In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import pickle

from warnings import filterwarnings
filterwarnings('ignore')
from matplotlib.collections import PathCollection
from statsmodels.graphics.gofplots import qqplot
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
from sklearn.metrics import classification_report, accuracy_score
```

In [3]:
```
df =pd.read_csv ("C:/Users/admin/OneDrive/Desktop/dataset/heart.csv")
```

In [4]:
```
df.describe()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	tha
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.64
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.90
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.50
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.00

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

Out[5]:
```
(303, 14)
```

In [6]:
```
df.sex.value_counts()
```

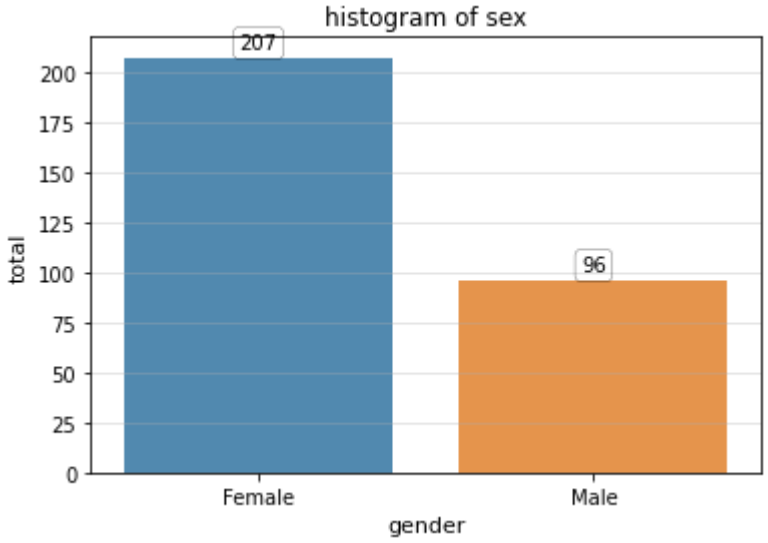
Out[6]:
```
1    207
0     96
Name: sex, dtype: int64
```

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

Out[7]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [8]:
```
labels = ['Female','Male']
order=df['sex']. value_counts().index
ax = sns.countplot(x = 'sex',data = df, order = order, alpha = 0.85)
plt.title('histogram of sex')

for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none',linewidth=0.25, boxstyle='round'))
plt.xlabel('gender',fontsize=11, fontfamily='sans-serif')
plt.ylabel('total',fontsize=11, fontfamily='sans-serif')
plt.xticks([0, 1], labels)
plt.grid(axis='y', alpha=0.4)
plt.show()
print('*' * 25)
print('\033[1m'+'.: Sex (Gender) Total :.'+'\033[0m')
print('*' * 25)
df.sex.value_counts(dropna=False)
```



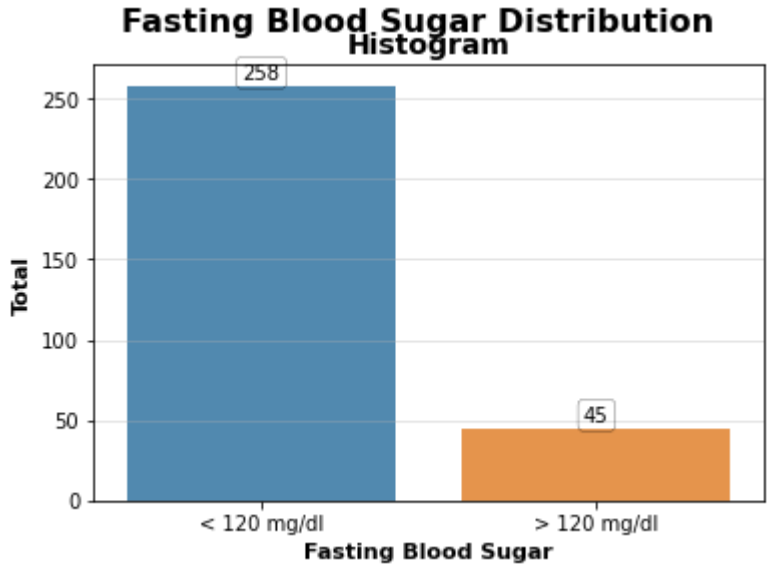

```
*************************
.: Sex (Gender) Total :.
*************************
```

Out[8]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [9]:
```
#The distribution of female patients are highest compared to male patients .
```

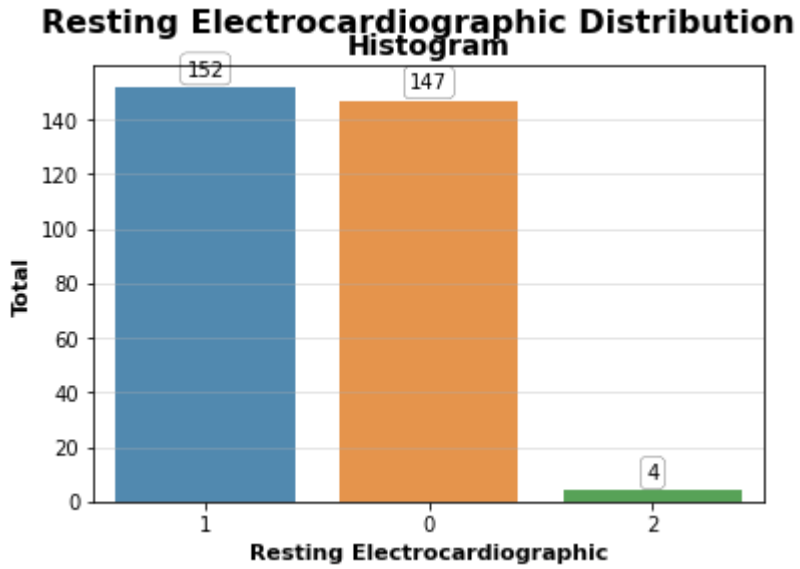
In [ ]:
```
#Chest pain type 0 have the highest number compared to other types of chest pain.
```

In [9]:
```
labels=['< 120 mg/dl', '> 120 mg/dl']
order=df['fbs'].value_counts().index
plt.suptitle('Fasting Blood Sugar Distribution', fontweight='heavy',
             fontsize=16, fontfamily='sans-serif')
plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='fbs', data=df, order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none', linewidth=0.25,
                       boxstyle='round'))
plt.xlabel('Fasting Blood Sugar', fontweight='bold', fontsize=11,
           fontfamily='sans-serif')
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.xticks([0, 1], labels)
plt.grid(axis='y', alpha=0.4)
plt.show()
```



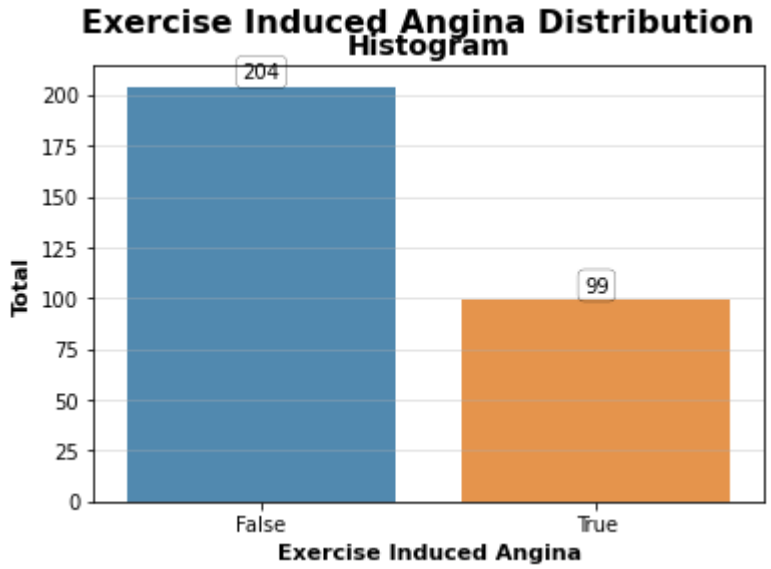
In [ ]:
```
#It can be seen that the number of patients with fasting blood sugar less than 12
```

In [10]:
```
labels=['1', '0', '2']
order=df['restecg'].value_counts().index
plt.suptitle('Resting Electrocardiographic Distribution', fontweight='heavy',
             fontsize=16, fontfamily='sans-serif')
plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='restecg', data=df,  order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none',  linewidth=0.25,
                       boxstyle='round'))
    plt.xlabel('Resting Electrocardiographic', fontweight='bold', fontsize=11,
           fontfamily='sans-serif')
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.grid(axis='y', alpha=0.4)
```



In [ ]:
```
# Resting electrocardiographic with results 1 and 0 has a higher distribution tha
#In addition, result 1 has the highest distribution compared to the other results
```

