Prediction of Cardio vascular diseases

This dataset has the information of about 300 patients. Our job here is to find out which factors are responsible for a patient getting a heart attack. Also we need to find out the metrics of a healthy person and a diseased person. We will use a classifier based algorithm to predict which person gets a heart attack based on various factos like cholestorol level, heart rate, chest pain type etc.

Step 1: Setting up the work environment

We are going to download the necessary packages for our work. We are going to view the dataset and check the datatypes.

In [187…

```
#importing packages
from pandas import Series;  from numpy.random import randn
from statsmodels.stats.weightstats import ttest_ind
import scipy.stats as stats
from scipy.stats import ttest_ind
import pandas as pd
import numpy as np

import seaborn as sb
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt
from scipy.stats import shapiro
from scipy import stats

from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures


from sklearn.linear_model import Ridge

from datetime import datetime
from datetime import date
import statsmodels.api as sm
from statsmodels.formula.api import ols

from sklearn.metrics import accuracy_score, make_scorer

from mpl_toolkits.mplot3d import Axes3D

from sklearn import preprocessing as preproc
from sklearn import metrics
from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler



from sklearn.metrics import accuracy_score, auc, balanced_accuracy_score, confusion_matrix, f1_score, precision


from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score, cross_val_predict, cross_validate, train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [2]:

```
#getting the dataset
cvd=pd.read_excel("C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Datasets for ML\\Classifier\\heart_dataset.xl
cvd.head()
```

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]:

```
#examining the dataset
cvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [4]:

```
#examining the dataset
cvd.shape
```

Out[4]:

```
(303, 14)
```

Step 2: Data cleaning

In this step, we are going to clean our dataset. We are going to look for null values and replace them with mean and mode. We are going to modify some variables if it is necessary and change datatypes for better analysis. We will also remove outliers from the dataset. Outliers hamper the machine learning algorithms and hence they have to be removed.

In [5]:

```
# column name cleaning

cvd.columns = cvd.columns.str.replace(' ', '')
cvd.columns
```

Out[5]:

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

In [6]:

```
#checking null values
cvd.isna().sum(axis=0)
```

Out[6]:

```
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 [7]:

```
#changing data type to right format
cvd.sex=cvd.sex.astype(object)
cvd.fbs=cvd.fbs.astype(object)
cvd.thal=cvd.thal.astype(object)
cvd.exang=cvd.exang.astype(object)
cvd.cp=cvd.cp.astype(object)
cvd.ca=cvd.ca.astype(object)
cvd.slope=cvd.slope.astype(object)
cvd.restecg=cvd.restecg.astype(object)
cvd.target=cvd.target.astype(object)
cvd.dtypes
```

Out[7]:

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

Univariate analysis

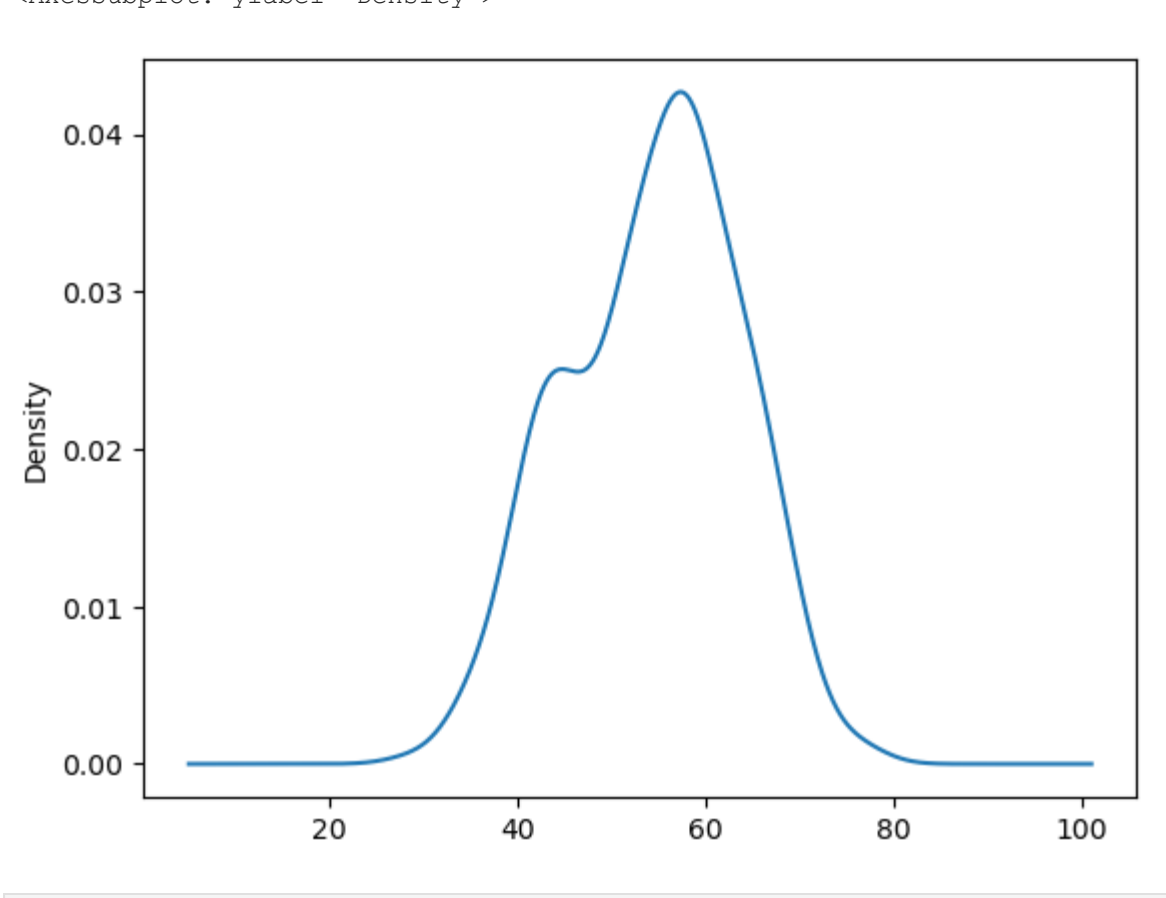
We are using boxplot,histogram and plot to find the outliers,extreme values and distribution of the numerical variables.

In [8]:

```
#checking distribution for age
cvd['age'].plot.density()
```

Out[8]:

```
<AxesSubplot: ylabel='Density'>
```

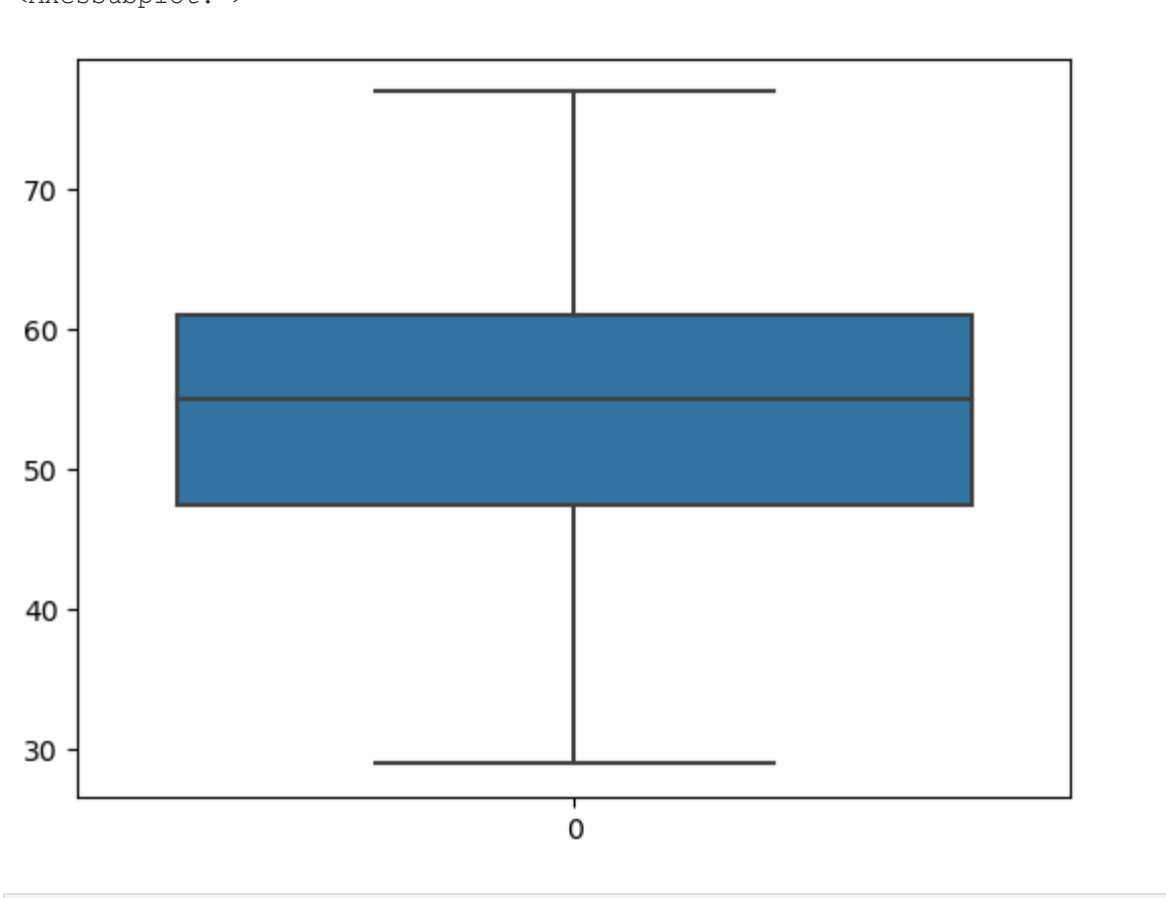


In [9]:

```
#checking outliers using boxplot Age
sb.boxplot(cvd['age'])
```

Out[9]:

```
<AxesSubplot: >
```

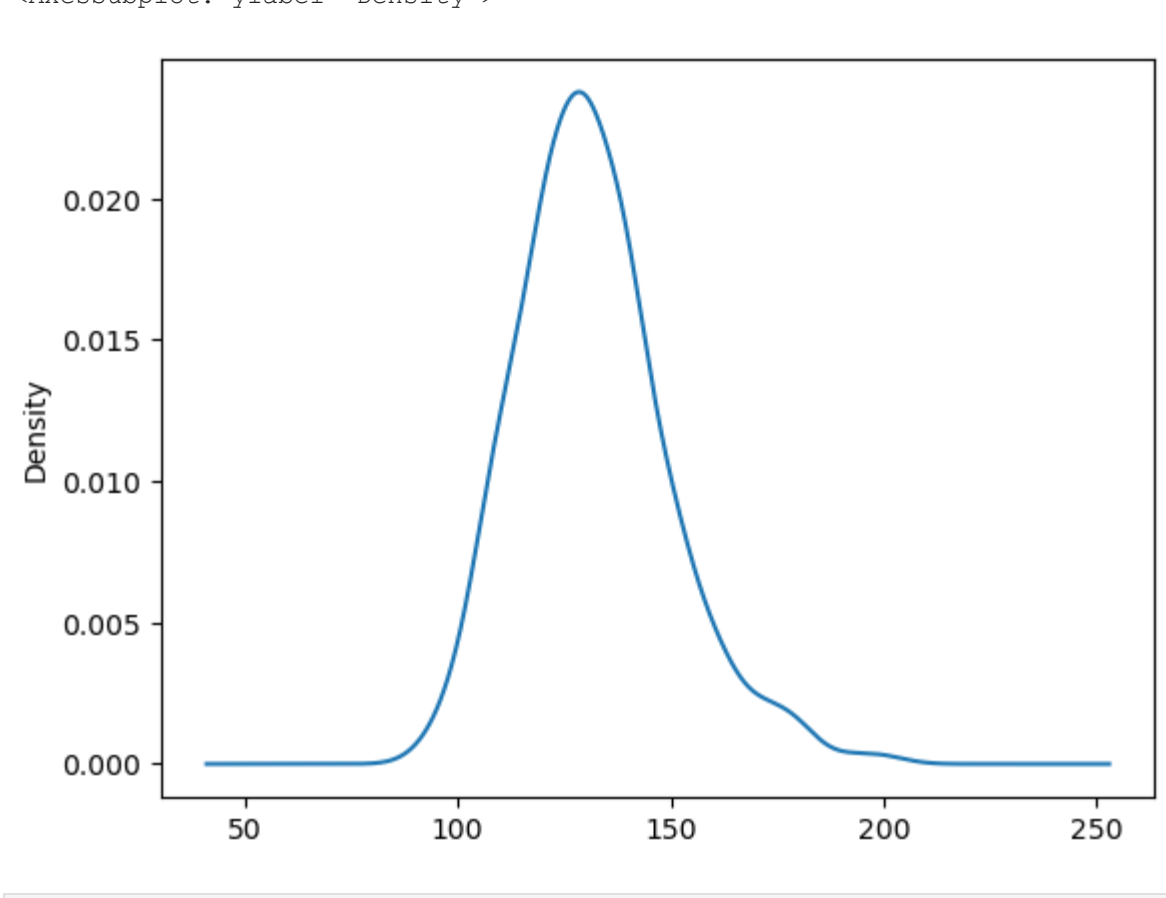


In [10]:

```
#checking distribution for trestbps
cvd['trestbps'].plot.density()
```

Out[10]:

