In [1]:
```
#Installing important necessary packages

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

random_state=42
```

In [2]:
```
df_description = pd.read_excel("Project3Data.xlsx")
df = pd.read_excel("Heartattact.xlsx")
print(df.shape)
df.head()
```

```
(303, 14)
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [3]:
```
df_description
```

Out[3]:

	variable	description
0	age	age in years
1	sex	(1 = male; 0 = female)
2	cp	chest pain type (0 = Non anginal; 1 = Non typ...
3	trestbps	resting blood pressure (in mm Hg on admission...
4	chol	serum cholestoral in mg/dl
5	fbs	(fasting blood sugar > 120 mg/dl) (1 = true; ...
6	restecg	resting electrocardiographic results
7	thalach	maximum heart rate achieved
8	exang	exercise induced angina (1 = yes; 0 = no)
9	oldpeak	ST depression induced by exercise relative to...
10	slope	the slope of the peak exercise ST segment
11	ca	number of major vessels (0-3) colored by flou...
12	thal	1 = fixed defect; 2 = normal; 3 = reversable...
13	target	0= Yes and 1=No

In [4]:
```
# checking for null
df.isnull().sum()
```

Out[4]:
```
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 [5]:
```
print(df[df.duplicated()])
df = df.drop_duplicates()
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
164   38    1   2       138   175    0        1      173      0      0.0

     slope  ca  thal  target
164      2   4     2       1
```

In [6]:
```
summary = df.describe()
summary
```

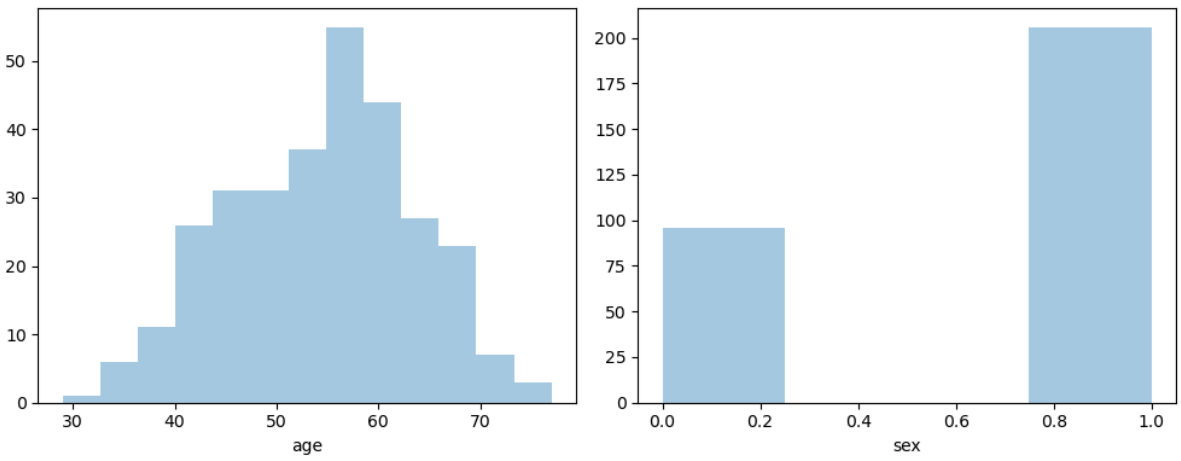
Out[6]:

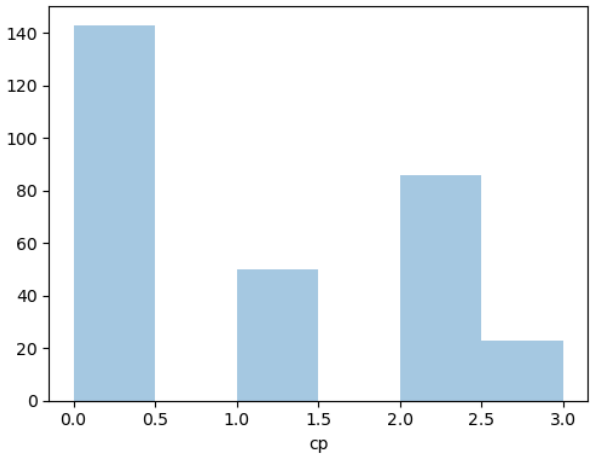
	age	sex	cp	trestbps	chol	fbs	restecg	thala
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000(
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	0.526490	149.569!
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686	0.526027	22.903!
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000(
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.250(
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	152.500(
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000	1.000000	166.000(
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000(

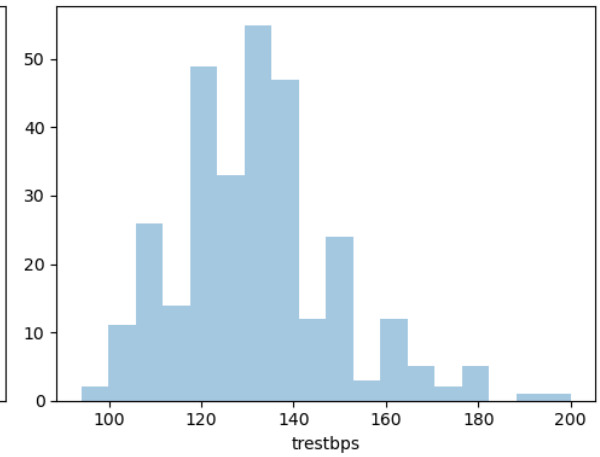
In [7]:
```
numerical_feature_columns = list(df._get_numeric_data().columns)
numerical_feature_columns
#All variables are categorical
```

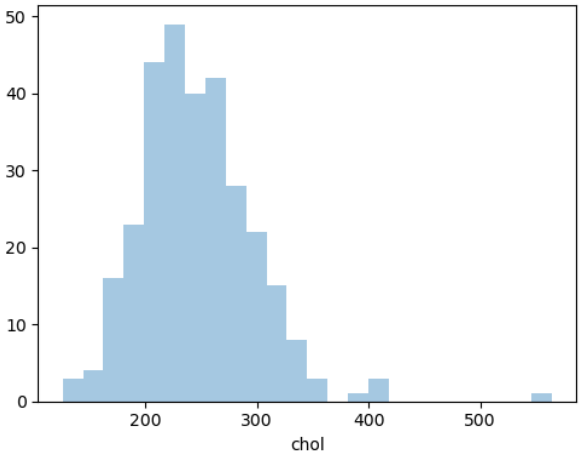
Out[7]:
```
['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal',
 'target']
```