In [11]:
```
labels=['False', 'True']
order=df['exang'].value_counts().index
plt.suptitle('Exercise Induced Angina Distribution', fontweight='heavy',
             fontsize=16, fontfamily='sans-serif')
plt.title('Pie Chart', fontweight='bold', fontsize=14, fontfamily='sans-serif')
plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='exang', data=df, order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none', linewidth=0.25,
                       boxstyle='round'))
plt.xlabel('Exercise Induced Angina', fontweight='bold', fontsize=11,
           fontfamily='sans-serif')
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.xticks([0, 1], labels)
plt.grid(axis='y', alpha=0.4)
```



In [ ]:
```
#Patients with no exercise induced angina are the highest compared to patients wi
```

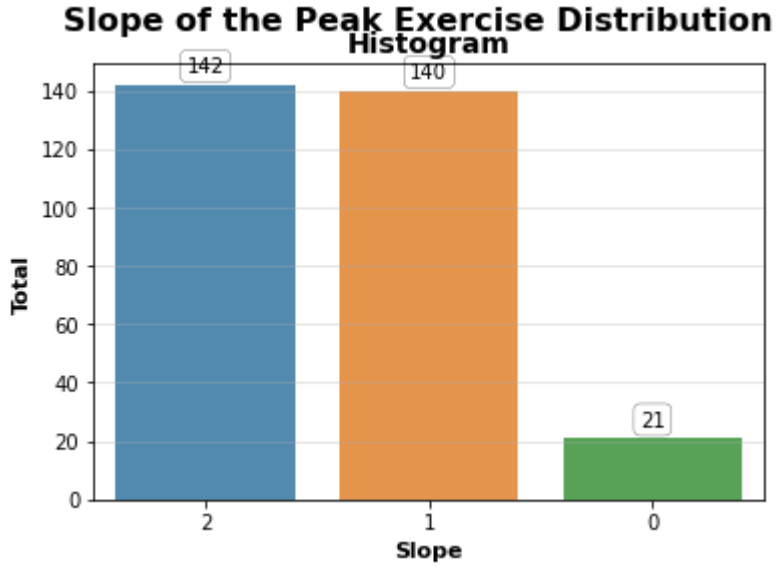
In [ ]:
```

```

In [ ]:
```
#slope (Slope of the Peak Exercise)
```

In [12]:
```
labels=['2', '1', '0']
order=df['slope'].value_counts().index
plt.suptitle('Slope of the Peak Exercise Distribution', fontweight='heavy',
             fontsize=16, fontfamily='sans-serif')
plt.title('Pie Chart', fontweight='bold', fontsize=14,
          fontfamily='sans-serif')

plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='slope', data=df, order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none', linewidth=0.25,
                       boxstyle='round'))
plt.xlabel('Slope', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.grid(axis='y', alpha=0.4)
```



In [ ]:
```
# The distribution of slope 1 and 2 are almost the same.
```

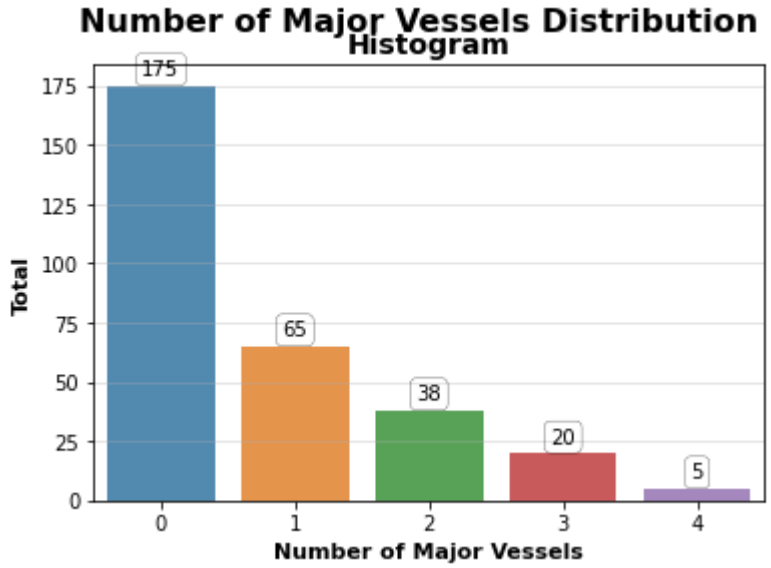
In [ ]:
```
# Moreover, slope 2 has the highest distribution compared to others.
```

In [ ]:
```

```

In [ ]:
```
#ca (Number of Major Vessels)
```

In [13]:
```
labels=['0', '1', '2', '3', '4']
order=df['ca'].value_counts().index
plt.suptitle('Number of Major Vessels Distribution', fontweight='heavy',
             fontsize=16, fontfamily='sans-serif')
plt.title('Pie Chart', fontweight='bold', fontsize=14, fontfamily='sans-serif')
plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='ca', data=df, order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none',  linewidth=0.25,
                       boxstyle='round'))
    plt.xlabel('Number of Major Vessels', fontweight='bold', fontsize=11,
           fontfamily='sans-serif',)
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.grid(axis='y', alpha=0.4)
```



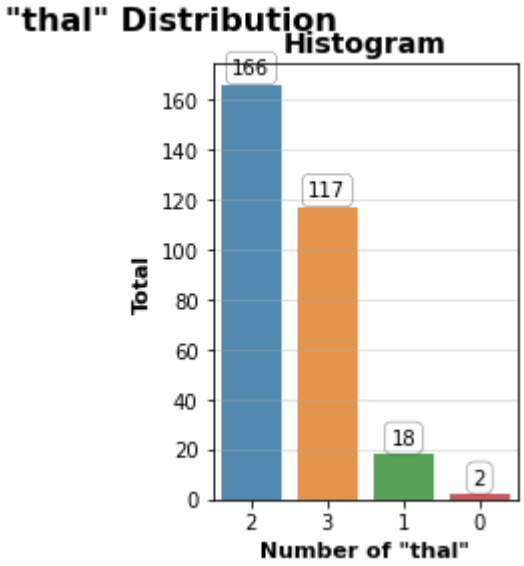
In [ ]:
```
#People with 0 major vessel has the highest distribution compared to others.
```

In [ ]:
```

```

In [ ]:
```
# thal
```

In [14]:
```
labels=['2', '3', '1', '0']
order=df['thal'].value_counts().index
plt.suptitle('"thal" Distribution', fontweight='heavy', fontsize=16,
             fontfamily='sans-serif')
plt.title('Pie Chart', fontweight='bold', fontsize=14, fontfamily='sans-serif')
countplt = plt.subplot(1, 2, 2)
plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='thal', data=df, order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none',  linewidth=0.25,
                       boxstyle='round'))
    plt.xlabel('Number of "thal"', fontweight='bold', fontsize=11,
           fontfamily='sans-serif')
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.grid(axis='y', alpha=0.4)
```

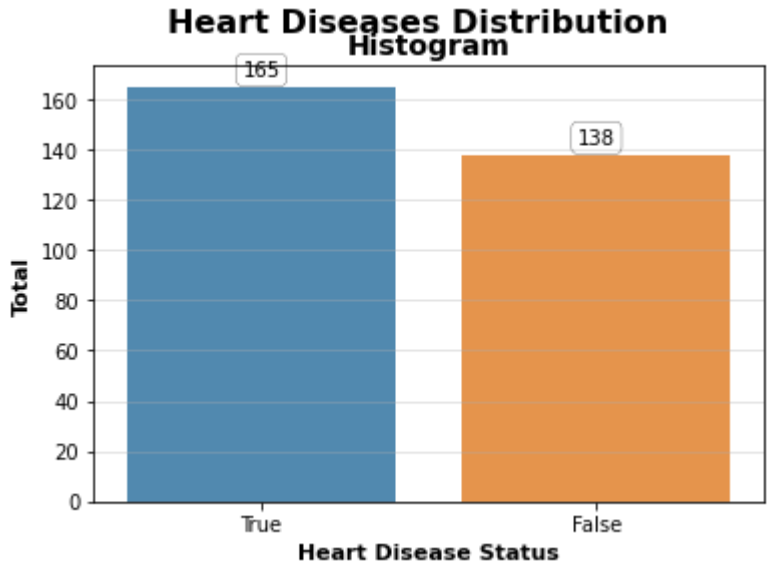


In [ ]:
```
#Patients with 2 "thal" has the highest distribution compared to others.
```

In [ ]:
```
df.describe()
```

In [ ]:
```
#target (Heart Diseases Status)
```

In [15]:
```
labels=['True', 'False']
order=df['target'].value_counts().index
plt.suptitle('Heart Diseases Distribution', fontweight='heavy',
             fontsize=16, fontfamily='sans-serif')
plt.title('Pie Chart', fontweight='bold', fontsize=14, fontfamily='sans-serif')
plt.title('Histogram', fontweight='bold', fontsize=14, fontfamily='sans-serif')
ax = sns.countplot(x='target', data=df,  order=order, alpha=0.85)
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+4.25,rect.get_height(),
             horizontalalignment='center', fontsize=10,
             bbox=dict(facecolor='none', linewidth=0.25,
                       boxstyle='round'))
    plt.xlabel('Heart Disease Status', fontweight='bold', fontsize=11,
           fontfamily='sans-serif')
plt.ylabel('Total', fontweight='bold', fontsize=11, fontfamily='sans-serif')
plt.xticks([0, 1], labels)
plt.grid(axis='y', alpha=0.4)
```



