Heart Patient Dataset for Classification.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('heart.csv')
pd.set_option('display.max_columns', None)
```

In [3]:
```
data.head()
```

Out[3]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40.0	M	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	Up	0
1	49.0	F	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	Flat	1
2	NaN	M	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	Up	0
3	NaN	F	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	Flat	1
4	NaN	M	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	Up	0

In [4]:
```
data.tail()
```

Out[4]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
913	45.0	M	TA	110.0	264.0	0.0	Normal	132.0	N	1.2	Flat	1
914	68.0	M	ASY	144.0	193.0	1.0	Normal	141.0	N	3.4	Flat	1
915	57.0	M	ASY	130.0	131.0	0.0	Normal	115.0	Y	1.2	Flat	1
916	57.0	F	ATA	130.0	236.0	0.0	LVH	174.0	N	0.0	Flat	1
917	38.0	M	NAP	138.0	175.0	0.0	Normal	173.0	N	0.0	Up	0

In [5]:
```
data.shape
```

Out[5]:
```
(918, 12)
```

In [6]:
```
data.columns
```

Out[6]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             900 non-null    float64
 1   Sex             918 non-null    object 
 2   ChestPainType   909 non-null    object 
 3   RestingBP       917 non-null    float64
 4   Cholesterol     902 non-null    float64
 5   FastingBS       909 non-null    float64
 6   RestingECG      918 non-null    object 
 7   MaxHR           909 non-null    float64
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        911 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(6), int64(1), object(5)
memory usage: 86.2+ KB
```

In [8]:
```
round(data.describe(),2)
```

Out[8]:

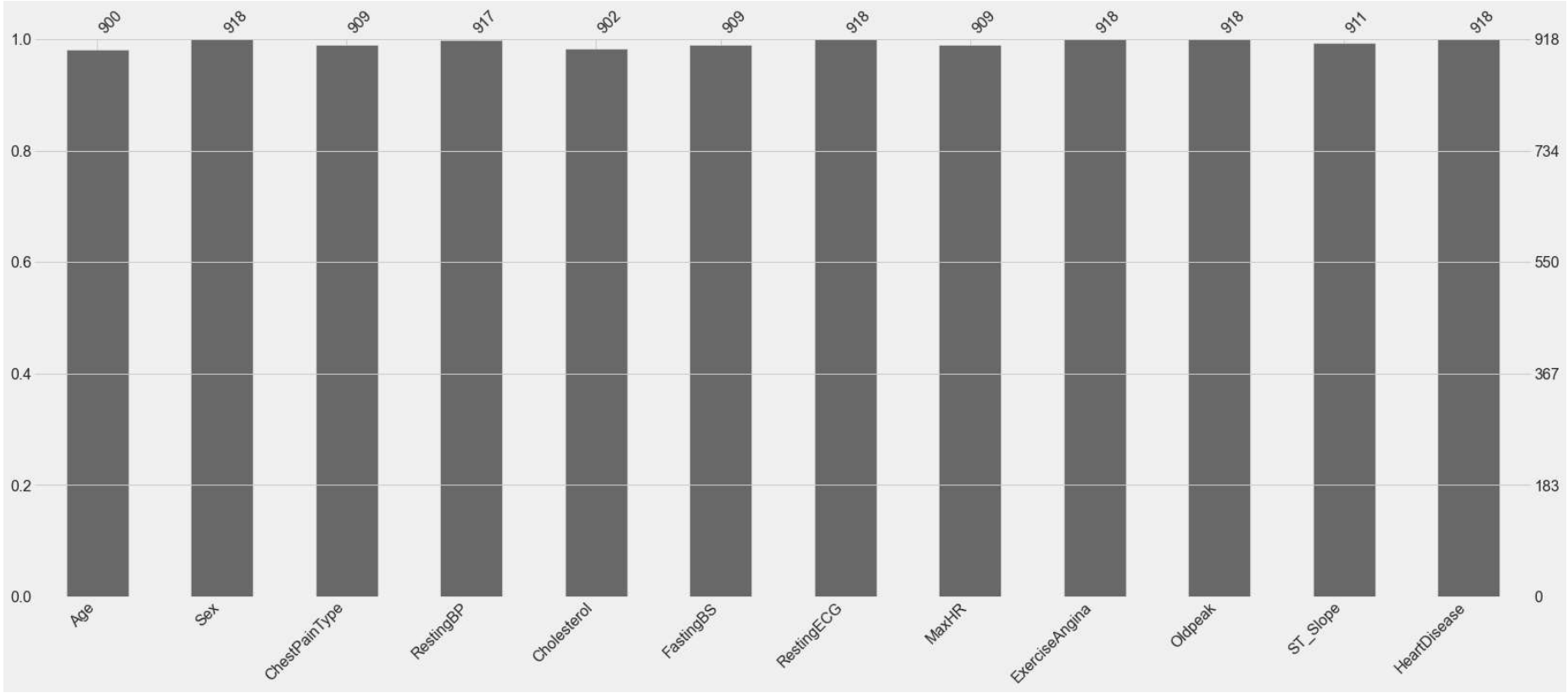
	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	900.00	917.00	902.00	909.00	909.00	918.00	918.00
mean	53.62	132.42	198.04	0.24	136.93	0.89	0.55
std	9.43	18.51	110.05	0.42	25.54	1.07	0.50
min	28.00	0.00	0.00	0.00	60.00	-2.60	0.00
25%	47.00	120.00	171.25	0.00	120.00	0.00	0.00
50%	54.00	130.00	222.50	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]:
```
data.dtypes
```

Out[9]:
```
Age               float64
Sex                object
ChestPainType      object
RestingBP         float64
Cholesterol       float64
FastingBS         float64
RestingECG         object
MaxHR             float64
ExerciseAngina     object
Oldpeak           float64
ST_Slope           object
HeartDisease        int64
dtype: object
```

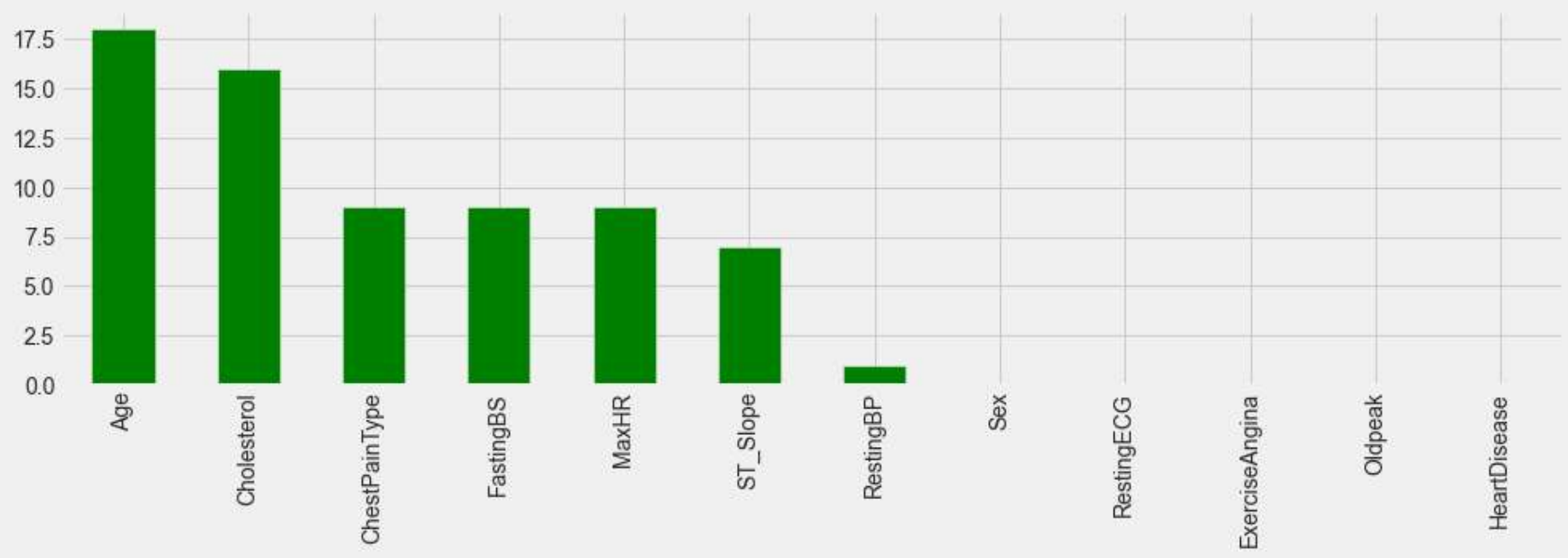
In [10]:
```
import missingno as msno
msno.bar(data)
```

Out[10]: <AxesSubplot:>



- Here, we can see that Age,Cholestrol,ChestPainType,fastingBS,MaxHR,ST_Slope & RestingBP having Null values so we need to fill them.

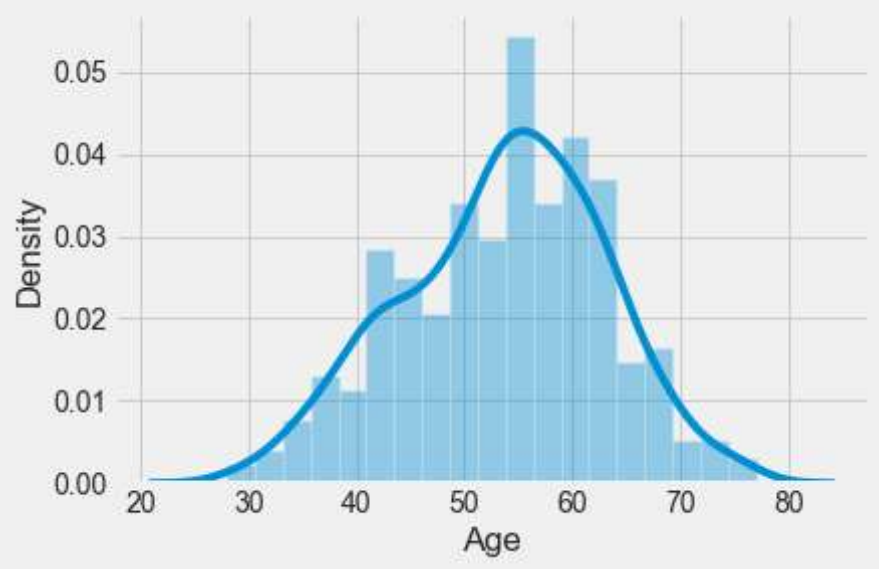
In [11]:
```
plt.figure(figsize=(15,4))
data.isnull().sum().sort_values(ascending=False).plot(kind='bar',color='g')
plt.show()
```



We have missing values so first we need to fill that missing values with the help of our statstical analysis.

In [12]:
```
for i in data:
    if data[i].dtypes=="int64" or data[i].dtypes=="float64":
        print('This is related:-',i)
        print()
        sns.distplot(data[i])
        plt.show()
        print()
        sns.boxplot(data[i])
        plt.show()
        print('**********************************************************')
```

```
This is related:- Age
```



In [13]:
```
for i in data:
    if data[i].dtypes=="int64" or data[i].dtypes=="float64":
        print('This is related:-',i)
        print("Skeness",round(data[i].skew(),2) )
        print("Kurtosis",round(data[i].kurtosis(),2) )
        print('mean',round(data[i].mean(),2))
        print('Median',round(data[i].median(),2))
        print('**********************************************************')
```

```
This is related:- Age
Skeness -0.21
Kurtosis -0.37
mean 53.62
Median 54.0
**********************************************************
This is related:- RestingBP
Skeness 0.18
Kurtosis 3.28
mean 132.42
Median 130.0
**********************************************************
This is related:- Cholesterol
Skeness -0.59
Kurtosis 0.08
mean 198.04
Median 222.5
**********************************************************
This is related:- FastingBS
Skeness 1.25
Kurtosis -0.44
mean 0.24
Median 0.0
**********************************************************
This is related:- MaxHR
Skeness -0.16
Kurtosis -0.45
mean 136.93
Median 138.0
**********************************************************
This is related:- Oldpeak
Skeness 1.02
```

```
Kurtosis 1.2
mean 0.89
Median 0.6
**********************************************************
This is related:- HeartDisease
Skeness -0.22
Kurtosis -1.96
mean 0.55
Median 1.0
**********************************************************
```

In [14]:
```python
def skewed_Check(col):
    if col.dtypes=="int64" or col.dtypes== "float64":
        if col.median()>col.mean():
            return 'Left Skewed data'
        else:
            return 'Right Skewed data'
    else:
        return "Mode :- "+col.mode()
```

In [15]:
```
for i in data:
    print(i,"-------",skewed_Check(data[i]))
    print()
    print("********************************")
```

```
Age ------- Left Skewed data

********************************
Sex ------- 0    Mode :- M
dtype: object

********************************
ChestPainType ------- 0    Mode :- ASY
dtype: object

********************************
RestingBP ------- Right Skewed data

********************************
Cholesterol ------- Left Skewed data

********************************
FastingBS ------- Right Skewed data

********************************
RestingECG ------- 0    Mode :- Normal
dtype: object

********************************
MaxHR ------- Left Skewed data

********************************
ExerciseAngina ------- 0    Mode :- N
dtype: object

********************************
Oldpeak ------- Right Skewed data

********************************
ST_Slope ------- 0    Mode :- Flat
dtype: object
```

```
******************************
HeartDisease ------- Left Skewed data

******************************
```

- If any numerical column contain the null values and having left skewed data then it will be fill by its mean.
- If any numerical column contain the null values and having right skewed data then it will be fill by its median.
- If any categorical column contain the null values then it will fill by its mode.
- There is another method to impute the missing values called as `KNNImputer` since we are going to traditional method.

- Since age having Normal distribution and there is no big gap between the mean & median but statstically its right skwed so we are imputing age with its median value.

In [16]:
```
data['Age'].fillna(data['Age'].median(),inplace=True)
```

- Since Cholesterol having multinomial distribution and there is big gap between the mean & median but statstically its left skwed so we are imputing age with its mean value.

In [17]:
```
data['Cholesterol'].fillna(data['Cholesterol'].mean(),inplace=True)
```

- Since ChestPainType having object as dtypes and there is mode is ASY so we we are imputing ChestPain with its mode value.

In [18]:
```
data['ChestPainType'].fillna('ASY',inplace=True)
```

- Since FastingBS having categories form as and there is highly frequent is 0 so we we are imputing FastingBS with its 0 value.

In [19]:
```
data['FastingBS'].fillna(0,inplace=True)
```

- Since MaxHR having Normal distribution and there is no big gap between the mean & median but statstically its left skwed so we are imputing MaxHR with its mean value.

In [20]:
```
data['MaxHR'].fillna(data['MaxHR'].mean(),inplace=True)
```

- Since ST_Slope having object as dtypes and there is mode is Flat so we we are imputing Flat with its mode value.

In [21]:
```
data['ST_Slope'].fillna('Flat',inplace=True)
```

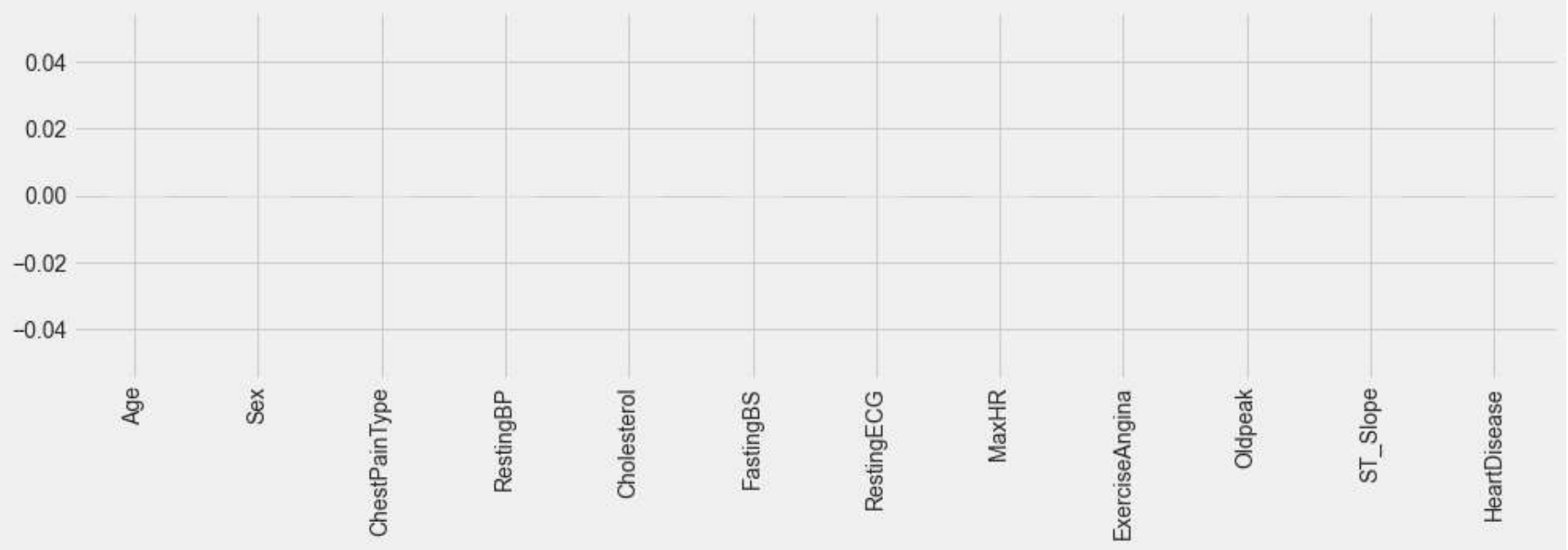
- Since RestingBP having Normal distribution and there is no big gap between the mean & median but statstically its right skwed so we are imputing RestingBP with its median value.

In [22]:
```
data['RestingBP'].fillna(data['RestingBP'].median(),inplace=True)
```

In [23]:
```
data.isnull().sum()
```

Out[23]:
```
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
```

In [24]:
```
plt.figure(figsize=(15,4))
data.isnull().sum().sort_values(ascending=False).plot(kind='bar',color='g')
plt.show()
```



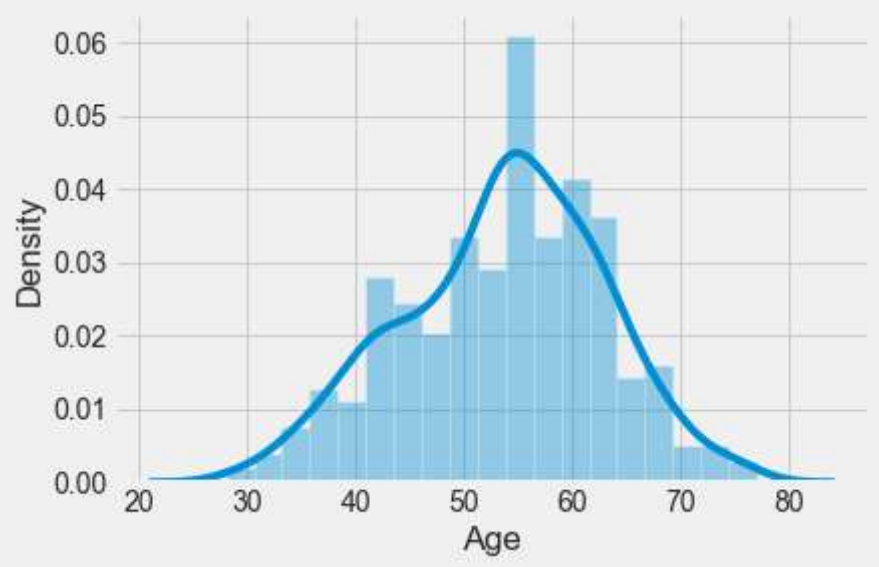
After filling all null value we are going recheck their distribition.

