### Import the libraries

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

### Read the dataset

In [5]:
```
df = pd.read_csv('Heart Disease data.csv')
df.head()
```

Out[5]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 52 | 1 | 0 | 125 | 212 | 0 | 1 | 168 | 0 | 1.0 | 2 | 2 | 3 | 0 |
| **1** | 53 | 1 | 0 | 140 | 203 | 1 | 0 | 155 | 1 | 3.1 | 0 | 0 | 3 | 0 |
| **2** | 70 | 1 | 0 | 145 | 174 | 0 | 1 | 125 | 1 | 2.6 | 0 | 0 | 3 | 0 |
| **3** | 61 | 1 | 0 | 148 | 203 | 0 | 1 | 161 | 0 | 0.0 | 2 | 1 | 3 | 0 |
| **4** | 62 | 0 | 0 | 138 | 294 | 1 | 1 | 106 | 0 | 1.9 | 1 | 3 | 2 | 0 |

In [6]:
```
df.shape
```

Out[6]:
```
(1025, 14)
```

## Data Preprocessing

### 1) Handling Null

In [7]:
```
df.isna().sum()
```

Out[7]:
```
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
```

### 2) Handling Duplicates

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
723
```

In [9]:
```
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

Out[9]:
```
0
```

### 3) Check data types

In [10]:
```
df.dtypes
```

Out[10]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

In [11]:
```
for i in df.columns:
    print(f'{i} - {df[i].nunique()}')
```

```
age - 41
sex - 2
cp - 4
trestbps - 49
chol - 152
fbs - 2
restecg - 3
thalach - 91
exang - 2
oldpeak - 40
slope - 3
ca - 5
thal - 4
target - 2
```

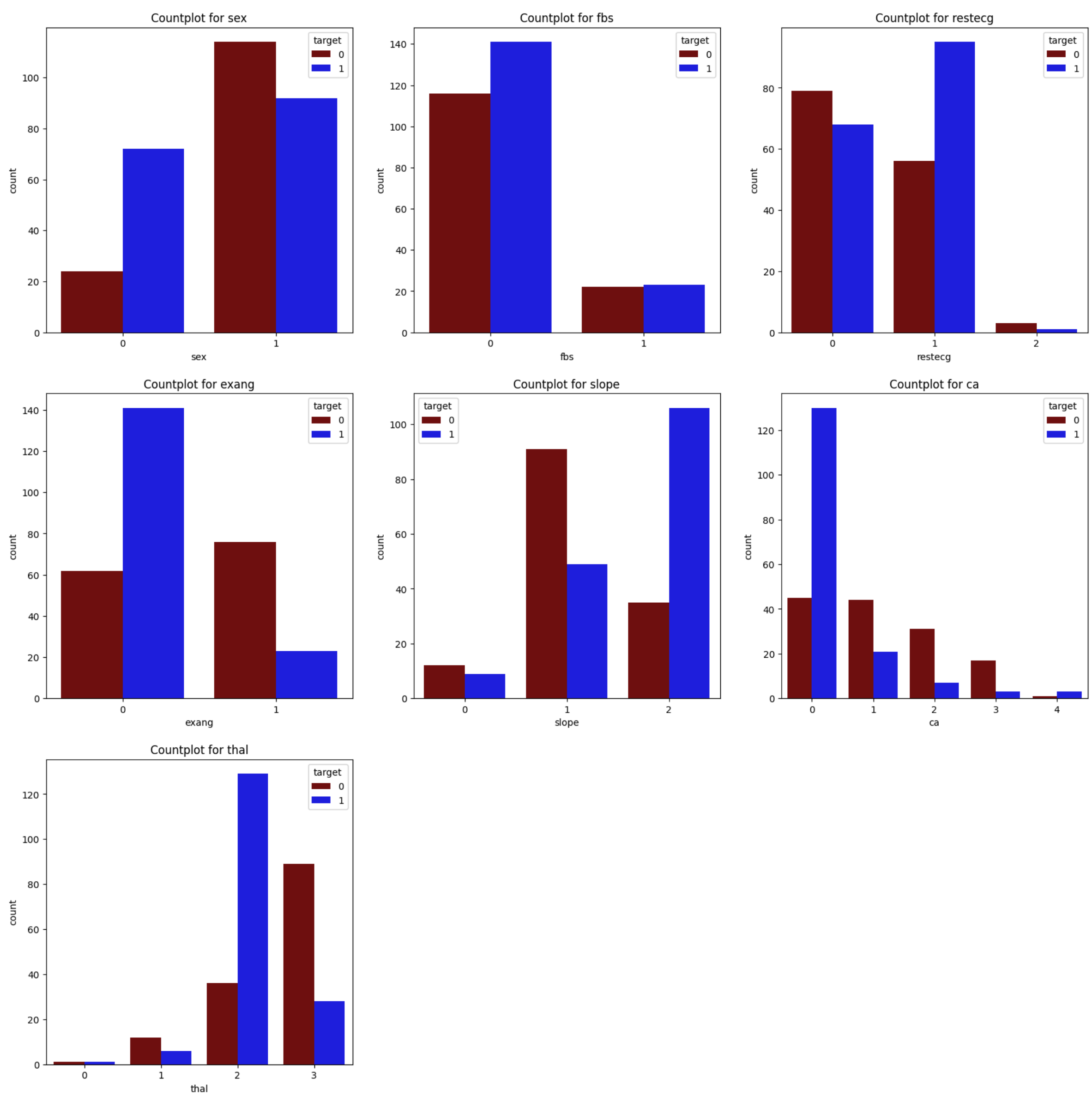
In [13]:
```
cat_cols = ['sex','fbs','restecg','exang','slope',
            'ca','thal']
cont_cols = ['age','trestbps','chol','thalach','oldpeak']
```