In [ ]:
```
#The total number of patients that have heart diseases are higher than patients t
```

In [ ]:
```

```

In [16]:
```
#Descriptive statistics
```

In [17]:
```
elect_dtypes(exclude='object').describe().T.style.background_gradient(cmap='PuRd'
```

Out[17]:

	count	mean	std	min	25%	50%	75%	m
age	303.000000	54.366337	9.082101	29.000000	47.500000	55.000000	61.000000	77.0000
sex	303.000000	0.683168	0.466011	0.000000	0.000000	1.000000	1.000000	1.0000
cp	303.000000	0.966997	1.032052	0.000000	0.000000	1.000000	2.000000	3.0000
trestbps	303.000000	131.623762	17.538143	94.000000	120.000000	130.000000	140.000000	200.0000
chol	303.000000	246.264026	51.830751	126.000000	211.000000	240.000000	274.500000	564.0000
fbs	303.000000	0.148515	0.356198	0.000000	0.000000	0.000000	0.000000	1.0000
restecg	303.000000	0.528053	0.525860	0.000000	0.000000	1.000000	1.000000	2.0000
thalach	303.000000	149.646865	22.905161	71.000000	133.500000	153.000000	166.000000	202.0000
exang	303.000000	0.326733	0.469794	0.000000	0.000000	0.000000	1.000000	1.0000
oldpeak	303.000000	1.039604	1.161075	0.000000	0.000000	0.800000	1.600000	6.2000
slope	303.000000	1.399340	0.616226	0.000000	1.000000	1.000000	2.000000	2.0000
ca	303.000000	0.729373	1.022606	0.000000	0.000000	0.000000	1.000000	4.0000
thal	303.000000	2.313531	0.612277	0.000000	2.000000	2.000000	3.000000	3.0000
target	303.000000	0.544554	0.498835	0.000000	0.000000	1.000000	1.000000	1.0000

In [ ]:
```
#From the descriptive statistics, it can be seen that age, resting blood pressure
```

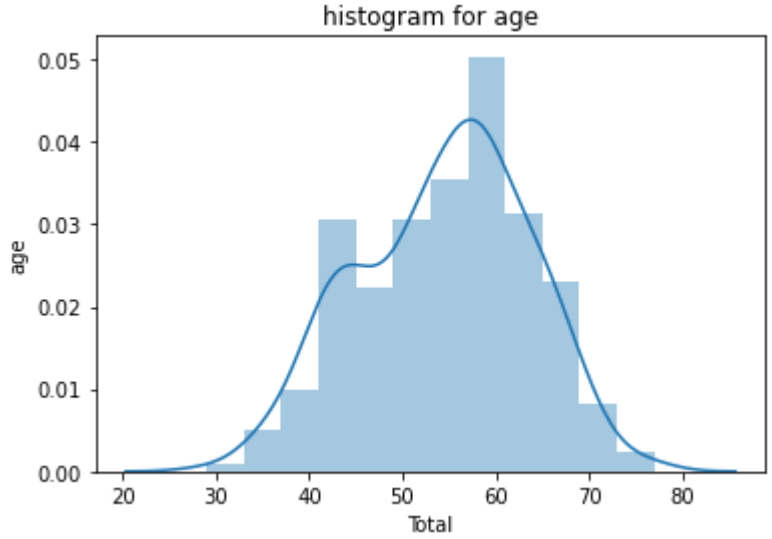
In [ ]:
```

```

In [ ]:
```
#continous columns distribution
```

In [ ]:
```
#age(patient age)
```

In [19]:
```
sns.distplot(df.age, kde=True)
plt.xlabel('Total')
plt.ylabel("age")
plt.title("histogram for age")
plt.show()
```



In [ ]:
```
# From the histogram and boxplot, it can be seen that this column is normally dis
```

In [ ]:
```

```

In [ ]:
```
#trestbps (Resting Blood Pressure in mm Hg)
```

In [ ]:
```
sns.boxplot(df.trestbps)
```

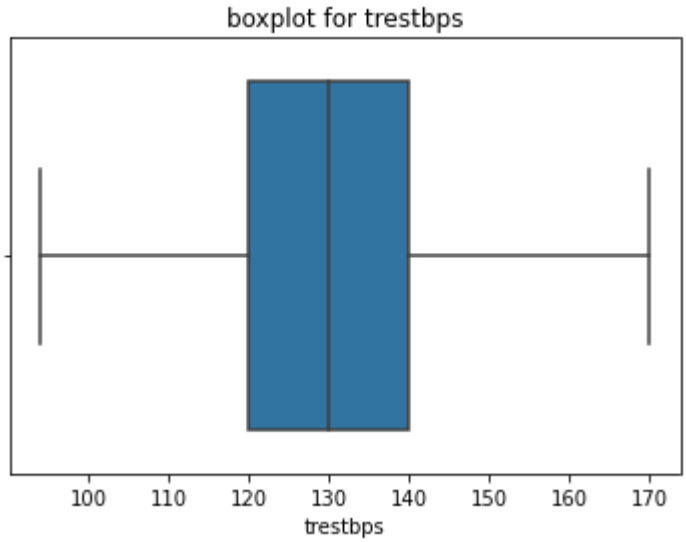
In [20]:
```
Q3=df.trestbps.quantile(0.75)
Q1=df.trestbps.quantile(0.25)
print('Q3:',Q3)
print('Q1:',Q1)
IQR=Q3-Q1
print('IQR:',IQR)
HE=Q3+1.5*IQR
LE=Q1-1.5*IQR
print('HE:',HE)
print('LE:',LE)
_HE=len(df[df.trestbps>HE])
_LE=len(df[df.trestbps<LE])
No_of_outliers=_HE+_LE
print('total_no_of_outliers:',No_of_outliers)
total=df.trestbps.value_counts().sum()
print('out_of:',total)
```

```
Q3: 140.0
Q1: 120.0
IQR: 20.0
HE: 170.0
LE: 90.0
total_no_of_outliers: 9
out_of: 303
```

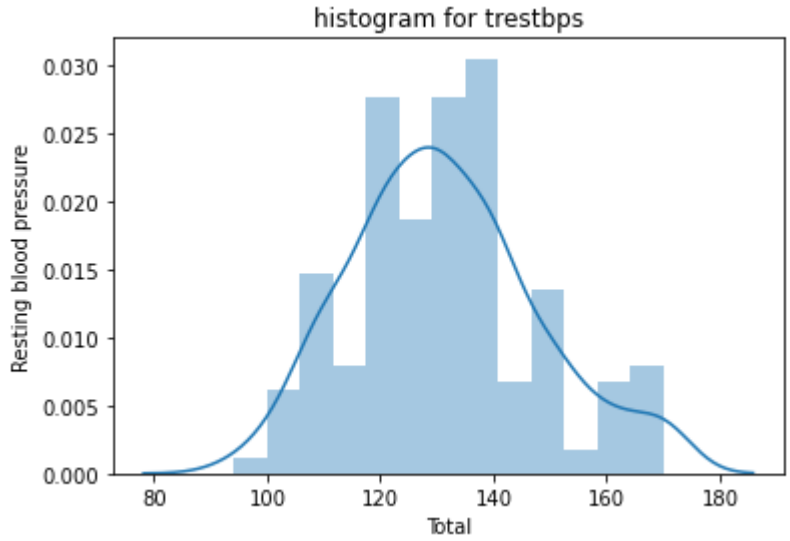
In [21]:
```
df.loc[df['trestbps']>=HE,'trestbps']=HE
df.loc[df['trestbps']<=LE,'trestbps']=LE
```

In [22]:
```
sns.boxplot(df.trestbps)
plt.title("boxplot for trestbps")
```

Out[22]:
```
Text(0.5, 1.0, 'boxplot for trestbps')
```

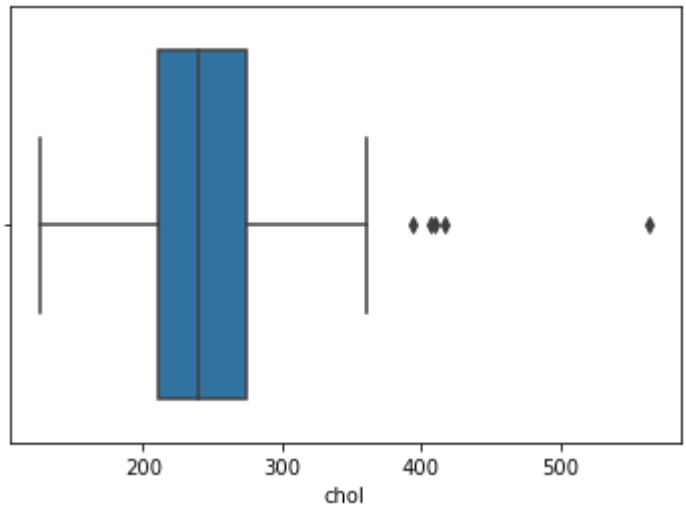


In [23]:
```
sns.distplot(df.trestbps, kde=True)
plt.xlabel('Total')
plt.ylabel("Resting blood pressure")
plt.title("histogram for trestbps")
plt.show()
```