In [25]:
```
data.columns
```

Out[25]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

In [26]:
```
for i in data:
    if data[i].dtypes=="int64" or data[i].dtypes=="float64":
        print('This is related:-',i)
        print()
        sns.distplot(data[i])
        plt.show()
        print()
        sns.boxplot(data[i])
        plt.show()
        print('************************************************************')
```

```
This is related:- Age
```



In [27]:
```
for i in data:
    if data[i].dtypes=="int64" or data[i].dtypes=="float64":
        print('This is related:-',i)
        print("Skeness",round(data[i].skew(),2) )
        print("Kurtosis",round(data[i].kurtosis(),2) )
        print('mean',round(data[i].mean(),2))
        print('Median',round(data[i].median(),2))
        print('Mode',data[i].mode())
        print('**********************************************************')
    else:
        print('This is related:-',i)
        print('Mode',data[i].mode())
        print('**********************************************************')
```

```
This is related:- Age
Skeness -0.21
Kurtosis -0.31
mean 53.63
Median 54.0
Mode 0    54.0
dtype: float64
**********************************************************
This is related:- Sex
Mode 0    M
dtype: object
**********************************************************
This is related:- ChestPainType
Mode 0    ASY
dtype: object
**********************************************************
This is related:- RestingBP
Skeness 0.18
Kurtosis 3.29
mean 132.42
Median 130.0
Mode 0    120.0
dtype: float64
**********************************************************
This is related:- Cholesterol
Skeness -0.6
Kurtosis 0.13
```

```
mean 198.04
Median 221.0
Mode 0    0.0
dtype: float64
**********************************************************
This is related:- FastingBS
Skeness 1.26
Kurtosis -0.4
mean 0.23
Median 0.0
Mode 0    0.0
dtype: float64
**********************************************************
This is related:- RestingECG
Mode 0    Normal
dtype: object
**********************************************************
This is related:- MaxHR
Skeness -0.16
Kurtosis -0.43
mean 136.93
Median 138.0
Mode 0    150.0
dtype: float64
**********************************************************
This is related:- ExerciseAngina
Mode 0    N
dtype: object
**********************************************************
This is related:- Oldpeak
Skeness 1.02
Kurtosis 1.2
mean 0.89
Median 0.6
Mode 0    0.0
dtype: float64
**********************************************************
This is related:- ST_Slope
Mode 0    Flat
dtype: object
**********************************************************
This is related:- HeartDisease
```

```
Skeness -0.22
Kurtosis -1.96
mean 0.55
Median 1.0
Mode 0    1
dtype: int64
**********************************************************
```

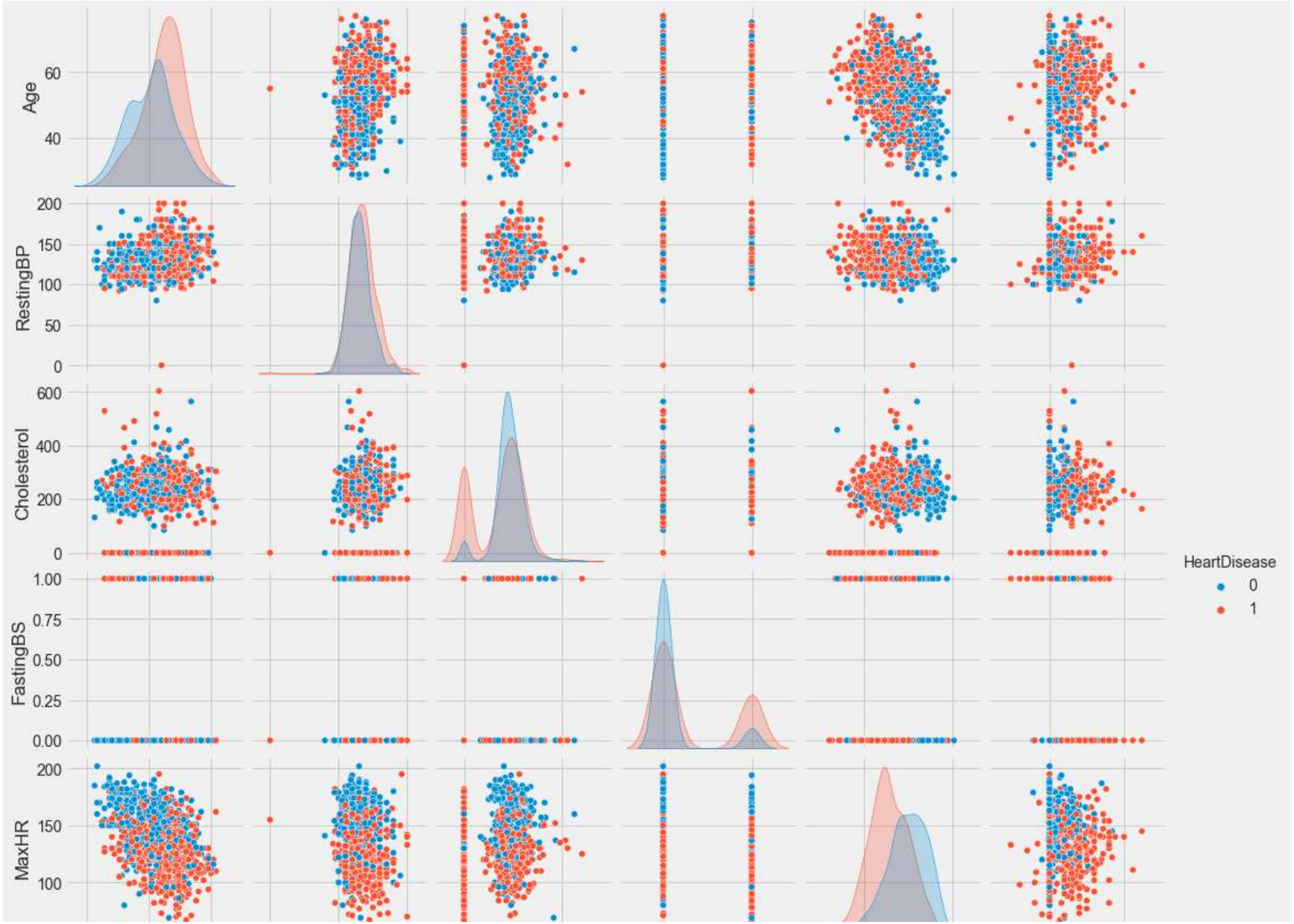
In [28]:
```
round(data.describe(),3) #from the descritive analysis we can not see any outliers in the data.
```

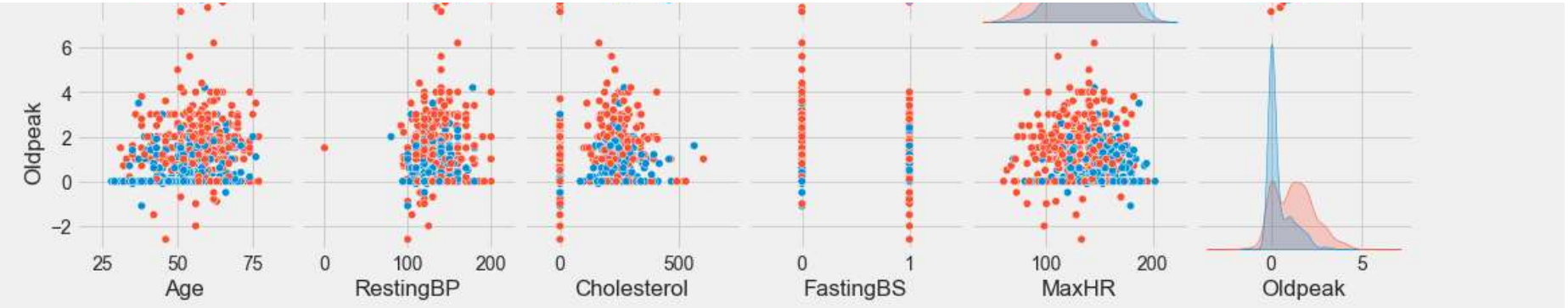
Out[28]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000	918.000	918.000	918.000	918.000	918.000	918.000
mean	53.631	132.418	198.037	0.233	136.934	0.887	0.553
std	9.335	18.500	109.081	0.423	25.412	1.067	0.497
min	28.000	0.000	0.000	0.000	60.000	-2.600	0.000
25%	47.000	120.000	173.250	0.000	120.000	0.000	0.000
50%	54.000	130.000	221.000	0.000	138.000	0.600	1.000
75%	60.000	140.000	266.000	0.000	156.000	1.500	1.000
max	77.000	200.000	603.000	1.000	202.000	6.200	1.000

Lets understand the data with help of EDA & Stastics

In [29]:
```
sns.pairplot(data,hue='HeartDisease')
plt.show()
```





- From this is visulization, we can undertand data is not so much linearly seperable.
- There is lot non-linearity avalible inside the data so in this logistics regression will give us better performance.

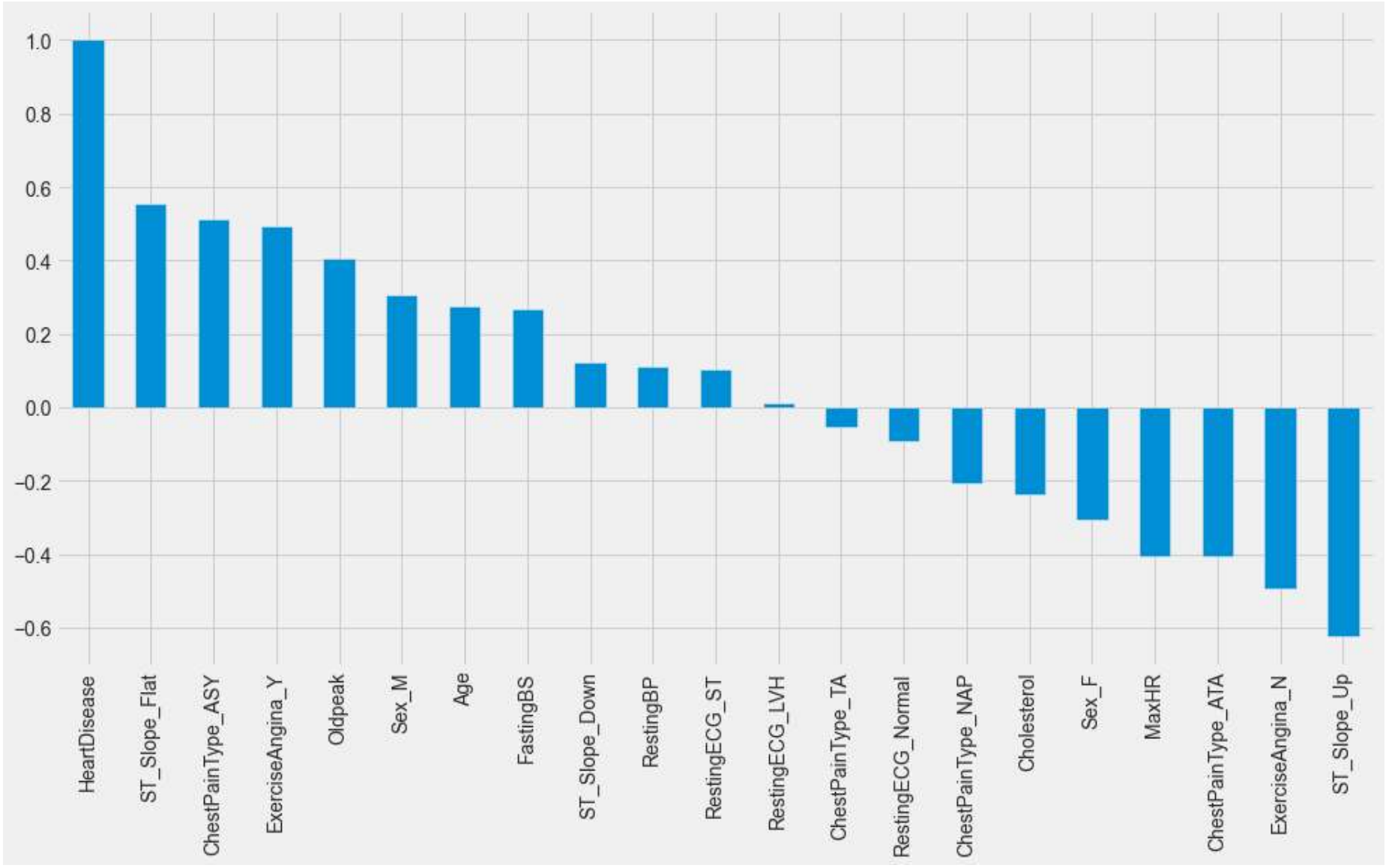
Understand the Correlation with target variable.

In [30]:
```
df_dummies = pd.get_dummies(data)
df_dummies.head()
```

Out[30]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	Sex_F	Sex_M	ChestPainType_ASY	ChestPainType_ATA	ChestPainTyp
0	40.0	140.0	289.0	0.0	172.0	0.0	0	0	1	0	1	
1	49.0	160.0	180.0	0.0	156.0	1.0	1	1	0	0	0	
2	54.0	130.0	283.0	0.0	98.0	0.0	0	0	1	0	1	
3	54.0	138.0	214.0	0.0	108.0	1.5	1	1	0	1	0	
4	54.0	150.0	195.0	0.0	122.0	0.0	0	0	1	0	0	

In [31]:
```
#Get Correlation of "Churn" with other variables:
plt.figure(figsize=(15,8))
df_dummies.corr()['HeartDisease'].sort_values(ascending = False).plot(kind='bar')
plt.show()
```



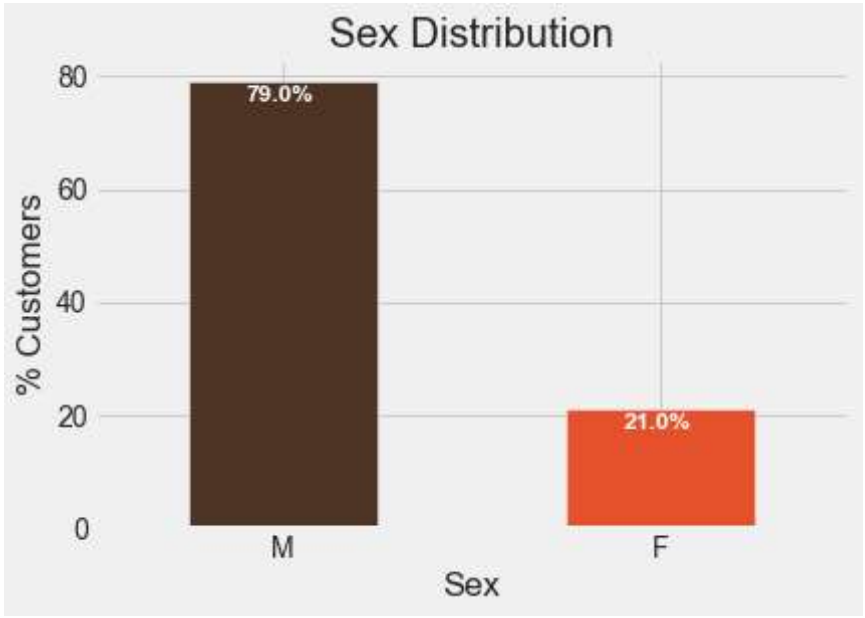
- ST_Slope with `FLAT` ,ChestPain with `ASY` ,ExeciseAngina with `Y` , `Oldpeak` ,Sex with `Male` having positive correlation with `HearFailure` .
- Whereas,ST_Slope with `UP` ,ExeciseAngina with `N` ,MaxHR ,ChestPainType with `ATA` ,Sex with `Female` having less correlation with `HeartFailure` .
- We will explore the patterns for the above correlations below before we delve into modelling and identifying the important variables.

Data Exploration & Data Visualization

Let us first start with exploring our data set, to better understand the patterns in the data and potentially form some hypothesis. First we will look at the distribution of individual variables and then slice and dice our data for any interesting trends.

- There are ~80% are men and remaining are Female.

In [32]:
```
colors = ['#4D3425','#E4512B']
ax = (data['Sex'].value_counts()*100.0 /len(data)).plot(kind='bar',
                                                        stacked = True,
                                                        rot = 0,
                                                        color = colors)
ax.set_ylabel('% Customers')
ax.set_xlabel('Sex')
ax.set_ylabel('% Customers')
ax.set_title('Sex Distribution')

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())
# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-3.5, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
           weight = 'bold')
```



- Based on the age we range we have classified the data in Citizen class in below way.

In [33]:
```
data['Age'].max()
```

Out[33]: 77.0

In [34]:
```
data['Age'].min()
```

Out[34]: 28.0

In [35]:
```
data['Age'].median()
```

Out[35]: 54.0

In [36]:
```
data['Age'] = data['Age'].astype('int64')
```

In [37]:
```
citizen_class =[]

for i in data['Age']:
    if i<=28:
        citizen_class.append('Young Citizen')
    elif i >28 and i<=45:
        citizen_class.append('Junior Citizen')
    else:
        citizen_class.append('Senior Citizen')
```

In [38]:
```
set(citizen_class)
```

Out[38]:
```
{'Junior Citizen', 'Senior Citizen', 'Young Citizen'}
```

In [39]:
```
data['Citizen'] = citizen_class
```

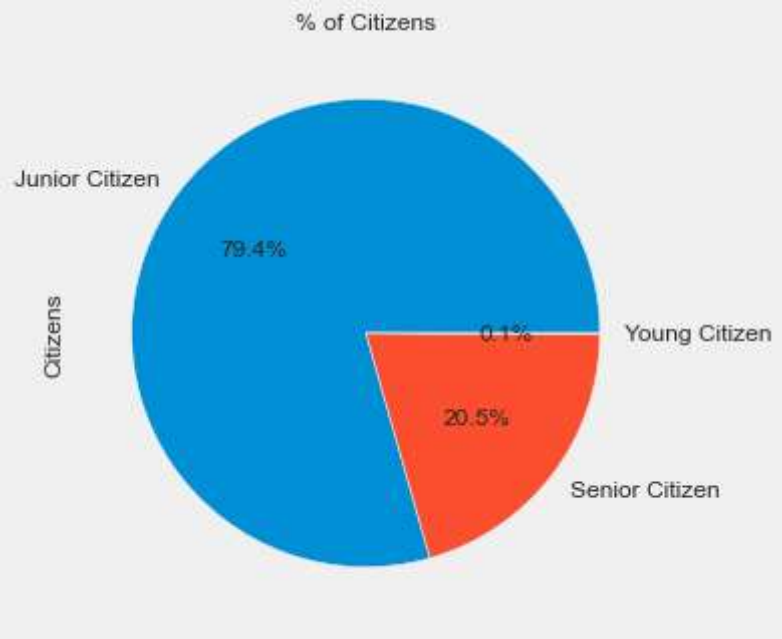
In [40]:
```
data.head()
```

Out[40]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease	Citizen
0	40	M	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	Up	0	Junior Citizen
1	49	F	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	Flat	1	Senior Citizen
2	54	M	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	Up	0	Senior Citizen
3	54	F	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	Flat	1	Senior Citizen
4	54	M	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	Up	0	Senior Citizen

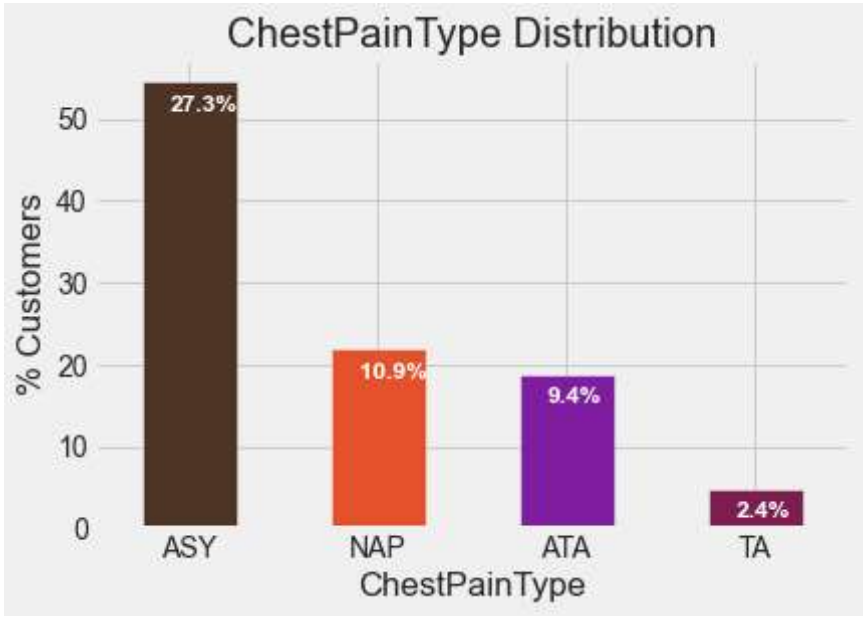
- The data showing there are 20.5% are Senior citizen
- 79.4% are Junior Citizen
- 0.1% are the young citizen.
- Its obvious to have senior and junior citizen because they might have lot of problems with heart disease than the Young generation which is comes under the Young Citizen.