In [ ]:

## Bivariate Analysis

### Countplot

In [14]:
```
plt.figure(figsize=(20,20))
for i in range(0,len(cat_cols)):
    plt.subplot(3,3,i+1)
    sns.countplot(x=df[cat_cols[i]],hue= df['target'],palette=['maroon','blue'
    plt.title(f'Countplot for {cat_cols[i]}')
plt.show()
```

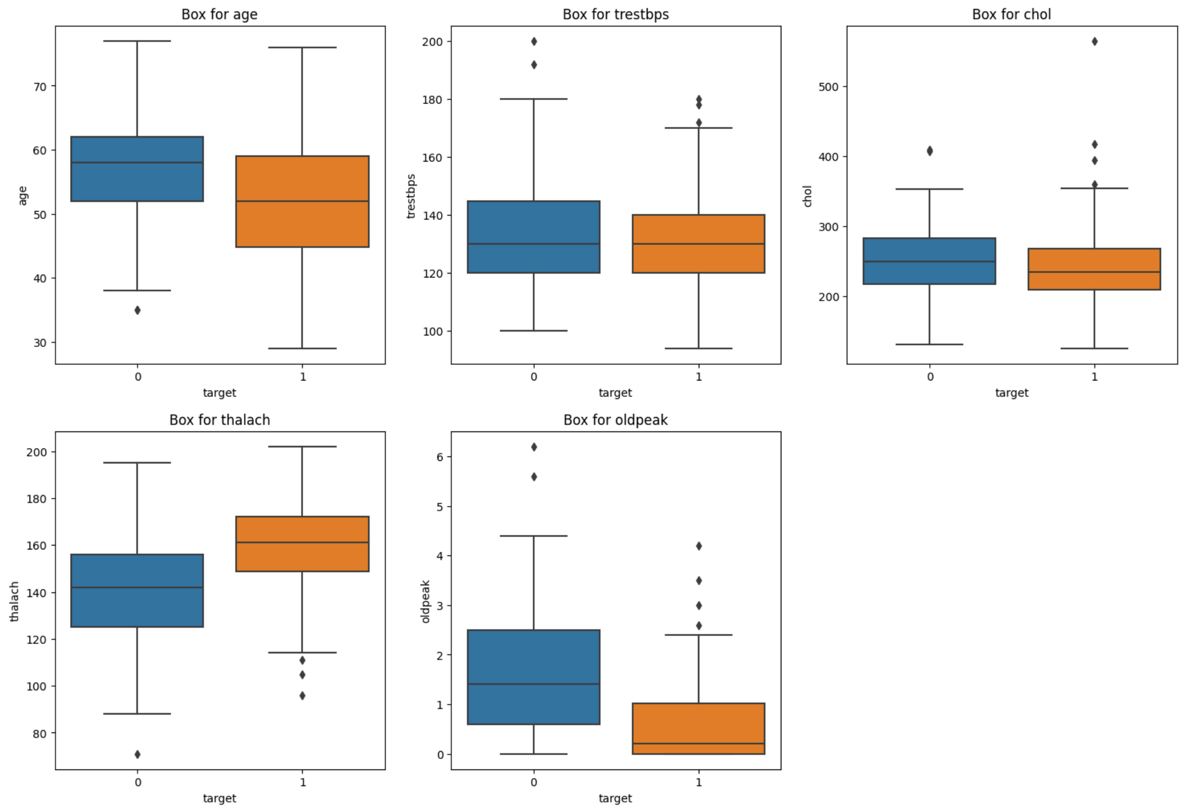


### Boxplot - cont vs target

In [15]:
```
print(cont_cols)
```

```
['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

In [16]:
```
plt.figure(figsize=(18,12))
for i in range(0,len(cont_cols)):
    plt.subplot(2,3,i+1)
    sns.boxplot(y=df[cont_cols[i]],x=df['target'])
    plt.title(f'Box for {cont_cols[i]}')
plt.show()
```




In [17]:
```
df.columns
```

Out[17]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [19]:
```
r1 = df.groupby('target')[['age','trestbps','chol','thalach']].agg(['min','max
r1
```

Out[19]:

| | age | | | trestbps | | | chol | | | thalach | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | max | mean | min | max | mean | min | max | mean | min | max | mean |
| target | | | | | | | | | | | | |
| 0 | 35 | 77 | 56.601449 | 100 | 200 | 134.398551 | 131 | 409 | 251.086957 | 71 | 195 | 139.1014 |
| 1 | 29 | 76 | 52.585366 | 94 | 180 | 129.250000 | 126 | 564 | 242.640244 | 96 | 202 | 158.3780 |

In [20]:
```
cont_cols