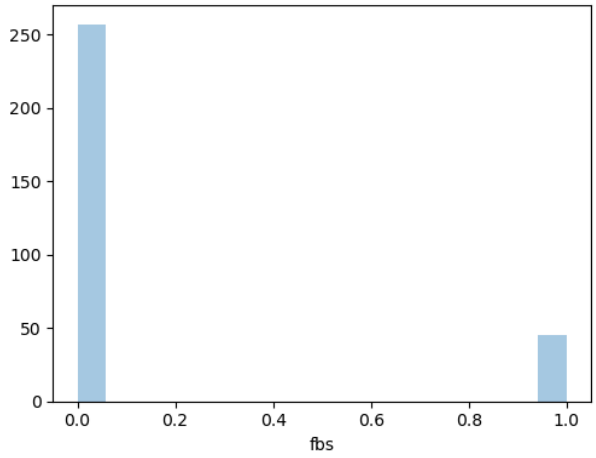
In [8]:
```
num_columns = ['age','sex','cp','trestbps','chol','fbs','restecg','thalach','exang'
for i in range(0, len(num_columns), 2):
    plt.figure(figsize=(10,4))
    plt.subplot(121)
    sns.distplot(df[num_columns[i]], kde=False)
    plt.subplot(122)
    sns.distplot(df[num_columns[i+1]], kde=False)
    plt.tight_layout()
    plt.show()
```