In [24]:
```
sns.boxplot(df.chol)
```

Out[24]: <AxesSubplot:xlabel='chol'>



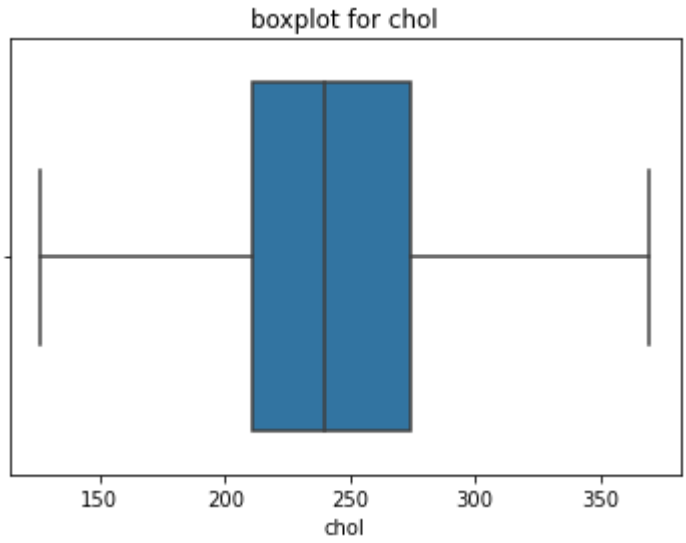
In [25]:
```
Q3=df.chol.quantile(0.75)
Q1=df.chol.quantile(0.25)
print('Q3:',Q3)
print('Q1:',Q1)
IQR=Q3-Q1
print('IQR:',IQR)
HE=Q3+1.5*IQR
LE=Q1-1.5*IQR
print('HE:',HE)
print('LE:',LE)
_HE=len(df[df.chol>HE])
_LE=len(df[df.chol<LE])
No_of_outliers=_HE+_LE
print('total_no_of_outliers:',No_of_outliers)
total=df.chol.value_counts().sum()
print('out_of:',total)
```

```
Q3: 274.5
Q1: 211.0
IQR: 63.5
HE: 369.75
LE: 115.75
total_no_of_outliers: 5
out_of: 303
```

In [26]:
```
df.loc[df['chol']>=HE,'chol']=HE
df.loc[df['chol']<=LE,'chol']=LE
```

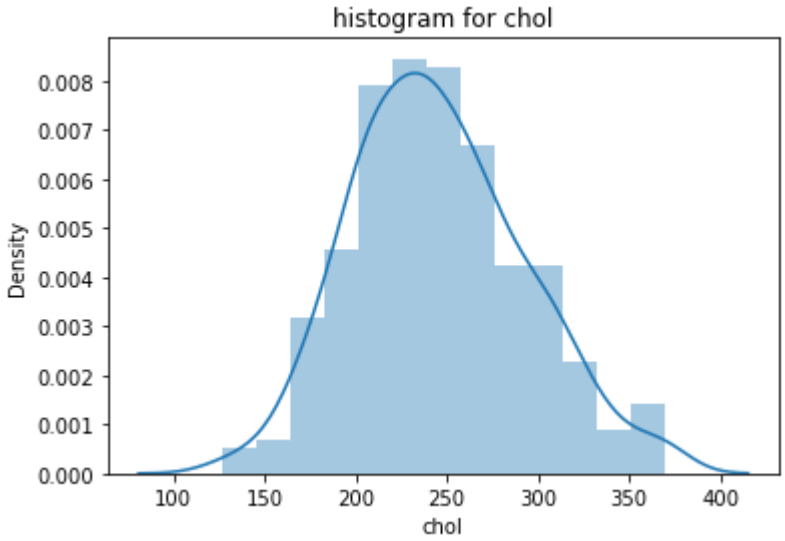
In [27]:
```
sns.boxplot(df.chol)
plt.title("boxplot for chol")
```

Out[27]:
```
Text(0.5, 1.0, 'boxplot for chol')
```



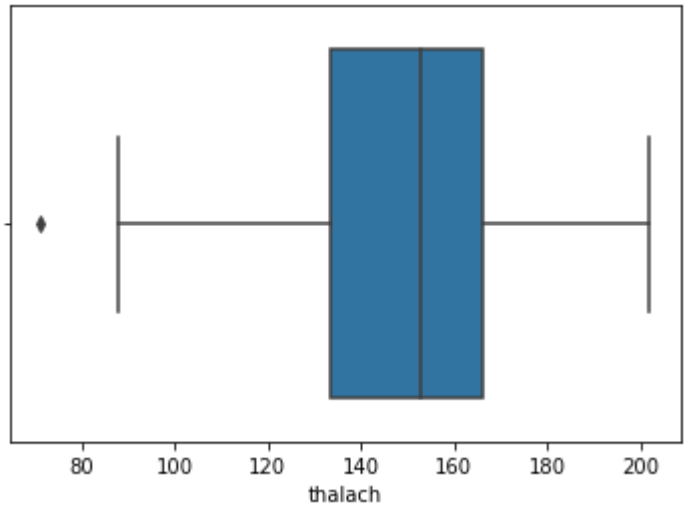
In [28]:
```
sns.distplot(df.chol)
plt.title("histogram for chol")
```

Out[28]:
```
Text(0.5, 1.0, 'histogram for chol')
```



In [29]:
```
sns.boxplot(df.thalach)
```

Out[29]:
```
<AxesSubplot:xlabel='thalach'>
```



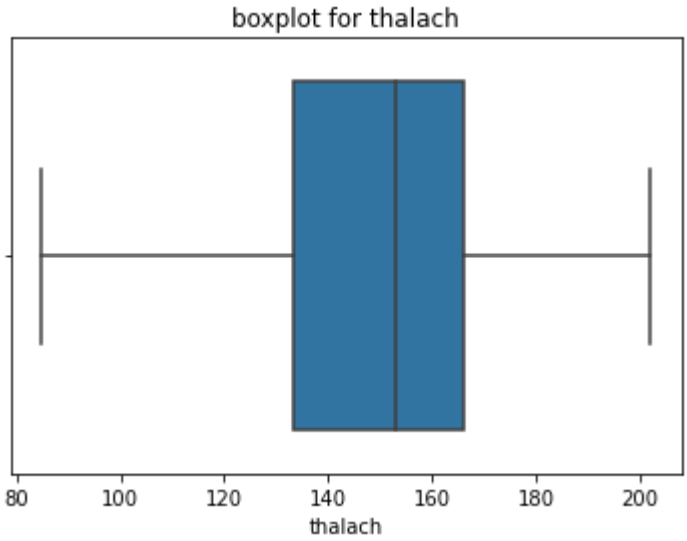
In [30]:
```
Q3=df.thalach.quantile(0.75)
Q1=df.thalach.quantile(0.25)
print('Q3:',Q3)
print('Q1:',Q1)
IQR=Q3-Q1
print('IQR:',IQR)
HE=Q3+1.5*IQR
LE=Q1-1.5*IQR
print('HE:',HE)
print('LE:',LE)
_HE=len(df[df.thalach>HE])
_LE=len(df[df.thalach<LE])
No_of_outliers=_HE+_LE
print('total_no_of_outliers:',No_of_outliers)
total=df.thalach.value_counts().sum()
print('out_of:',total)
```

```
Q3: 166.0
Q1: 133.5
IQR: 32.5
HE: 214.75
LE: 84.75
total_no_of_outliers: 1
out_of: 303
```

In [31]:
```
df.loc[df['thalach']>=HE,'thalach']=HE
df.loc[df['thalach']<=LE,'thalach']=LE
```

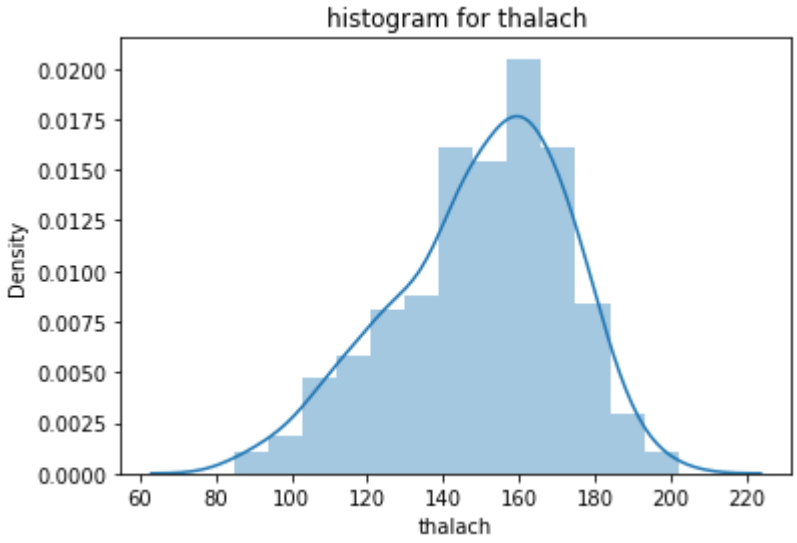
In [32]:
```
sns.boxplot(df.thalach)
plt.title("boxplot for thalach")
```