```

Out[20]: ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

In [21]:
```
r2 = df.groupby('target').agg({'age':['mean','median'],
                              'oldpeak':['min','max'],
                              'thalach':['min','max','mean']})
r2
```

Out[21]:

| | age | | oldpeak | | thalach | | |
|---|---|---|---|---|---|---|---|
| | mean | median | min | max | min | max | mean |
| target | | | | | | | |
| 0 | 56.601449 | 58.0 | 0.0 | 6.2 | 71 | 195 | 139.101449 |
| 1 | 52.585366 | 52.0 | 0.0 | 4.2 | 96 | 202 | 158.378049 |

### Find Age_bins based mean cholestrol for each target. use Pivot_table

In [22]:
```
df['age'].describe()  #where have we found age bins based mean????
```

Out[22]:
```
count    302.00000
mean      54.42053
std        9.04797
min       29.00000
25%       48.00000
50%       55.50000
75%       61.00000
max       77.00000
Name: age, dtype: float64
```

In [23]:
```
# Binning
df['Age_bins']  = pd.cut(df['age'],bins=list(range(25,85,5)))
df['Age_bins'].value_counts()
```

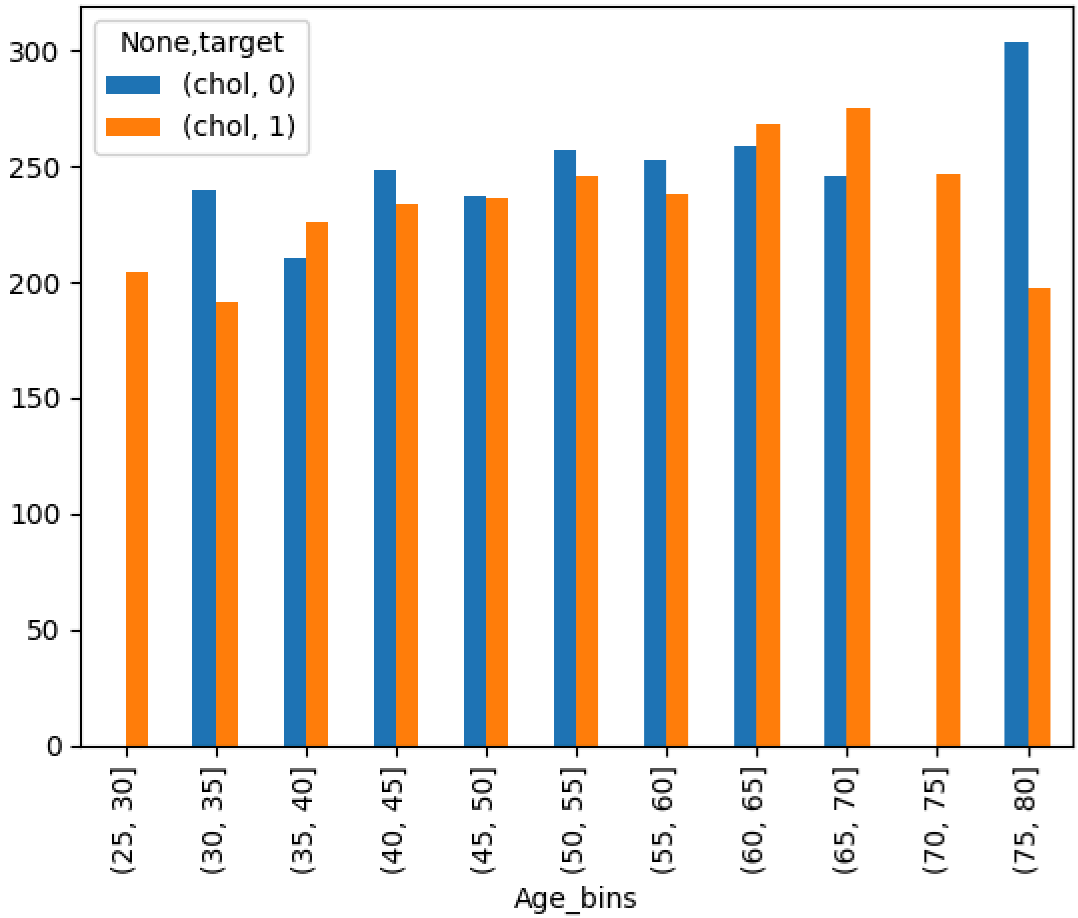
Out[23]:
```
(55, 60]    72
(50, 55]    57
(60, 65]    46
(40, 45]    45
(45, 50]    31
(65, 70]    27
(35, 40]    11
(30, 35]     6
(70, 75]     4
(75, 80]     2
(25, 30]     1
Name: Age_bins, dtype: int64
```

In [25]:
```