```
<AxesSubplot: ylabel='Density'>
```

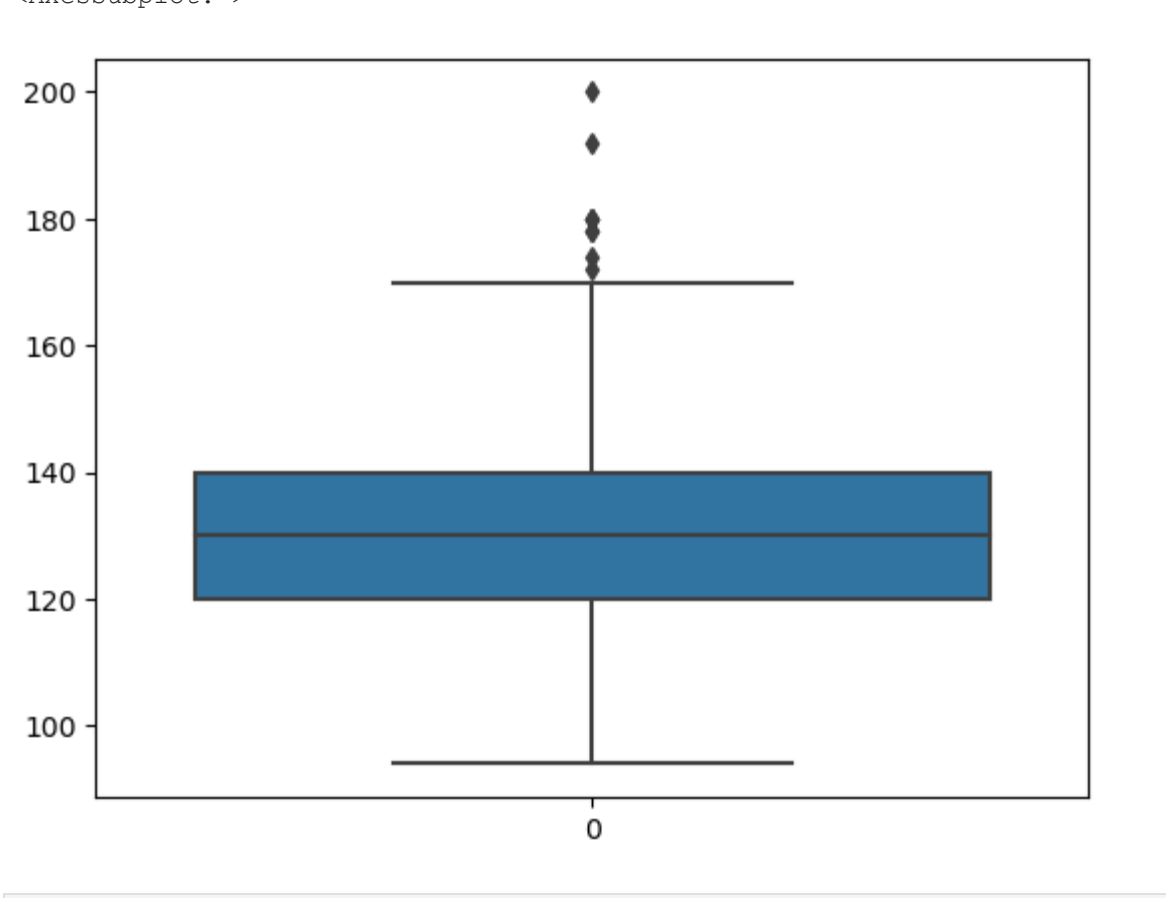


In [11]:

```
#checking outliers using boxplot trestbps
sb.boxplot(cvd['trestbps'])
```

Out[11]:

```
<AxesSubplot: >
```



In [12]:

```
#outlier treatment
cvd['trestbps'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[12]:

```
0.10    110.0
0.25    120.0
0.50    130.0
0.70    140.0
0.90    152.0
0.95    160.0
0.99    180.0
Name: trestbps, dtype: float64
```

In [13]:

```
#seeing limits
trestbps_HE=cvd[cvd['trestbps'] > 160.0].copy()
```

In [14]:

```
#putting them at one place
print(len(trestbps_HE))
```

```
15
```

In [15]:

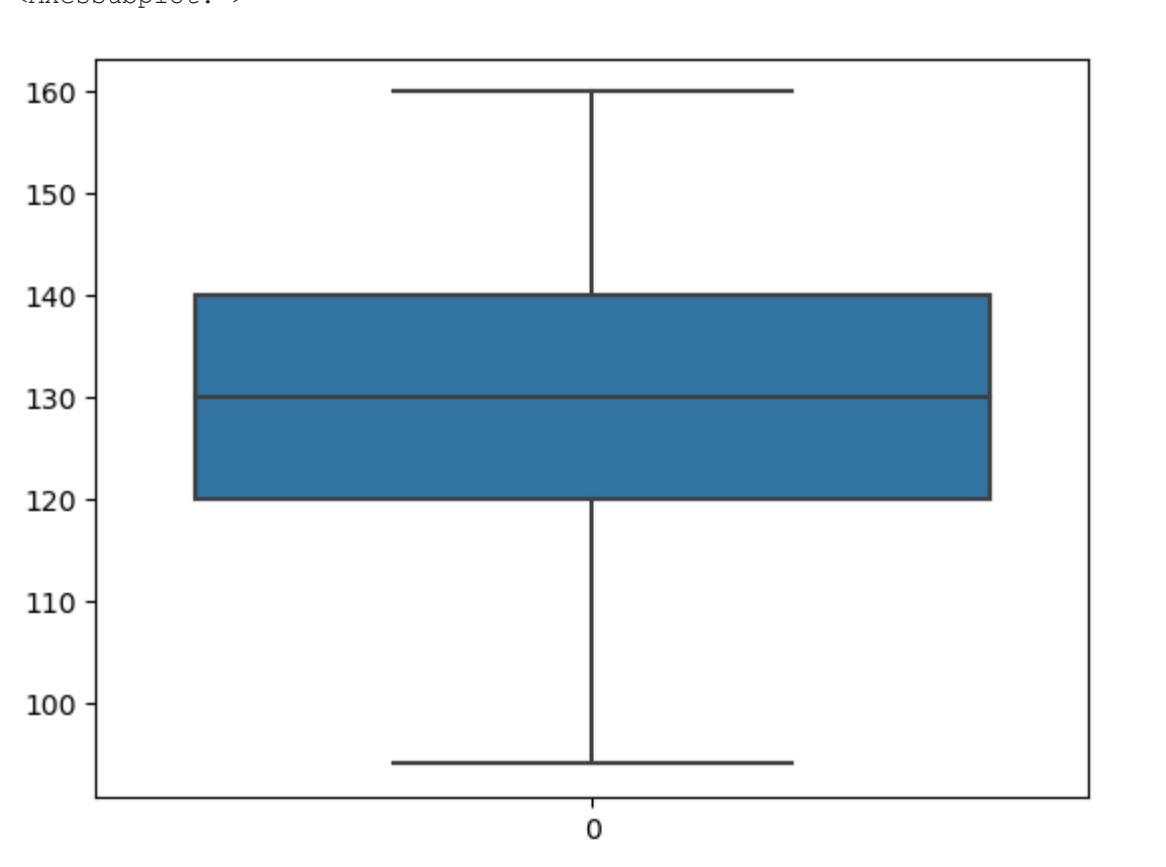
```
#removing outliers
for x in cvd['trestbps']:
    if x > 160.0:
        cvd['trestbps'].replace(x,np.nan,inplace=True)
```

In [16]:

```
#checking outliers gone or not
sb.boxplot(cvd['trestbps'])
```

Out[16]:

```
<AxesSubplot: >
```



In [17]:

```
#replacing with mean values

medbps=cvd['trestbps'].mean()
print(medbps)
cvd['trestbps'].replace(np.nan, medbps, inplace=True)
```

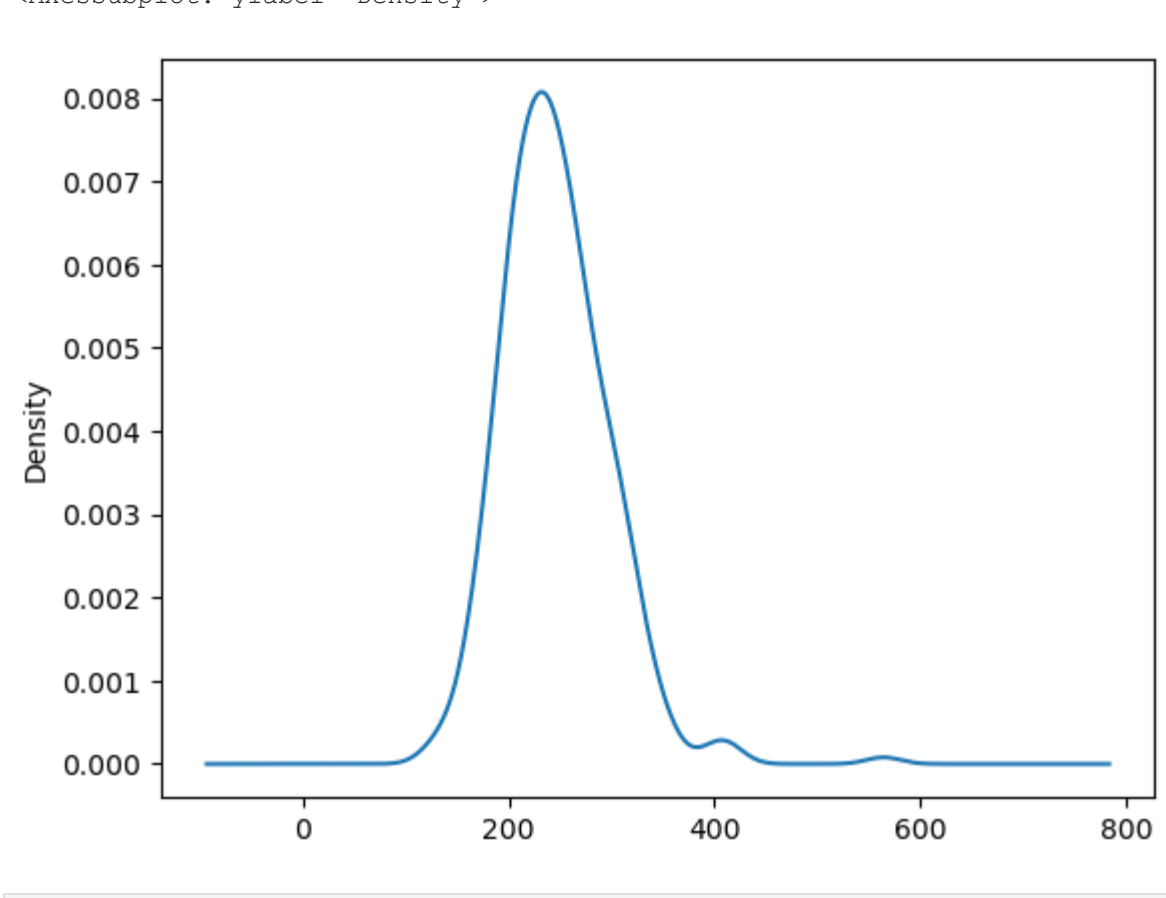
```
129.30208333333334
```

In [18]:

```
#checking distribution for chol
cvd['chol'].plot.density()
```

Out[18]:

```
<AxesSubplot: ylabel='Density'>
```

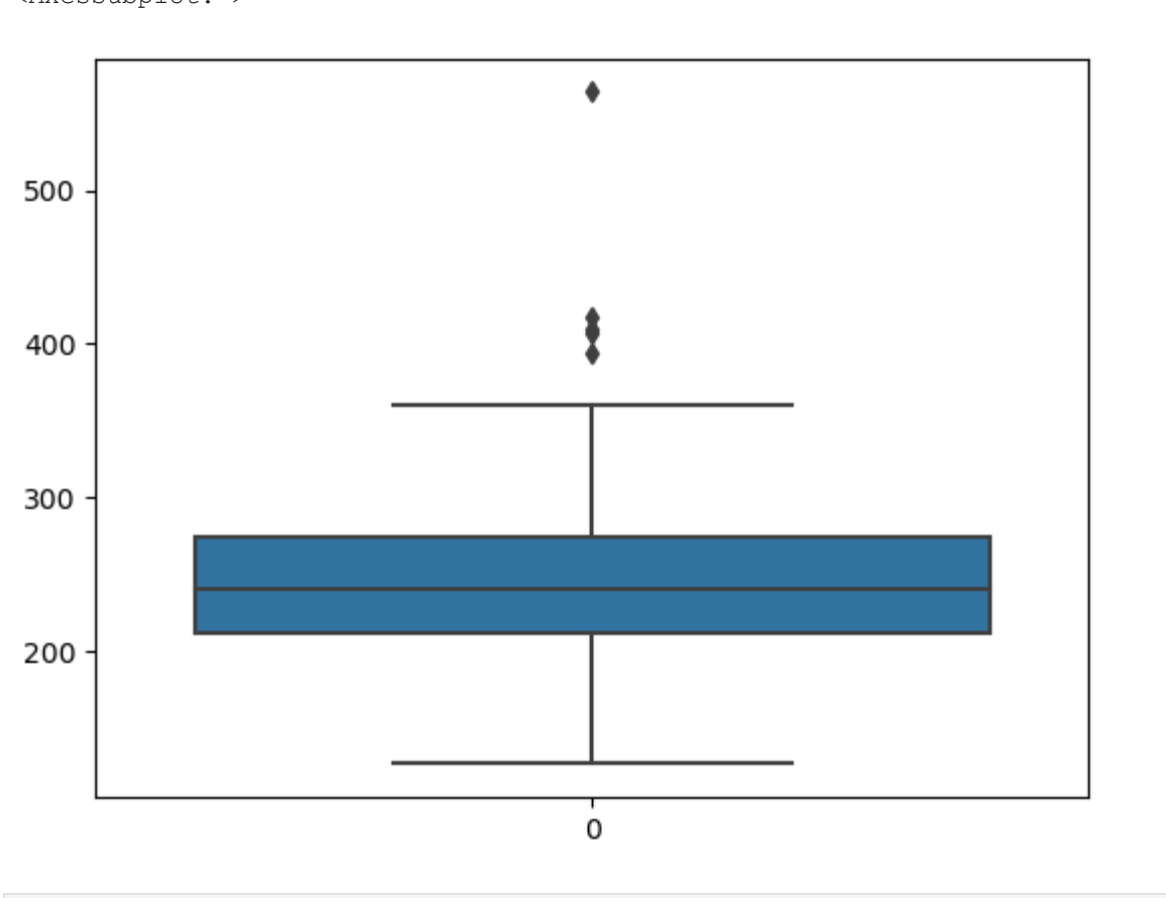


In [19]:

```
#checking outliers using boxplot chol
sb.boxplot(cvd['chol'])
```

Out[19]:

```
<AxesSubplot: >
```



In [20]:

```
#outlier treatment
cvd['chol'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[20]:

```
0.10    188.00
0.25    211.00
0.50    240.00
0.70    268.00
0.90    308.80
0.95    326.90
0.99    406.74
Name: chol, dtype: float64
```

In [21]:

```
#seeing limits
chol_HE=cvd[cvd['chol'] > 330].copy()
```

In [22]:

```
#putting them at one place
print(len(chol_HE))
```

```
13
```

In [23]:

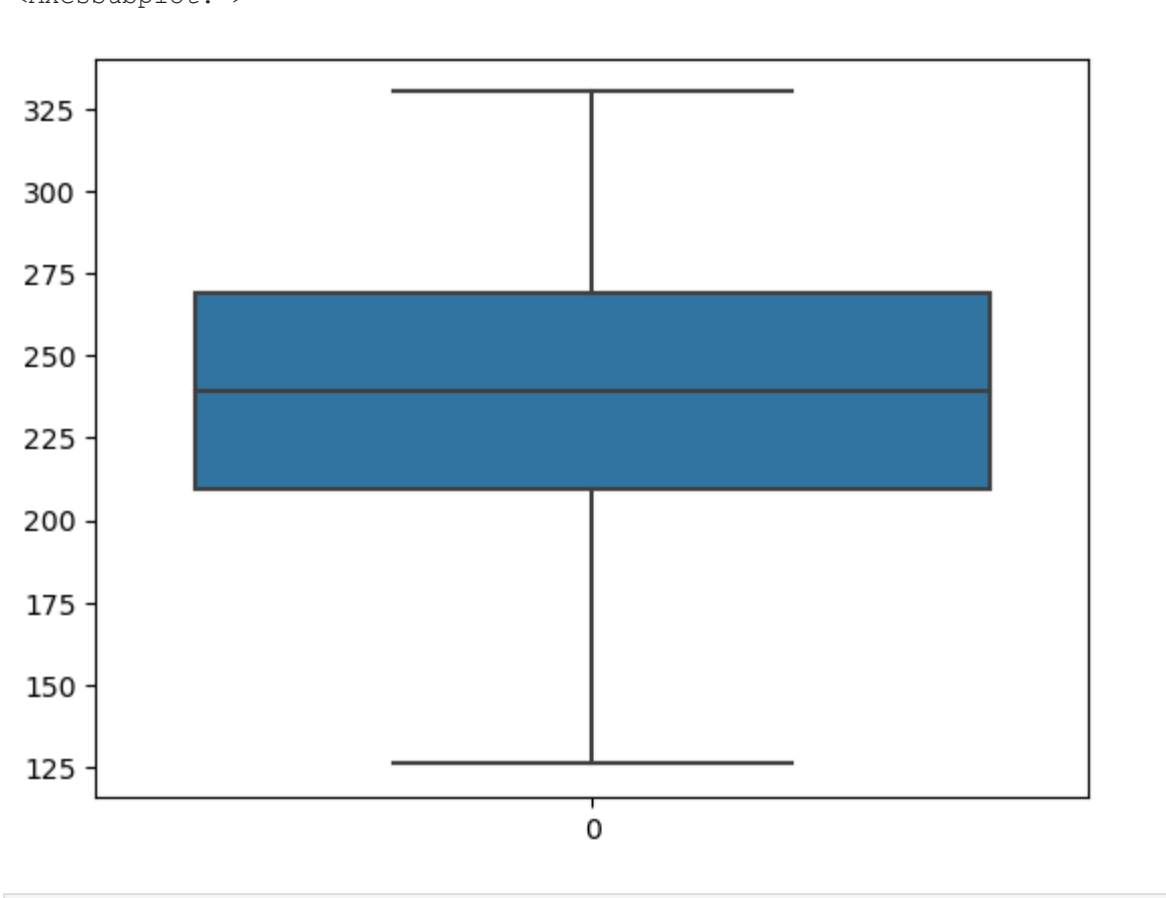
```
#removing outliers
for x in cvd['chol']:
    if x > 330:
        cvd['chol'].replace(x,np.nan,inplace=True)
```

In [24]:

```
#checking outliers gone or not
sb.boxplot(cvd['chol'])
```

Out[24]:

```
<AxesSubplot: >
```



In [25]:

```
#replacing with median values

medchol=cvd['chol'].median()
print(medchol)
cvd['chol'].replace(np.nan, medchol, inplace=True)
```

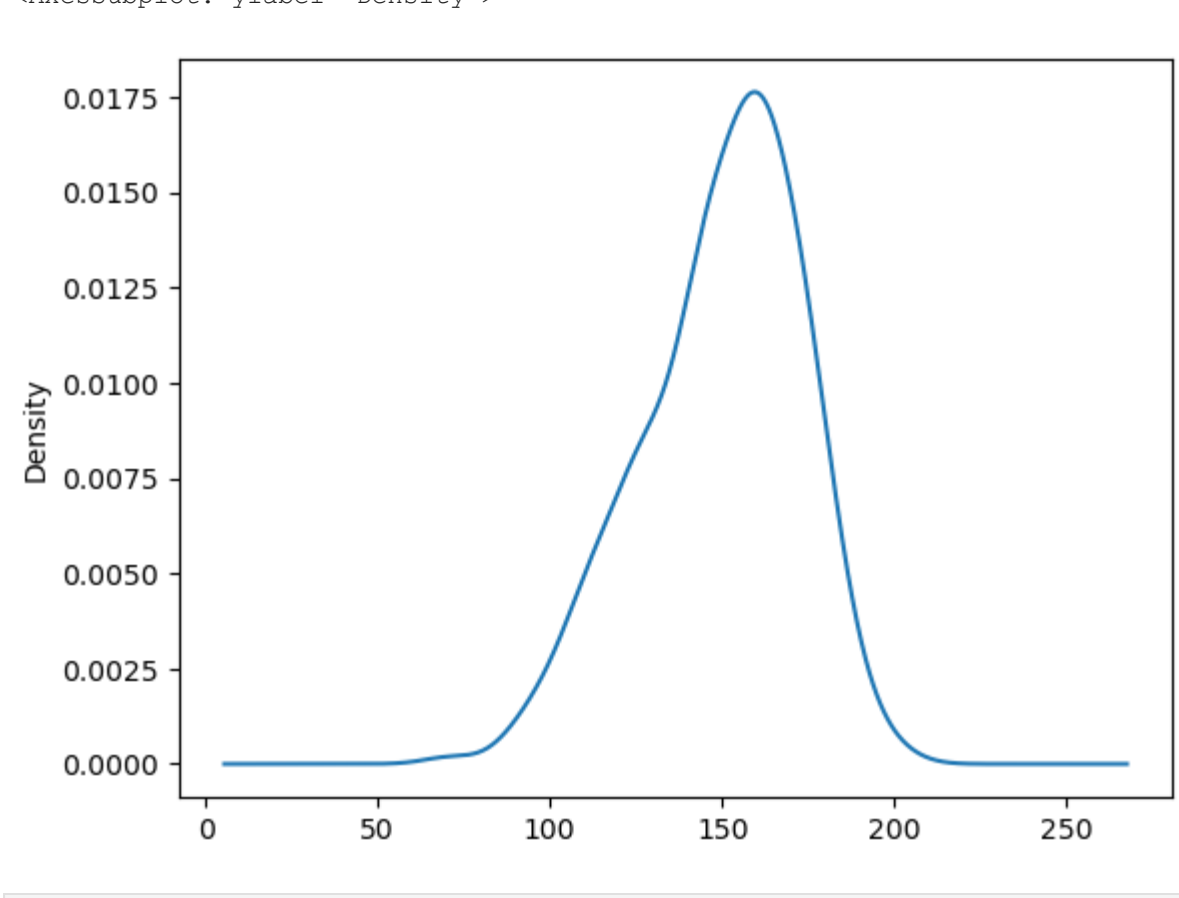
```
239.0
```

In [26]:

```
#checking distribution for thalach
cvd['thalach'].plot.density()
```

Out[26]:

```
<AxesSubplot: ylabel='Density'>
```

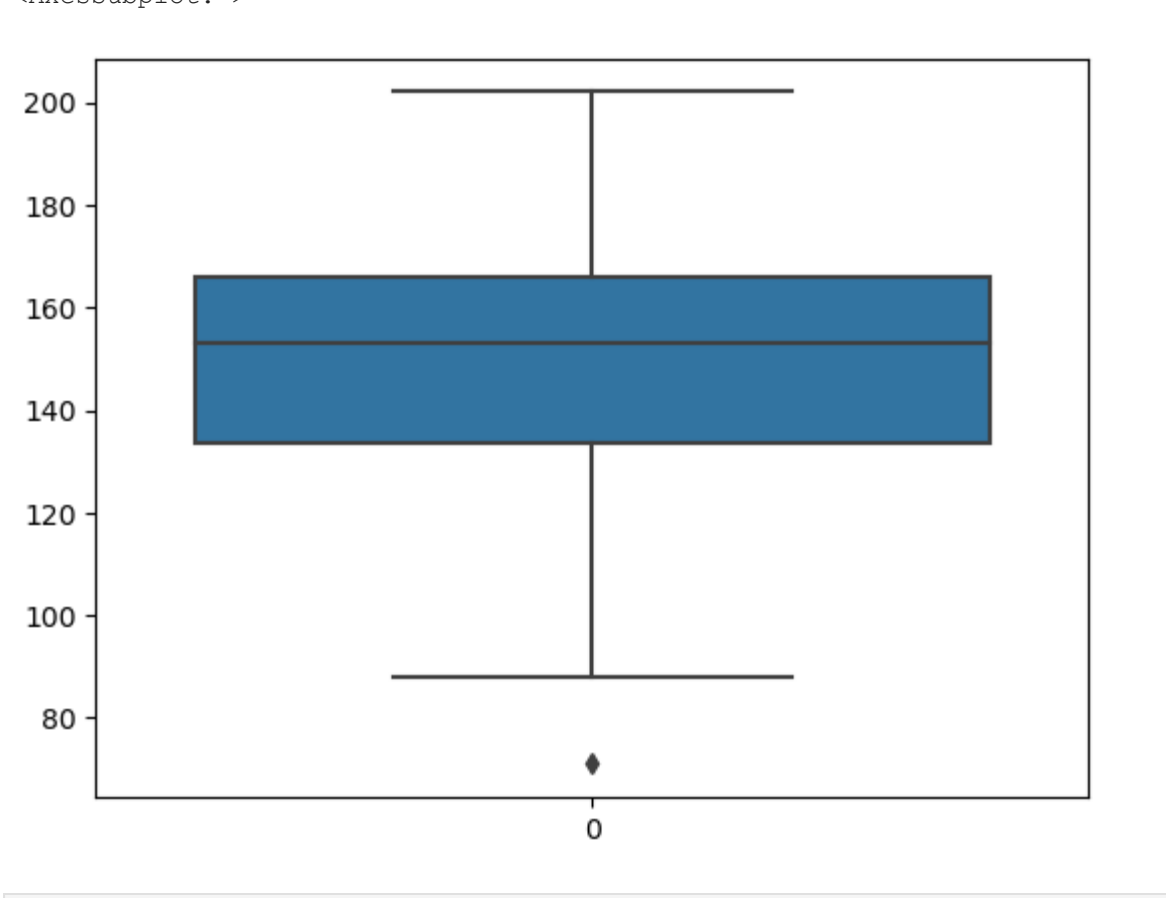


In [27]:

```
#checking outliers using boxplot thalach
sb.boxplot(cvd['thalach'])
```

Out[27]:

```
<AxesSubplot: >
```



In [28]:

```
#outlier treatment
cvd['thalach'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[28]:

```
0.10    116.00
0.25    133.50
0.50    153.00
0.70    163.00
0.90    176.60
0.95    181.90
0.99    191.96
Name: thalach, dtype: float64
```

In [29]:

```
#seeing limits
thal_LE=cvd[cvd['thalach'] < 116].copy()
```

In [30]:

```
#putting them at one place
print(len(thal_LE))
```

```
30
```

In [31]:

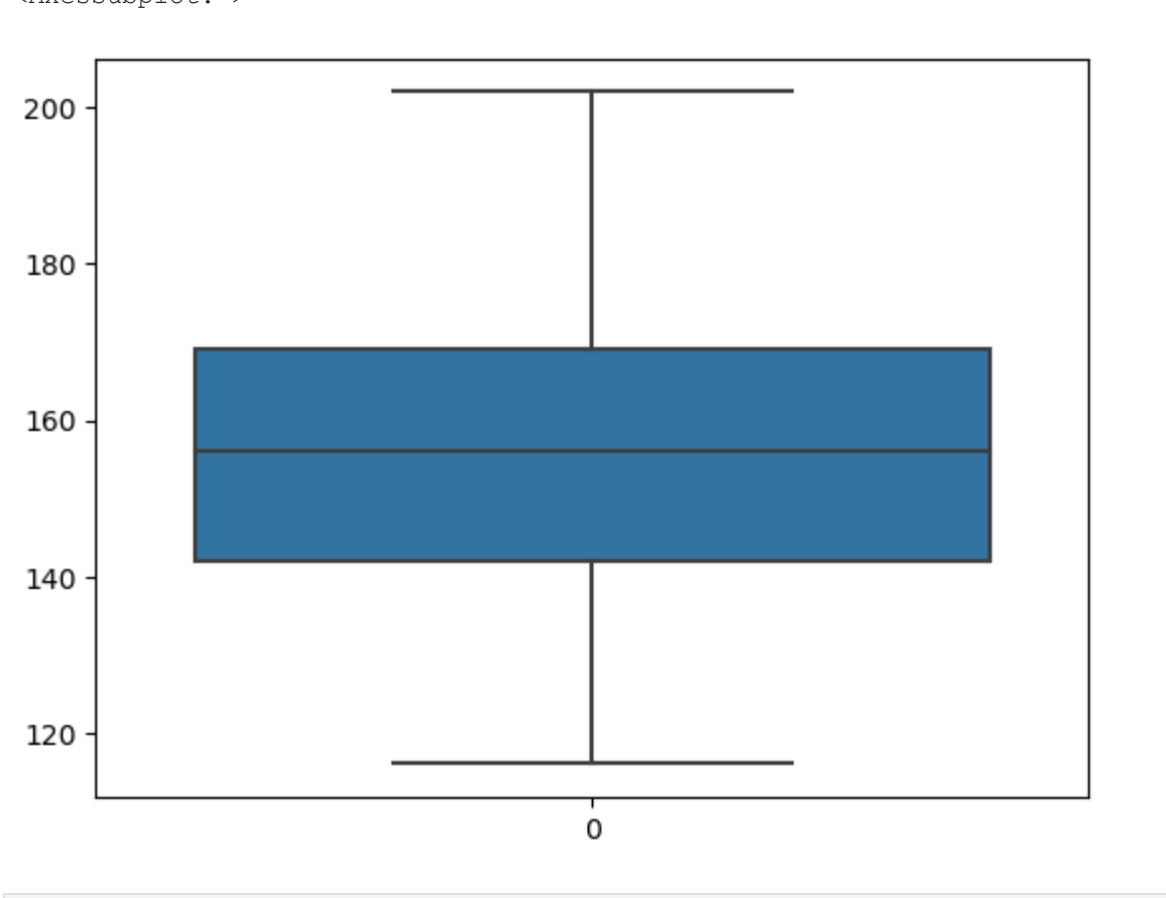
```
#removing outliers
for x in cvd['thalach']:
    if x < 116:
        cvd['thalach'].replace(x,np.nan,inplace=True)
```

In [32]:

```
#checking outliers gone or not
sb.boxplot(cvd['thalach'])
```

Out[32]:

```
<AxesSubplot: >
```



In [33]:

```
#replacing with median values

medthal=cvd['thalach'].median()
print(medthal)
cvd['thalach'].replace(np.nan, medthal, inplace=True)
```