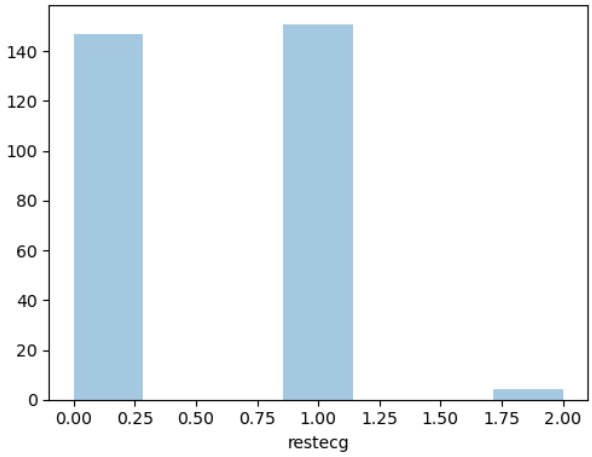


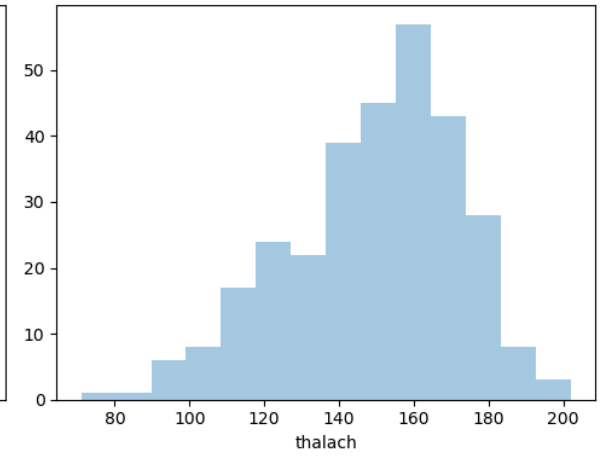

cp

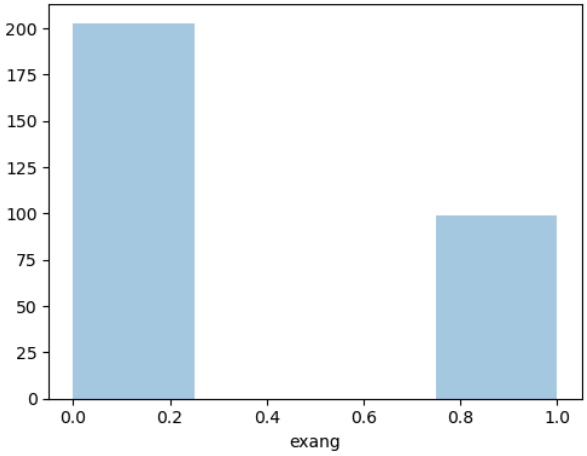

trestbps

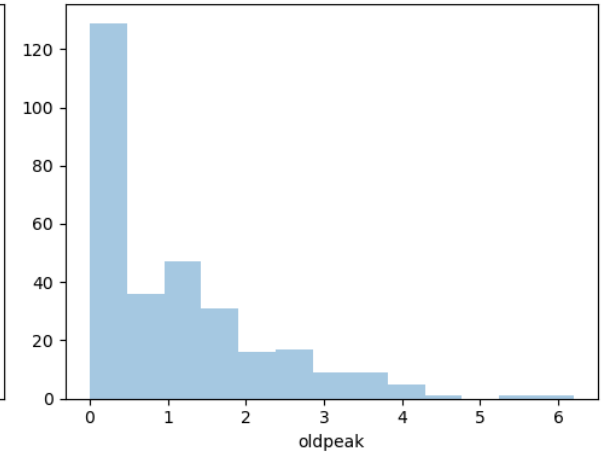

chol

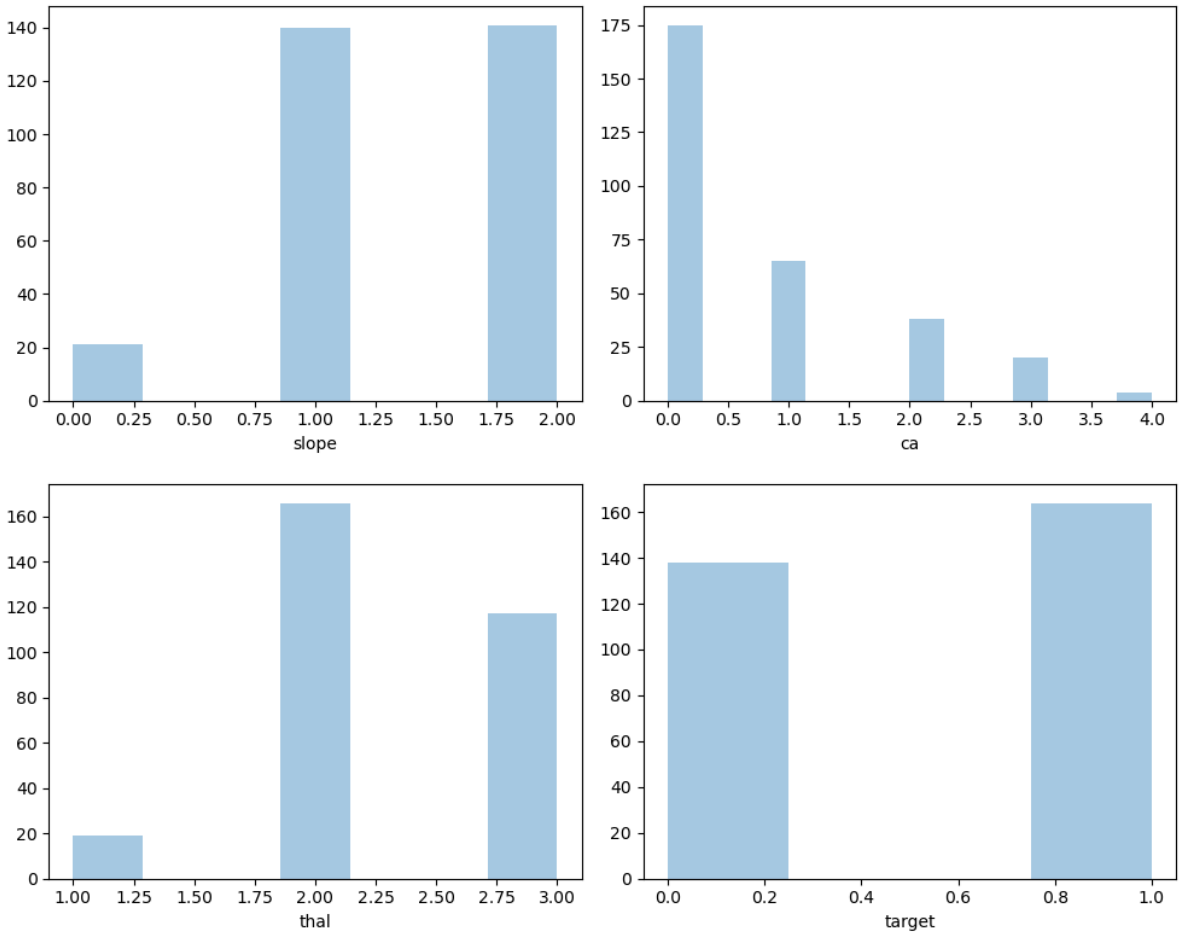

fbs


restecg


thalach


exang


oldpeak



In [9]:
```
#in the age range of 50-60 has highest rate of CVD
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 302 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       302 non-null    int64  
 1   sex       302 non-null    int64  
 2   cp        302 non-null    int64  
 3   trestbps  302 non-null    int64  
 4   chol      302 non-null    int64  
 5   fbs       302 non-null    int64  
 6   restecg   302 non-null    int64  
 7   thalach   302 non-null    int64  
 8   exang     302 non-null    int64  
 9   oldpeak   302 non-null    float64
 10  slope     302 non-null    int64  
 11  ca        302 non-null    int64  
 12  thal      302 non-null    int64  
 13  target    302 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 35.4 KB
```

In [11]:
```
df.describe().transpose()
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
age	302.0	54.420530	9.047970	29.0	48.00	55.5	61.00	77.0
sex	302.0	0.682119	0.466426	0.0	0.00	1.0	1.00	1.0
cp	302.0	0.963576	1.032044	0.0	0.00	1.0	2.00	3.0
trestbps	302.0	131.602649	17.563394	94.0	120.00	130.0	140.00	200.0
chol	302.0	246.500000	51.753489	126.0	211.00	240.5	274.75	564.0
fbs	302.0	0.149007	0.356686	0.0	0.00	0.0	0.00	1.0
restecg	302.0	0.526490	0.526027	0.0	0.00	1.0	1.00	2.0
thalach	302.0	149.569536	22.903527	71.0	133.25	152.5	166.00	202.0
exang	302.0	0.327815	0.470196	0.0	0.00	0.0	1.00	1.0
oldpeak	302.0	1.043046	1.161452	0.0	0.00	0.8	1.60	6.2
slope	302.0	1.397351	0.616274	0.0	1.00	1.0	2.00	2.0
ca	302.0	0.718543	1.006748	0.0	0.00	0.0	1.00	4.0
thal	302.0	2.324503	0.588366	1.0	2.00	2.0	3.00	3.0
target	302.0	0.543046	0.498970	0.0	0.00	1.0	1.00	1.0

In [12]:
```
df.columns
```

Out[12]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

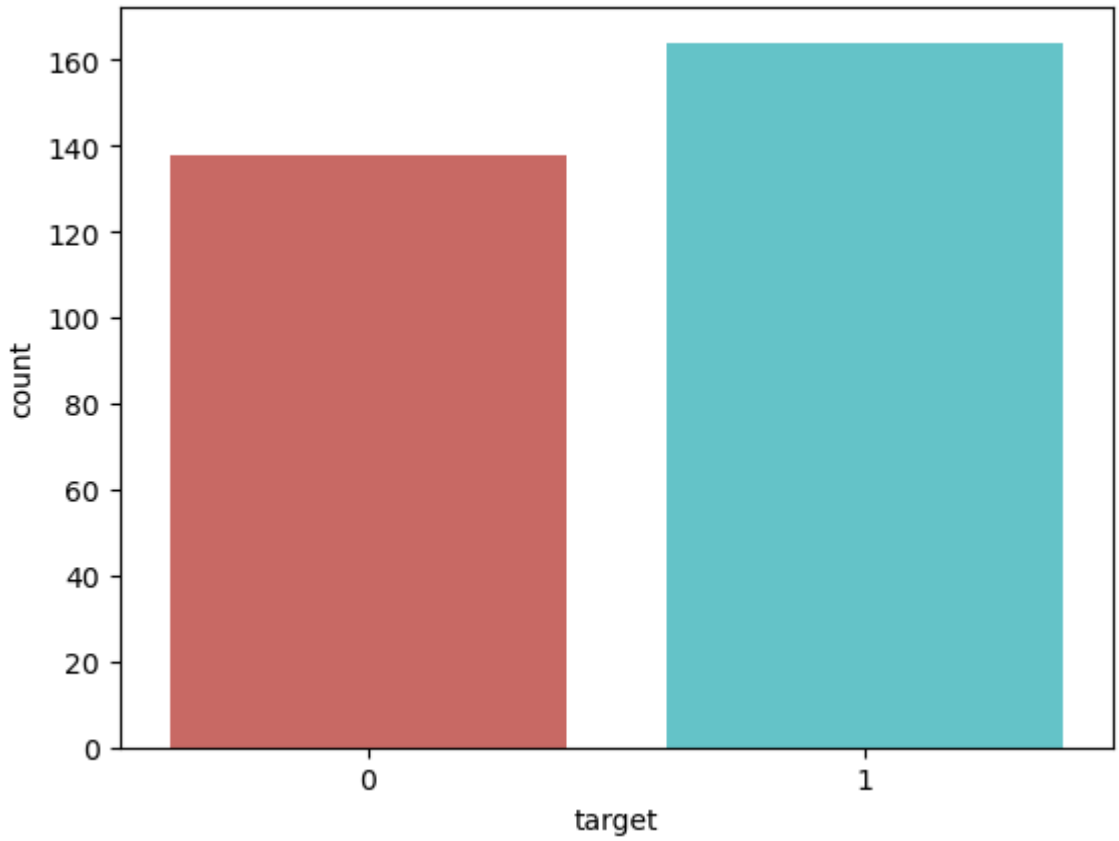
In [14]:
```
df['target'].value_counts()
```

Out[14]:
```
1    164
0    138
Name: target, dtype: int64
```

In [16]:
```
target_counts = df.target.value_counts()
print('Class 0:', target_counts[0])
print('Class 1:', target_counts[1])
print('Proportion:', round(target_counts[0] / target_counts[1], 2), ': 1')
```