pt1 = pd.pivot_table(data=df,columns=['target'],index=['Age_bins'],values=['ch
pt1  # default agg used is mean
```

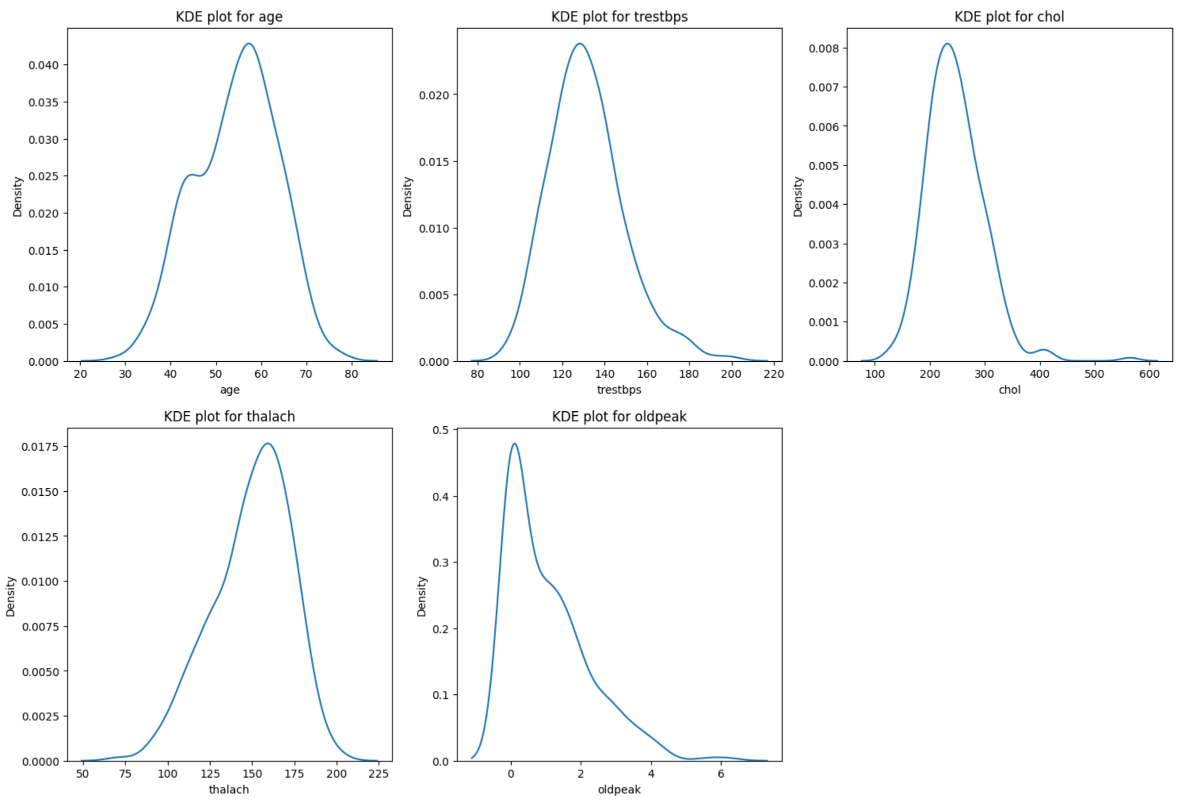
Out[25]:

| | chol | |
|---|---|---|
| target | 0 | 1 |
| Age_bins | | |
| (25, 30] | NaN | 204.000000 |
| (30, 35] | 240.000000 | 191.750000 |
| (35, 40] | 210.000000 | 225.571429 |
| (40, 45] | 248.100000 | 233.371429 |
| (45, 50] | 237.000000 | 236.166667 |
| (50, 55] | 257.350000 | 245.675676 |
| (55, 60] | 253.000000 | 237.703704 |
| (60, 65] | 258.678571 | 268.333333 |
| (65, 70] | 245.933333 | 275.333333 |
| (70, 75] | NaN | 246.250000 |
| (75, 80] | 304.000000 | 197.000000 |

In [26]:
```
pt1.plot(kind='bar')
plt.show()
```



In [27]:
```
plt.figure(figsize=(18,12))
for i in range(0,len(cont_cols)):
    plt.subplot(2,3,i+1)
    sns.kdeplot(x=df[cont_cols[i]])
    plt.title(f'KDE plot for {cont_cols[i]}')
plt.show()
```




## Correlation

In [28]:
```
df[cont_cols].head()
```

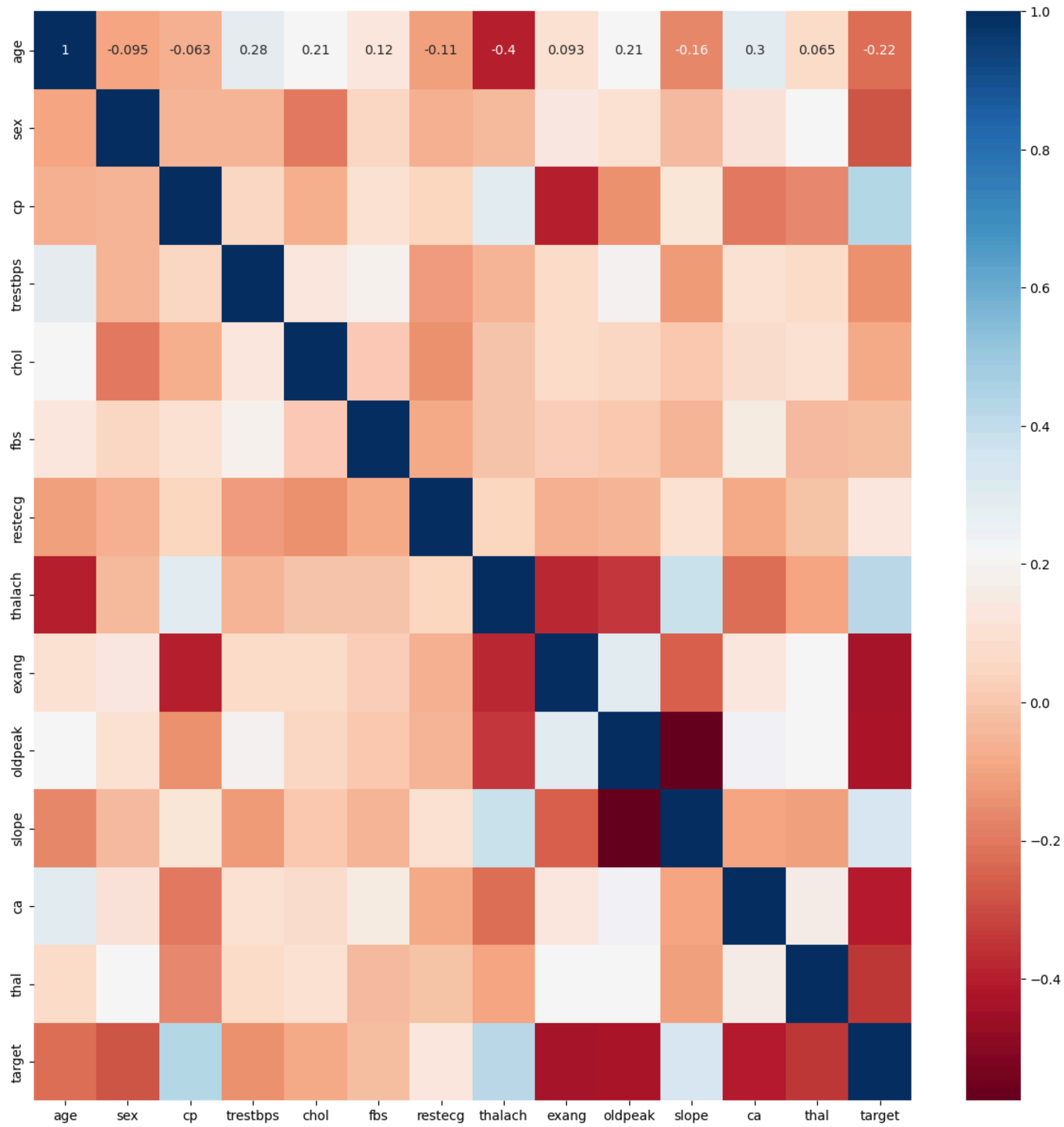
Out[28]:

| | age | trestbps | chol | thalach | oldpeak |
|---|---|---|---|---|---|
| **0** | 52 | 125 | 212 | 168 | 1.0 |
| **1** | 53 | 140 | 203 | 155 | 3.1 |
| **2** | 70 | 145 | 174 | 125 | 2.6 |
| **3** | 61 | 148 | 203 | 161 | 0.0 |
| **4** | 62 | 138 | 294 | 106 | 1.9 |

In [29]:
```
corr = df.corr()