In [41]:
```
ax = (data['Citizen'].value_counts()*100.0 /len(data))\
.plot.pie(autopct='%.1f%%', labels = ['Junior Citizen', 'Senior Citizen', 'Young Citizen'],figsize =(5,5), fontsize = 12
ax.set_ylabel('Citizens',fontsize = 12)
ax.set_title('% of Citizens', fontsize = 12)
plt.show()
```



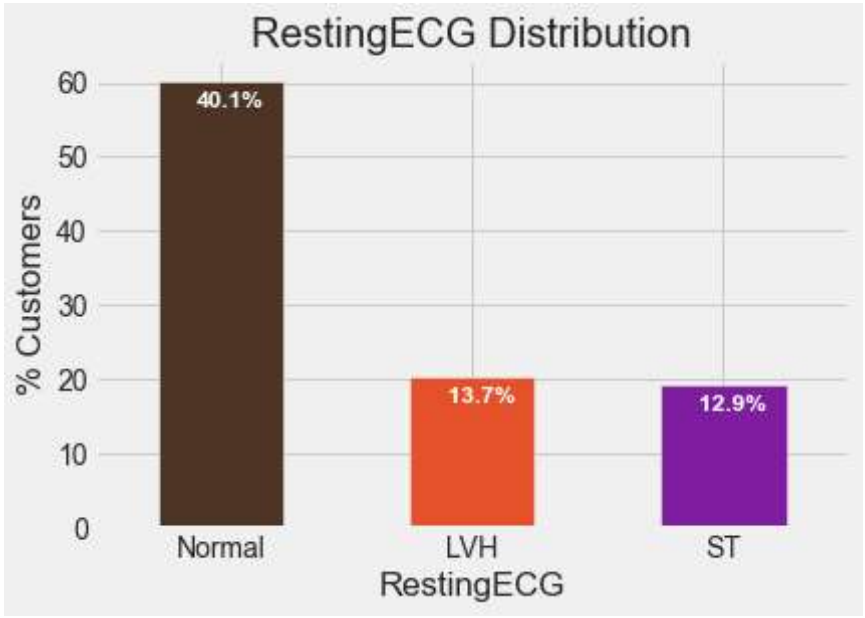
- Lot of patience are avaliable which comes under ASY ChestPainType i.e.27.3%.
- Afterwards, NAP,TAT & TA available in composition of 10.9% , 9.4% & 2.4% respectively.

In [42]:
```
colors = ['#4D3425','#E4512B','#7E1D9F','#7E1D4F']
ax = (data['ChestPainType'].value_counts()*100.0 /len(data)).plot(kind='bar',
                                                    stacked = True,
                                                    rot = 0,
                                                    color = colors)
ax.set_ylabel('% Customers')
ax.set_xlabel('ChestPainType')
ax.set_ylabel('% Customers')
ax.set_title('ChestPainType Distribution')

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())
# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-3.5, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
           weight = 'bold')
```



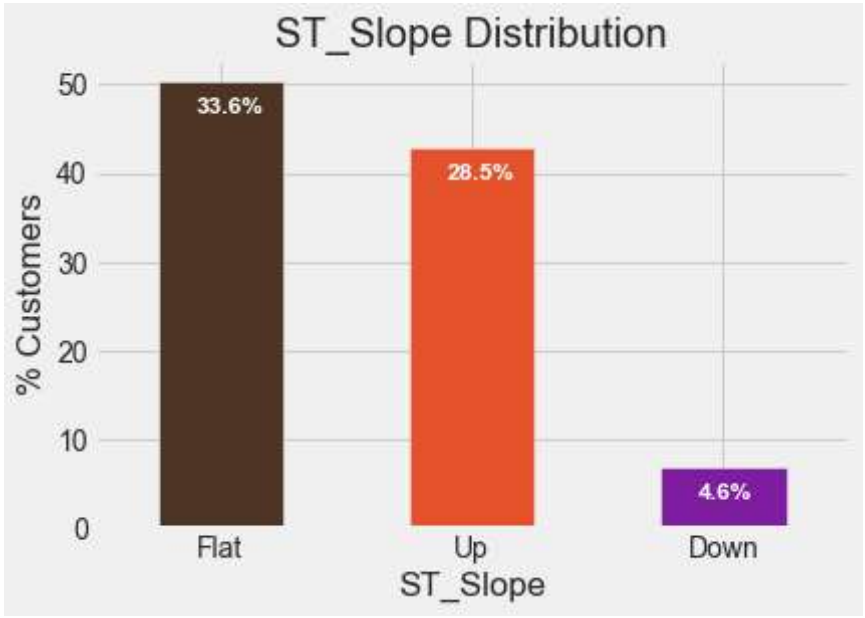
- Lot of patience are avaliable which comes under Normal RestingECG i.e. 40.1%.
- Afterwards, LVH & ST available in composition of 13.7% & 12.9% respectively.

In [43]:
```
colors = ['#4D3425','#E4512B','#7E1D9F']
ax = (data['RestingECG'].value_counts()*100.0 /len(data)).plot(kind='bar',
                                                               stacked = True,
                                                               rot = 0,
                                                               color = colors)
ax.set_ylabel('% Customers')
ax.set_xlabel('RestingECG')
ax.set_ylabel('% Customers')
ax.set_title('RestingECG Distribution')

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())
# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-3.5, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
           weight = 'bold')
```



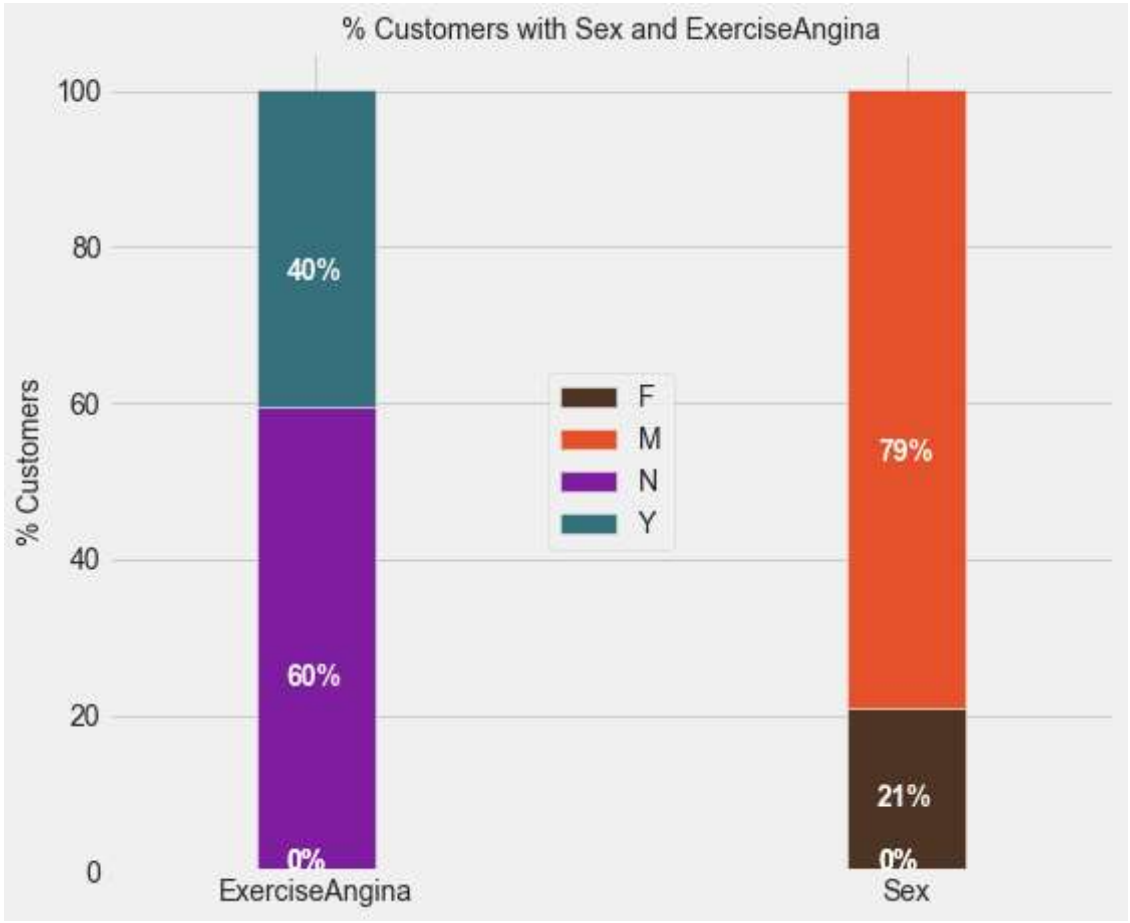
- Lot of patience are avaliable which comes under Flat ST_Slope i.e. 33.6%.
- Afterwards, Up & Down available in composition of 28.5% & 4.6% respectively.

In [44]:
```
colors = ['#4D3425','#E4512B','#7E1D9F']
ax = (data['ST_Slope'].value_counts()*100.0 /len(data)).plot(kind='bar',
                                                             stacked = True,
                                                             rot = 0,
                                                             color = colors)
ax.set_ylabel('% Customers')
ax.set_xlabel('ST_Slope')
ax.set_ylabel('% Customers')
ax.set_title('ST_Slope Distribution')

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())
# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-3.5, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
           weight = 'bold')
```



In [45]:
```
data1=data.reset_index()
```

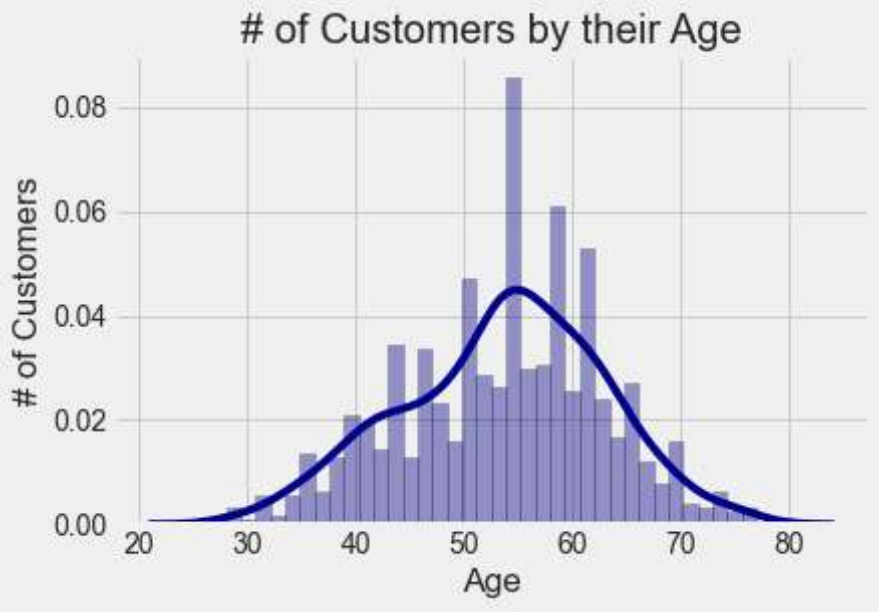
In [46]:
```
df2 = pd.melt(data1, id_vars=['index'],value_vars=['Sex','ExerciseAngina'])
df3 = df2.groupby(['variable','value']).count().unstack()
df3 = df3*100/len(data)
colors = ['#4D3425','#E4512B','#7E1D9F','#36707C']
ax = df3.loc[:,'index'].plot.bar(stacked=True, color=colors,
                                  figsize=(8,7),rot = 0,
                                 width = 0.2)
ax.set_ylabel('% Customers',size = 14)
ax.set_xlabel('')
ax.set_title('% Customers with Sex and ExerciseAngina',size = 14)
ax.legend(loc = 'center',prop={'size':14})

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```



- Since the distrubution of the Age is normal Distribution & lot of patience are comes in the range of 50-60.

In [47]:
```
ax = sns.distplot(data['Age'], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
ax.set_ylabel('# of Customers')
ax.set_xlabel('Age')
ax.set_title('# of Customers by their Age')
```

Out[47]:
```
Text(0.5, 1.0, '# of Customers by their Age')
```

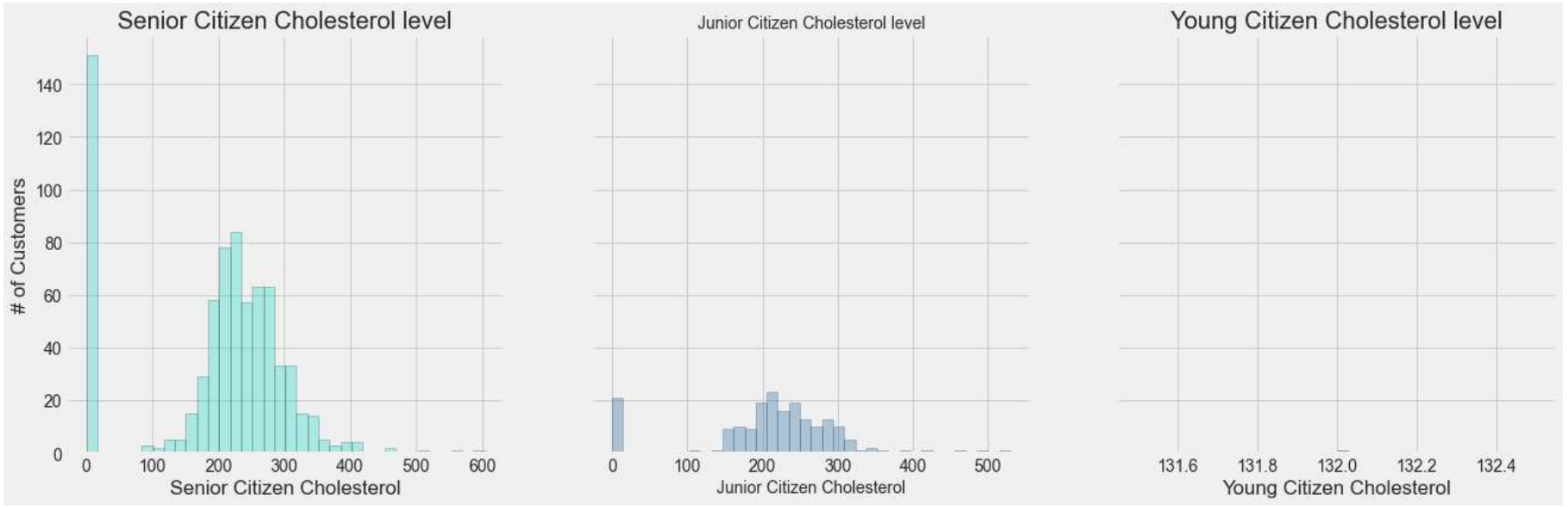


- Here,we can see that those patients are comes under senior citizen having more Cholesterol than Junior Citizen.

- Junior Citizen having more Cholesterol than the Young Citizen.

In [48]:
```
fig, (ax1,ax2,ax3) = plt.subplots(nrows=1, ncols=3, sharey = True, figsize = (20,6))

ax = sns.distplot(data[data['Citizen']=='Senior Citizen']['Cholesterol'],
                  hist=True, kde=False,
                  bins=int(180/5), color = 'turquoise',
                  hist_kws={'edgecolor':'black'},
                  kde_kws={'linewidth': 4},
                 ax=ax1)
ax.set_ylabel('# of Customers')
ax.set_xlabel('Senior Citizen Cholesterol')
ax.set_title('Senior Citizen Cholesterol level')

ax = sns.distplot(data[data['Citizen']=='Junior Citizen']['Cholesterol'],
                  hist=True, kde=False,
                  bins=int(180/5), color = 'steelblue',
                  hist_kws={'edgecolor':'black'},
                  kde_kws={'linewidth': 4},
                 ax=ax2)
ax.set_xlabel('Junior Citizen Cholesterol',size = 14)
ax.set_title('Junior Citizen Cholesterol level',size = 14)

ax = sns.distplot(data[data['Citizen']=='Young Citizen']['Cholesterol'],
                  hist=True, kde=False,
                  bins=int(180/5), color = 'darkblue',
                  hist_kws={'edgecolor':'black'},
                  kde_kws={'linewidth': 4},
                 ax=ax3)