```
Class 0: 138
Class 1: 164
Proportion: 0.84 : 1
```

In [18]:
```
sns.countplot(x='target', data = df, palette = 'hls')
plt.show()
```



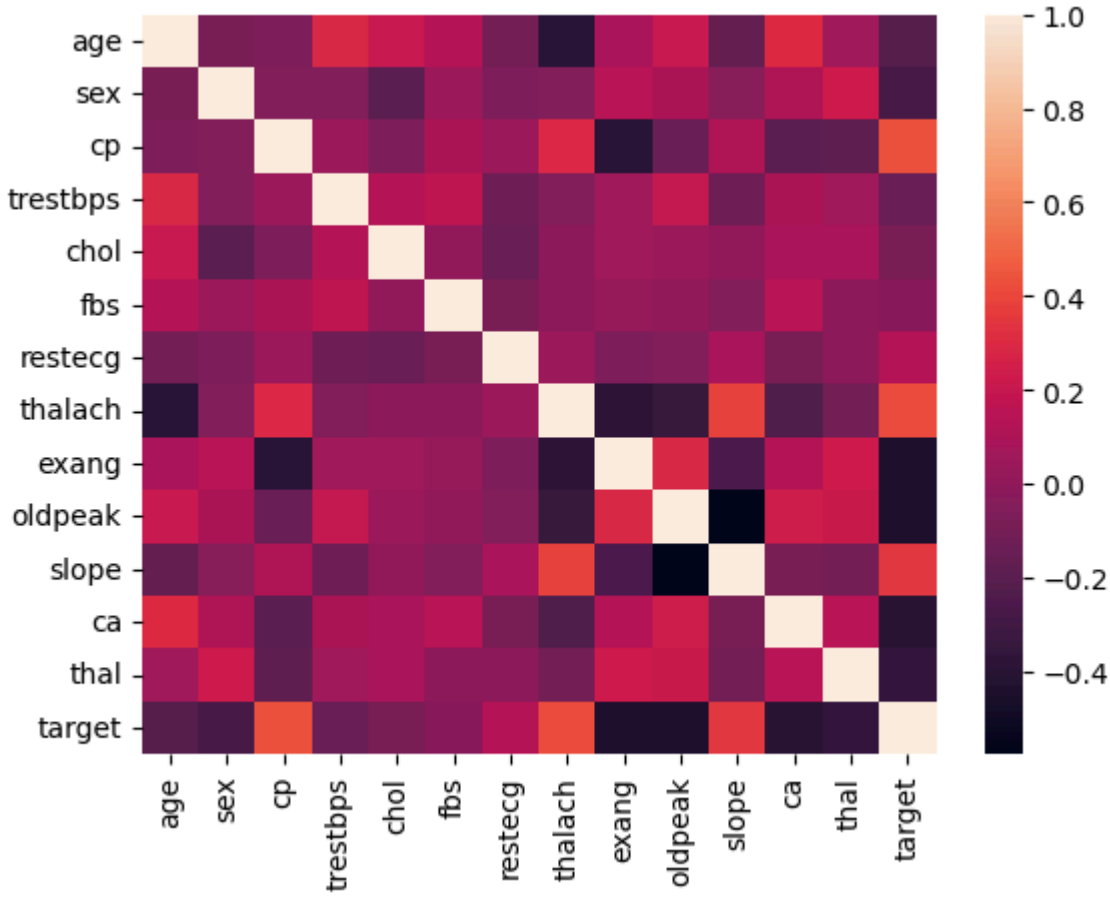
In [19]:
```
df.groupby('target').mean()
```

Out[19]:

target	age	sex	cp	trestbps	chol	fbs	restecg	thalach	e
0	56.601449	0.826087	0.478261	134.398551	251.086957	0.159420	0.449275	139.101449	0.55
1	52.585366	0.560976	1.371951	129.250000	242.640244	0.140244	0.591463	158.378049	0.14

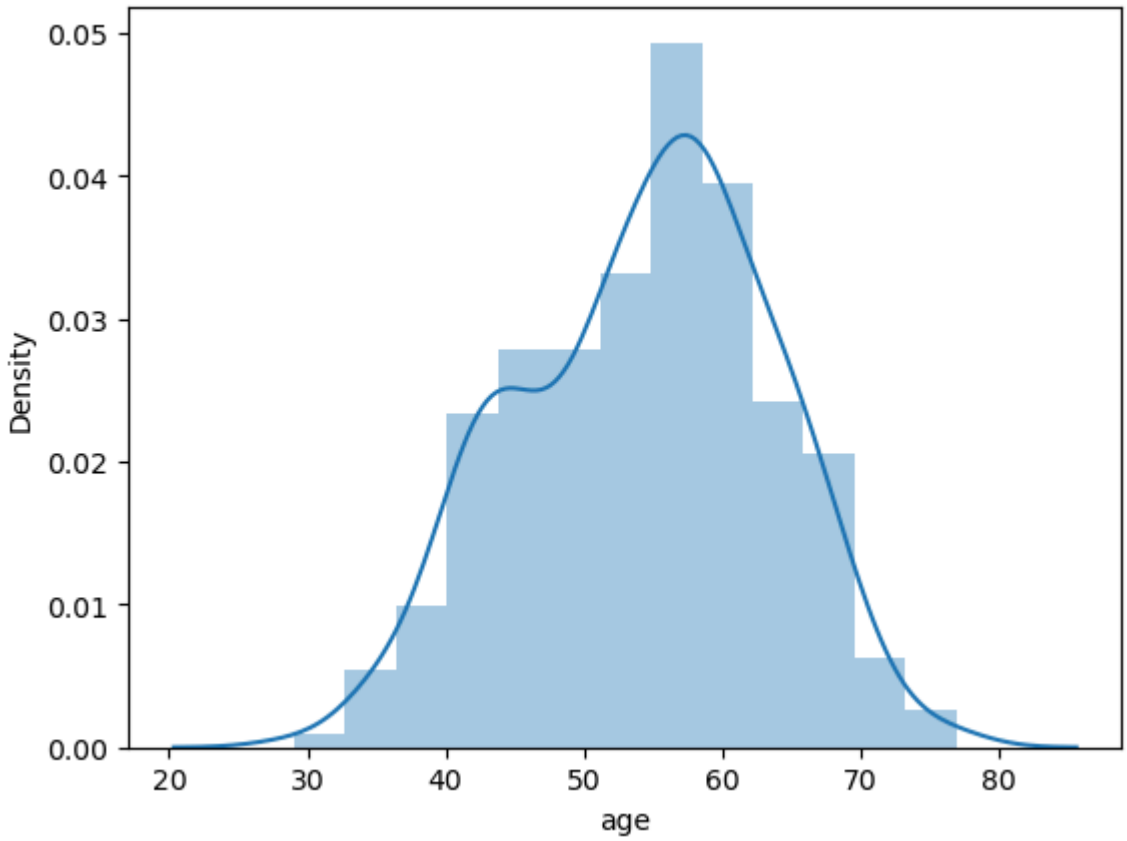
In [20]:
```
sns.heatmap(df.corr())
```

Out[20]:
```
<Axes: >
```