plt.figure(figsize=(15,15))
sns.heatmap(corr,annot=True,cmap='RdBu')
plt.show()
```

```
C:\Users\win 8.1\AppData\Local\Temp\ipykernel_10732\3849691348.py:1: FutureW
arning: The default value of numeric_only in DataFrame.corr is deprecated. I
n a future version, it will default to False. Select only valid columns or s
pecify the value of numeric_only to silence this warning.
  corr = df.corr()
```




## Inference

Highly correlated features are not present

### Outlier Treatment

In [31]:
```
a = df[cont_cols].describe(percentiles=[0.01,0.02,0.05,0.95,0.97,0.98,0.99]).T
a = a.iloc[:,3:]
a
```

Out[31]:

| | min | 1% | 2% | 5% | 50% | 95% | 97% | 98% | 99% | max |
|---|---|---|---|---|---|---|---|---|---|---|
| **age** | 29.0 | 35.00 | 35.04 | 40.00 | 55.5 | 68.00 | 69.97 | 70.00 | 71.00 | 77.0 |
| **trestbps** | 94.0 | 100.00 | 101.02 | 108.00 | 130.0 | 160.00 | 170.00 | 177.92 | 180.00 | 200.0 |
| **chol** | 126.0 | 149.00 | 160.08 | 175.05 | 240.5 | 326.95 | 340.97 | 353.98 | 406.87 | 564.0 |
| **thalach** | 71.0 | 95.01 | 97.04 | 108.05 | 152.5 | 181.95 | 184.97 | 186.98 | 191.98 | 202.0 |
| **oldpeak** | 0.0 | 0.00 | 0.00 | 0.00 | 0.8 | 3.40 | 3.60 | 4.00 | 4.20 | 6.2 |

In [32]:
```
def outlier_treatment(x):
    x = x.clip(upper=x.quantile(0.99))
    x = x.clip(lower=x.quantile(0.01))
    return x
```

In [33]:
```
df1 = df.copy()
```

In [34]:
```
df[cont_cols] = df[cont_cols].apply(outlier_treatment)
```

In [35]:
```
a = df[cont_cols].describe(percentiles=[0.01,0.02,0.05,0.95,0.97,0.98,0.99]).T
a = a.iloc[:,3:]
a
```

Out[35]:

| | min | 1% | 2% | 5% | 50% | 95% | 97% | 98% | 99% | max |
|---|---|---|---|---|---|---|---|---|---|---|
| **age** | 35.00 | 35.0000 | 35.04 | 40.00 | 55.5 | 68.00 | 69.97 | 70.00 | 71.0000 | 71.00 |
| **trestbps** | 100.00 | 100.0000 | 101.02 | 108.00 | 130.0 | 160.00 | 170.00 | 177.92 | 180.0000 | 180.00 |
| **chol** | 149.00 | 149.0000 | 160.08 | 175.05 | 240.5 | 326.95 | 340.97 | 353.98 | 406.7413 | 406.87 |
| **thalach** | 95.01 | 95.0199 | 97.04 | 108.05 | 152.5 | 181.95 | 184.97 | 186.98 | 191.9602 | 191.98 |
| **oldpeak** | 0.00 | 0.0000 | 0.00 | 0.00 | 0.8 | 3.40 | 3.60 | 4.00 | 4.2000 | 4.20 |

### Select x and y

In [36]:
```
df.columns
```

Out[36]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target', 'Age_bins'],
      dtype='object')
```

In [37]:
```
x = df.drop(['target', 'Age_bins'],axis=1)
y = df['target']
print(x.shape)
print(y.shape)
```

```
(302, 13)
(302,)
```