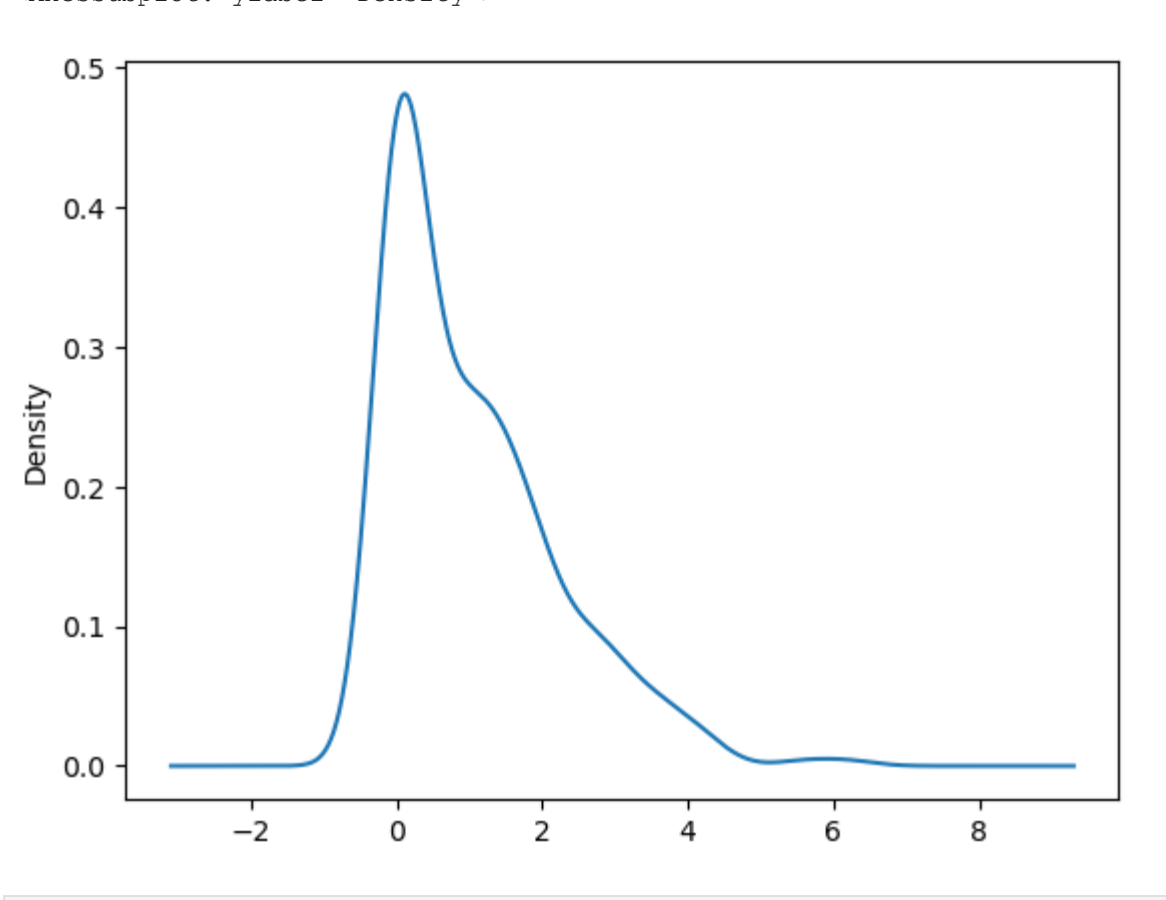
```
156.0
```

In [34]:

```
#checking distribution for oldpeak
cvd['oldpeak'].plot.density()
```

Out[34]:

```
<AxesSubplot: ylabel='Density'>
```

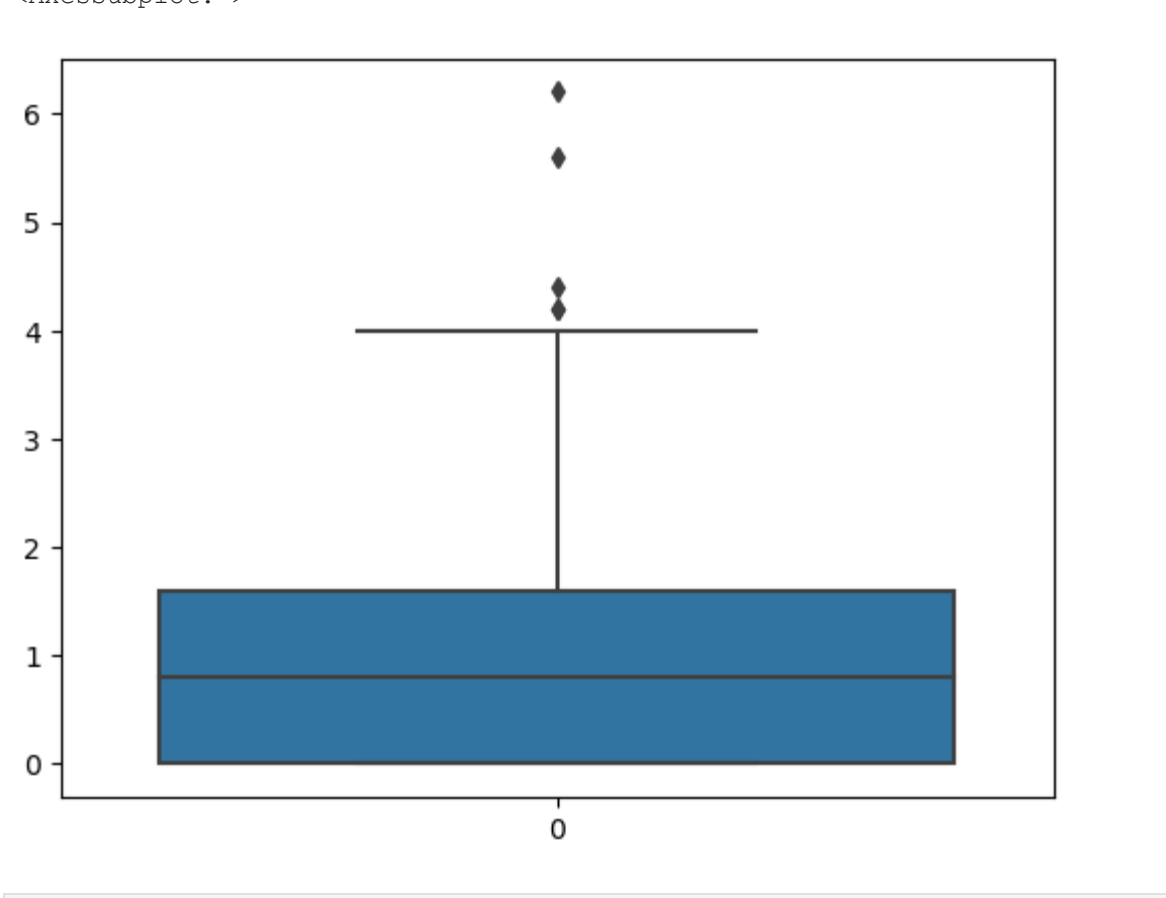


In [35]:

```
#checking outliers using boxplot oldpeak
sb.boxplot(cvd['oldpeak'])
```

Out[35]:

```
<AxesSubplot: >
```



In [36]:

```
#outlier treatment
cvd['oldpeak'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[36]:

```
0.10    0.0
0.25    0.0
0.50    0.8
0.70    1.4
0.90    2.8
0.95    3.4
0.99    4.2
Name: oldpeak, dtype: float64
```

In [37]:

```
#seeing limits
op_HE=cvd[cvd['oldpeak'] > 4].copy()
```

In [38]:

```
#putting them at one place
print(len(op_HE))
```

```
5
```

In [39]:

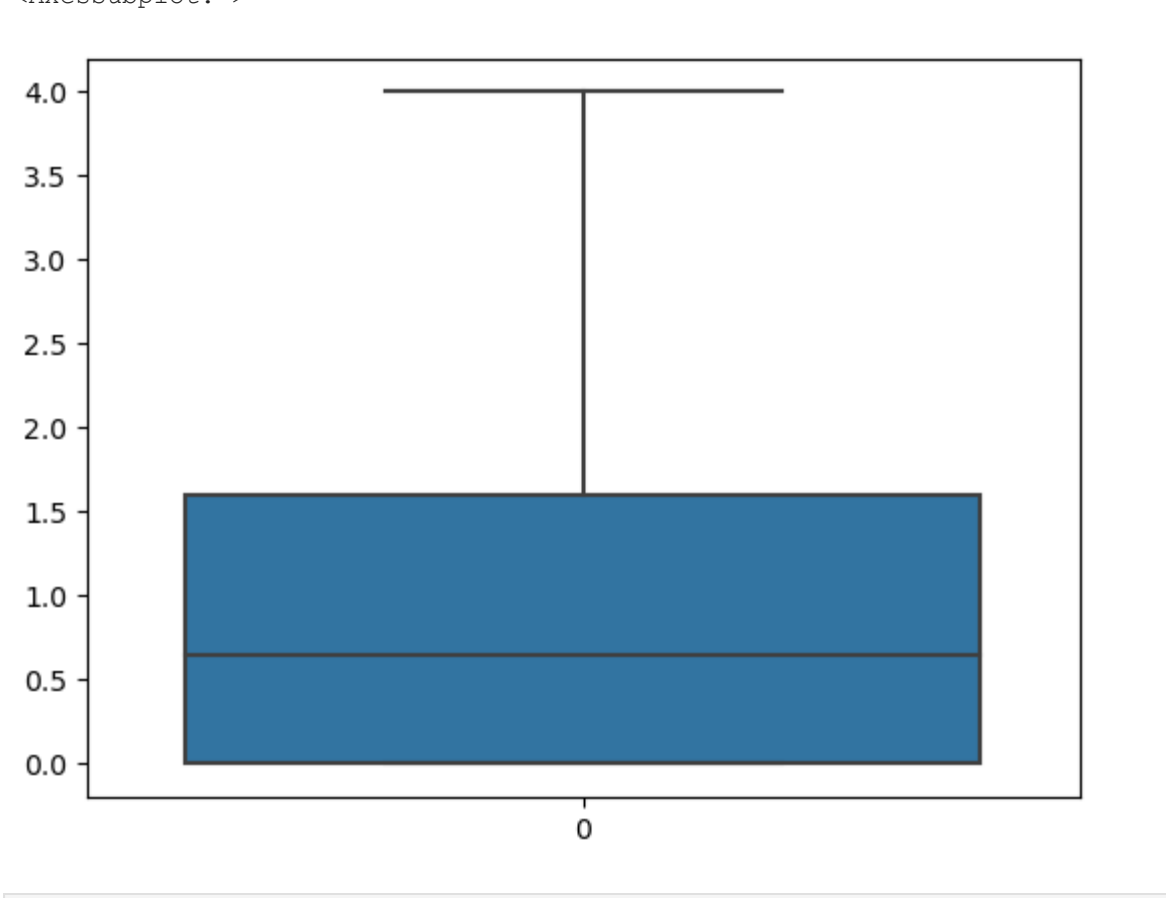
```
#removing outliers
for x in cvd['oldpeak']:
    if x >4:
        cvd['oldpeak'].replace(x,np.nan,inplace=True)
```

In [40]:

```
#checking outliers gone or not
sb.boxplot(cvd['oldpeak'])
```

Out[40]:

```
<AxesSubplot: >
```



In [41]:

```
#replacing with median values

medop=cvd['oldpeak'].median()
print(medop)
cvd['oldpeak'].replace(np.nan, medop, inplace=True)
```

```
0.6499999999999999
```

Step 3:Exploratory data analysis

In this step, we are going to explore the dataset. Perform hypothesis tests,bivariate analysis and check for correlation between variables.

In [42]:

```
#summary stats
cvd.describe().round(2)
```

Out[42]:

	age	trestbps	chol	thalach	oldpeak
count	303.00	303.00	303.00	303.00	303.00
mean	54.37	129.30	240.18	154.70	0.97
std	9.08	14.12	41.41	17.19	1.04
min	29.00	94.00	126.00	116.00	0.00
25%	47.50	120.00	211.00	143.00	0.00
50%	55.00	130.00	239.00	156.00	0.65
75%	61.00	140.00	269.00	166.00	1.60
max	77.00	160.00	330.00	202.00	4.00

In [43]:

```
#seeing correlation of numerical variables

cordata =cvd.corr()
cordata.round(2)
```

Out[43]:

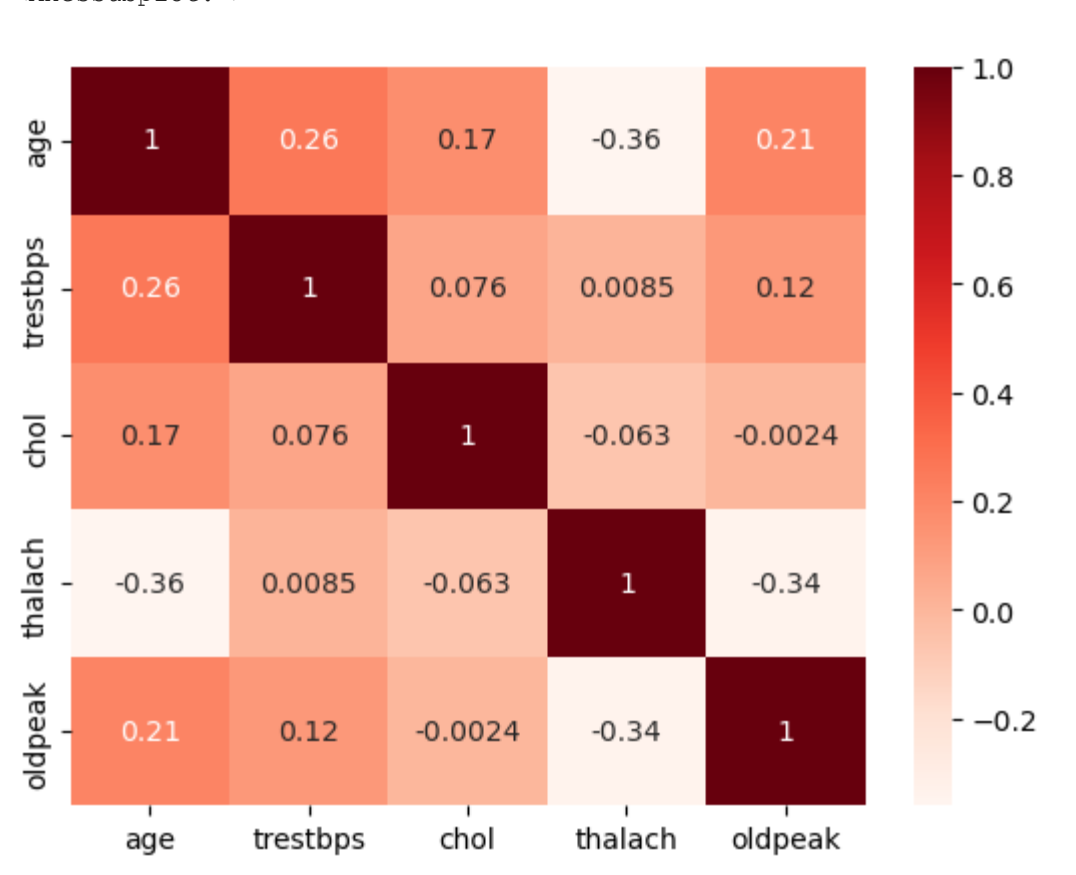
	age	trestbps	chol	thalach	oldpeak
age	1.00	0.26	0.17	-0.36	0.21
trestbps	0.26	1.00	0.08	0.01	0.12
chol	0.17	0.08	1.00	-0.06	-0.00
thalach	-0.36	0.01	-0.06	1.00	-0.34
oldpeak	0.21	0.12	-0.00	-0.34	1.00

In [44]:

```
#heatmap of correlation
sb.heatmap(cordata, annot=True, cmap="Reds")
```

Out[44]:

```
<AxesSubplot: >
```



In [45]:

```
#subsetting dataset between healthy and unhealthy
healthy=cvd[cvd['target']==0]
unhealthy=cvd[cvd['target']==1]
```

In [46]:

```
#getting summary stats of healthy
healthy.describe().round(2)
```

Out[46]:

	age	trestbps	chol	thalach	oldpeak
count	138.00	138.00	138.00	138.00	138.00
mean	56.60	130.63	245.68	148.16	1.46
std	7.96	13.65	42.73	15.59	1.15
min	35.00	100.00	131.00	116.00	0.00
25%	52.00	120.00	217.25	136.50	0.60
50%	58.00	129.65	245.00	150.00	1.20
75%	62.00	140.00	279.75	156.00	2.20
max	77.00	160.00	330.00	195.00	4.00

In [47]:

```
#getting summary stats of unhealthy
unhealthy.describe().round(2)
```

Out[47]:

	age	trestbps	chol	thalach	oldpeak
count	165.00	165.00	165.00	165.00	165.00
mean	52.50	128.20	235.58	160.16	0.56
std	9.55	14.45	39.83	16.58	0.73
min	29.00	94.00	126.00	116.00	0.00
25%	44.00	120.00	208.00	151.00	0.00
50%	52.00	130.00	234.00	161.00	0.20
75%	59.00	138.00	262.00	172.00	1.00
max	76.00	160.00	325.00	202.00	3.50

In [48]:

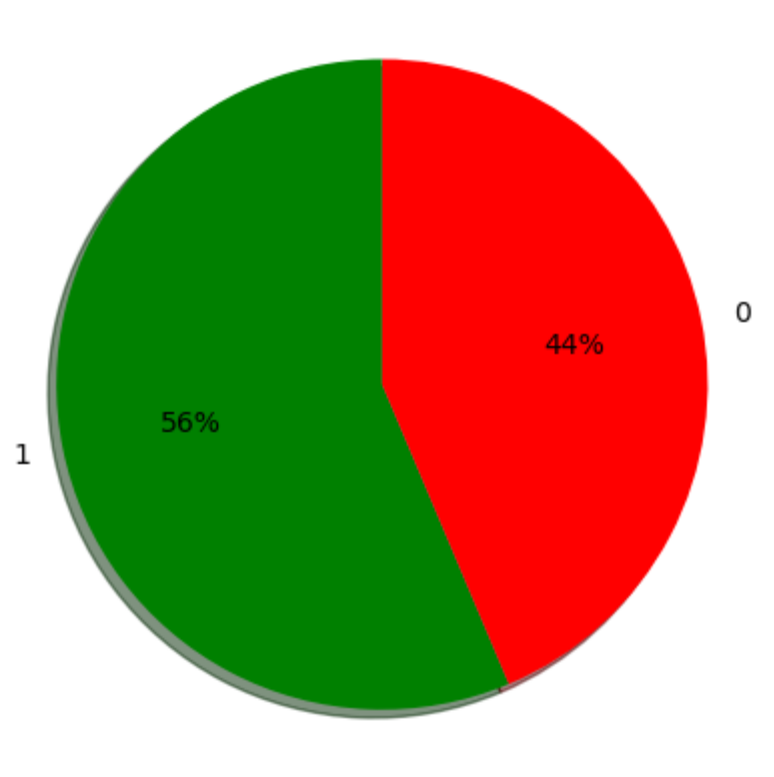
```
#what percent population is vulnerable
vulpop=len(unhealthy)/len(cvd)*100
print("The percent of population that is vulnerable is",vulpop)
```

```
The percent of population that is vulnerable is 54.45544554455446
```

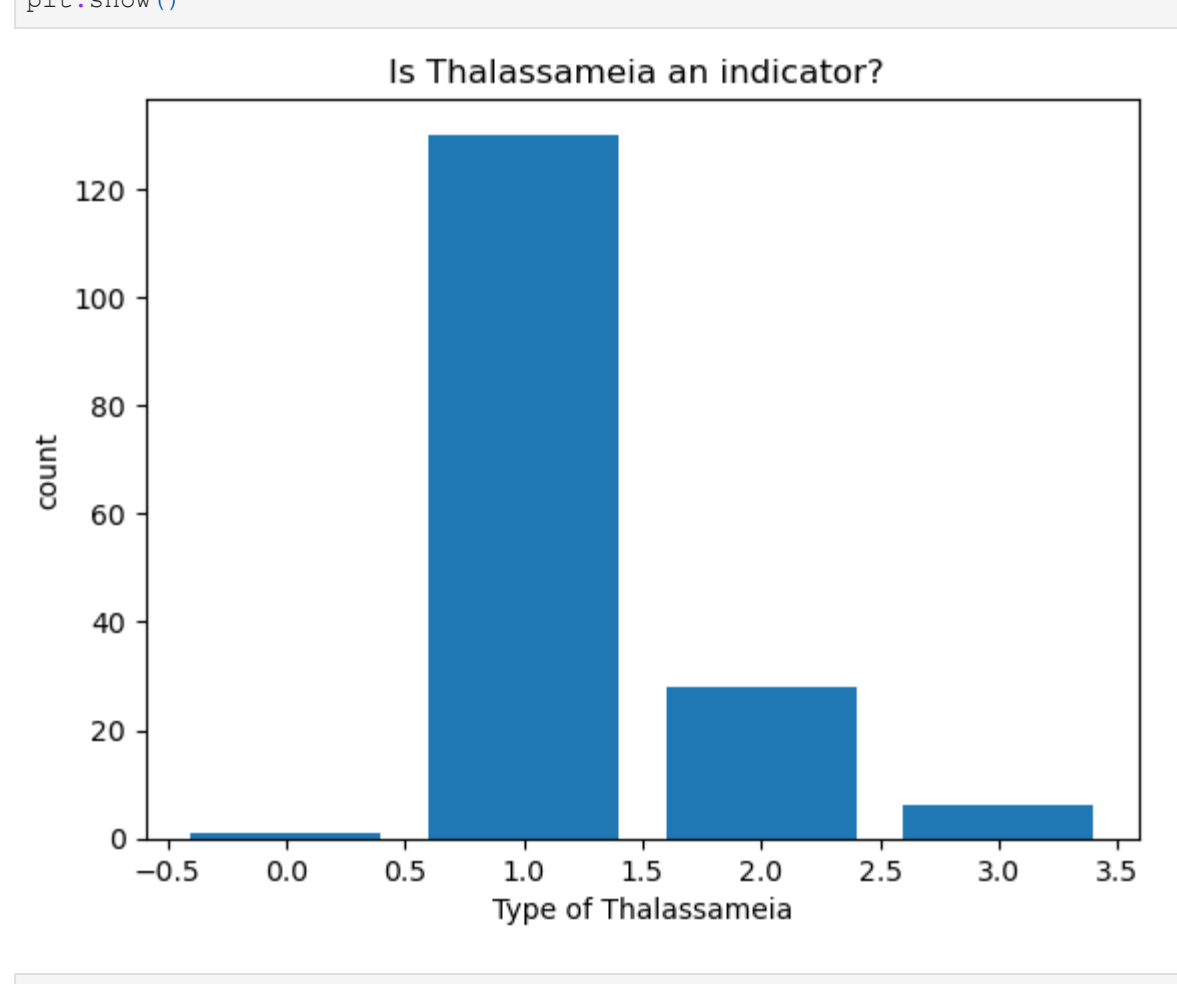
In [49]:

```
#is heart attack dependent on gender?
gender=unhealthy['sex'].unique()
counts=unhealthy['sex'].value_counts()

# color for each label
colors = ['green','red']

# plotting the pie chart
plt.pie(counts, labels =
gender, colors=colors,
        startangle=90, shadow = True,
        radius = 1.1, autopct = '%1.0f%%')
```

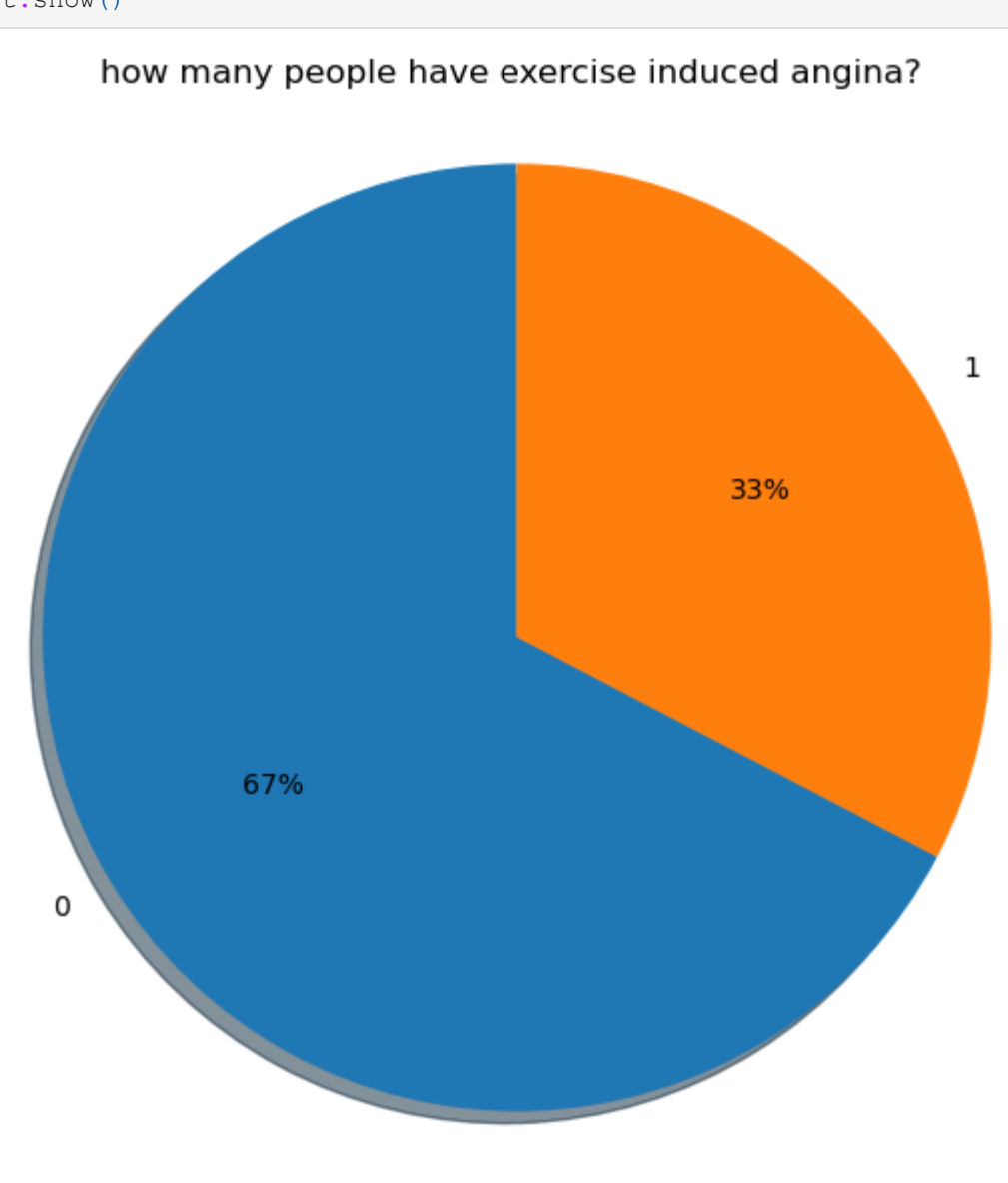
```
Out[49]: ([<matplotlib.patches.Wedge at 0x1ee8ec29460>,
           <matplotlib.patches.Wedge at 0x1ee8ec29d30>],
          [Text(-1.185899899739905, -0.24029446060382614, '1'),
           Text(1.185899899739905, 0.24029446060382548, '0')],
          [Text(-0.6468544907672208, -0.13106970578390514, '56%'),
           Text(0.6468544907672209, 0.13106970578390478, '44%')])
```



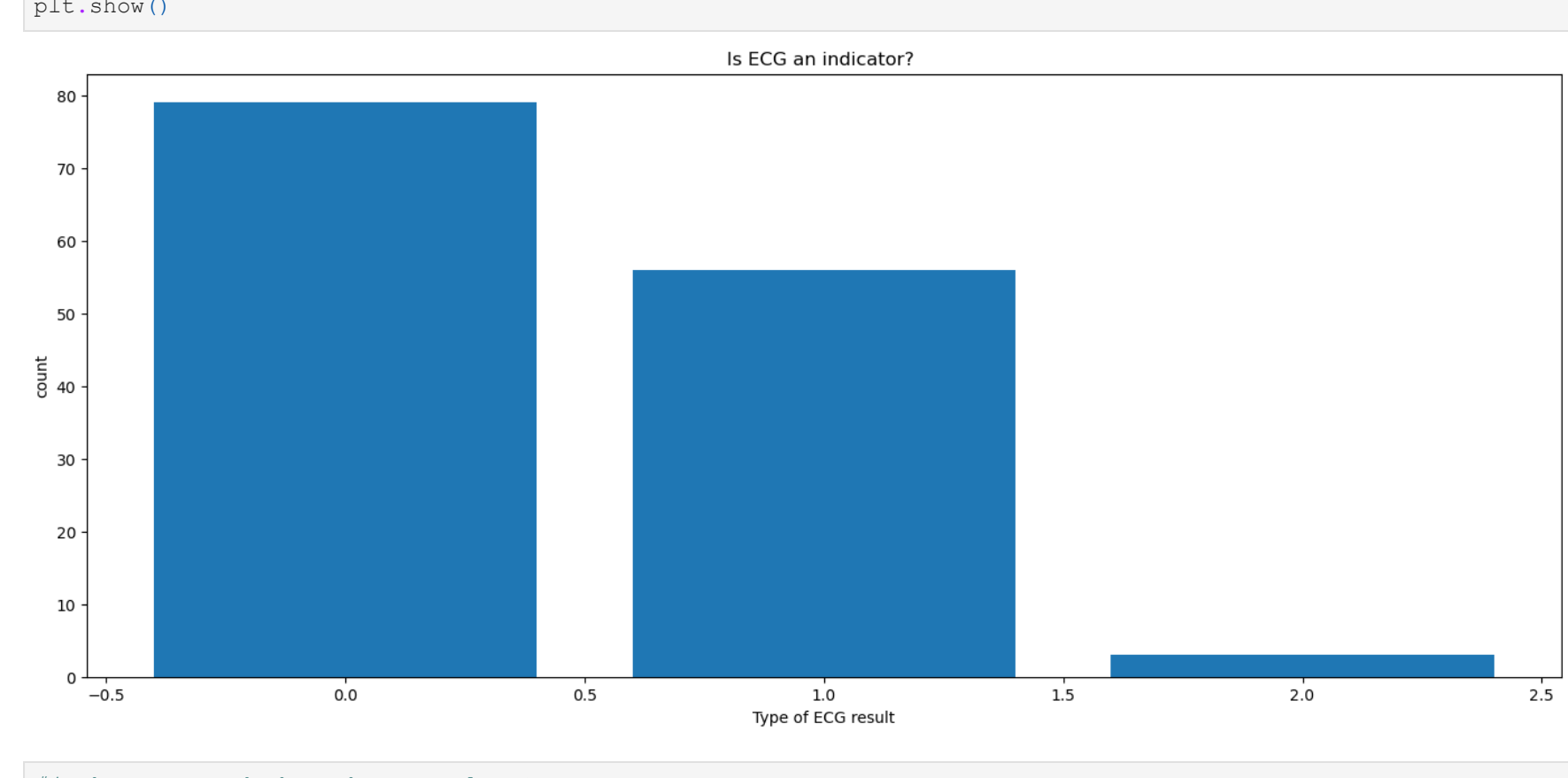