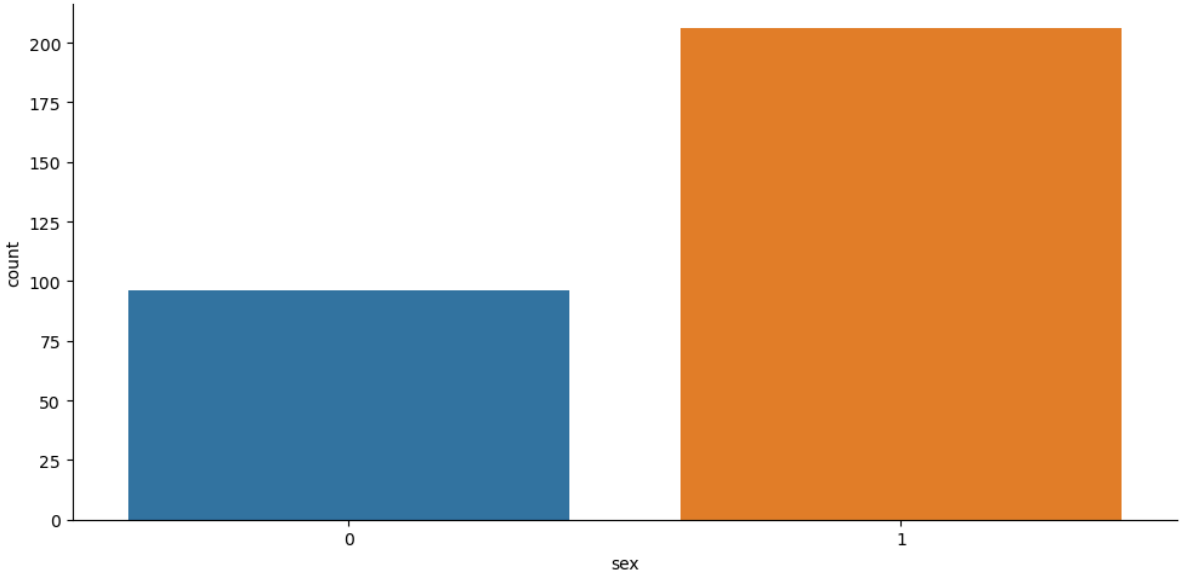
In [21]:
```
sns.distplot(df.age)
```

Out[21]:
```
<Axes: xlabel='age', ylabel='Density'>
```



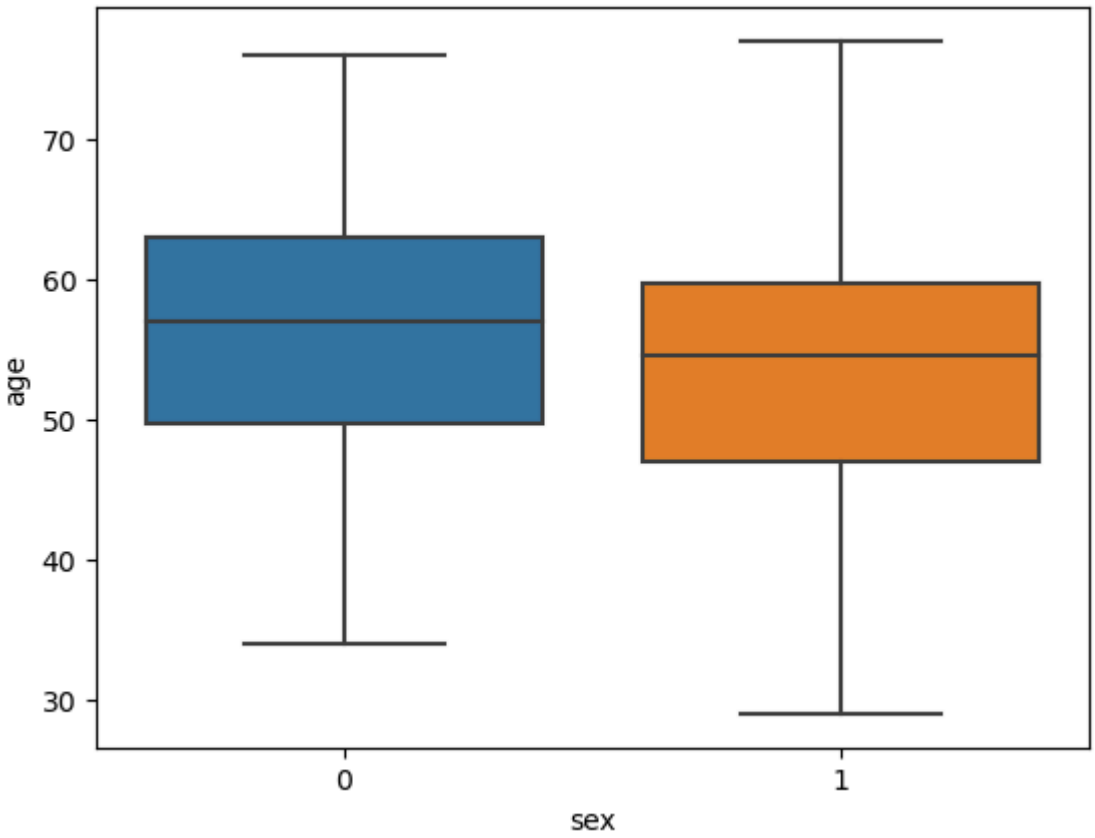
In [22]:
```
# There are significantly more men in the data than women
sns.catplot(x ='sex', data = df, kind = 'count', aspect = 2.0)
```

Out[22]: <seaborn.axisgrid.FacetGrid at 0x21117aad5d0>



In [23]:
```
sns.boxplot(data = df, x = 'sex', y = 'age')
```

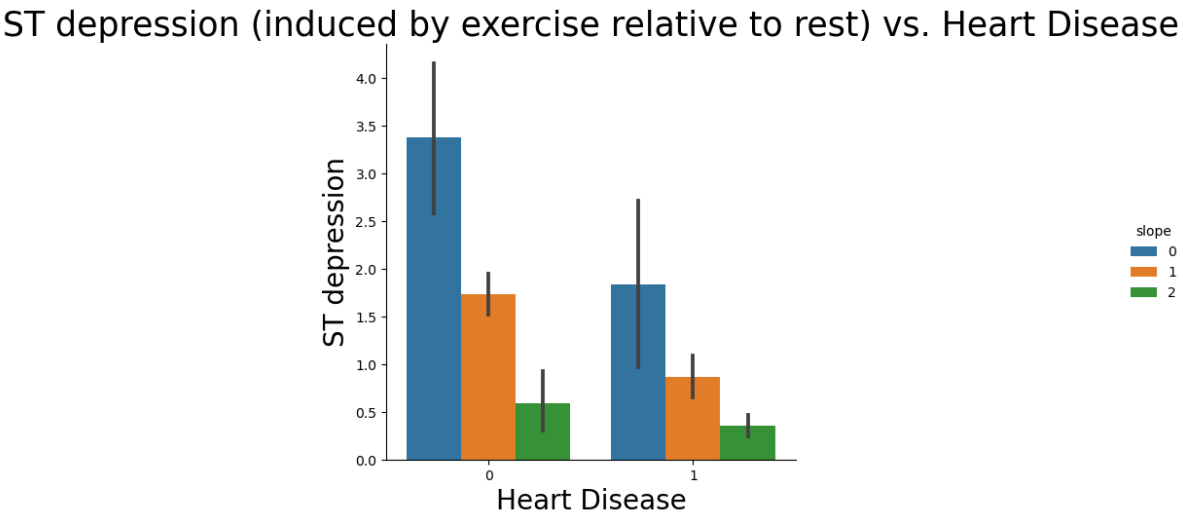
Out[23]: <Axes: xlabel='sex', ylabel='age'>



In [24]:
```
sns.catplot(x="target", y="oldpeak", hue="slope", kind="bar", data=df);