### Split data into train and test

In [38]:
```
from sklearn.model_selection import train_test_split
```

In [39]:
```
x_train,x_test,y_train,y_test  = train_test_split(x,y,test_size=0.3,random_sta
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(211, 13)
(91, 13)
(211,)
(91,)
```

### Function to evalaute model

In [40]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_sc
from sklearn.metrics import precision_score,recall_score
```

In [41]:
```
def eval_model(ytest,ypred):
    cm = confusion_matrix(ytest,ypred)
    cr = classification_report(ytest,ypred)
    print('Confusion Matrix\n',cm)
    print('Classification Report\n',cr)

def gen_res(model,xtrain,xtest,ytrain,ytest,ypred,model_name):
    eval_model(ytest,ypred)
    train_acc = model.score(xtrain,ytrain)  # Train Acc
    test_acc = model.score(xtest,ytest)     # Test Acc
    pre1 = precision_score(ytest,ypred)  # pre score = 1
    rec1 = recall_score(ytest,ypred)     # rec score = 1
    res = pd.DataFrame({'Train Acc':train_acc,'Test Acc':test_acc,
                       'Pre1':pre1,'Rec1':rec1},index=[model_name])
    return res
```

### Train the model

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
```

### Model-1 : DT1

In [43]:
```
dt1 = DecisionTreeClassifier(criterion='gini',random_state=25)
dt1.fit(x_train,y_train)
```

Out[43]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=25)
```

In [44]:
```
ypred_dt1 = dt1.predict(x_test)
dt1_res = gen_res(dt1,x_train,x_test,y_train,y_test,ypred_dt1,'DT1(gini)')
dt1_res
```

```
Confusion Matrix
 [[37 11]
 [10 33]]
Classification Report
               precision    recall  f1-score   support

           0       0.79      0.77      0.78        48
           1       0.75      0.77      0.76        43

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

Out[44]:

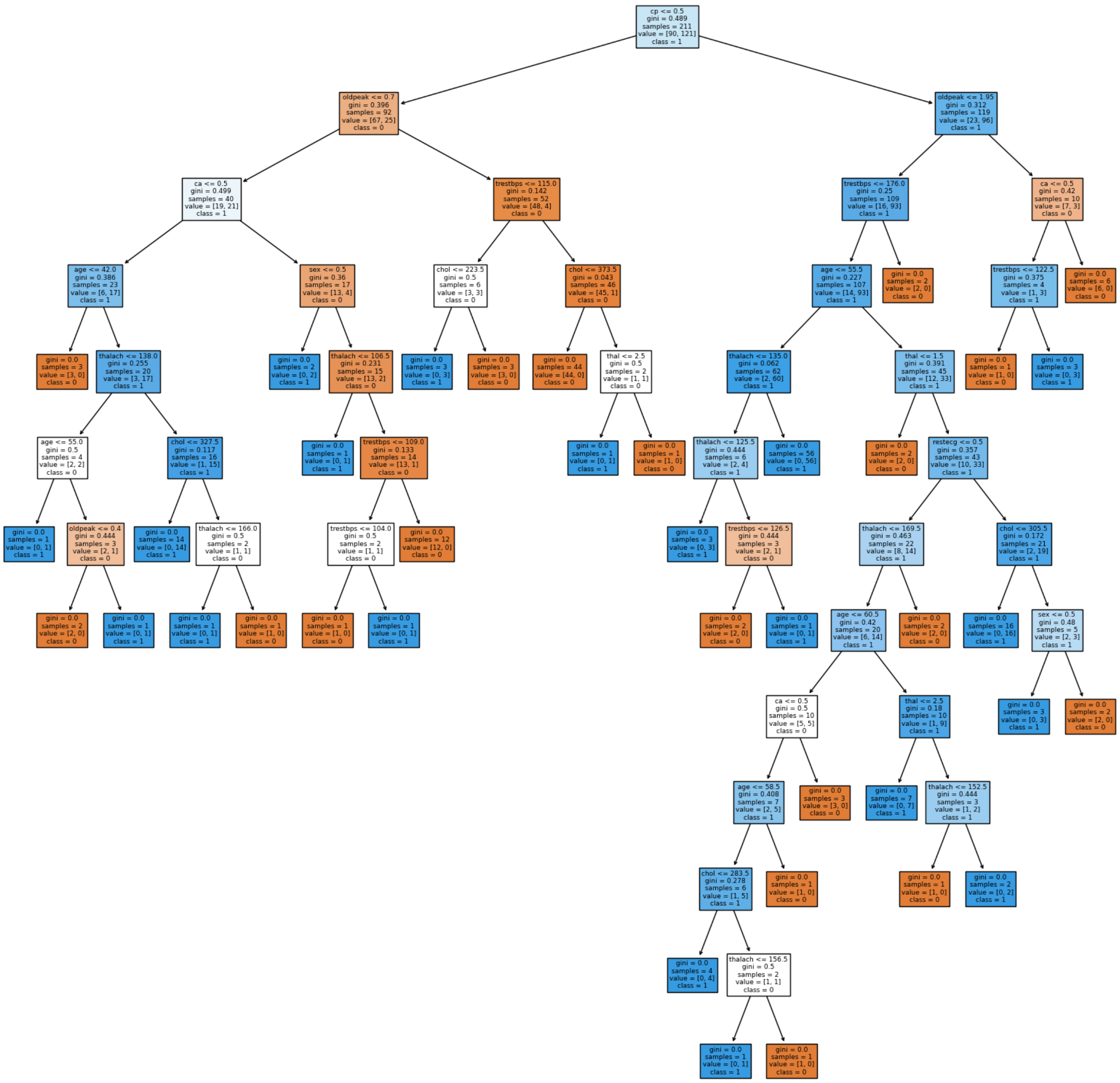
| | Train Acc | Test Acc | Pre1 | Rec1 |
|---|---|---|---|---|
| **DT1(gini)** | 1.0 | 0.769231 | 0.75 | 0.767442 |