Out[32]:
```
Text(0.5, 1.0, 'boxplot for thalach')
```



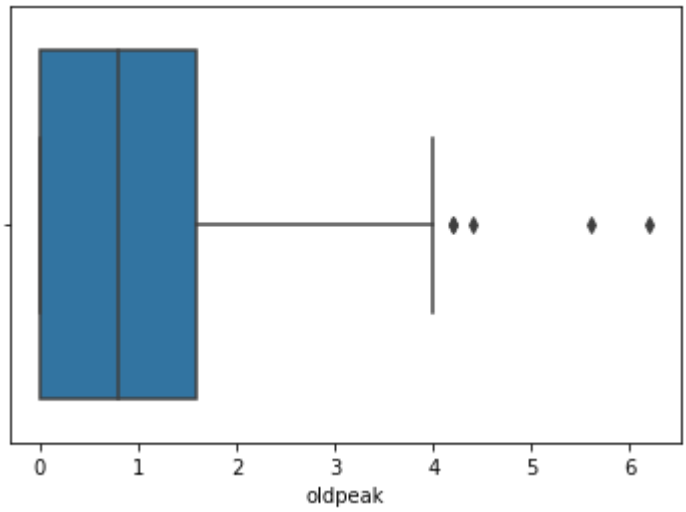
In [33]:
```
sns.distplot(df.thalach)
plt.title("histogram for thalach")
```

Out[33]:
```
Text(0.5, 1.0, 'histogram for thalach')
```



In [34]:
```
sns.boxplot(df.oldpeak)
```

Out[34]:
```
<AxesSubplot:xlabel='oldpeak'>
```



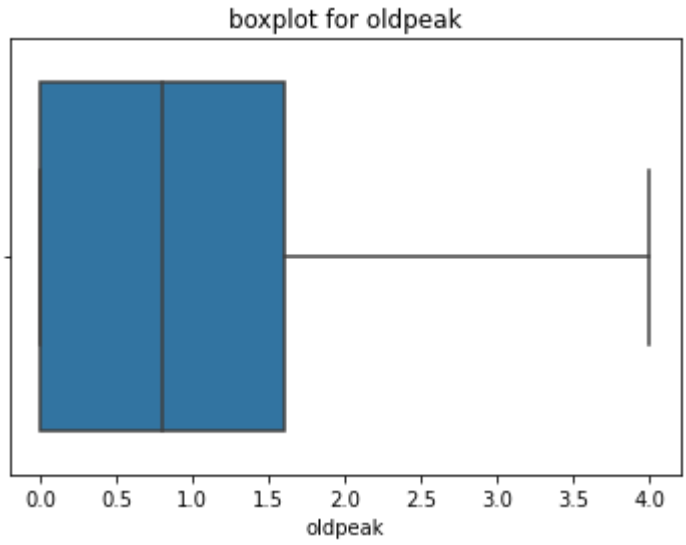
In [35]:
```
Q3=df.oldpeak.quantile(0.75)
Q1=df.oldpeak.quantile(0.25)
print('Q3:',Q3)
print('Q1:',Q1)
IQR=Q3-Q1
print('IQR:',IQR)
HE=Q3+1.5*IQR
LE=Q1-1.5*IQR
print('HE:',HE)
print('LE:',LE)
_HE=len(df[df.oldpeak>HE])
_LE=len(df[df.oldpeak<LE])
No_of_outliers=_HE+_LE
print('total_no_of_outliers:',No_of_outliers)
total=df.oldpeak.value_counts().sum()
print('out_of:',total)
```

```
Q3: 1.6
Q1: 0.0
IQR: 1.6
HE: 4.0
LE: -2.4000000000000004
total_no_of_outliers: 5
out_of: 303
```

In [36]:
```
df.loc[df['oldpeak']>=HE,'oldpeak']=HE
df.loc[df['oldpeak']<=LE,'oldpeak']=LE
```

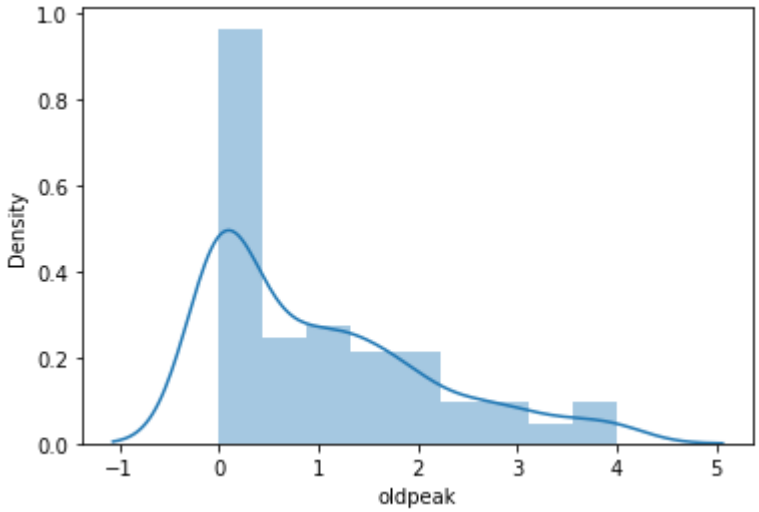
In [37]:
```
sns.boxplot(df.oldpeak)
plt.title("boxplot for oldpeak")
```