ax.set_xlabel('Young Citizen Cholesterol')
ax.set_title('Young Citizen Cholesterol level')
```

Out[48]:
```
Text(0.5, 1.0, 'Young Citizen Cholesterol level')
```



In [49]:
```
data.head()
```

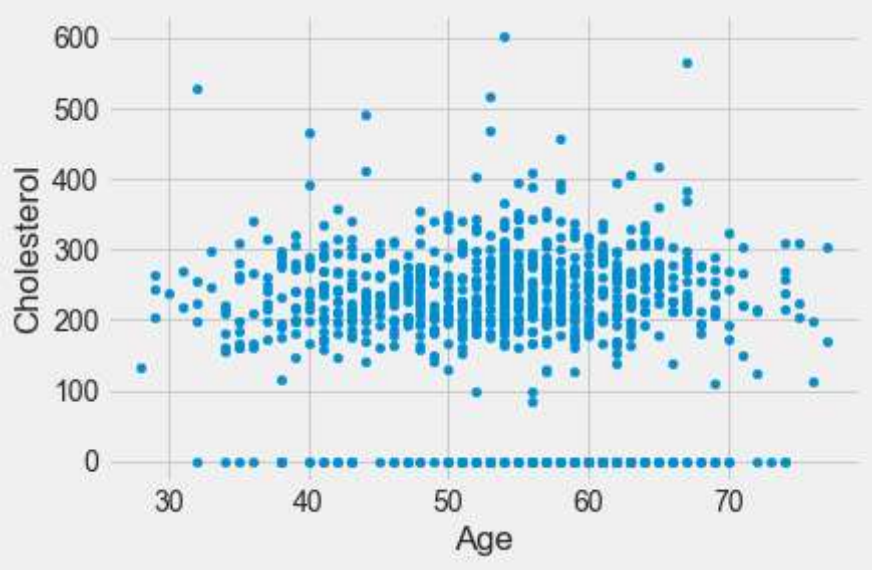
Out[49]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease	Citizen
0	40	M	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	Up	0	Junior Citizen
1	49	F	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	Flat	1	Senior Citizen
2	54	M	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	Up	0	Senior Citizen
3	54	F	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	Flat	1	Senior Citizen
4	54	M	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	Up	0	Senior Citizen

- Age & Cholesterol does not having proper linear relationship.

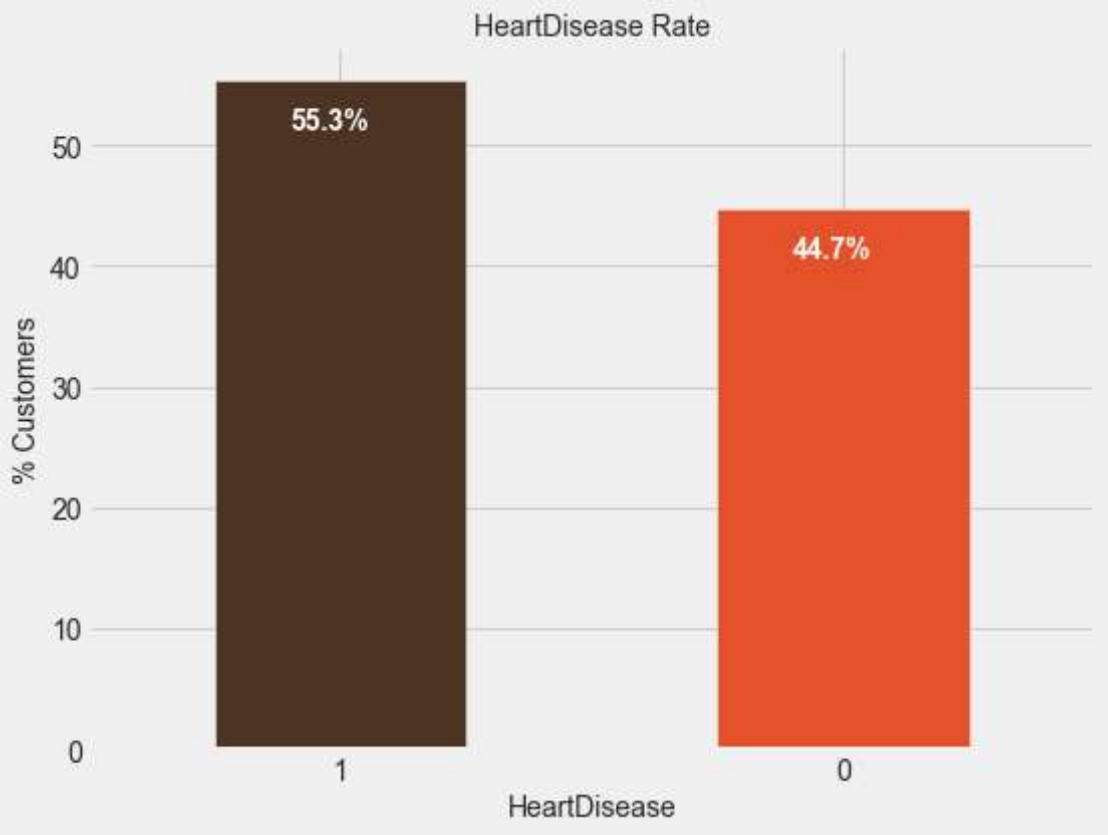
In [50]:
```
data[['Age', 'Cholesterol']].plot.scatter(x = 'Age',y='Cholesterol')
```

Out[50]:
```
<AxesSubplot:xlabel='Age', ylabel='Cholesterol'>
```



- Rate of HeartDisease is greater in this data.

In [51]:
```
colors = ['#4D3425','#E4512B']
ax = (data['HeartDisease'].value_counts()*100.0 /len(data)).plot(kind='bar',
                                                                 stacked = True,
                                                                 rot = 0,
                                                                 color = colors,
                                                                 figsize = (8,6))
ax.set_ylabel('% Customers',size = 14)
ax.set_xlabel('HeartDisease',size = 14)
ax.set_title('HeartDisease Rate', size = 14)

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())
# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-4.0, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
           weight = 'bold',
           size = 14)
```



In [52]:
```
data.columns
```

Out[52]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease', 'Citizen'],
      dtype='object')
```

- ASY ChestPainType having higher rate of heartFailure problem.

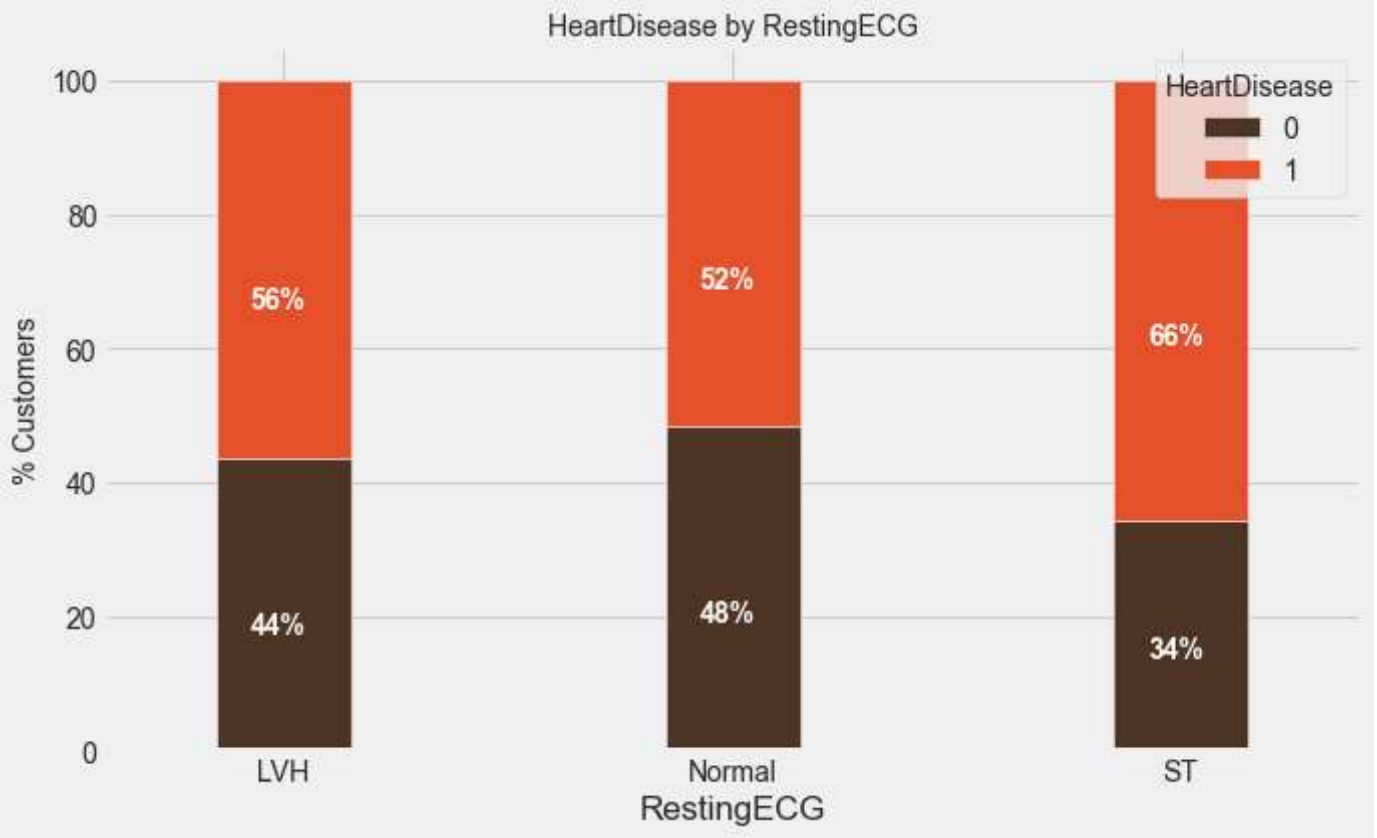
In [53]:
```
colors = ['#4D3425','#E4512B']
contract_heart = data.groupby(['ChestPainType','HeartDisease']).size().unstack()

ax = (contract_heart.T*100.0 / contract_heart.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.legend(loc='best',prop={'size':14},title = 'HeartDisease')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('HeartDisease by ChestPainType',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```

- Male category having higher rate of heartFailure problems.

In [54]:
```
colors = ['#4D3425','#E4512B']
contract_heart = data.groupby(['Sex','HeartDisease']).size().unstack()

ax = (contract_heart.T*100.0 / contract_heart.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.legend(loc='best',prop={'size':14},title = 'HeartDisease')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('HeartDisease by Sex',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```

- ST type of RestingECG having more heartFailure problems.

In [55]:
```
colors = ['#4D3425','#E4512B']
contract_heart = data.groupby(['RestingECG','HeartDisease']).size().unstack()

ax = (contract_heart.T*100.0 / contract_heart.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.legend(loc='best',prop={'size':14},title = 'HeartDisease')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('HeartDisease by RestingECG',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```

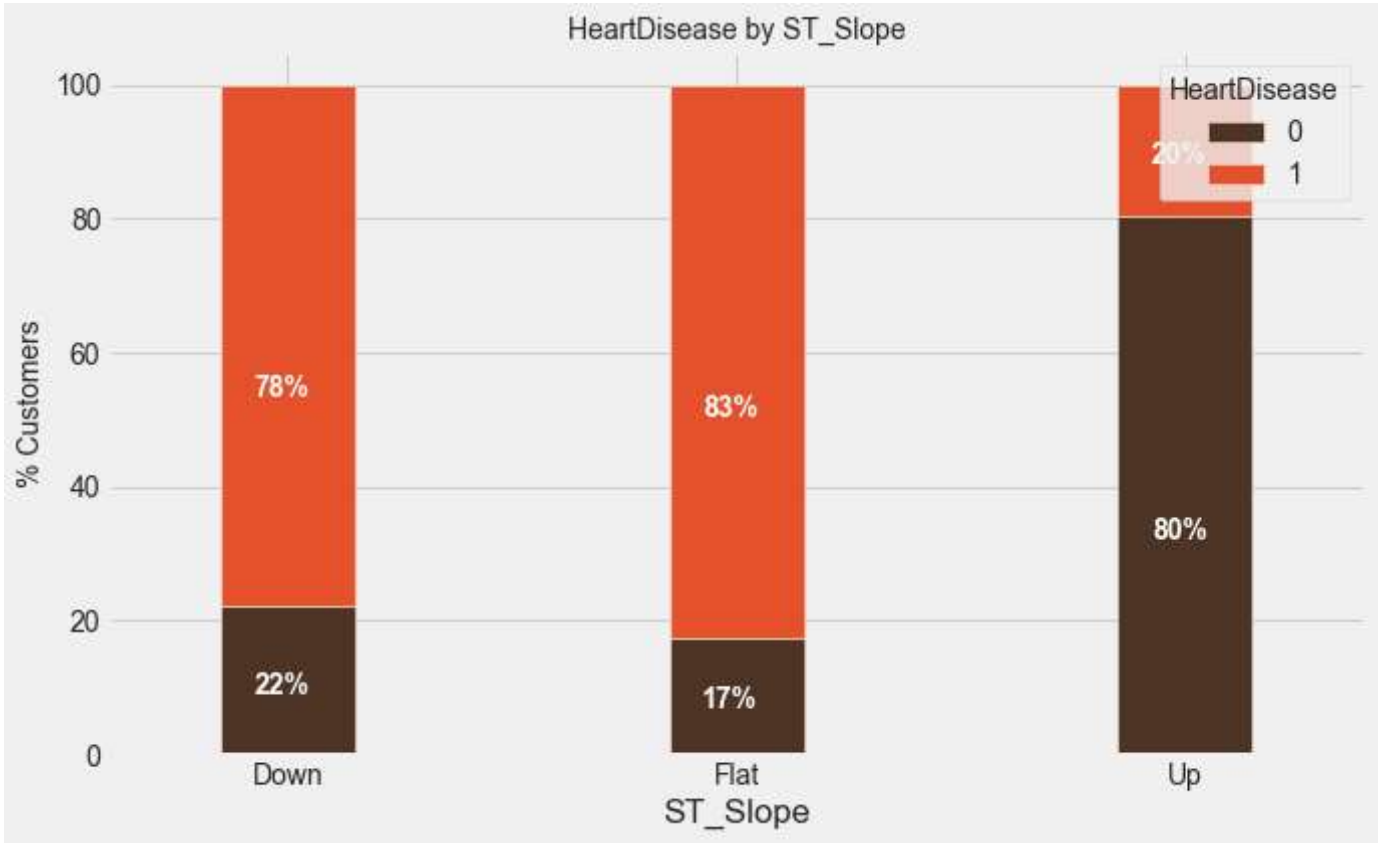


- If ExerciseAngina is Y then there is higher possibility of heartFailure problem.

In [56]:
```
colors = ['#4D3425','#E4512B']
contract_heart = data.groupby(['ExerciseAngina','HeartDisease']).size().unstack()

ax = (contract_heart.T*100.0 / contract_heart.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.legend(loc='best',prop={'size':14},title = 'HeartDisease')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('HeartDisease by ExerciseAngina',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```

- If the ST_Slope in the Down & Flat category then there will be the higher chances of HeartFailure.

In [57]:
```
colors = ['#4D3425','#E4512B']
contract_heart = data.groupby(['ST_Slope','HeartDisease']).size().unstack()

ax = (contract_heart.T*100.0 / contract_heart.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.legend(loc='best',prop={'size':14},title = 'HeartDisease')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('HeartDisease by ST_Slope',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```



- In the case of heartFailure senior citizen having more problem than the Junior & Young Citizen.

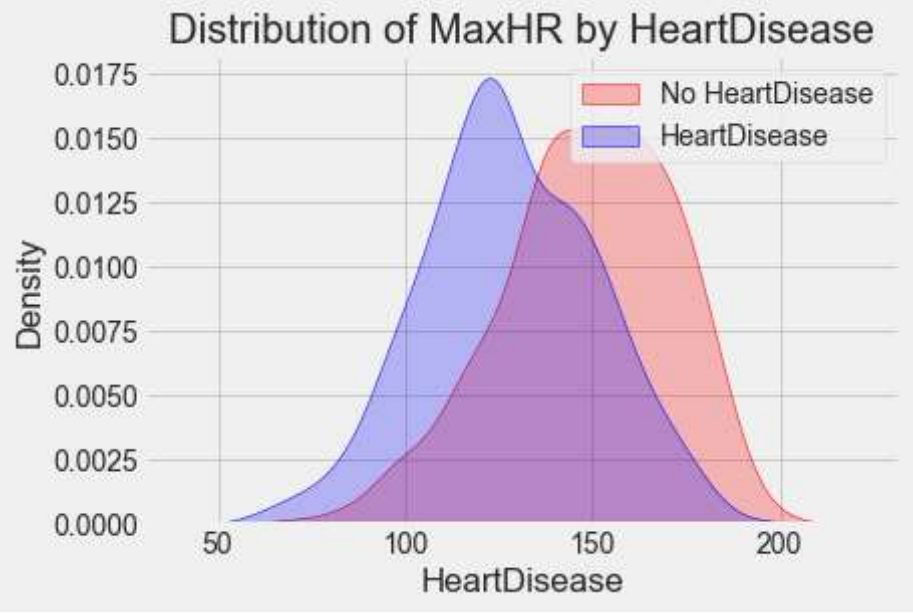
In [58]:
```
colors = ['#4D3425','#E4512B']
contract_heart = data.groupby(['Citizen','HeartDisease']).size().unstack()

ax = (contract_heart.T*100.0 / contract_heart.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.legend(loc='best',prop={'size':14},title = 'HeartDisease')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('HeartDisease by Citizen',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```

Lets Compare the Distribution which comes under heartFailure.

- If MaxHR is going to below 60 then there will lesser probablity of getting heartFailure problem.
- If MaxHR is going to above 190 then there will higher probablity of getting heartFailure problem.

In [59]:
```
ax = sns.kdeplot(data.MaxHR[(data["HeartDisease"] == 0) ],
                 color="Red", shade = True)
ax = sns.kdeplot(data.MaxHR[(data["HeartDisease"] == 1) ],
                 ax =ax, color="Blue", shade= True)
ax.legend(["No HeartDisease","HeartDisease"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('HeartDisease')
ax.set_title('Distribution of MaxHR by HeartDisease')
```

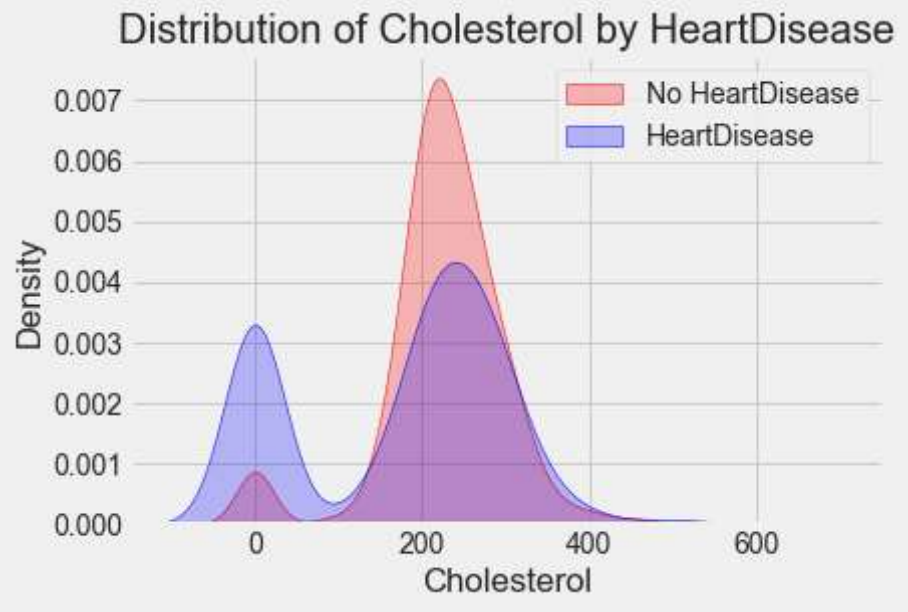
Out[59]:
```
Text(0.5, 1.0, 'Distribution of MaxHR by HeartDisease')
```



- If lesser Cholesterol then lesser probablity of getting heartFailure problem.
- If higher Cholesterol then there will higher probablity of getting heartFailure problem.