### Inference

Overfitting exists

In [45]:
```
from sklearn.tree import plot_tree
```

In [46]:
```
cn = ['0','1']
plt.figure(figsize=(20,20))
plot_tree(dt1,feature_names=x_train.columns,class_names=cn,filled=True)
plt.show()
```



### Model-2 : DT2

In [47]:
```
dt2 = DecisionTreeClassifier(criterion='gini',max_depth=6,min_samples_split=10
dt2.fit(x_train,y_train)
```

Out[47]:
```
DecisionTreeClassifier

DecisionTreeClassifier(max_depth=6, min_samples_split=10, random_state=25)
```

In [48]:
```
ypred_dt2 = dt2.predict(x_test)
dt2_res =  gen_res(dt2,x_train,x_test,y_train,y_test,ypred_dt2,'DT2(gini,md=6,
dt2_res
```

```
Confusion Matrix
 [[36 12]
 [ 9 34]]
Classification Report
               precision    recall  f1-score   support

           0       0.80      0.75      0.77        48
           1       0.74      0.79      0.76        43

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

Out[48]:

| | Train Acc | Test Acc | Pre1 | Rec1 |
|---|---|---|---|---|
| **DT2(gini,md=6,mss=15)** | 0.900474 | 0.769231 | 0.73913 | 0.790698 |

### Model-3 : DT3

In [49]:
```
dt3 = DecisionTreeClassifier(criterion='entropy',max_depth=7,
                             min_samples_split=20,random_state=25)
dt3.fit(x_train,y_train)
```

Out[49]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=7, min_samples_split=
20,
                       random_state=25)
```

In [50]:
```
ypred_dt3 = dt3.predict(x_test)
dt3_res =  gen_res(dt3,x_train,x_test,y_train,y_test,ypred_dt2,'DT3(ent,md=7,m
dt3_res
```

```
Confusion Matrix
 [[36 12]
 [ 9 34]]
Classification Report
               precision    recall  f1-score   support

           0       0.80      0.75      0.77        48
           1       0.74      0.79      0.76        43

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

Out[50]:

| | Train Acc | Test Acc | Pre1 | Rec1 |
|---|---|---|---|---|
| **DT3(ent,md=7,mss=20)** | 0.895735 | 0.769231 | 0.73913 | 0.790698 |

#### Inference

By changing the hyperparameters(criterion,max_depth,min_samples_split), the performace of the model changes. So we nned to tune the hyperparameters

#### HyperParameter Tuning

1. GridSearchCV
   a) It consumes a lot of time. Time inefficient.
   b) It works on all combination of hyperparameters and then it returns the best set of hyperparemeters that generated the best results on different splits.
2. RandomizedSearchCV
   a) It consumes a comparatively less time. Time efficient.
   b) It works on all random subset of hyperparameters and then it returns the best set of hyperparemeters that generated the best results.

In [51]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [52]:
```
hparams = {'criterion':['gini','entropy'],
          'max_depth':[4,5,6,7,8],
          'min_samples_split':[8,10,12,15,20]}

# 2 * 5 * 5 = 50

# 'min_samples_split':list(range(7,16))
```

#### GridSearchCV

In [53]:
```
dt_model = DecisionTreeClassifier(random_state=0)

gs1 = GridSearchCV(dt_model,param_grid=hparams,scoring='accuracy',cv=5)
gs1.fit(x_train,y_train)
```

Out[53]:
```
GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [54]:
```
print(gs1.best_params_)
print(gs1.best_estimator_)
print(gs1.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 8}
DecisionTreeClassifier(max_depth=6, min_samples_split=8, random_state=0)
0.8059800664451828
```

#### Analysis of Grid SearchCV Results

In [55]:
```
gs1_res = pd.DataFrame(gs1.cv_results_)
gs1_res.head()
```

Out[55]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_criterion | param_max_de |
|---|---|---|---|---|---|---|
| **0** | 0.004999 | 0.000963 | 0.002996 | 0.000801 | gini | |
| **1** | 0.005593 | 0.004036 | 0.003808 | 0.004682 | gini | |
| **2** | 0.004529 | 0.005784 | 0.000610 | 0.000805 | gini | |
| **3** | 0.003125 | 0.006250 | 0.003121 | 0.006243 | gini | |
| **4** | 0.003125 | 0.006250 | 0.003124 | 0.006248 | gini | |

In [56]:
```
print(gs1_res.shape)
print(gs1_res.columns)
```

```
(50, 16)
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_criterion', 'param_max_depth', 'param_min_samples_split',
       'params', 'split0_test_score', 'split1_test_score', 'split2_test_scor
e',
       'split3_test_score', 'split4_test_score', 'mean_test_score',
       'std_test_score', 'rank_test_score'],
      dtype='object')
```

In [57]:
```
gs1_res = gs1_res[['param_criterion', 'param_max_depth', 'param_min_samples_sp
                  'mean_test_score','rank_test_score']]
gs1_res.head()
```