Out[37]:
```
Text(0.5, 1.0, 'boxplot for oldpeak')
```



In [38]:
```
sns.distplot(df.oldpeak)
```

Out[38]:
```
<AxesSubplot:xlabel='oldpeak', ylabel='Density'>
```



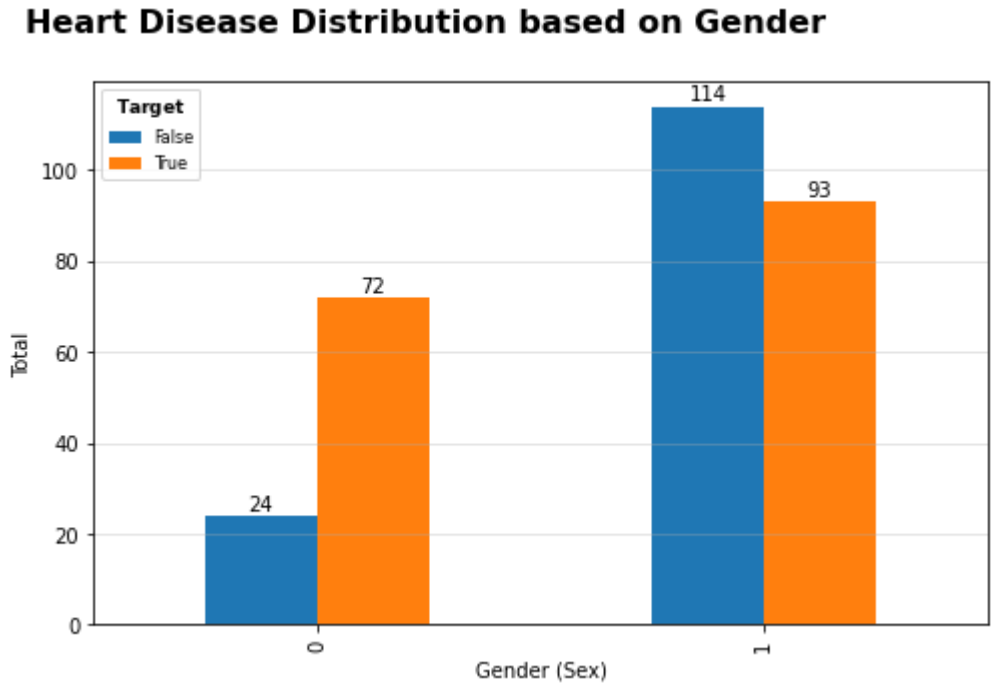
In [ ]:
```

```

In [ ]:
```
#Heart Disease Distribution based on Gender
```

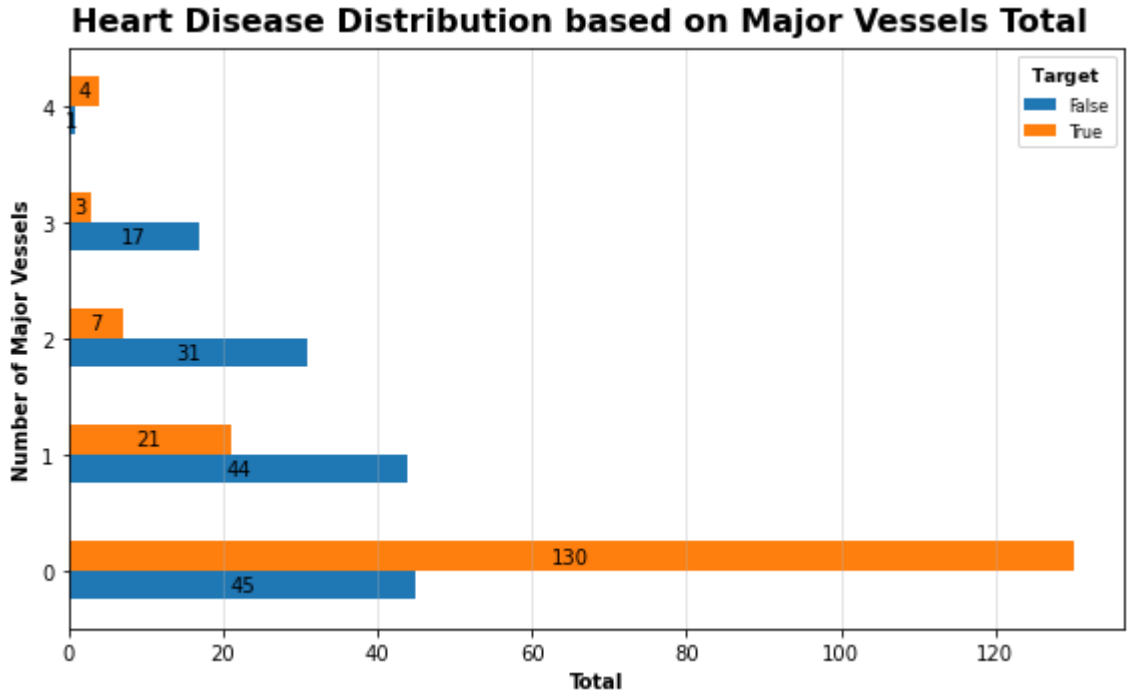
In [39]:
```
labels = ['False', 'True']
label_gender = np.array([0, 1])
label_gender2 = ['Male', 'Female']

ax = pd.crosstab(df.sex, df.target).plot(kind = 'bar', figsize = (8,5))
for rect in ax.patches:
    ax.text (rect.get_x()+rect.get_width()/2,
             rect.get_height()+1.25,rect.get_height(),
             horizontalalignment='center', fontsize=10)
plt.suptitle('Heart Disease Distribution based on Gender', fontweight='heavy',
             x=0.065, y=0.98, ha='left', fontsize='16', fontfamily='sans-serif')
plt.xlabel('Gender (Sex)')
plt.ylabel('Total')
plt.grid(axis='y', alpha=0.4)
plt.legend(labels=labels, title='$\\bf{Target}$', fontsize='8',
           title_fontsize='9', loc='upper left', frameon=True);
```



In [ ]:
```
#Heart Disease Distribution based on Major Vessels Total
```

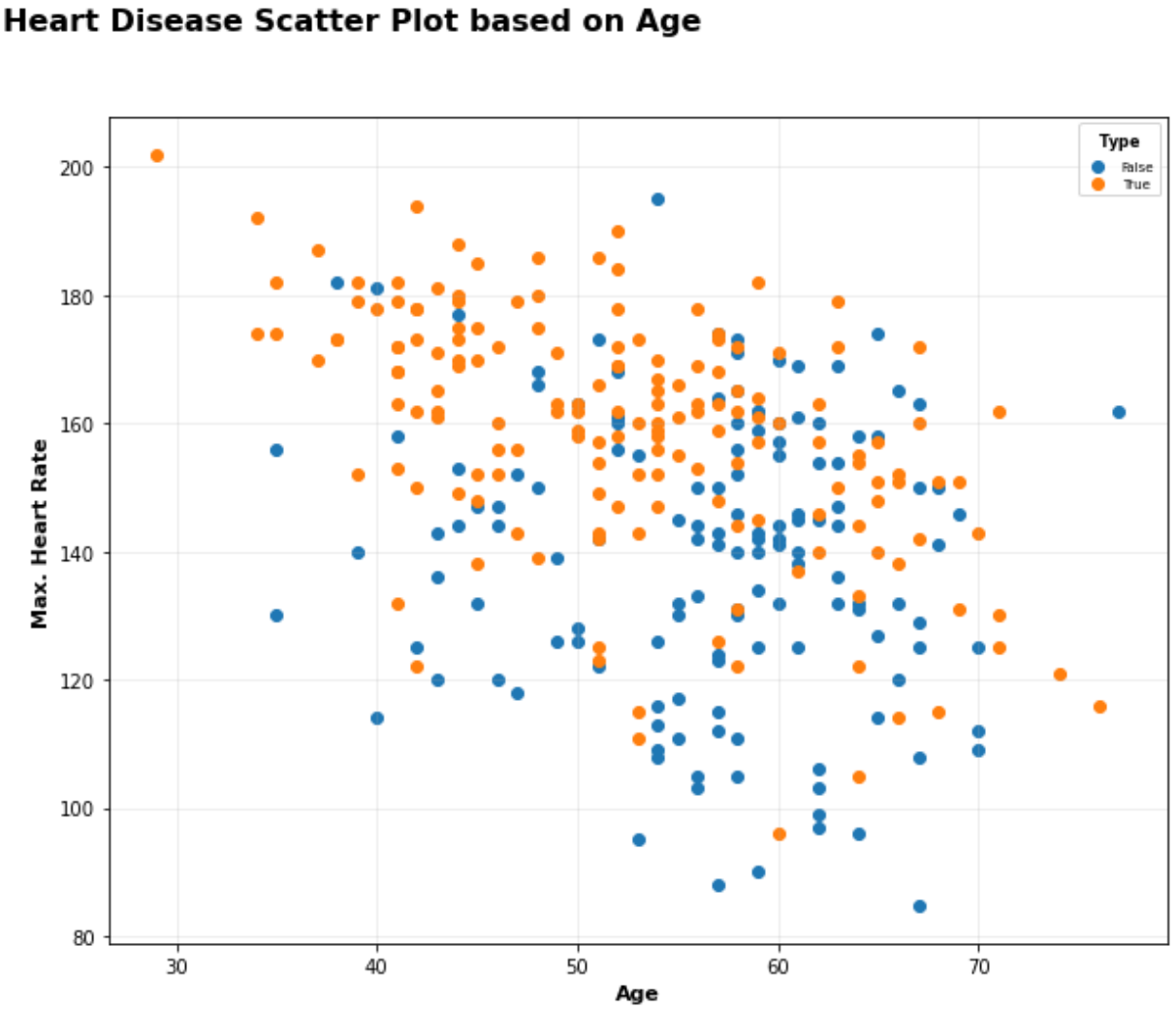
In [40]:
```
labels = ['False', 'True']
ax = pd.crosstab(df.ca, df.target).plot(kind='barh', figsize=(8, 5))
for rect in ax.patches:
    width, height = rect.get_width(), rect.get_height()
    x, y = rect.get_xy()
    ax.text (x+width/2, y+height/2, '{:.0f}'.format(width),
             horizontalalignment='center', verticalalignment='center')
plt.suptitle('Heart Disease Distribution based on Major Vessels Total',
             fontweight='heavy', x=0.069, y=0.98, ha='left', fontsize='16',
             fontfamily='sans-serif')
plt.tight_layout(rect=[0, 0.04, 1, 1.025])
plt.xlabel('Total', fontfamily='sans-serif', fontweight='bold')
plt.ylabel('Number of Major Vessels', fontfamily='sans-serif', fontweight='bold')
plt.yticks(rotation=0)
plt.grid(axis='x', alpha=0.4)
plt.grid(axis='y', alpha=0)
plt.legend(labels=labels, title='$\\bf{Target}$', fontsize='8', frameon=True,
           title_fontsize='9', loc='upper right');
```



In [ ]:

In [ ]:
```
# Heart Disease Scatter Plot based on Age
```

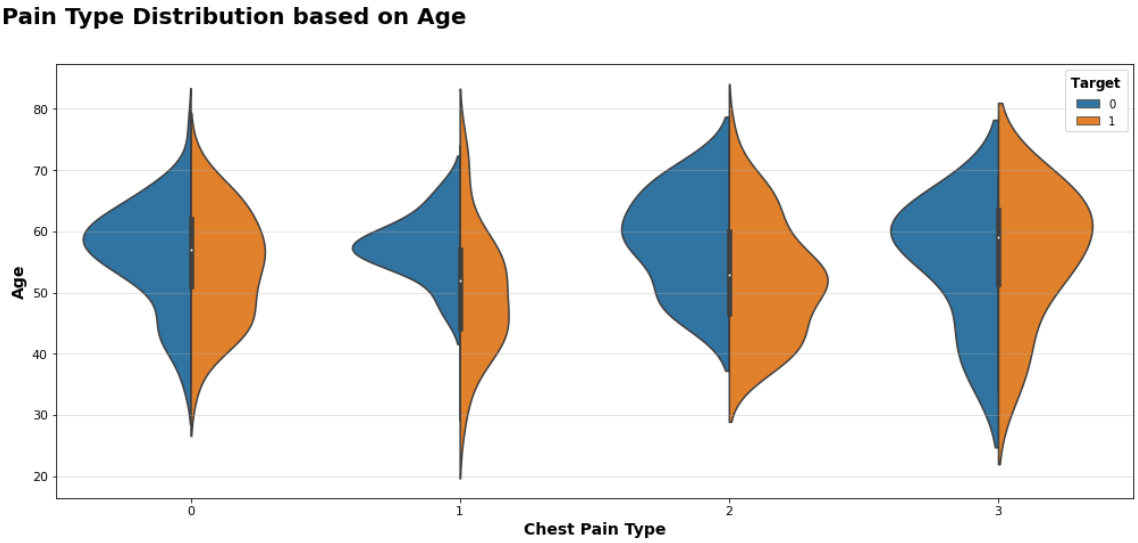
In [41]:
```
plt.figure(figsize=(10, 8))
plt.suptitle('Heart Disease Scatter Plot based on Age', fontweight='heavy',
             x=0.048, y=0.98, ha='left', fontsize='16', fontfamily='sans-serif')
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)])
plt.legend(['False', 'True'], title='$\\bf{Type}$', fontsize='7',
           title_fontsize='8', loc='upper right', frameon=True)
plt.xlabel('Age', fontweight='bold', fontsize='11',
           fontfamily='sans-serif')
plt.ylabel('Max. Heart Rate', fontweight='bold', fontsize='11',
           fontfamily='sans-serif')
plt.ticklabel_format(style='plain', axis='both')
plt.grid(axis='both', alpha=0.4, lw=0.5)
plt.show()
```