```
In [50]: #Is type of thalassameia an indicator
Thal=unhealthy['thal'].unique()
count=unhealthy['thal'].value_counts()

#plotting people per thalassameia type
plt.bar(Thal,count)


plt.title('Is Thalassameia an indicator? ')
plt.xlabel('Type of Thalassameia')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```



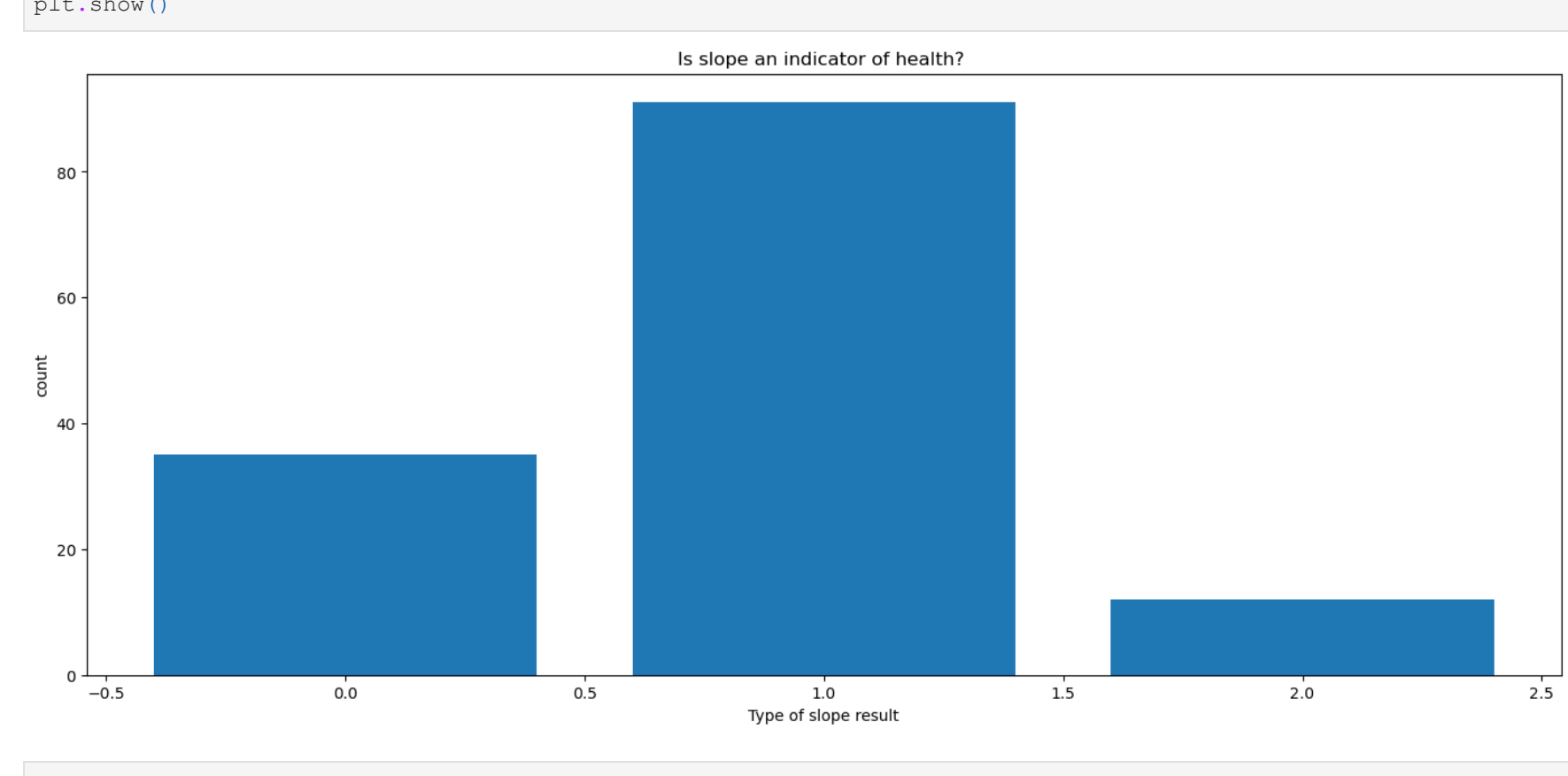

```
In [51]: # percent of people have angina due to exercise

ang=cvd['exang'].unique()
count=cvd['exang'].value_counts()

#plotting people per thalassameia type
plt.pie(count, labels =
ang,
        startangle=90, shadow = True,
        radius = 1.1, autopct = '%1.0f%%')


plt.title('how many people have exercise induced angina? ')

plt.show()
```



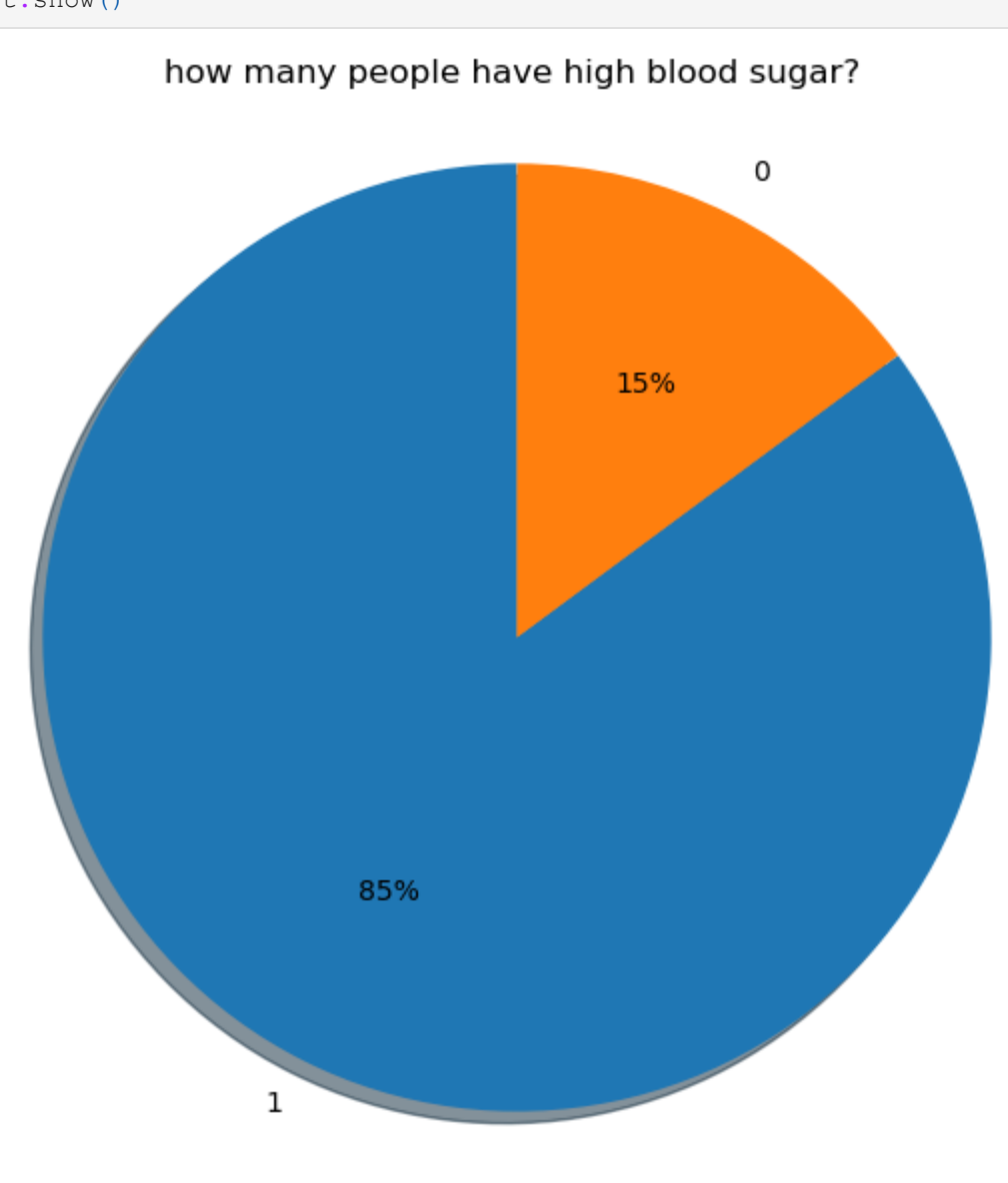

```
In [52]: #Rest ECG type and heart attack
ECG=healthy['restecg'].unique()
count=healthy['restecg'].value_counts()

#plotting people per ECG type
plt.bar(ECG,count)


plt.title('Is ECG an indicator? ')
plt.xlabel('Type of ECG result')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```



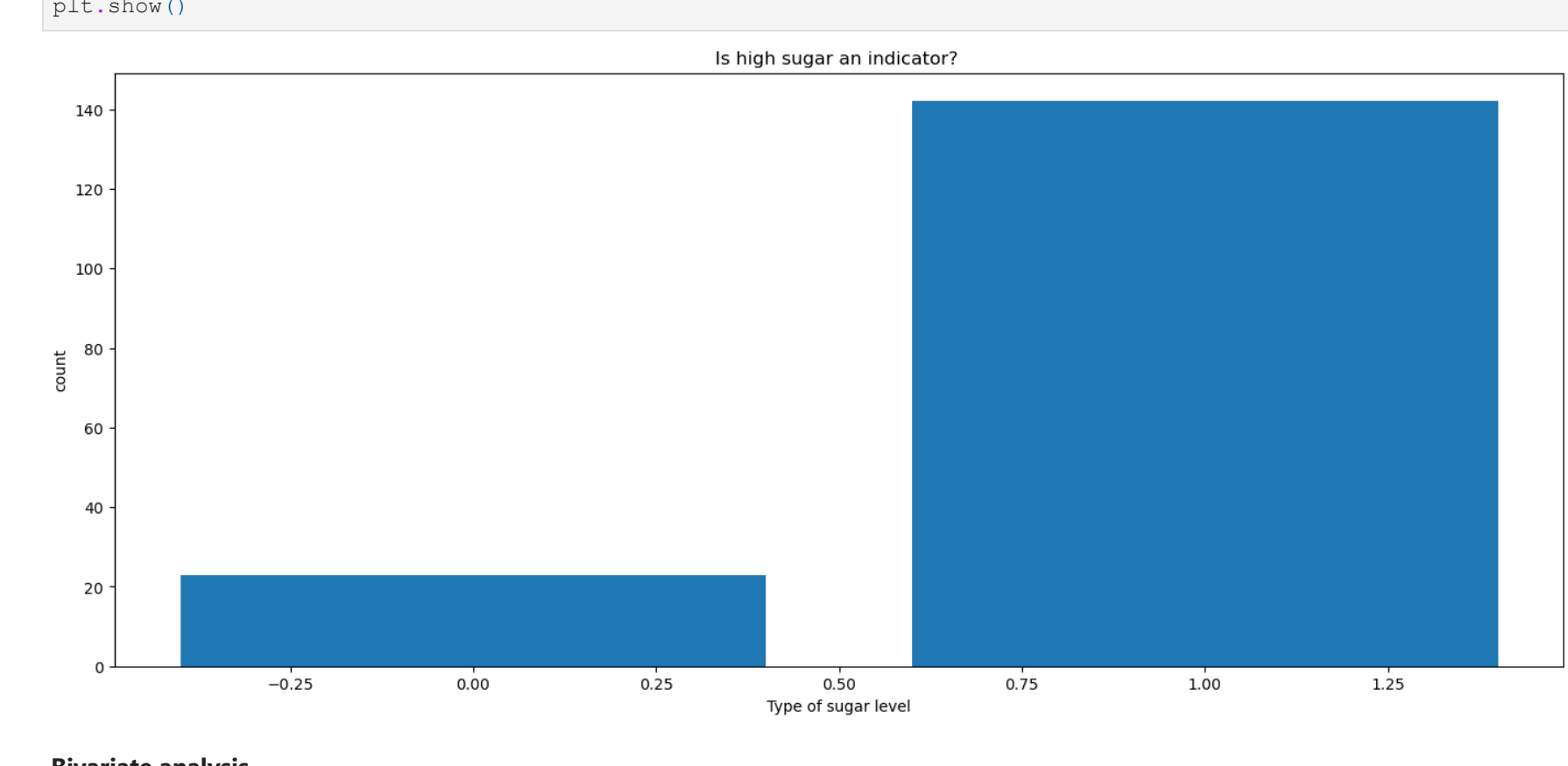

```
In [53]: #is heart attack dependent on slope?
slopetp=healthy['slope'].unique()
count=healthy['slope'].value_counts()


#plotting people per slope type
plt.bar(slopetp,count)


plt.title('Is slope an indicator of health? ')
plt.xlabel('Type of slope result')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```




```
In [54]: #Percentage of people having high sugar
sug=cvd['fbs'].unique()
count=cvd['fbs'].value_counts()

#plotting people per sugar level
plt.pie(count, labels =
sug,
        startangle=90, shadow = True,
        radius = 1.1, autopct = '%1.0f%%')


plt.title('how many people have high blood sugar? ')

plt.show()
```