In [60]:
```
ax = sns.kdeplot(data.Cholesterol[(data["HeartDisease"] == 0) ],
                 color="Red", shade = True)
ax = sns.kdeplot(data.Cholesterol[(data["HeartDisease"] == 1) ],
                 ax =ax, color="Blue", shade= True)
ax.legend(["No HeartDisease","HeartDisease"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Cholesterol')
ax.set_title('Distribution of Cholesterol by HeartDisease')
```

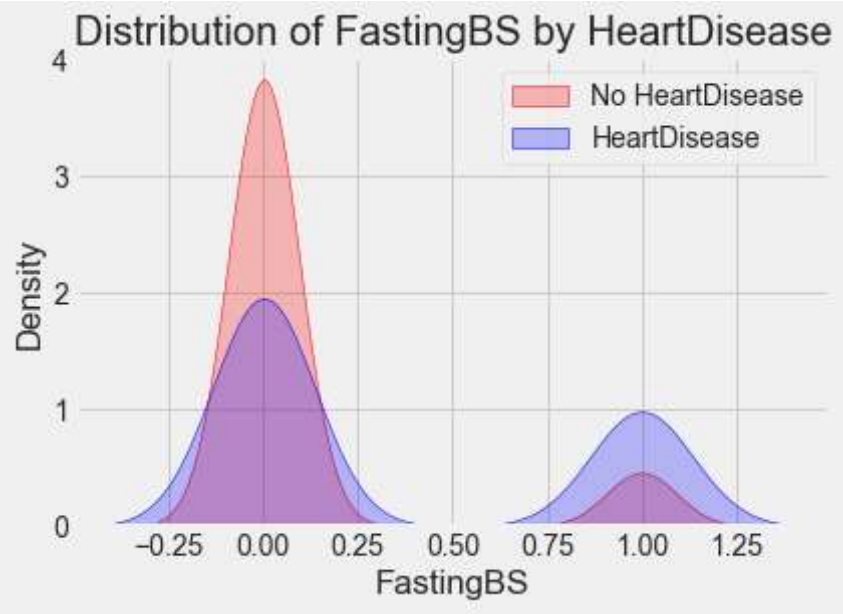
Out[60]:
```
Text(0.5, 1.0, 'Distribution of Cholesterol by HeartDisease')
```



- If FastingBS is near to zero then there is less chances of HeartFailure
- IF FastingBS is positive there higher chances of heartFailure.

In [61]:
```
ax = sns.kdeplot(data.FastingBS[(data["HeartDisease"] == 0) ],
                 color="Red", shade = True)
ax = sns.kdeplot(data.FastingBS[(data["HeartDisease"] == 1) ],
                 ax =ax, color="Blue", shade= True)
ax.legend(["No HeartDisease","HeartDisease"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('FastingBS')
ax.set_title('Distribution of FastingBS by HeartDisease')
```

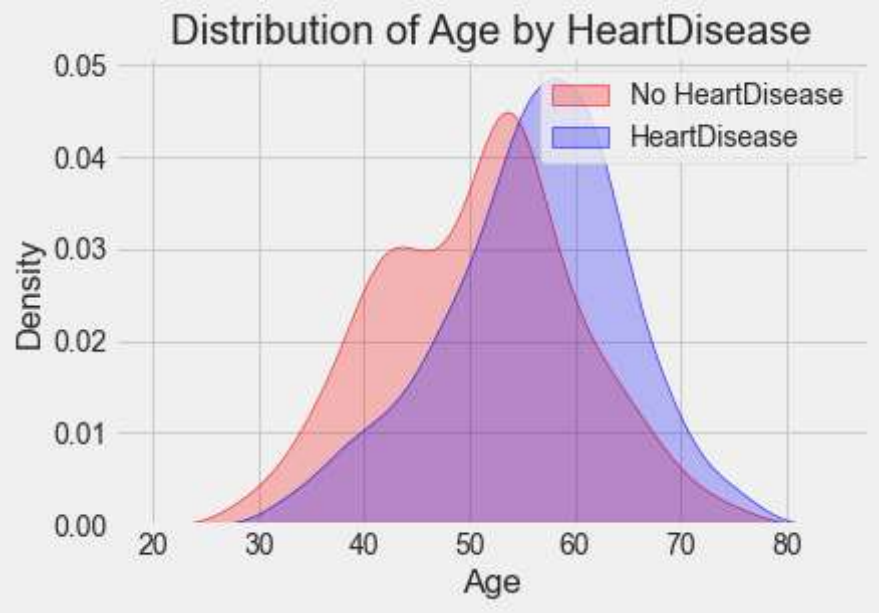
Out[61]:
```
Text(0.5, 1.0, 'Distribution of FastingBS by HeartDisease')
```



- As the Age increase the heart disease problem is also increased proportionally & vice versa.

In [62]:
```
ax = sns.kdeplot(data.Age[(data["HeartDisease"] == 0) ],
                 color="Red", shade = True)
ax = sns.kdeplot(data.Age[(data["HeartDisease"] == 1) ],
                 ax =ax, color="Blue", shade= True)
ax.legend(["No HeartDisease","HeartDisease"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Age')
ax.set_title('Distribution of Age by HeartDisease')
```

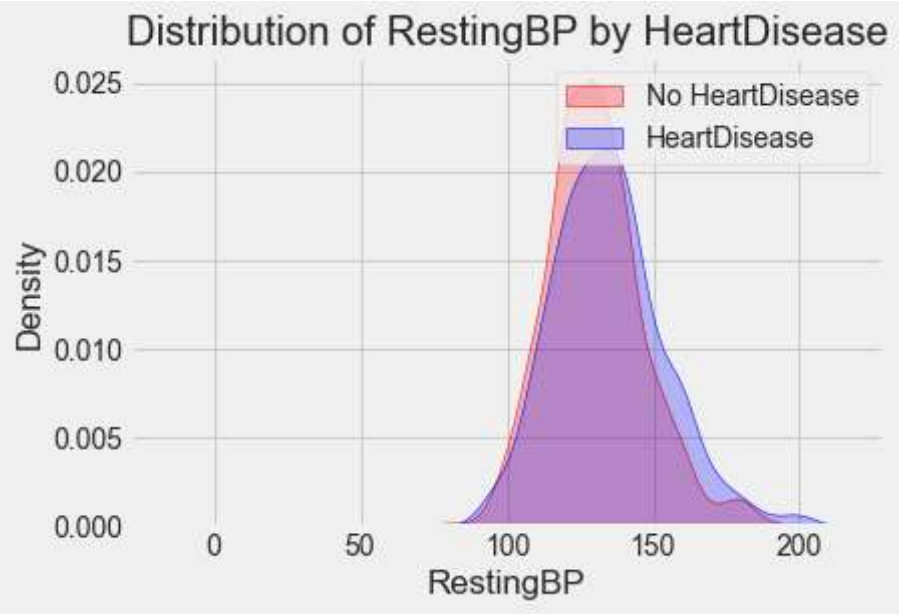
Out[62]:
```
Text(0.5, 1.0, 'Distribution of Age by HeartDisease')
```



- If RestingBP is higher then there is quit less probablity of getting heartproblem but still can't say that by watching the high overlapping distribution

In [63]:
```
ax = sns.kdeplot(data.RestingBP[(data["HeartDisease"] == 0) ],
                 color="Red", shade = True)
ax = sns.kdeplot(data.RestingBP[(data["HeartDisease"] == 1) ],
                 ax =ax, color="Blue", shade= True)
ax.legend(["No HeartDisease","HeartDisease"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('RestingBP')
ax.set_title('Distribution of RestingBP by HeartDisease')
```

Out[63]:
```
Text(0.5, 1.0, 'Distribution of RestingBP by HeartDisease')
```



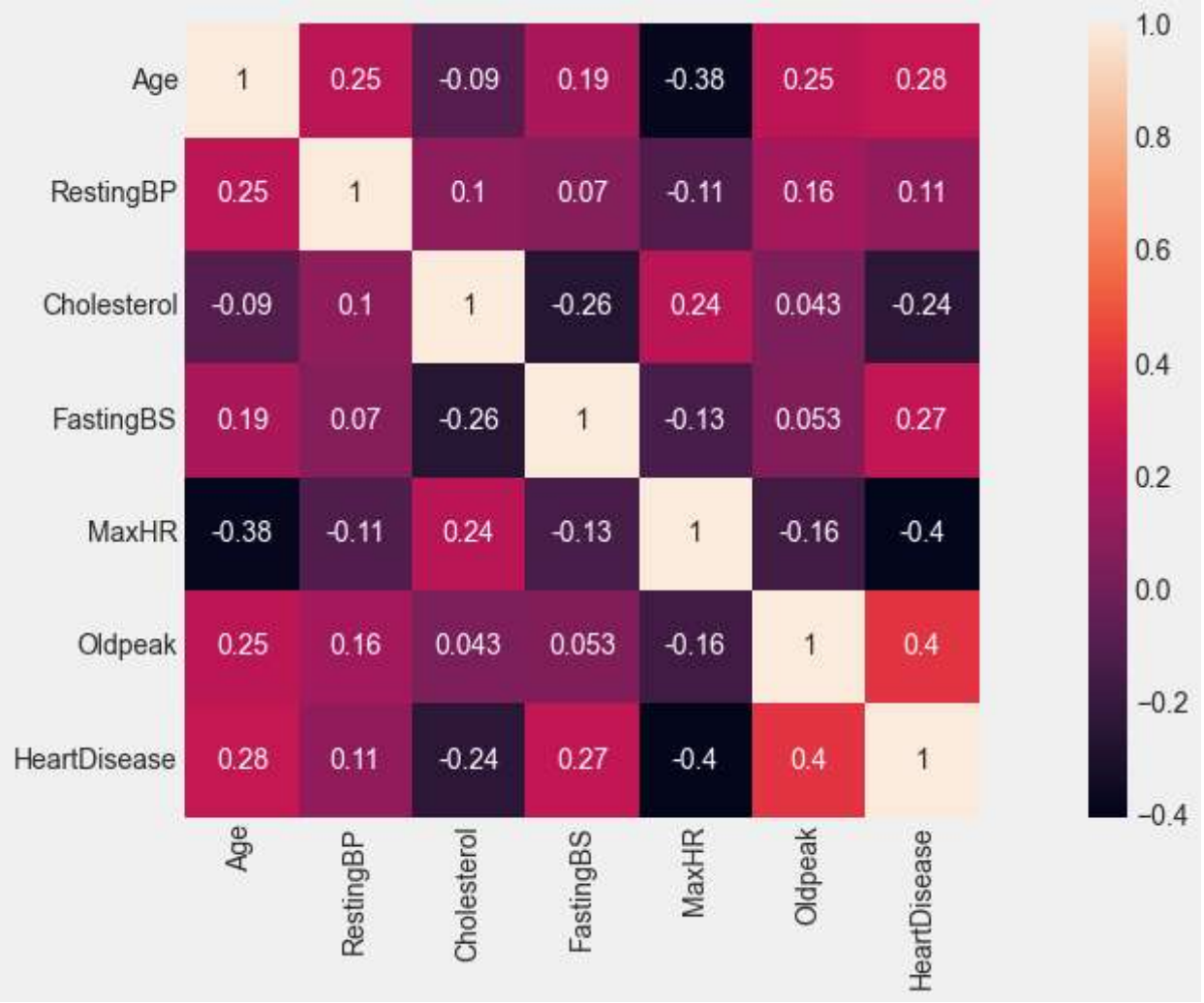
Let's Understand the Correlation between the feature.

In [64]:
```
corrmat = data.corr()
corrmat
```

Out[64]:

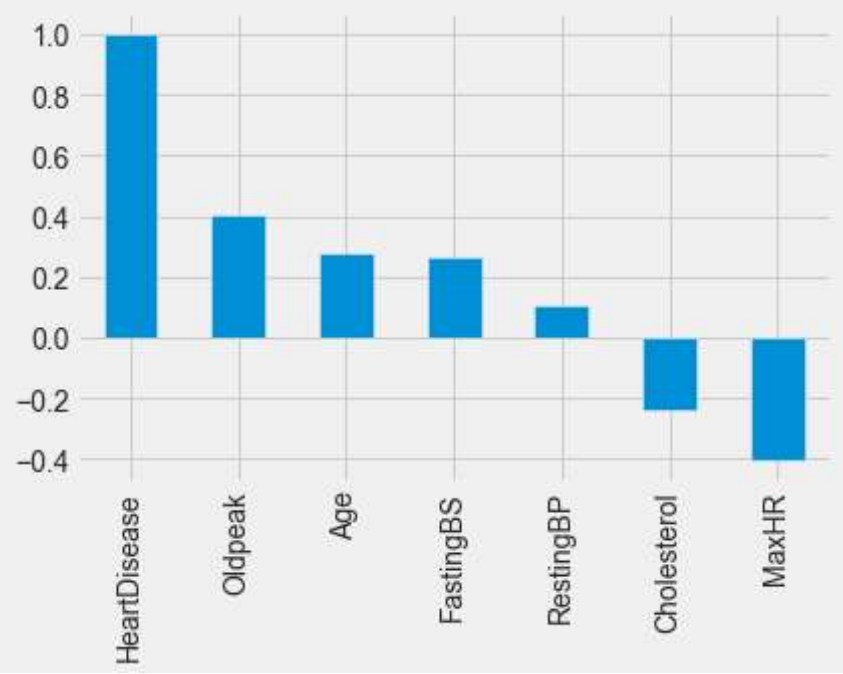
	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
Age	1.000000	0.251571	-0.090477	0.193024	-0.383049	0.251012	0.276086
RestingBP	0.251571	1.000000	0.102622	0.069599	-0.113134	0.163952	0.108733
Cholesterol	-0.090477	0.102622	1.000000	-0.257840	0.239484	0.043471	-0.235760
FastingBS	0.193024	0.069599	-0.257840	1.000000	-0.134395	0.052698	0.267291
MaxHR	-0.383049	-0.113134	0.239484	-0.134395	1.000000	-0.160312	-0.404162
Oldpeak	0.251012	0.163952	0.043471	0.052698	-0.160312	1.000000	0.403951
HeartDisease	0.276086	0.108733	-0.235760	0.267291	-0.404162	0.403951	1.000000

In [65]:
```
plt.figure(figsize=(18,7))
sns.heatmap(corrmat,annot=True,square=True)
plt.show()
```



Correlation of HeartDisease with Other Features.

In [66]:
```
corr = data.corr()['HeartDisease']
corr.sort_values(ascending=False).plot(kind='bar')
plt.show()
```



Let's Build the Model for Predicting the class.

In [67]:
```
data.head()
```

Out[67]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease	Citizen
0	40	M	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	Up	0	Junior Citizen
1	49	F	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	Flat	1	Senior Citizen
2	54	M	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	Up	0	Senior Citizen
3	54	F	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	Flat	1	Senior Citizen
4	54	M	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	Up	0	Senior Citizen

- From the above data we have some categories which is not possible to put inside the model building process so we need to first convert them into numerical data types.

- Sex is nominal category.
- RestingECG seems like nominal category
- ST_Slop is seems like nominal category.
- ExerciseAngina seems like oridinal Categories.
- Citizen is like ordinal category
- ChestPain is like ordinal category.

Lets do some feature engineering.

In [68]:
```
sex_dummy = pd.get_dummies(data['Sex'],drop_first=True)
```

In [69]:
```
sex_dummy.head()
```

Out[69]:

	M
0	1
1	0
2	1
3	0
4	1

In [70]:
```
St_slope_dummy = pd.get_dummies(data['ST_Slope'],drop_first=True)
```

In [71]:
```
St_slope_dummy.head()
```

Out[71]:

	Flat	Up
0	0	1
1	1	0
2	0	1
3	1	0
4	0	1

In [72]:
```
data.drop(columns=['ST_Slope','Sex'],inplace=True)
```

In [73]:
```
data = pd.concat([data,St_slope_dummy,sex_dummy],axis=1)
```

In [74]:
```
data.head()
```

Out[74]:

	Age	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	HeartDisease	Citizen	Flat	Up	M
0	40	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	0	Junior Citizen	0	1	1
1	49	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	1	Senior Citizen	1	0	0
2	54	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	0	Senior Citizen	0	1	1
3	54	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	1	Senior Citizen	1	0	0
4	54	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	0	Senior Citizen	0	1	1

- While doing dummy dataframe creation we have been dropping a column to avoid multi-colinearity Trap.

In [75]:
```
restingECG = pd.get_dummies(data['RestingECG'],drop_first=True)
restingECG.head()
```

Out[75]:

	Normal	ST
0	1	0
1	1	0
2	0	1
3	1	0
4	1	0

In [76]:
```
data = pd.concat([data,restingECG],axis=True)
```

In [77]:
```
data.drop(columns=['RestingECG'],inplace=True)
```

In [78]:
```
data.head()
```

Out[78]:

	Age	ChestPainType	RestingBP	Cholesterol	FastingBS	MaxHR	ExerciseAngina	Oldpeak	HeartDisease	Citizen	Flat	Up	M	Normal	ST
0	40	ATA	140.0	289.0	0.0	172.0	N	0.0	0	Junior Citizen	0	1	1	1	0
1	49	NAP	160.0	180.0	0.0	156.0	N	1.0	1	Senior Citizen	1	0	0	1	0
2	54	ATA	130.0	283.0	0.0	98.0	N	0.0	0	Senior Citizen	0	1	1	0	1
3	54	ASY	138.0	214.0	0.0	108.0	Y	1.5	1	Senior Citizen	1	0	0	1	0
4	54	NAP	150.0	195.0	0.0	122.0	N	0.0	0	Senior Citizen	0	1	1	1	0

In [79]:
```
dict_Chestpain = {"ASY":3,'TA':2,'NAP':1,'ATA':0} #This order made by refeencing how much impact gain insight the heartf
```

In [80]:
```
data['ChestPainType'] = data['ChestPainType'].map(dict_Chestpain)
```

In [81]:
```
dict_ExerciseAngina = {"Y":1,'N':0}
```

In [82]:
```
data['ExerciseAngina']=data['ExerciseAngina'].map(dict_ExerciseAngina)
##This order made by refeencing how much impact gain insight the heartfail.
```

In [83]:
```
dict_citizen = {"Senior Citizen":2,"Junior Citizen":1,"Young Citizen":0}
#This order made by refeencing how much impact gain insight the heartfail.
```

In [84]:
```
data['Citizen']=data['Citizen'].map(dict_citizen)
```

In [85]:
```
data.head()
```

Out[85]:

	Age	ChestPainType	RestingBP	Cholesterol	FastingBS	MaxHR	ExerciseAngina	Oldpeak	HeartDisease	Citizen	Flat	Up	M	Normal	ST
0	40	0	140.0	289.0	0.0	172.0	0	0.0	0	1	0	1	1	1	0
1	49	1	160.0	180.0	0.0	156.0	0	1.0	1	2	1	0	0	1	0
2	54	0	130.0	283.0	0.0	98.0	0	0.0	0	2	0	1	1	0	1
3	54	3	138.0	214.0	0.0	108.0	1	1.5	1	2	1	0	0	1	0
4	54	1	150.0	195.0	0.0	122.0	0	0.0	0	2	0	1	1	1	0

Lets implement the machine learning model.