In [ ]:

In [ ]:
```
#Chest Pain Type based on Age
```

In [42]:
```
fig, ax = plt.subplots()
fig.set_size_inches(17, 7)
plt.suptitle('Chest Pain Type Distribution based on Age', fontweight='heavy',
             x=0.028, y=0.98, ha='left', fontsize='20', fontfamily='sans-serif')
sns.violinplot(x='cp', y='age', hue='target', data=df, ax=ax,
 boxprops=dict(alpha=0.9), linewidth=1.5,
               split=True)
plt.legend(title='$\\bf{Target}$', fontsize='10', title_fontsize='12', frameon=Tr
           loc='upper right')
plt.xlabel('Chest Pain Type', fontweight='bold', fontsize='14',
           fontfamily='sans-serif')
plt.ylabel('Age', fontweight='bold', fontsize='14', fontfamily='sans-serif')
plt.xticks(fontsize='11')
plt.yticks(fontsize='11')
plt.grid(axis='y', alpha=0.4)
plt.show()
```

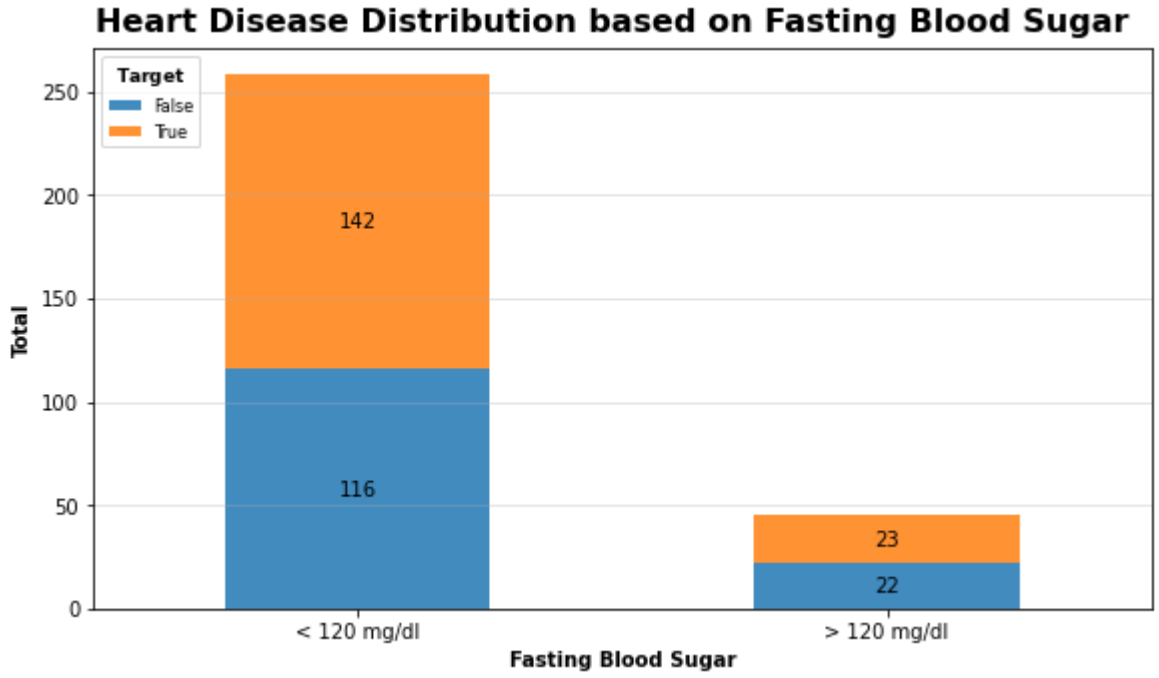


In [ ]:
```

```

In [ ]:
```
# Heart Disease Distribution based on Fasting Blood Sugar
```

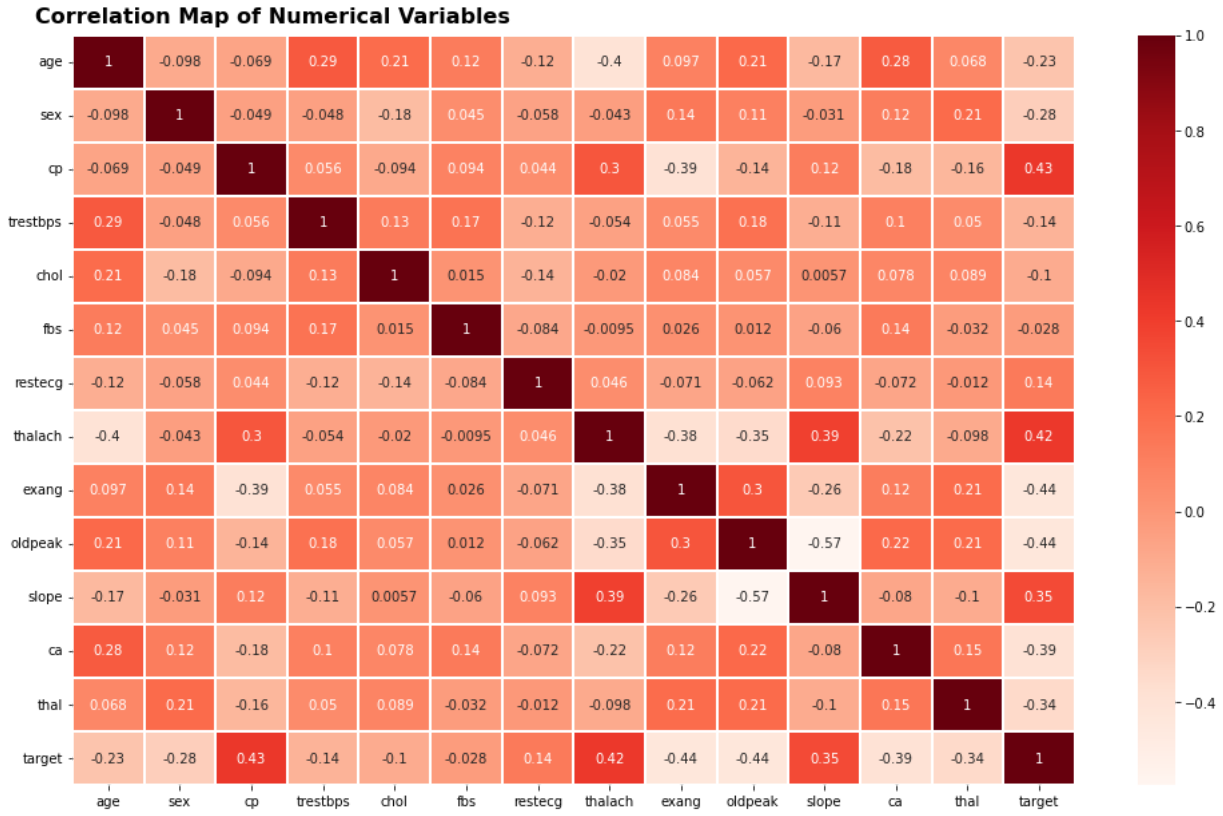
In [43]:
```
labels = ['False', 'True']
label_gender = np.array([0, 1])
label_gender2 = ['< 120 mg/dl', '> 120 mg/dl']
ax = pd.crosstab(df.fbs, df.target).plot(kind='bar', figsize=(8, 5),
                                         stacked=True,
                                         alpha=0.85)
for rect in ax.patches:
    width, height = rect.get_width(), rect.get_height()
    x, y = rect.get_xy()
    ax.text (x+width/2, y+height/2, '{:.0f}'.format(height),
             horizontalalignment='center', verticalalignment='center')
    plt.suptitle('Heart Disease Distribution based on Fasting Blood Sugar',
             fontweight='heavy', x=0.065, y=0.98, ha='left', fontsize='16',
             fontfamily='sans-serif')
    plt.tight_layout(rect=[0, 0.04, 1, 1.025])
plt.xlabel('Fasting Blood Sugar', fontfamily='sans-serif', fontweight='bold')
plt.ylabel('Total', fontfamily='sans-serif', fontweight='bold')
plt.xticks(label_gender, label_gender2, rotation=0)
plt.grid(axis='y', alpha=0.4)
plt.grid(axis='x', alpha=0)
plt.legend(labels=labels, title='$\\bf{Target}$', fontsize='8',
           title_fontsize='9', loc='upper left', frameon=True);
```