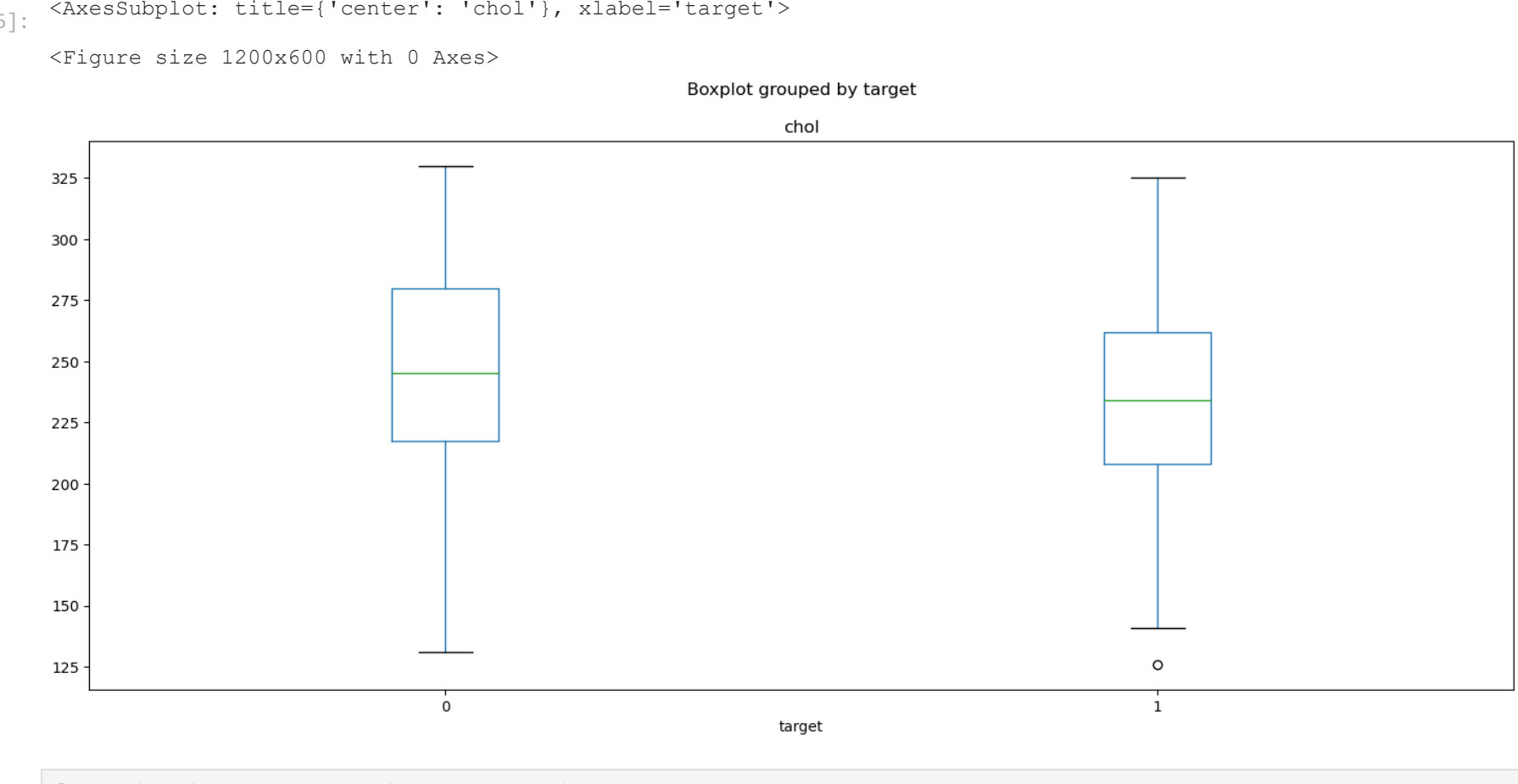

```
In [55]: #sugar level and heart attack
sug=unhealthy['fbs'].unique()
count=unhealthy['fbs'].value_counts()

#plotting people per sugar level
plt.bar(sug,count)


plt.title('Is high sugar an indicator? ')
plt.xlabel('Type of sugar level')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```



Bivariate analysis

Now we are going to analyse two variables with respect to one another.

```
In [56]: #is cholestrol responsible for heart attack
plt.figure(figsize=[12,6])
cvd.boxplot(by="target",column="chol", grid = False)

Out[56]: <AxesSubplot: title={'center': 'chol'}, xlabel='target'>

<Figure size 1200x600 with 0 Axes>
```

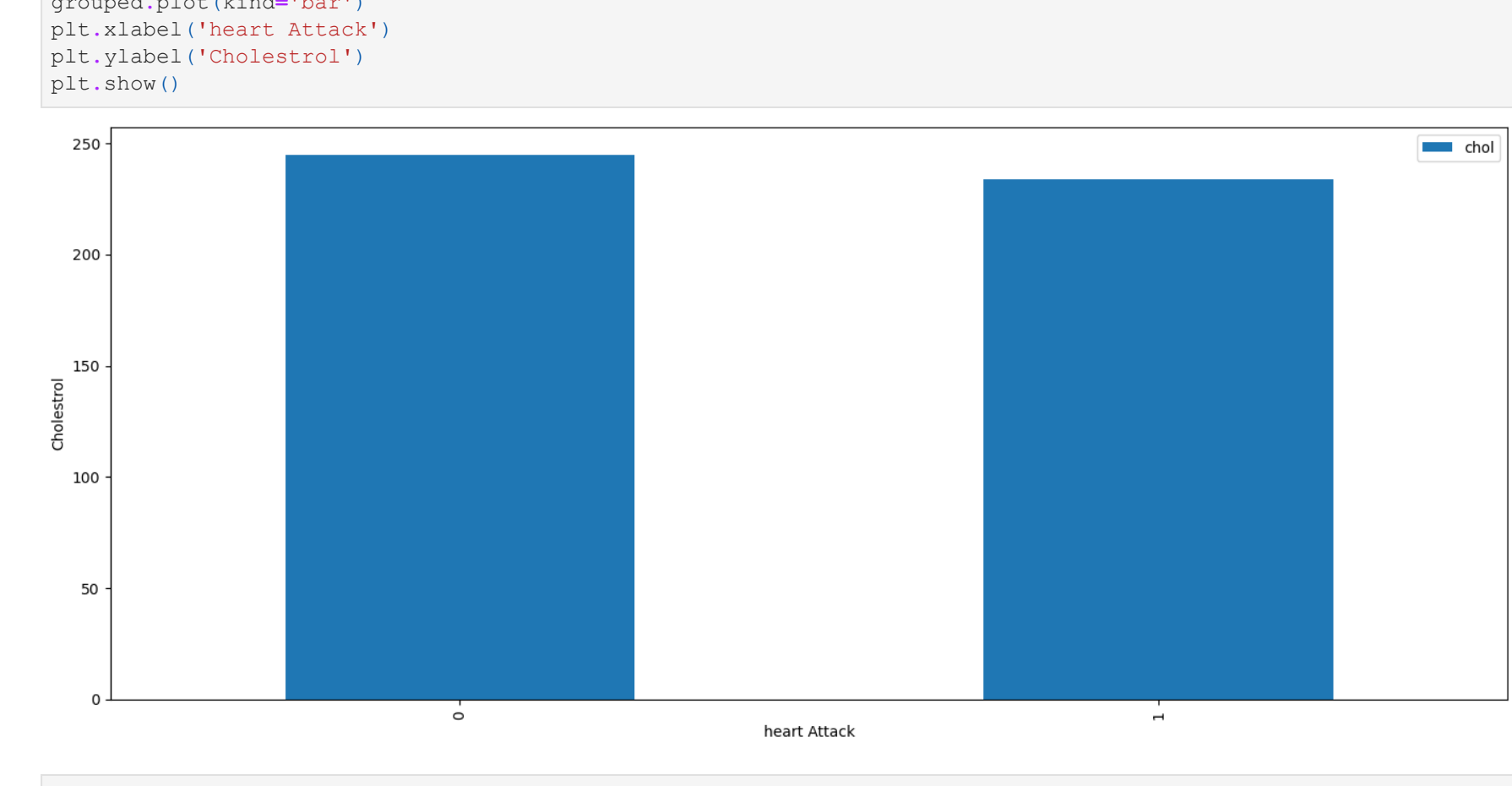



```
In [57]: #checking if cholestrol differs according to the heart attack
chol=cvd['chol']
ha=cvd['target']
mod = ols("chol ~ ha", data=cvd).fit()
anov_table = sm.stats.anova_lm(mod)
anov_table
```

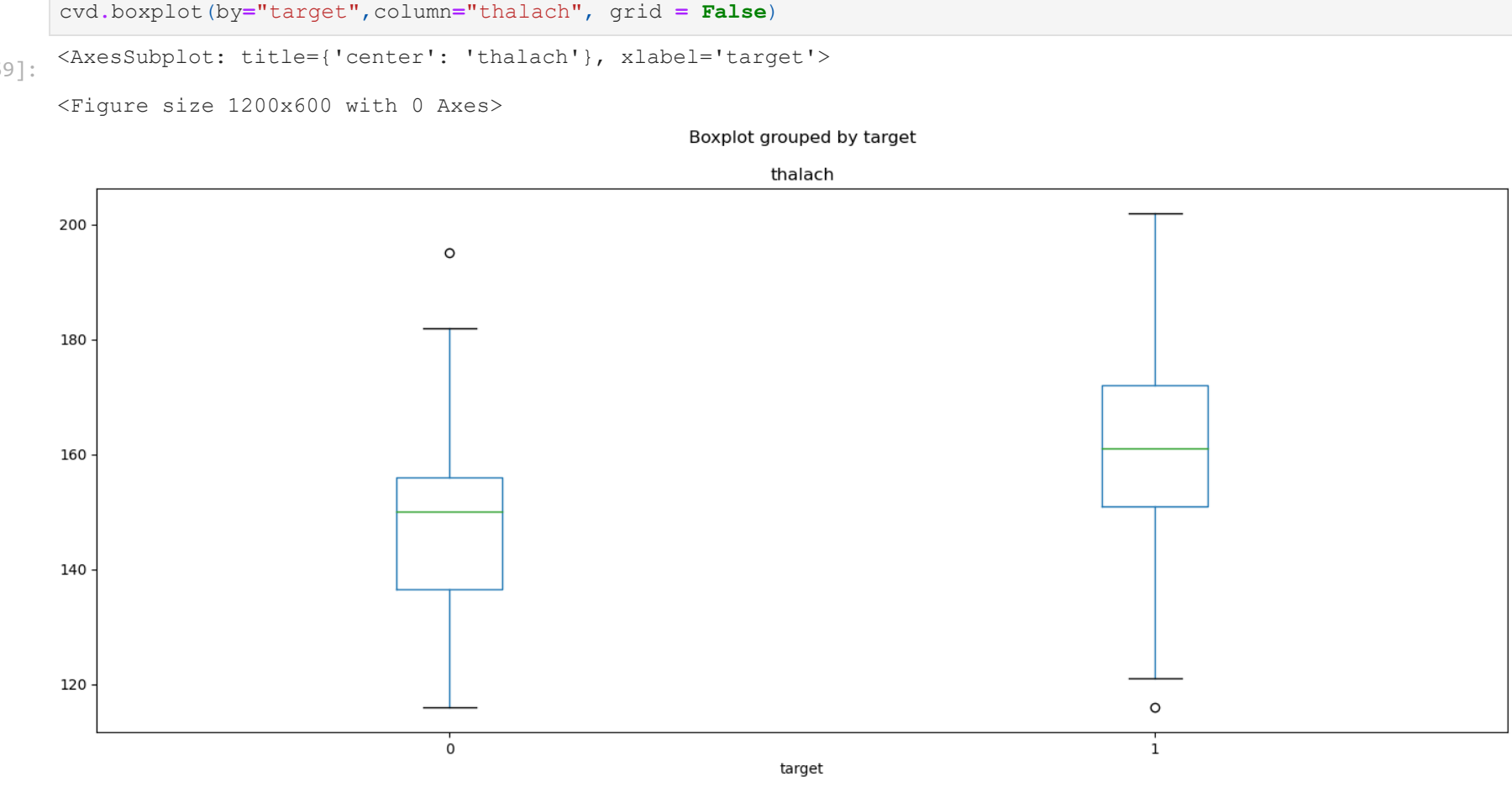
Out[57]:

	df	sum_sq	mean_sq	F	PR(>F)
ha	1.0	7674.102193	7674.102193	4.527199	0.034173
Residual	301.0	510228.274045	1695.110545	NaN	NaN

```
In [58]: #seeing the average cholestrol level by heart attack type
grouped = cvd.groupby('target').agg({'chol': 'median'})

grouped.plot(kind='bar')
plt.xlabel('heart Attack')
plt.ylabel('Cholestrol')
plt.show()
```




```
In [59]: #is high heart rate responsible for heart attack
plt.figure(figsize=[12,6])
cvd.boxplot(by="target",column="thalach", grid = False)