In [86]:
```
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.metrics import classification_report,confusion_matrix,precision_recall_curve
from sklearn.metrics import accuracy_score,recall_score,auc,roc_auc_score
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier,VotingClassifier
from sklearn.ensemble import GradientBoostingClassifier,AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score,train_test_split,GridSearchCV,RandomizedSearchCV
```

In [87]:
```
x = data.drop(columns='HeartDisease')
y = data['HeartDisease']
```

In [88]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.80,random_state=40,stratify=y)
```

In [89]:
```
x_train.shape,x_test.shape
```

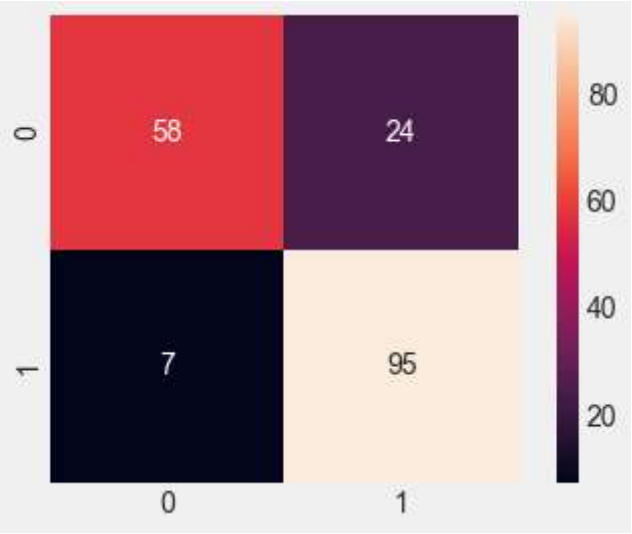
Out[89]:
```
((734, 14), (184, 14))
```

Check Model performance Without Scalling Data.

RandomForest

In [90]:
```
model = RandomForestClassifier(n_estimators=100,criterion='gini',min_samples_leaf=1)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
print("**********************")
```

```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.89      0.71      0.79        82
           1       0.80      0.93      0.86       102

    accuracy                           0.83       184
   macro avg       0.85      0.82      0.82       184
weighted avg       0.84      0.83      0.83       184

*****************
```

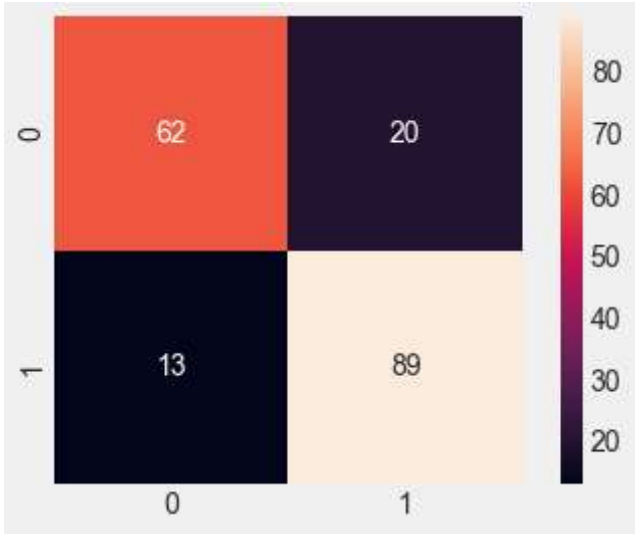


****

AdaboostClassifier

In [91]:
```
adamodel = AdaBoostClassifier(n_estimators=100)
adamodel.fit(x_train,y_train)
y_pred = adamodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

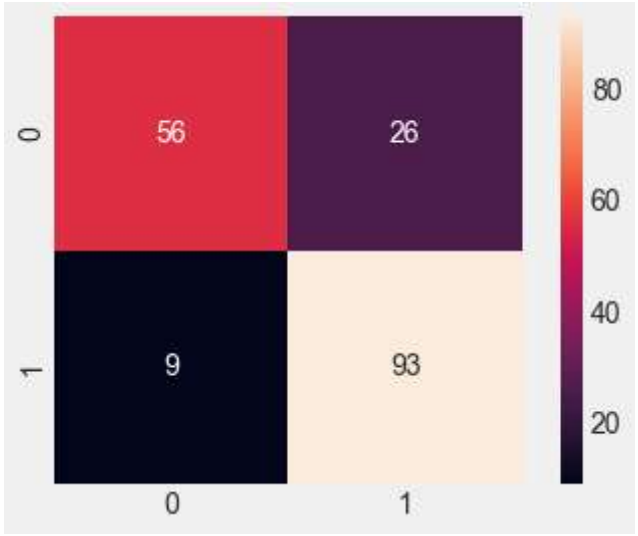
```
Accuarcy :- 0.8206521739130435
*****************
              precision    recall  f1-score   support

           0       0.83      0.76      0.79        82
           1       0.82      0.87      0.84       102

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

*****************
```



GradientBoostClassifier

In [92]:
```
Gradmodel = GradientBoostingClassifier(n_estimators=100,learning_rate=0.01,ccp_alpha=0.01)
Gradmodel.fit(x_train,y_train)
y_pred =Gradmodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```

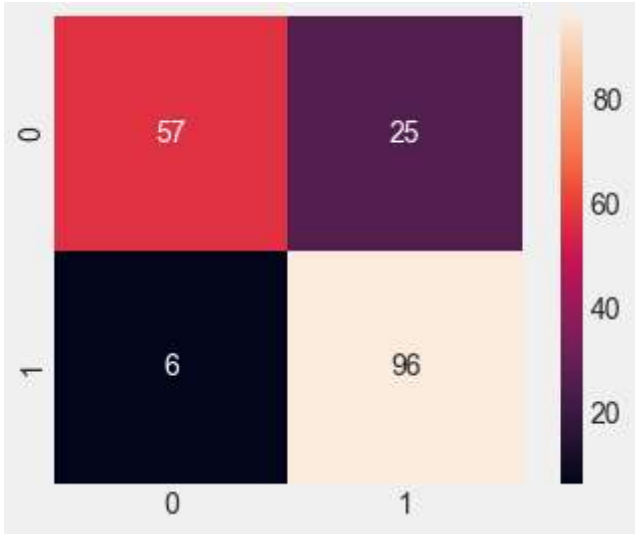
```
Accuarcy :- 0.8097826086956522
*****************
              precision    recall  f1-score   support

           0       0.86      0.68      0.76        82
           1       0.78      0.91      0.84       102

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

*****************
```



ExtratreeClassifier

In [93]:
```
Extramodel = ExtraTreesClassifier(n_estimators=600,ccp_alpha=0.01)
Extramodel.fit(x_train,y_train)
y_pred =Extramodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

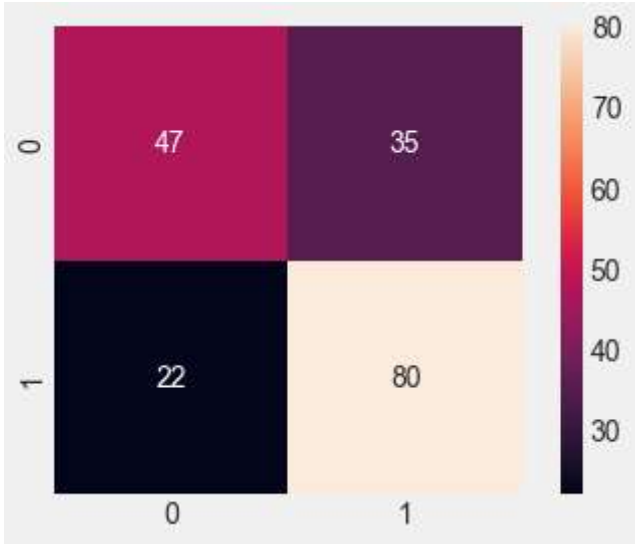
```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.90      0.70      0.79        82
           1       0.79      0.94      0.86       102

    accuracy                           0.83       184
   macro avg       0.85      0.82      0.82       184
weighted avg       0.84      0.83      0.83       184

*****************
```



SVC

In [94]:
```
svc =SVC(kernel='rbf')
svc.fit(x_train,y_train)
y_pred = svc.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

```
Accuarcy :- 0.6902173913043478
*****************
              precision    recall  f1-score   support

           0       0.68      0.57      0.62        82
           1       0.70      0.78      0.74       102

    accuracy                           0.69       184
   macro avg       0.69      0.68      0.68       184
weighted avg       0.69      0.69      0.69       184

*****************
```



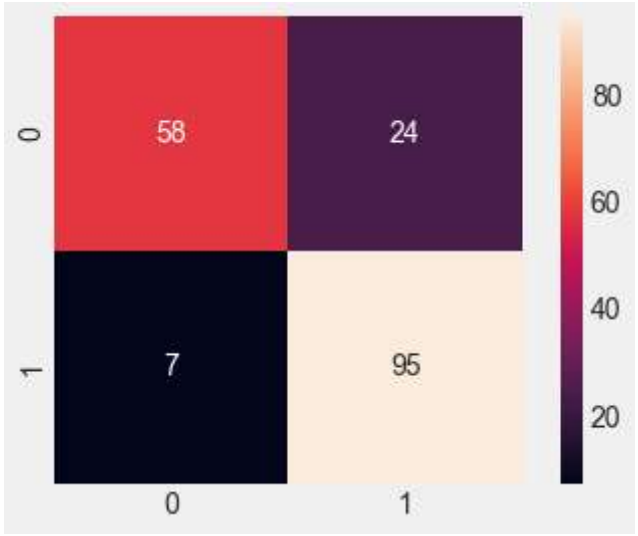
Check model performace with Scalled Data.

In [95]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test  = scaler.fit_transform(x_test)
```

RandomForest

In [96]:
```
model = RandomForestClassifier(n_estimators=100,criterion='gini',min_samples_leaf=1)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

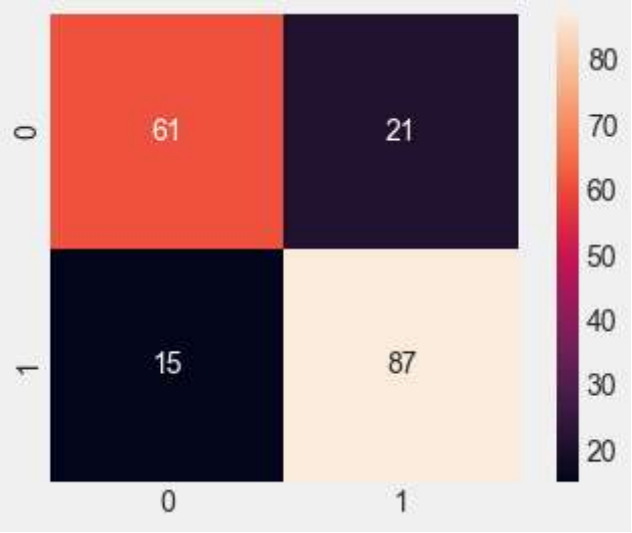
```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.89      0.71      0.79        82
           1       0.80      0.93      0.86       102

    accuracy                           0.83       184
   macro avg       0.85      0.82      0.82       184
weighted avg       0.84      0.83      0.83       184

*****************
```



AdaBoostClassifier

In [97]:
```
adamodel = AdaBoostClassifier(n_estimators=100)
adamodel.fit(X_train,y_train)
y_pred = adamodel.predict(X_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

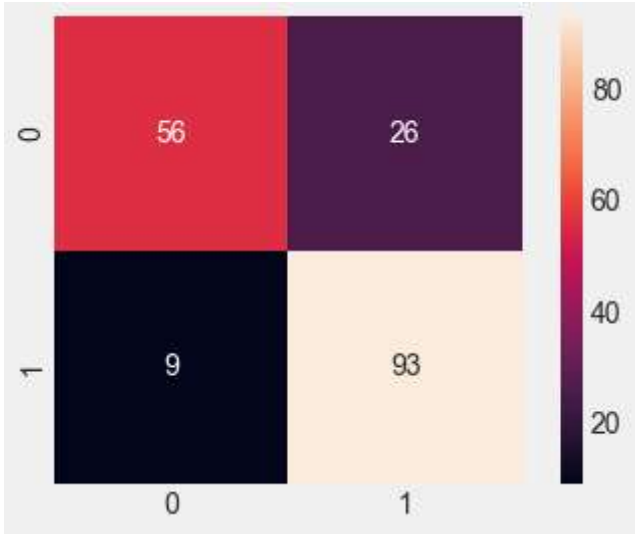
```
Accuarcy :- 0.8043478260869565
*****************
              precision    recall  f1-score   support

           0       0.80      0.74      0.77        82
           1       0.81      0.85      0.83       102

    accuracy                           0.80       184
   macro avg       0.80      0.80      0.80       184
weighted avg       0.80      0.80      0.80       184

*****************
```



GradientBoostClassifier

In [98]:
```
Gradmodel = GradientBoostingClassifier(n_estimators=100,learning_rate=0.01,ccp_alpha=0.01)
Gradmodel.fit(X_train,y_train)
y_pred =Gradmodel.predict(X_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```

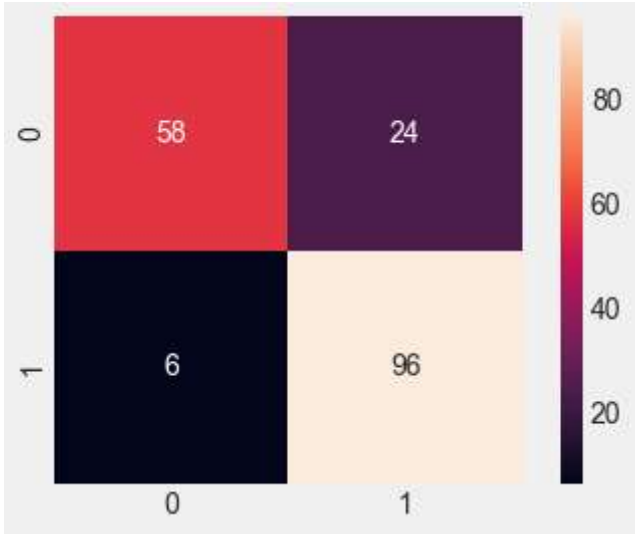
```
Accuarcy :- 0.8097826086956522
*****************
              precision    recall  f1-score   support

           0       0.86      0.68      0.76        82
           1       0.78      0.91      0.84       102

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

*****************
```



ExtraTreeClassifier

In [99]:
```
Extramodel = ExtraTreesClassifier(n_estimators=600,ccp_alpha=0.01)
Extramodel.fit(X_train,y_train)
y_pred =Extramodel.predict(X_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

```
Accuarcy :- 0.8369565217391305
*****************
              precision    recall  f1-score   support

           0       0.91      0.71      0.79        82
           1       0.80      0.94      0.86       102

    accuracy                           0.84       184
   macro avg       0.85      0.82      0.83       184
weighted avg       0.85      0.84      0.83       184

*****************
```



Support Vector Classifier

In [100]:
```
svc =SVC(kernel='rbf')
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

```
weighted avg       0.83      0.83      0.82       184

*****************
```

Model Perormance with Balanced Data.

In [101]:
```
y_train.value_counts()
```

Out[101]:
```
1    406
0    328
Name: HeartDisease, dtype: int64
```

In [102]:
```
y_test.value_counts()
```

Out[102]:
```
1    102
0     82
Name: HeartDisease, dtype: int64
```

- We are going to do oversampling notundersampling because it will not going loss data.

In [103]:
```
from imblearn.over_sampling import SMOTE
```

In [104]:
```
y_train.value_counts()
```

Out[104]:
```
1    406
0    328
Name: HeartDisease, dtype: int64
```

In [105]:
```
#transform the data
oversample = SMOTE()
X_trains,Y_trains= oversample.fit_resample(x_train,y_train)
```

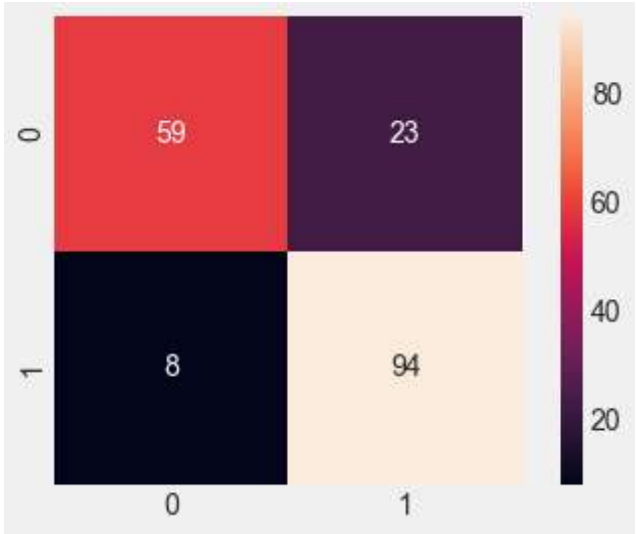
In [106]:
```
Y_trains.value_counts()
```

Out[106]:
```
1    406
0    406
Name: HeartDisease, dtype: int64
```

RandomForest

In [107]:
```
model = RandomForestClassifier(n_estimators=100,criterion='gini',min_samples_leaf=1)
model.fit(X_trains,Y_trains)
y_pred = model.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
print("**********************")
```

```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.88      0.72      0.79        82
           1       0.80      0.92      0.86       102

    accuracy                           0.83       184
   macro avg       0.84      0.82      0.83       184
weighted avg       0.84      0.83      0.83       184

*****************
```

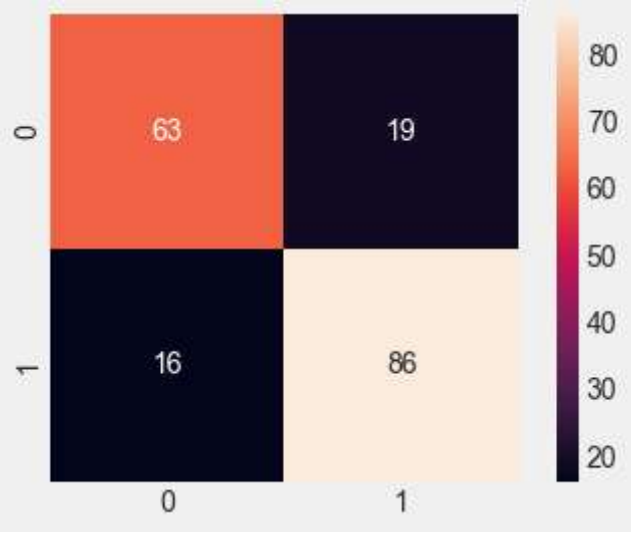


****

AdaboostClassifier

In [108]:
```
adamodel = AdaBoostClassifier(n_estimators=100)
adamodel.fit(X_trains,Y_trains)
y_pred = adamodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