Out[57]:

| | param_criterion | param_max_depth | param_min_samples_split | params | mean_test_score |
|---|---|---|---|---|---|
| **0** | gini | 4 | 8 | {'criterion': 'gini', 'max_depth': 4, 'min_sam... | 0.772425 |
| **1** | gini | 4 | 10 | {'criterion': 'gini', 'max_depth': 4, 'min_sam... | 0.772425 |
| **2** | gini | 4 | 12 | {'criterion': 'gini', 'max_depth': 4, 'min_sam... | 0.777187 |
| **3** | gini | 4 | 15 | {'criterion': 'gini', 'max_depth': 4, 'min_sam... | 0.777076 |
| **4** | gini | 4 | 20 | {'criterion': 'gini', 'max_depth': 4, 'min_sam... | 0.762791 |

In [58]:
```
gs1_res.sort_values('rank_test_score').head()  # asc order or rank
```

Out[58]:

| | param_criterion | param_max_depth | param_min_samples_split | params | mean_test_score |
|---|---|---|---|---|---|
| **10** | gini | 6 | 8 | {'criterion': 'gini', 'max_depth': 6, 'min_sam... | 0.805980 |
| **20** | gini | 8 | 8 | {'criterion': 'gini', 'max_depth': 8, 'min_sam... | 0.796456 |
| **15** | gini | 7 | 8 | {'criterion': 'gini', 'max_depth': 7, 'min_sam... | 0.796456 |
| **5** | gini | 5 | 8 | {'criterion': 'gini', 'max_depth': 5, 'min_sam... | 0.796456 |
| **6** | gini | 5 | 10 | {'criterion': 'gini', 'max_depth': 5, 'min_sam... | 0.786932 |

In [59]:
```
print(gs1.best_params_)
# {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10}
# {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 12}
```

```
{'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 8}
```

### Model 4 : DT4 (based on GridSearchCV Results)

In [60]:
```
dt4 = DecisionTreeClassifier(**gs1.best_params_)
dt4.fit(x_train,y_train)
```

Out[60]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, min_samples_split=8)
```

In [61]:
```
ypred_dt4 = dt4.predict(x_test)
dt4_res = gen_res(dt4,x_train,x_test,y_train,y_test,ypred_dt4,'DT4_GS1(ent,md=
dt4_res
```

```
Confusion Matrix
 [[35 13]
 [10 33]]
Classification Report
               precision    recall  f1-score   support

           0       0.78      0.73      0.75        48
           1       0.72      0.77      0.74        43

    accuracy                           0.75        91
   macro avg       0.75      0.75      0.75        91
weighted avg       0.75      0.75      0.75        91
```

Out[61]:

| | Train Acc | Test Acc | Pre1 | Rec1 |
|---|---|---|---|---|
| **DT4_GS1(ent,md=5,mss=10)** | 0.900474 | 0.747253 | 0.717391 | 0.767442 |

### Randomized SearchCV

In [62]:
```
dt_model1 = DecisionTreeClassifier(random_state=0)
rs1 = RandomizedSearchCV(dt_model1,param_distributions=hparams,scoring='accura
                        n_iter=20)
# n_iter = number of random cobintaion to select best hparams from
rs1.fit(x_train,y_train)
```

Out[62]:

```
▸ RandomizedSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [63]:
```
print(rs1.best_params_)
print(rs1.best_estimator_)
print(rs1.best_score_)
```

```
{'min_samples_split': 8, 'max_depth': 6, 'criterion': 'gini'}
DecisionTreeClassifier(max_depth=6, min_samples_split=8, random_state=0)
0.8059800664451828
```

In [64]:
```
rs1_res = pd.DataFrame(rs1.cv_results_)
print(rs1_res.shape)
```

```
(20, 16)
```

### Log Reg

In [65]:
```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(x_train,y_train)
```

Out[65]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [66]:
```
ypred_lr1 = lr1.predict(x_test)
lr1_res = gen_res(lr1,x_train,x_test,y_train,y_test,ypred_lr1,'LogReg')
```

```
Confusion Matrix
 [[38 10]
 [ 5 38]]
Classification Report
               precision    recall  f1-score   support

           0       0.88      0.79      0.84        48
           1       0.79      0.88      0.84        43

    accuracy                           0.84        91
   macro avg       0.84      0.84      0.84        91
weighted avg       0.84      0.84      0.84        91
```

In [67]:
```
model_res = pd.concat([dt1_res,dt2_res,dt3_res,dt4_res,lr1_res])
model_res
```

Out[67]:

| | Train Acc | Test Acc | Pre1 | Rec1 |
|---|---|---|---|---|
| **DT1(gini)** | 1.000000 | 0.769231 | 0.750000 | 0.767442 |
| **DT2(gini,md=6,mss=15)** | 0.900474 | 0.769231 | 0.739130 | 0.790698 |
| **DT3(ent,md=7,mss=20)** | 0.895735 | 0.769231 | 0.739130 | 0.790698 |
| **DT4_GS1(ent,md=5,mss=10)** | 0.900474 | 0.747253 | 0.717391 | 0.767442 |
| **LogReg** | 0.853081 | 0.835165 | 0.791667 | 0.883721 |

In [68]:
```
# Recall = TP/(TP+FN)
```

### Saving the Model

In [72]:
```
import pickle
```

In [73]:
```
pickle.dump(lr1,open('heart_disease_lr.pkl','wb'))  # write binary
```

In [76]:
```
model = pickle.load(open('heart_disease_lr.pkl','rb'))
```

In [ ]:

```
```