plt.title('ST depression (induced by exercise relative to rest) vs. Heart Disease',
plt.xlabel('Heart Disease',size=20)
plt.ylabel('ST depression',size=20)
```

Out[24]: Text(36.818927083333335, 0.5, 'ST depression')

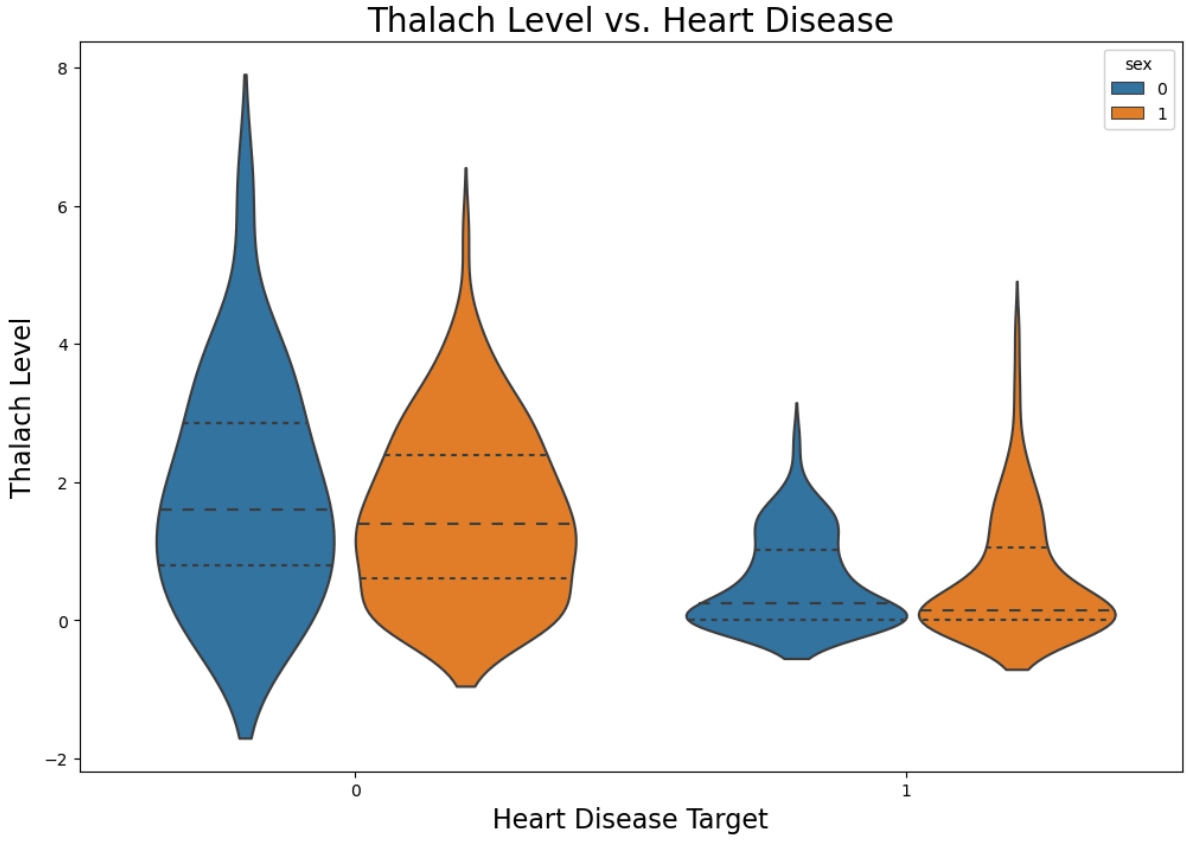


In [25]:
```
#Outlier detection Using box plot and violin plot