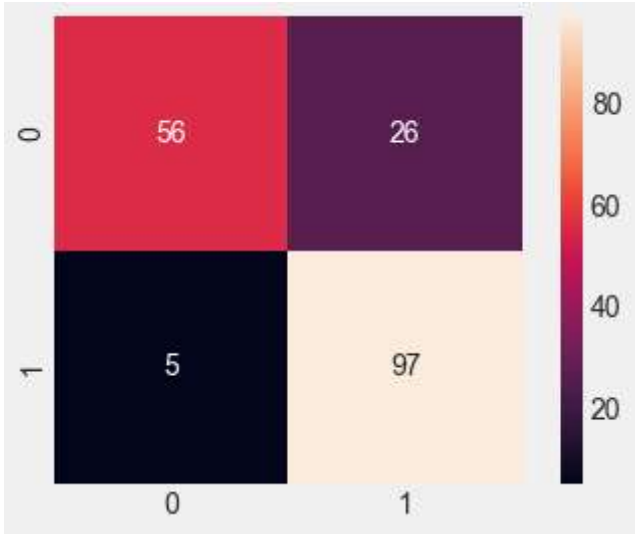
```
Accuarcy :- 0.8097826086956522
*****************
              precision    recall  f1-score   support

           0       0.80      0.77      0.78        82
           1       0.82      0.84      0.83       102

    accuracy                           0.81       184
   macro avg       0.81      0.81      0.81       184
weighted avg       0.81      0.81      0.81       184

*****************
```



GradientBoostingClassifier

In [109]:
```
Gradmodel = GradientBoostingClassifier(n_estimators=100,learning_rate=0.01,ccp_alpha=0.01)
Gradmodel.fit(X_trains,Y_trains)
y_pred =Gradmodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```

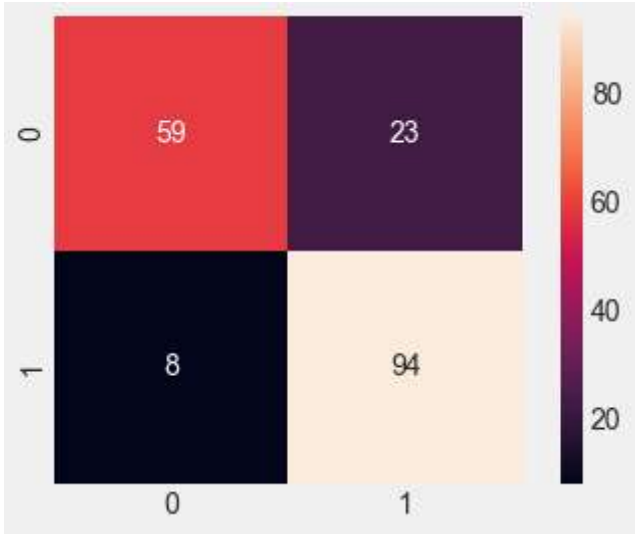
```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.92      0.68      0.78        82
           1       0.79      0.95      0.86       102

    accuracy                           0.83       184
   macro avg       0.85      0.82      0.82       184
weighted avg       0.85      0.83      0.83       184

*****************
```



ExtraTreeClassifer

In [110]:
```
Extramodel = ExtraTreesClassifier(n_estimators=600,ccp_alpha=0.01)
Extramodel.fit(X_trains,Y_trains)
y_pred =Extramodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt="d")
plt.show()
```

```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.88      0.72      0.79        82
           1       0.80      0.92      0.86       102

    accuracy                           0.83       184
   macro avg       0.84      0.82      0.83       184
weighted avg       0.84      0.83      0.83       184

*****************
```



- As we notied that model with Balanced dataset not produced good accuarcy as much as we required because there was not such kind of big

imbalancedment in our train dataset so this was not helpful in this case.

Model Selection Based on HyperParameterTunning

- Refer:- https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/ (https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/)
- Refer :- https://neptune.ai/blog/optuna-guide-how-to-monitor-hyper-parameter-optimization-runs (https://neptune.ai/blog/optuna-guide-how-to-monitor-hyper-parameter-optimization-runs)

RandomForest without Scalled data HyperParameter Tunning

In [111]:
```
import optuna
import sklearn

def objective(trial):
      data = x_train
      n_estimators = trial.suggest_int('n_estimators', 2,900)
      max_depth = int(trial.suggest_loguniform('max_depth', 1, 32))
      min_samples_leaf=int(trial.suggest_loguniform('min_samples_leaf', 1,10))
      criterion = trial.suggest_categorical('criterion', ["gini", "entropy", "log_loss"])
      ccp_alpha= float(trial.suggest_loguniform('ccp_alpha',0.01, 1))
      clf = sklearn.ensemble.RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth,criterion=criterion,c
      return sklearn.model_selection.cross_val_score(clf,x_train, y_train, scoring="accuracy",
           n_jobs=-1, cv=5).mean()
```

In [114]:
```
study = optuna.create_study(direction='maximize', study_name="experiment93", storage='sqlite:///starter.db')
```

```
[I 2022-10-14 21:43:26,885] A new study created in RDB with name: experiment93
```

In [115]:
```
study.optimize(objective, n_trials=10)
```

```
[I 2022-10-14 21:43:35,988] Trial 0 finished with value: 0.8365576367533315 and parameters: {'n_estimators': 789, 'max_
depth': 16.281512644044284, 'min_samples_leaf': 5.341739260515409, 'criterion': 'gini', 'ccp_alpha': 0.0789265341222504
2}. Best is trial 0 with value: 0.8365576367533315.
[I 2022-10-14 21:43:37,188] Trial 1 finished with value: 0.83654831795732 and parameters: {'n_estimators': 151, 'max_de
pth': 3.23218578571228, 'min_samples_leaf': 6.8087338823940495, 'criterion': 'entropy', 'ccp_alpha': 0.1336111977647739
5}. Best is trial 0 with value: 0.8365576367533315.
[I 2022-10-14 21:43:42,556] Trial 2 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 865, 'max_
depth': 18.255461906385115, 'min_samples_leaf': 5.526843121198774, 'criterion': 'entropy', 'ccp_alpha': 0.5234641043838
497}. Best is trial 0 with value: 0.8365576367533315.
[I 2022-10-14 21:43:44,464] Trial 3 finished with value: 0.8597148448420462 and parameters: {'n_estimators': 312, 'max_
depth': 3.433645164876308, 'min_samples_leaf': 7.305350353636437, 'criterion': 'entropy', 'ccp_alpha': 0.01895143383928
2168}. Best is trial 3 with value: 0.8597148448420462.
[I 2022-10-14 21:43:47,316] Trial 4 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 462, 'max_
depth': 12.855362595516697, 'min_samples_leaf': 2.00939687595647, 'criterion': 'gini', 'ccp_alpha': 0.4983732687868584
5}. Best is trial 3 with value: 0.8597148448420462.
[I 2022-10-14 21:43:51,426] Trial 5 finished with value: 0.8597055260460349 and parameters: {'n_estimators': 782, 'max_
depth': 3.257561756320972, 'min_samples_leaf': 1.606867392235131, 'criterion': 'gini', 'ccp_alpha': 0.02694430044419906
7}. Best is trial 3 with value: 0.8597148448420462.
[I 2022-10-14 21:43:53,277] Trial 6 finished with value: 0.8597055260460348 and parameters: {'n_estimators': 284, 'max_
depth': 4.003860960895839, 'min_samples_leaf': 4.057136694192428, 'criterion': 'gini', 'ccp_alpha': 0.01099273823641450
6}. Best is trial 3 with value: 0.8597148448420462.
[I 2022-10-14 21:43:56,462] Trial 7 finished with value: 0.8174354673376201 and parameters: {'n_estimators': 469, 'max_
depth': 16.629469492295065, 'min_samples_leaf': 1.1675082447373373, 'criterion': 'log_loss', 'ccp_alpha': 0.25602388946
43896}. Best is trial 3 with value: 0.8597148448420462.
[I 2022-10-14 21:44:01,396] Trial 8 finished with value: 0.8433603578417668 and parameters: {'n_estimators': 720, 'max_
depth': 11.723029093293803, 'min_samples_leaf': 2.7222742825395616, 'criterion': 'log_loss', 'ccp_alpha': 0.10392882278
780796}. Best is trial 3 with value: 0.8597148448420462.
[I 2022-10-14 21:44:06,160] Trial 9 finished with value: 0.862426614481409 and parameters: {'n_estimators': 748, 'max_d
epth': 3.2827574208393275, 'min_samples_leaf': 1.248636351910885, 'criterion': 'entropy', 'ccp_alpha': 0.01610037602828
985}. Best is trial 9 with value: 0.862426614481409.
```

In [117]:
```
trial = study.best_trial

print('Accuaracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

```
Accuaracy: 0.862426614481409
Best hyperparameters: {'ccp_alpha': 0.01610037602828985, 'criterion': 'entropy', 'max_depth': 3.2827574208393275, 'min_
samples_leaf': 1.248636351910885, 'n_estimators': 748}
```

- The above we can get hyperparameter tunning with the help `Otuna` python library.

Final Model Selection Criteria

- As we have seen lot of model performing well since we are with good accuracy but we have data that is regarding to health sector and every patient is in the risk if we provide wrong classification for those who have heart diesase.
- Since we have decide to go ahead with the model who have reached the best Recall & F1 Score with desent accuracy.
- Because Recall is giving us prediction from actual data that means it will provide accurate prediction for those who heart diease and those who dont have heart diease.
- In the scalled data we have found GradientBoost and Randomforest perfomed well in terms of Recall.
- We are going to take only these model.
- Lets see their hyperparameter tunning GradientBoost with Scalled data.

Hyperparameter Tunning of RandormForest On Scalled Data.

In [118]:
```
import optuna
import sklearn

def objective(trial):
      data = X_train
      n_estimators = trial.suggest_int('n_estimators', 2,900)
      max_depth = int(trial.suggest_loguniform('max_depth', 1, 32))
      min_samples_leaf=int(trial.suggest_loguniform('min_samples_leaf', 1,10))
      criterion = trial.suggest_categorical('criterion', ["gini", "entropy", "log_loss"])
      ccp_alpha= float(trial.suggest_loguniform('ccp_alpha',0.01, 1))
      clf = sklearn.ensemble.RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth,criterion=criterion,c
      return sklearn.model_selection.cross_val_score(clf,data, y_train, scoring="accuracy",
           n_jobs=-1, cv=5).mean()
```

In [121]:
```
study = optuna.create_study(direction='maximize', study_name="experiment88", storage='sqlite:///starter.db')
```

```
[I 2022-10-14 21:45:02,499] A new study created in RDB with name: experiment88
```

In [122]:
```
study.optimize(objective, n_trials=10)
```

```
[I 2022-10-14 21:45:04,098] Trial 0 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 52, 'max_depth': 5.749253167070718, 'min_samples_leaf': 2.974350785577008, 'criterion': 'gini', 'ccp_alpha': 0.2569245303506593}. Best is trial 0 with value: 0.5531357748578885.
[I 2022-10-14 21:45:08,659] Trial 1 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 635, 'max_depth': 1.0974588075384268, 'min_samples_leaf': 3.1149854084397055, 'criterion': 'log_loss', 'ccp_alpha': 0.6646710154463822}. Best is trial 0 with value: 0.5531357748578885.
[I 2022-10-14 21:45:10,735] Trial 2 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 223, 'max_depth': 1.7433232300777288, 'min_samples_leaf': 1.8812367226812605, 'criterion': 'entropy', 'ccp_alpha': 0.37847245963729037}. Best is trial 0 with value: 0.5531357748578885.
[I 2022-10-14 21:45:17,330] Trial 3 finished with value: 0.8583449818283476 and parameters: {'n_estimators': 526, 'max_depth': 23.854181035457003, 'min_samples_leaf': 1.808318844286502, 'criterion': 'log_loss', 'ccp_alpha': 0.03582308648061634}. Best is trial 3 with value: 0.8583449818283476.
[I 2022-10-14 21:45:21,075] Trial 4 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 595, 'max_depth': 2.743550584198023, 'min_samples_leaf': 2.1685998175116943, 'criterion': 'gini', 'ccp_alpha': 0.8033681321759675}. Best is trial 3 with value: 0.8583449818283476.
[I 2022-10-14 21:45:23,470] Trial 5 finished with value: 0.8447209020594538 and parameters: {'n_estimators': 382, 'max_depth': 1.529940999165397, 'min_samples_leaf': 5.3424140627269345, 'criterion': 'gini', 'ccp_alpha': 0.011805991784429308}. Best is trial 3 with value: 0.8583449818283476.
[I 2022-10-14 21:45:25,569] Trial 6 finished with value: 0.5980337340415618 and parameters: {'n_estimators': 273, 'max_depth': 11.269802541098219, 'min_samples_leaf': 7.5957971038465795, 'criterion': 'log_loss', 'ccp_alpha': 0.31034624799840255}. Best is trial 3 with value: 0.8583449818283476.
[I 2022-10-14 21:45:28,607] Trial 7 finished with value: 0.5531357748578885 and parameters: {'n_estimators': 450, 'max_depth': 10.613088492553386, 'min_samples_leaf': 8.662297943182447, 'criterion': 'entropy', 'ccp_alpha': 0.7469643484640409}. Best is trial 3 with value: 0.8583449818283476.
[I 2022-10-14 21:45:33,115] Trial 8 finished with value: 0.8556145745969619 and parameters: {'n_estimators': 661, 'max_depth': 10.97563092714708, 'min_samples_leaf': 1.4224298621349944, 'criterion': 'entropy', 'ccp_alpha': 0.034797409292355529}. Best is trial 3 with value: 0.8583449818283476.
[I 2022-10-14 21:45:37,161] Trial 9 finished with value: 0.8406392694063927 and parameters: {'n_estimators': 723, 'max_depth': 2.408554242483022, 'min_samples_leaf': 1.9254902291912424, 'criterion': 'gini', 'ccp_alpha': 0.050175429685477314}. Best is trial 3 with value: 0.8583449818283476.
```

In [123]:
```
trial = study.best_trial

print('Accuaracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

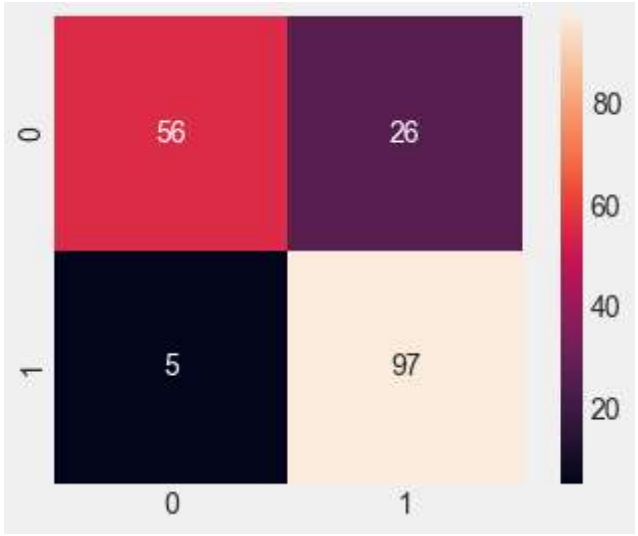
```
Accuaracy: 0.8583449818283476
Best hyperparameters: {'ccp_alpha': 0.03582308648061634, 'criterion': 'log_loss', 'max_depth': 23.854181035457003, 'min
_samples_leaf': 1.808318844286502, 'n_estimators': 526}
```

Hyperparameter Tunning of GradientBoost On Balanced Data.

Below is the base model which we had created

In [124]:
```
Gradmodel = GradientBoostingClassifier(n_estimators=100,learning_rate=0.01,ccp_alpha=0.01)
Gradmodel.fit(X_trains,Y_trains)
y_pred =Gradmodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```

```
Accuarcy :- 0.8315217391304348
*****************
              precision    recall  f1-score   support

           0       0.92      0.68      0.78        82
           1       0.79      0.95      0.86       102

    accuracy                           0.83       184
   macro avg       0.85      0.82      0.82       184
weighted avg       0.85      0.83      0.83       184

*****************
```



Let's Do RandomSearchCV Hyperparameter Tunning.

In [125]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [126]:
```
Parameter_Trials={'n_estimators':[int(x) for x in np.linspace(start = 100, stop = 500, num = 10)],
                  "loss": ['log_loss', 'deviance', 'exponential'],
                  "learning_rate":[float(x) for x in np.linspace(start = 1, stop = 0.01, num = 50)],
                  "criterion":['friedman_mse', 'squared_error', 'mse'],
                  'max_depth':[int(x) for x in np.arange(1,10,1)],
                 "ccp_alpha":[float(x) for x in np.linspace(start = 0.01, stop = 0.3, num = 100)]}
```

In [127]:
```
# Create a based model
Grad = GradientBoostingClassifier()
# Instantiate the grid search model
grid_search = RandomizedSearchCV( Grad,Parameter_Trials,cv = 50, n_jobs = -1, verbose = 0,random_state=42)
```

In [128]:
```
grid_search.fit(X_trains,Y_trains)
```

Out[128]:
```
RandomizedSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

In [131]:
```
grid_search.best_params_
```

Out[131]:
```
{'n_estimators': 100,
 'max_depth': 4,
 'loss': 'exponential',
 'learning_rate': 0.7777551020408163,
 'criterion': 'friedman_mse',
 'ccp_alpha': 0.06272727272727271}
```

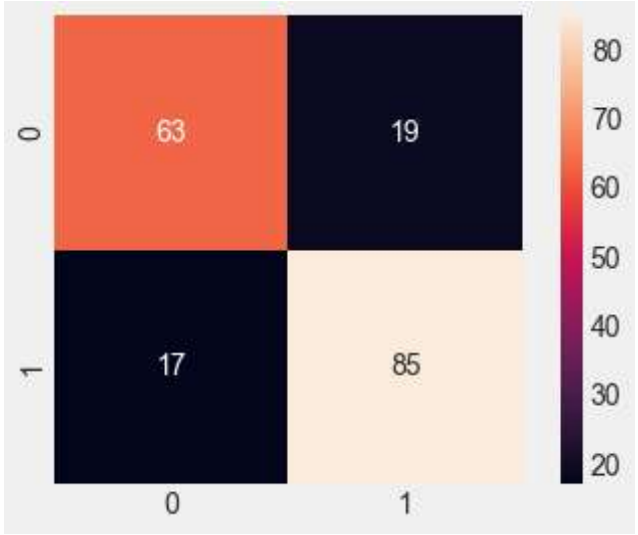
In [132]:
```
grid_search.best_score_
```

Out[132]:
```
0.8372794117647058
```

Apply The above Hyperparameter & Start Building our Final Model.

In [134]:
```
Gradmodel = GradientBoostingClassifier(n_estimators=100,max_depth= 4,loss='exponential',learning_rate= 0.777755102040816
 ccp_alpha=0.06272727272727271)
Gradmodel.fit(X_trains,Y_trains)
y_pred =Gradmodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```

```
Accuarcy :- 0.8043478260869565
*****************
              precision    recall  f1-score   support

           0       0.79      0.77      0.78        82
           1       0.82      0.83      0.83       102

    accuracy                           0.80       184
   macro avg       0.80      0.80      0.80       184
weighted avg       0.80      0.80      0.80       184

*****************
```



Overfit or Underfit Model understanding

- Overfitting :- overfitting, where a model matches the training data almost perfectly, but does poorly in validation and other new data.
- Underfitting :- When a model fails to capture important distinctions and patterns in the data, so it performs poorly even in training data, that is called underfitting.

In [135]:
```
error_accuarcy =1- accuracy_score(y_test,y_pred)
error_recall = 1-recall_score(y_test,y_pred)
print('Accuarcy error :-',error_accuarcy)
print('Recall error :-',error_recall)
```

```
Accuarcy error :- 0.19565217391304346
Recall error :- 0.16666666666666663
```

- Since we are not concern about the accuaracy in this case
- Then this time Training_Score & Testing_Score are not so important.