In [ ]:

In [ ]:
```
# Heatmap
```

In [44]:
```
plt.figure(figsize=(14, 9))
sns.heatmap(df.corr(), annot=True, cmap='Reds', linewidths=0.1)
plt.suptitle('Correlation Map of Numerical Variables', fontweight='heavy',
             x=0.03, y=0.98, ha='left', fontsize='16', fontfamily='sans-serif')
plt.tight_layout(rect=[0, 0.04, 1, 1.01])
```



In [ ]:

In [ ]:
```
# Dataset Pre-processing
```

In [ ]:
```
#This section will prepare the dataset before building the machine learning model
```

In [ ]:
```
# 1. One-Hot Encoding
```

In [ ]:
```
#The data pre-processing will be transforming categorical variables using one-hot
```

In [45]:
```
# --- Creating Dummy Variables for cp, thal and slope ---
cp = pd.get_dummies(df['cp'], prefix='cp')
thal = pd.get_dummies(df['thal'], prefix='thal')
slope = pd.get_dummies(df['slope'], prefix='slope')

# --- Merge Dummy Variables to Main Data Frame ---
frames = [df, cp, thal, slope]
df = pd.concat(frames, axis = 1)
```

In [ ]:
```
# --- Display New Data Frame ---
df.head().style.background_gradient(cmap='PuRd').hide_index().set_properties(**{'
```

In [ ]:
```
#After creating dummy variables, there are some unnecessary variables in the data
```

In [ ]:
```
#Dropping Unnecessary Variables
```

In [ ]:
```
# The variables that unnecessary will be deleted.
```

In [46]:
```
# --- Drop Unnecessary Variables ---
df = df.drop(columns = ['cp', 'thal', 'slope'])
```

In [47]:
```
# --- Display New Data Frame ---
df.head().style.background_gradient(cmap='Reds').hide_index().set_properties(**{'
```

Out[47]:

age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	target	cp_0	cp_1
63	1	145	233.000000	1	0	150.000000	0	2.300000	0	1	0	0
37	1	130	250.000000	0	1	187.000000	0	3.500000	0	1	0	0
41	0	130	204.000000	0	0	172.000000	0	1.400000	0	1	0	1
56	1	120	236.000000	0	1	178.000000	0	0.800000	0	1	0	1
57	0	120	354.000000	0	1	163.000000	1	0.600000	0	1	1	0

In [ ]:
```

```

In [ ]:
```
#features separating
```

In [ ]:
```
#In this section, the 'target' (dependent) column will be seperated from independ
```

In [48]:
```
# --- Seperating Dependent Features ---
x = df.drop(['target'], axis=1)
y = df['target']
```

In [ ]:
```
# In this section, data normalization will be performed to normalize the range of
# Data normalization will use min-max normalization.
```

In [49]:
```
# --- Data Normalization using Min-Max Method ---
x = MinMaxScaler().fit_transform(x)
```

In [ ]:
```
#Splitting the Dataset
```

In [ ]:
```
#The dataset will be splitted into 80:20 ratio (80% training and 20% testing).
```

In [50]:
```
# --- Splitting Dataset into 80:20 ---
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_s
```

In [ ]:
```
# Model Implementation
```

In [ ]:
```
#Logistic Regression
```

In [51]:
```
# --- Applying Logistic Regression ---
LRclassifier = LogisticRegression(max_iter=1000, random_state=1, solver='liblinea
LRclassifier.fit(x_train, y_train)

y_pred_LR = LRclassifier.predict(x_test)
```

In [ ]:
```
# --- LR Accuracy ---
LRAcc = accuracy_score(y_pred_LR, y_test)
print('.:. Logistic Regression Accuracy:'+'\033[1m {:.2f}%'.format(LRAcc*100)+' 

# --- LR Classification Report ---
print('\n\033[1m'+'.: Classification Report'+'\033[0m')
print('*' * 25)
print(classification_report(y_test, y_pred_LR))

# --- Performance Evaluation ---
#print('\n\033[1m'+'.: Performance Evaluation'+'\033[0m')
#print('*' * 26)
#fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(14, 10))

# --- LR Confusion Matrix ---
#logmatrix = ConfusionMatrix(LRclassifier, ax=ax1, cmap='PuRd',
          #                  title='Logistic Regression Confusion Matrix')
#logmatrix.fit(x_train, y_train)
#logmatrix.score(x_test, y_test)
#logmatrix.finalize()

# --- LR ROC AUC ---
#logrocauc = ROCAUC(LRclassifier, classes=['False', 'True'], ax=ax2,
       #            title='Logistic Regression ROC AUC Plot')
#logrocauc.fit(x_train, y_train)
#logrocauc.score(x_test, y_test)
#@logrocauc.finalize()

# --- LR Learning Curve ---
#loglc = LearningCurve(LRclassifier, ax=ax3, title='Logistic Regression Learning 
#loglc.fit(x_train, y_train)
#loglc.finalize()

# --- LR Precision Recall Curve ---
#logcurve = PrecisionRecallCurve(LRclassifier, ax=ax4, ap_score=True, iso_f1_curv
               #                 title='Logistic Regression Precision-Recall Curv
#logcurve.fit(x_train, y_train)
#logcurve.score(x_test, y_test)
#logcurve.finalize()


#plt.tight_layout()
```

In [52]:
```
# --- Applying Random Forest ---
RFclassifier = RandomForestClassifier(n_estimators=1000, random_state=1, max_leaf

RFclassifier.fit(x_train, y_train)
y_pred_RF = RFclassifier.predict(x_test)
```

In [18]:
```
# --- Random Forest Accuracy ---
RFAcc = accuracy_score(y_pred_RF, y_test)
print('.:. Random Forest Accuracy:'+'\033[1m {:.2f}%'.format(RFAcc*100)+' .:.')

# --- Random FOrest Classification Report ---
print('\n\033[1m'+'.: Classification Report'+'\033[0m')
print('*' * 25)
print(classification_report(y_test, y_pred_RF))
```

.:. Random Forest Accuracy: **91.80%** .:.

.: Classification Report

****

	precision	recall	f1-score	support
0	0.92	0.88	0.90	25
1	0.92	0.94	0.93	36
accuracy			0.92	61
macro avg	0.92	0.91	0.91	61
weighted avg	0.92	0.92	0.92	61

In [53]:
```
# --- Applying KNN ---
KNNClassifier = KNeighborsClassifier(n_neighbors=3)
KNNClassifier.fit(x_train, y_train)

y_pred_KNN = KNNClassifier.predict(x_test)
```

In [20]:
```

# --- KNN Accuracy ---
KNNAcc = accuracy_score(y_pred_KNN, y_test)
print('.:. K-Nearest Neighbour Accuracy:'+'\033[1m {:.2f}%'.format(KNNAcc*100)+'

# --- KNN Classification Report ---
print('\n\033[1m'+'.: Classification Report'+'\033[0m')
print('*' * 25)
print(classification_report(y_test, y_pred_KNN))
```

.:. K-Nearest Neighbour Accuracy: **83.61%** .:.

.: Classification Report

****

	precision	recall	f1-score	support
0	0.76	0.88	0.81	25
1	0.91	0.81	0.85	36
accuracy			0.84	61
macro avg	0.83	0.84	0.83	61
weighted avg	0.85	0.84	0.84	61

In [ ]:
```
#Gradient bossting
```

In [54]:
```
# --- Applying Gradient Boosting ---
GBclassifier = GradientBoostingClassifier(random_state=1, n_estimators=100, max_l
                                          min_samples_leaf=20)

GBclassifier.fit(x_train, y_train)
y_pred_GB = GBclassifier.predict(x_test)
```

In [23]:
```
# --- Gradient Boosting Accuracy ---
GBAcc = accuracy_score(y_pred_GB, y_test)
print('.:. Gradient Boosting Accuracy:'+'\033[1m {:.2f}%'.format(GBAcc*100)+' .:.

# --- Gradient Boosting Classification Report ---
print('\n\033[1m'+'.: Classification Report'+'\033[0m')
print('*' * 25)
print(classification_report(y_test, y_pred_GB))
```

```
.:. Gradient Boosting Accuracy: 90.16% .:.

.: Classification Report
*************************
              precision    recall  f1-score   support

           0       0.85      0.92      0.88        25
           1       0.94      0.89      0.91        36

    accuracy                           0.90        61
   macro avg       0.90      0.90      0.90        61
weighted avg       0.90      0.90      0.90        61
```

In [ ]:
```

```