# Violin plot


plt.figure(figsize=(12,8))
sns.violinplot(x= 'target', y= 'oldpeak',hue="sex", inner='quartile',data= df )
plt.title("Thalach Level vs. Heart Disease",fontsize=20)
plt.xlabel("Heart Disease Target", fontsize=16)
plt.ylabel("Thalach Level", fontsize=16)
```

Out[25]:
```
Text(0, 0.5, 'Thalach Level')
```

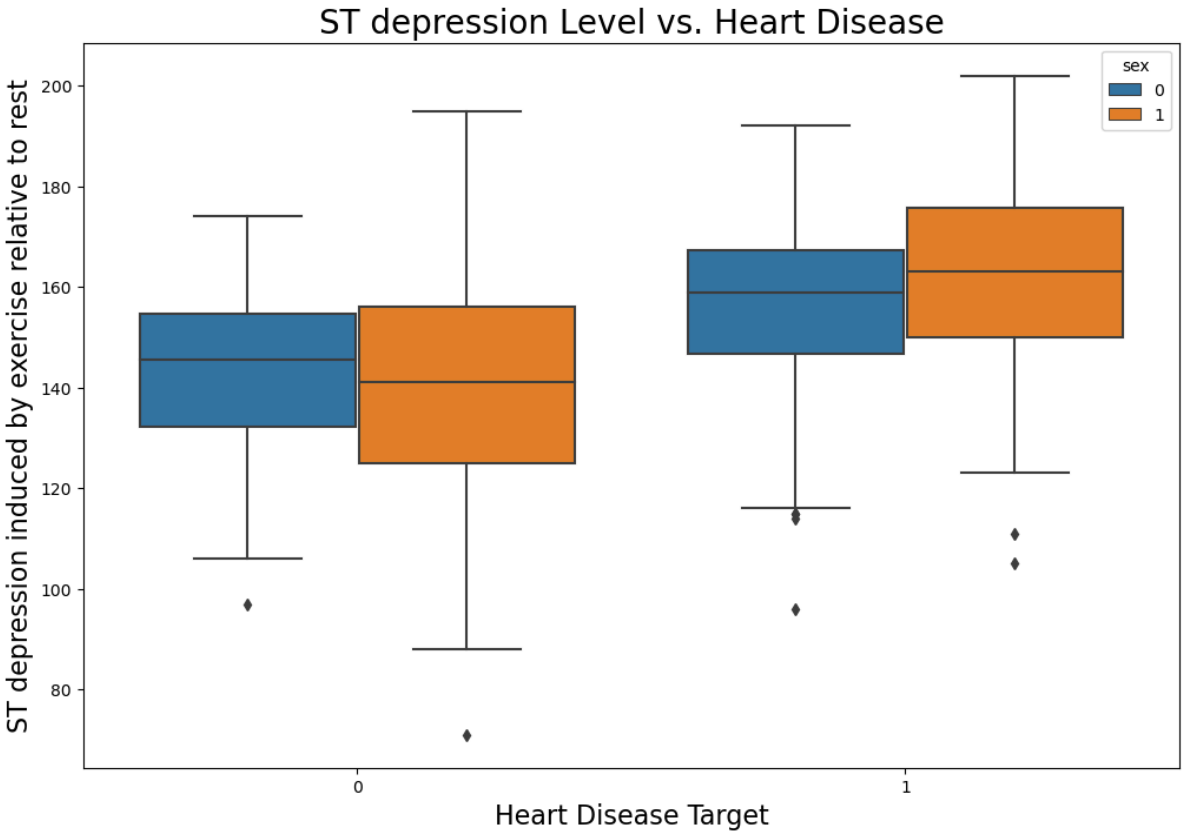


In [26]:
```
# Box Plot

plt.figure(figsize=(12,8))
sns.boxplot(x= 'target', y= 'thalach',hue="sex", data=df )
plt.title("ST depression Level vs. Heart Disease", fontsize=20)
```