In [136]:
```
Training_Score = Gradmodel.score(X_train,y_train)
Testing_Score  = Gradmodel.score(X_test,y_test)
print("Training_Score :-",Training_Score)
print("Testing_Score :-",Testing_Score)
```

```
Training_Score :- 0.5490463215258855
Testing_Score :- 0.5380434782608695
```

In [137]:
```
train_score = cross_val_score(Gradmodel,X_trains,Y_trains,scoring='accuracy',cv=10,n_jobs=-1,verbose=0)
train_score
```

Out[137]:
```
array([0.90243902, 0.82926829, 0.80246914, 0.81481481, 0.80246914,
       0.81481481, 0.87654321, 0.85185185, 0.81481481, 0.85185185])
```

In [138]:
```
np.std(train_score)
```

Out[138]:
```
0.031938486666871664
```

In [139]:
```
np.mean(train_score)
```

Out[139]:
```
0.83613369467028
```

In [140]:
```
test_score = cross_val_score(Gradmodel,x_test,y_test,scoring='accuracy',cv=10,n_jobs=-1,verbose=0)
test_score
```

Out[140]:
```
array([0.78947368, 0.73684211, 0.68421053, 0.73684211, 0.83333333,
       1.        , 0.72222222, 0.83333333, 0.83333333, 0.83333333])
```

In [141]:
```
np.std(test_score)
```

Out[141]:
```
0.08484622388292809
```

In [142]:
```
np.mean(test_score)
```

Out[142]:
```
0.8002923976608187
```

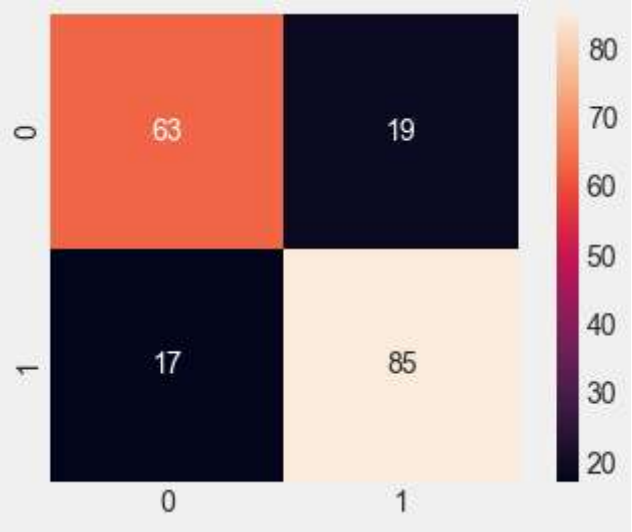
- Since it is prividing us accuaracy less standard deviation and consitant enought to give better accuarcy at unique unseen dataset.

- As we required **high recall/sentivity/True Positive Rate than False Negative Rate.**
- **TPR > FPR**

AUC-ROC Curve.

- The metrics change with the changing threshold values. We can generate different confusion matrices and compare the various metrics. But that would not be a prudent thing to do. Instead, what we can do is generate a plot between some of these metrics so that we can easily visualize which threshold is giving us a better result.
- The AUC-ROC curve solves just that problem!

In [143]:
```
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```



Sensitivity / True Positive Rate / Recall

$$Sensitivity = \frac{TP}{TP + FN}$$

Specificity / True Negative Rate

$$Specificity = \frac{TN}{TN + FP}$$

Specificity tells us what proportion of the negative class got correctly classified.

Taking the same example as in Sensitivity, Specificity would mean determining the proportion of healthy people who were correctly identified by the model.

False Positive Rate

$$FPR = \frac{FP}{TN + FP} = 1 - Specificity$$

FPR tells us what proportion of the negative class got incorrectly classified by the classifier.

Refer :- https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/ (https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/)

Probablities For The Prediction

- A machine learning classification model can be used to predict the actual class of the data point directly or predict its probability of belonging to different classes. The latter gives us more control over the result. We can determine our own threshold to interpret the result of the classifier. This is sometimes more prudent than just building a completely new model!
- Setting different thresholds for classifying positive class for data points will inadvertently change the Sensitivity and Specificity of the model. And one of these thresholds will probably give a better result than the others, depending on whether we are aiming to lower the number of False Negatives or False Positives

- When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives
- In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

In [144]:
```
from sklearn.metrics import roc_curve
pred_prob = Gradmodel.predict_proba(x_test)
fpr, tpr, thresh = roc_curve(y_test, pred_prob[:,1], pos_label=1)
```

In [145]:
```
fpr
```

Out[145]:
```
array([0.        , 0.        , 0.06097561, 0.08536585, 0.14634146,
       0.18292683, 0.23170732, 0.29268293, 0.37804878, 1.        ])
```

In [146]:
```
tpr
```

Out[146]:
```
array([0.        , 0.10784314, 0.52941176, 0.53921569, 0.65686275,
       0.80392157, 0.83333333, 0.91176471, 0.96078431, 1.        ])
```

In [147]:
```
thresh
```

Out[147]:
```
array([1.91598773, 0.91598773, 0.90107655, 0.77264496, 0.73952682,
       0.62464061, 0.52846776, 0.48355547, 0.3415396 , 0.14607125])
```

In [148]:
```
# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

In [149]:
```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score = roc_auc_score(y_test, pred_prob[:,1])


print(auc_score)
```

```
0.8784672405547584
```

In [150]:
```
dictionary = dict(tpr=tpr,fpr=fpr,thresh=thresh)
```

In [151]:
```
dictionary
```

Out[151]:
```
{'tpr': array([0.        , 0.10784314, 0.52941176, 0.53921569, 0.65686275,
       0.80392157, 0.83333333, 0.91176471, 0.96078431, 1.        ]),
 'fpr': array([0.        , 0.        , 0.06097561, 0.08536585, 0.14634146,
       0.18292683, 0.23170732, 0.29268293, 0.37804878, 1.        ]),
 'thresh': array([1.91598773, 0.91598773, 0.90107655, 0.77264496, 0.73952682,
       0.62464061, 0.52846776, 0.48355547, 0.3415396 , 0.14607125])}
```

In [152]:
```
summary = pd.DataFrame(dictionary,columns=['tpr','fpr','thresh'])
summary
```

Out[152]:

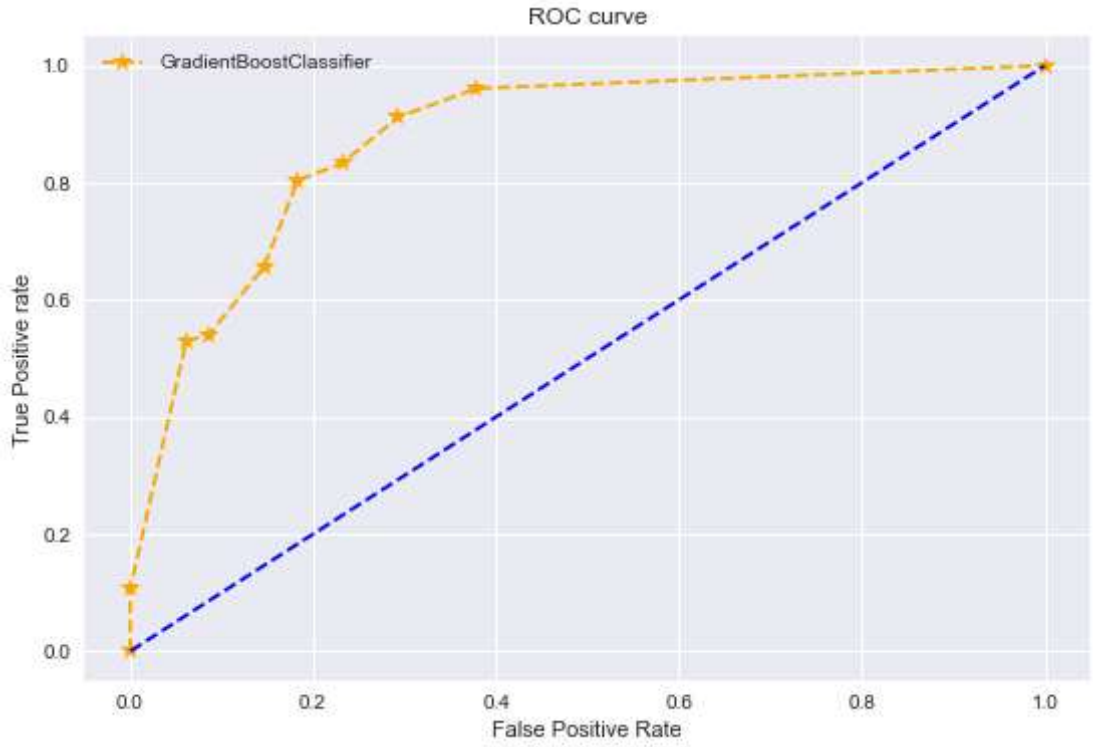
	tpr	fpr	thresh
0	0.000000	0.000000	1.915988
1	0.107843	0.000000	0.915988
2	0.529412	0.060976	0.901077
3	0.539216	0.085366	0.772645
4	0.656863	0.146341	0.739527
5	0.803922	0.182927	0.624641
6	0.833333	0.231707	0.528468
7	0.911765	0.292683	0.483555
8	0.960784	0.378049	0.341540
9	1.000000	1.000000	0.146071

In [153]:
```
summary.sort_values('tpr',ascending=False)[1:5]
```

Out[153]:

	tpr	fpr	thresh
8	0.960784	0.378049	0.341540
7	0.911765	0.292683	0.483555
6	0.833333	0.231707	0.528468
5	0.803922	0.182927	0.624641

In [154]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr, tpr, marker="*",markersize=10,linestyle='--',color='orange',label='GradientBoostClassifier')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```



- We need model with higher TPR and lesser FPR.
- we need select the model on the threshold nearest one where FPR will be less.

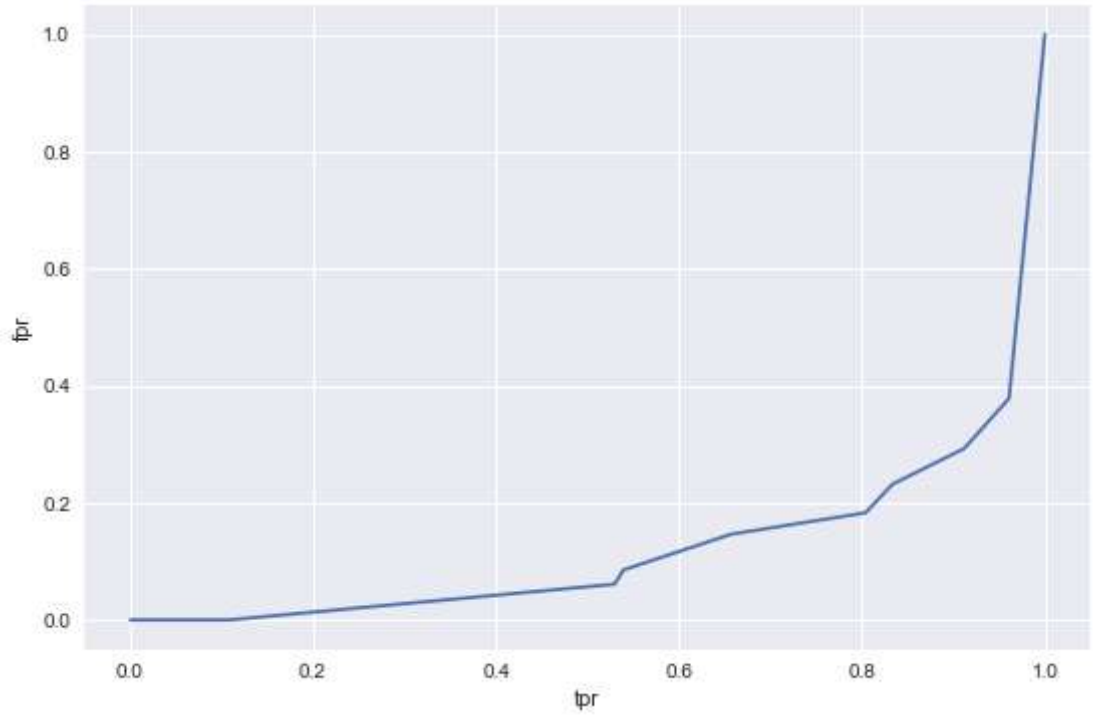
In [155]:
```
summary.sort_values('tpr',ascending=False)[1:5]
```

Out[155]:

	tpr	fpr	thresh
8	0.960784	0.378049	0.341540
7	0.911765	0.292683	0.483555
6	0.833333	0.231707	0.528468
5	0.803922	0.182927	0.624641

In [156]:
```
sns.lineplot(x=summary['tpr'],y=summary['fpr'],data=summary)
```

Out[156]:
```
<AxesSubplot:xlabel='tpr', ylabel='fpr'>
```



In [157]:
```
summary.sort_values('tpr',ascending=False)[1:8]
```

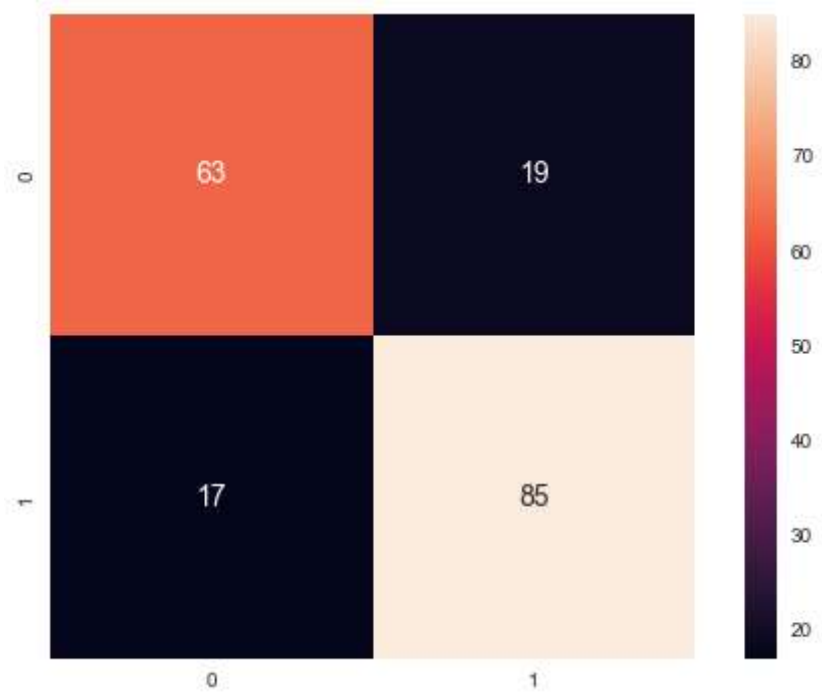
Out[157]:

	tpr	fpr	thresh
8	0.960784	0.378049	0.341540
7	0.911765	0.292683	0.483555
6	0.833333	0.231707	0.528468
5	0.803922	0.182927	0.624641
4	0.656863	0.146341	0.739527
3	0.539216	0.085366	0.772645
2	0.529412	0.060976	0.901077

- Here we have been focusing on the TPR & threshold
- we need to know when tpr is high and fpr low and at which point thershold will classify our classes optimally.
- first we are taking 0.75 is the point here TPR > FPR and threshold is also 60 at position.
- 0.75 is taken by me its upto ur business understanding (Domain knowledge)
- Once class probablity is greater than 75% then he will be the heart patient otherwise he will not

In [158]:
```
Gradmodel = GradientBoostingClassifier(n_estimators=100,max_depth= 4,loss='exponential',learning_rate= 0.777755102040816
 ccp_alpha=0.06272727272727271)
Gradmodel.fit(X_trains,Y_trains)
y_pred =Gradmodel.predict(x_test)
print('Accuarcy :-',accuracy_score(y_test,y_pred))
print("*****************")
print(classification_report(y_test,y_pred))
print("*****************")
matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(matrix,annot=True,square=True,fmt='d')
plt.show()
```

```
Accuarcy :- 0.8043478260869565
*****************
              precision    recall  f1-score   support

           0       0.79      0.77      0.78        82
           1       0.82      0.83      0.83       102

    accuracy                           0.80       184
   macro avg       0.80      0.80      0.80       184
weighted avg       0.80      0.80      0.80       184

*****************
```



In [222]:
```
from sklearn.metrics import roc_curve
pred_prob = Gradmodel.predict_proba(x_test)
fpr, tpr, thresh = roc_curve(y_test, pred_prob[:,1])
```

In [223]:
```
y_predict_prob_class_1 =  pred_prob[:,1]
```

In [224]:
```
y_predict_class = [1 if prob > 0.75 else 0 for prob in y_predict_prob_class_1]
print("Accuracy:",accuracy_score(y_test, y_predict_class))
print("Recall:",recall_score(y_test, y_predict_class))
```

```
Accuracy: 0.7065217391304348
Recall: 0.5392156862745098
```

- Here 75 is not otimal threshold to predict theb classses optimally.
- Here,at the 5% I am getting best classification ever with higher accuaracy and higher recall.

In [225]:
```
from sklearn.metrics import f1_score,precision_score
```

In [226]:
```
y_predict_prob_class_1 =  pred_prob[:,1]
y_predict_class = [1 if prob > 0.5 else 0 for prob in y_predict_prob_class_1]
print("Accuracy (Sensitivity):",accuracy_score(y_test, y_predict_class))
print("Recall:",recall_score(y_test, y_predict_class))
print("F1-Score:",f1_score(y_test, y_predict_class))
print("Precision-Score:",precision_score(y_test, y_predict_class))
```

```
Accuracy (Sensitivity): 0.8043478260869565
Recall: 0.8333333333333334
F1-Score: 0.825242718446602
Precision-Score: 0.8173076923076923
```

Final Evalution on model

- As we know at the at the 0.5 FPR is very less than TPR
- TPR means the Recall that highly concerned with True value from the data which has to be accurately predicted.
- Hence we have Heart patient data we alway make sure patient who have heart dieases need to be predited accuractely.
- We don't care about patient who do not have heart diease that's why we can see precision score is lesser than the recall & FPR is also low.

In [227]:
```
data_sequence_info=['Age', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'MaxHR', 'ExerciseAngina', 'Oldpeak', 'Citizen', 'Flat', 'Up', 'M',
       'Normal', 'ST']
```

In [228]:
```
probablity= Gradmodel.predict_proba([[2,2,0,0,0.0,100,0,0.0,0,0,0,0,0,1]])
```

In [229]:
```
probablity[0]
```

Out[229]:
```
array([0.22735504, 0.77264496])
```

In [230]:
```
if probablity[0][1]>=0.5:
    print('He is the Heart Patient.')
elif probablity[0][0]<0.5:
    print('He is safe.')
```

```
He is the Heart Patient.
```

In [206]:
```
x_train.columns
```

Out[206]:
```
Index(['Age', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'MaxHR', 'ExerciseAngina', 'Oldpeak', 'Citizen', 'Flat', 'Up', 'M',
       'Normal', 'ST'],
      dtype='object')
```

- Here is our business choice to chose the best threshould as per busienss requirement or Domain knowledge.

Thank You !