```
plt.xlabel("Heart Disease Target",fontsize=16)
plt.ylabel("ST depression induced by exercise relative to rest", fontsize=16)
```

Out[26]:
```
Text(0, 0.5, 'ST depression induced by exercise relative to rest')
```



In [28]:
```
# Filtering data by POSITIVE Heart Disease patient
pos_data = df[df['target']==1]
pos_data.describe()
```

Out[28]:

	age	sex	cp	trestbps	chol	fbs	restecg	tha
count	164.000000	164.000000	164.000000	164.000000	164.000000	164.000000	164.000000	164.00
mean	52.585366	0.560976	1.371951	129.250000	242.640244	0.140244	0.591463	158.37
std	9.511957	0.497788	0.953878	16.204739	53.456580	0.348303	0.505358	19.19
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	96.00
25%	44.750000	0.000000	1.000000	120.000000	208.750000	0.000000	0.000000	148.75
50%	52.000000	1.000000	2.000000	130.000000	234.500000	0.000000	1.000000	161.00
75%	59.000000	1.000000	2.000000	140.000000	267.250000	0.000000	1.000000	172.00
max	76.000000	1.000000	3.000000	180.000000	564.000000	1.000000	2.000000	202.00

In [29]:
```
# Filtering data by NEGATIVE Heart Disease patient
pos_data = df[df['target']==0]
pos_data.describe()
```

Out[29]:

	age	sex	cp	trestbps	chol	fbs	restecg	tha
count	138.000000	138.000000	138.000000	138.000000	138.000000	138.000000	138.000000	138.00(
mean	56.601449	0.826087	0.478261	134.398551	251.086957	0.159420	0.449275	139.10
std	7.962082	0.380416	0.905920	18.729944	49.454614	0.367401	0.541321	22.598
min	35.000000	0.000000	0.000000	100.000000	131.000000	0.000000	0.000000	71.00(
25%	52.000000	1.000000	0.000000	120.000000	217.250000	0.000000	0.000000	125.00(
50%	58.000000	1.000000	0.000000	130.000000	249.000000	0.000000	0.000000	142.00(
75%	62.000000	1.000000	0.000000	144.750000	283.000000	0.000000	1.000000	156.00(
max	77.000000	1.000000	3.000000	200.000000	409.000000	1.000000	2.000000	195.00(

In [31]:
```
# Filtering data by NEGATIVE Heart Disease patient
neg_data = df[df['target']==0]
neg_data.describe()
```

Out[31]:

	age	sex	cp	trestbps	chol	fbs	restecg	tha
count	138.000000	138.000000	138.000000	138.000000	138.000000	138.000000	138.000000	138.00(
mean	56.601449	0.826087	0.478261	134.398551	251.086957	0.159420	0.449275	139.10
std	7.962082	0.380416	0.905920	18.729944	49.454614	0.367401	0.541321	22.598
min	35.000000	0.000000	0.000000	100.000000	131.000000	0.000000	0.000000	71.00(
25%	52.000000	1.000000	0.000000	120.000000	217.250000	0.000000	0.000000	125.00(
50%	58.000000	1.000000	0.000000	130.000000	249.000000	0.000000	0.000000	142.00(
75%	62.000000	1.000000	0.000000	144.750000	283.000000	0.000000	1.000000	156.00(
max	77.000000	1.000000	3.000000	200.000000	409.000000	1.000000	2.000000	195.00(

In [32]:
```
print("(Positive Patients ST depression): " + str(pos_data['oldpeak'].mean()))
print("(Negative Patients ST depression): " + str(neg_data['oldpeak'].mean()))
```

```
(Positive Patients ST depression): 1.5855072463768116
(Negative Patients ST depression): 1.5855072463768116
```

In [33]:
```
print("(Positive Patients thalach): " + str(pos_data['thalach'].mean()))
print("(Negative Patients thalach): " + str(neg_data['thalach'].mean()))
```

```
(Positive Patients thalach): 139.1014492753623
(Negative Patients thalach): 139.1014492753623
```

In [34]:
```
#Machine Learning + Predictive Analytics
#Prepare Data for Modeling

#To prepare data for modeling, just remember ASN (Assign,Split, Normalize).

#Assign the 13 features to X, & the last column to our classification predictor, y

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [35]:
```
# Split: the data set into the Training set and Test set
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_sta
```

In [36]:
```
#Normalize: Standardizing the data will transform the data so that its distributior
#mean of 0 and a standard deviation of 1

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [37]:
```
# Prediction using the Classification model Logistic Regression

from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression

model1 = LogisticRegression(random_state=1) # get instance of model
model1.fit(x_train, y_train) # Train/Fit model

y_pred1 = model1.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred1)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.81      0.72      0.76        29
           1       0.77      0.84      0.81        32

    accuracy                           0.79        61
   macro avg       0.79      0.78      0.78        61
weighted avg       0.79      0.79      0.79        61
```

In [38]:
```
# Prediction using Random Forest Classifier

from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

model6 = RandomForestClassifier(random_state=1)# get instance of model
model6.fit(x_train, y_train) # Train/Fit model

y_pred6 = model6.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred6)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.85      0.76      0.80        29
           1       0.80      0.88      0.84        32

    accuracy                           0.82        61
   macro avg       0.82      0.82      0.82        61
weighted avg       0.82      0.82      0.82        61
```

In [39]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred6)
print(cm)http://localhost:8889/notebooks/Examining%20Factors%20Responsible%20for%20
accuracy_score(y_test, y_pred6)
```

Out[39]:
```
[[22  7]
 [ 4 28]]
0.819672131147541
```

In [40]:
```
# We get a good accuracy of 80% from the confusion matrix
pip install nbconvert[webpdf]
```

```
  Cell In[40], line 2
    pip install nbconvert[webpdf]
        ^
SyntaxError: invalid syntax
```

In [ ]:
```
```