Out[59]: <AxesSubplot: title={'center': 'thalach'}, xlabel='target'>

<Figure size 1200x600 with 0 Axes>
```

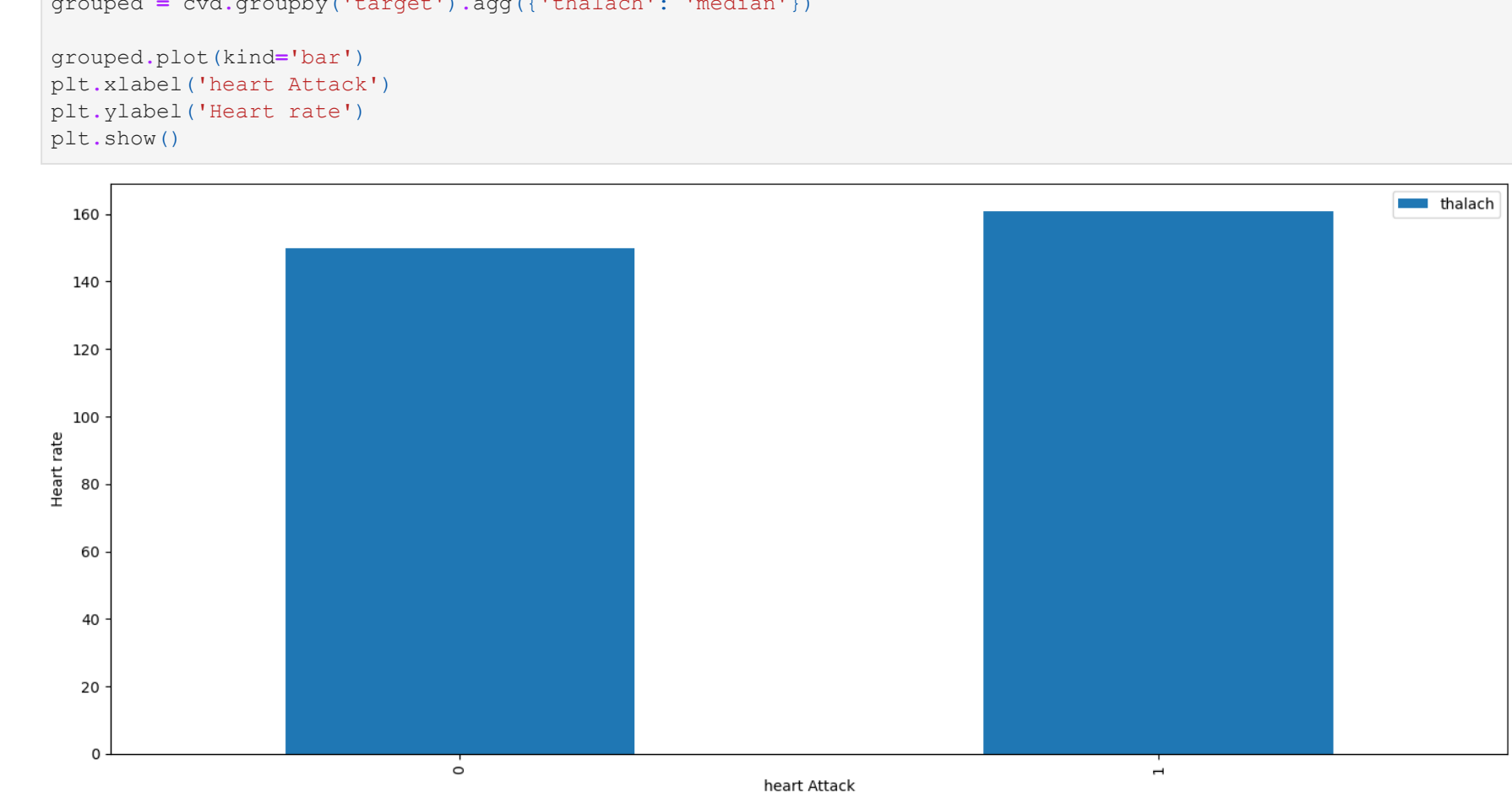



```
In [60]: #checking if high heart rate differs according to the heart attack
hr=cvd['thalach']
ha=cvd['target']
mod = ols("hr ~ ha", data=cvd).fit()
anov_table = sm.stats.anova_lm(mod)
anov_table
```

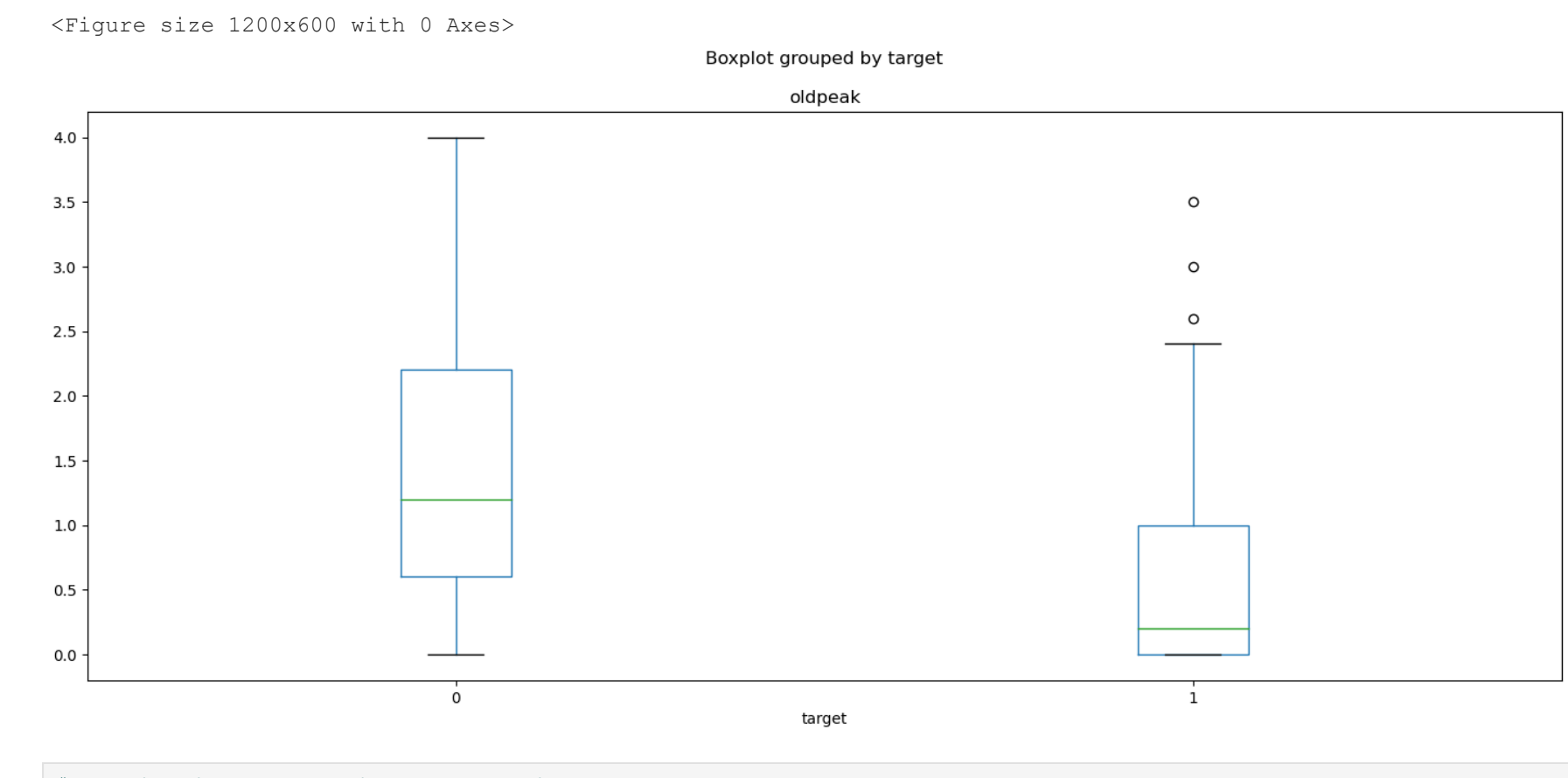
Out[60]:

	df	sum_sq	mean_sq	F	PR(>F)
ha	1.0	10828.991435	10828.991435	41.578204	4.501307e-10
Residual	301.0	78395.074572	260.448753	NaN	NaN

```
In [61]: #seeing the average heart rate by heart attack type
grouped = cvd.groupby('target').agg({'thalach': 'median'})

grouped.plot(kind='bar')
plt.xlabel('heart Attack')
plt.ylabel('Heart rate')
plt.show()
```




```
In [62]: #is oldpeak responsible for heart attack
plt.figure(figsize=[12,6])
cvd.boxplot(by="target",column="oldpeak", grid = False)

Out[62]: <AxesSubplot: title={'center': 'oldpeak'}, xlabel='target'>

<Figure size 1200x600 with 0 Axes>
```

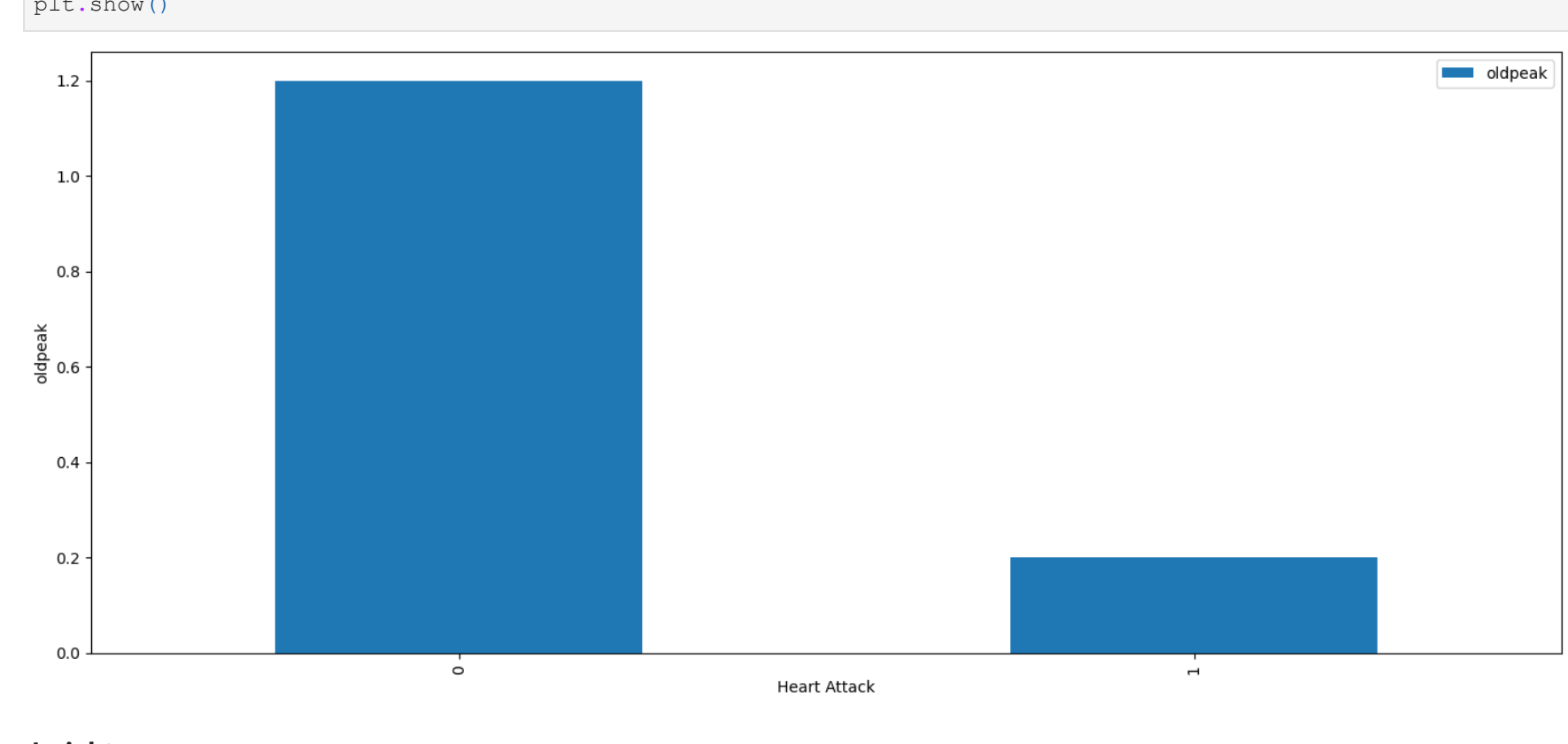



```
In [63]: #checking if oldpeak differs according to the heart attack
op=cvd['oldpeak']
ha=cvd['target']
mod = ols("op ~ ha", data=cvd).fit()
anov_table = sm.stats.anova_lm(mod)
anov_table
```

Out[63]:

	df	sum_sq	mean_sq	F	PR(>F)
ha	1.0	60.196725	60.196725	67.738609	5.717304e-15
Residual	301.0	267.487252	0.888662	NaN	NaN

```
In [64]: #seeing the average oldpeak by heart attack type
grouped = cvd.groupby('target').agg({'oldpeak': 'median'})

grouped.plot(kind='bar')
plt.xlabel('Heart Attack')
plt.ylabel('oldpeak')
plt.show()
```



Insights

1.People who get heart attack are relatively younger than those who dont.

2.Unhealthy people have a higher heart rate and lower cholestrol level,oldpeak and resting BP than healthy.

3.There is a moderately positive correlation between age and variables like oldpeak,cholestrol and resting BP.

4.There is a moderately negative correlation between age and heart rate achieved.

5.Slope type 1 is an indicator of good health mostly.

6.Men are most likely to have heart attacks.

7.Unhealthy people have low oldpeak and cholestrol but a higher heart rate.

8.High sugar level can be associated with heart attacks.

9.Around 85% of the respondents have high sugar.

10.Thalassemia type 1 is found to be associated with heart attacks.

11.More than half of the respondents had a heart attack.

12.Most people didnt have exercise induced angina.

13.Healthy people have ECG type 0 mostly.

Step 4: Model Building

Now we are going to build the classifier models to predict who is more likely to have a heart attack. We will build several models and select the best one.

```
In [65]: #creating target variable which is the outcome of the heart attack
target=cvd['target']

In [66]: #removing target variable from main dataframe
cvd1= cvd.drop(['target'], axis=1)

In [71]: #listing categorical variables
obj_vars=['sex','cp','restecg','fbs','exang','slope',
          'ca','thal']

In [255… cvd1.head()
```

Out[255]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145.0	233.0	1	0	150.0	0	2.3	0	0	1
1	37	1	2	130.0	250.0	0	1	187.0	0	3.5	0	0	2
2	41	0	1	130.0	204.0	0	0	172.0	0	1.4	2	0	2
3	56	1	1	120.0	236.0	0	1	178.0	0	0.8	2	0	2
4	57	0	0	120.0	239.0	0	1	163.0	1	0.6	2	0	2

```
In [68]: #initializing label encoder
le=LabelEncoder()
le

Out[68]: ▾ LabelEncoder
         LabelEncoder()

In [69]: #transforming target variable
target=le.fit_transform(target)
target

Out[69]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

In [72]: #creating dummy variables


for category in cvd1[obj_vars]:
    cvd1[category] = le.fit_transform(cvd1[category])

In [75]: #converting to Dataframe
cvd1= pd.DataFrame(cvd1)
target= pd.DataFrame(target)

In [76]: #getting Decision Tree classifier

dct=DecisionTreeClassifier()

dct

Out[76]: ▾ DecisionTreeClassifier
         DecisionTreeClassifier()

In [77]: #getting Logistic regression classifier

lgr=LogisticRegression()

lgr

Out[77]: ▾ LogisticRegression
         LogisticRegression()

In [78]: #getting GradientB oosting Classifier

gbc=GradientBoostingClassifier()

gbc

Out[78]: ▾ GradientBoostingClassifier
         GradientBoostingClassifier()

In [79]: #getting Gaussian NB classifier

gn=GaussianNB()

gn

Out[79]: ▾ GaussianNB
         GaussianNB()

In [80]: #getting KNeighbors Classifier

knc=KNeighborsClassifier()

knc

Out[80]: ▾ KNeighborsClassifier
         KNeighborsClassifier()

In [81]: #getting Random Forest Classifier

rfc=RandomForestClassifier()

rfc

Out[81]: ▾ RandomForestClassifier
         RandomForestClassifier()

In [82]: #getting Support vector classifier

svc=SVC()

svc

Out[82]: ▾ SVC
         SVC()

In [83]: #standard scaler initiated
sc=StandardScaler()

In [125… #creating X and y
X = cvd1.values
y= target.values

In [126… #train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.25, random_state=1)

In [127… #using standard scaler
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

In [137… #fitting the dct model
model1=dct.fit(X_train,y_train)
model1

Out[137]: ▾ DecisionTreeClassifier
          DecisionTreeClassifier()

In [138… #predict
ypred1= model1.predict(X_test)
ypred1

Out[138]: array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0])

In [139… #accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred1)*100))

Accuracy of the model: 69.73684210526315%

In [140… #confusion matrix
cm=confusion_matrix(y_test,ypred1)
print(cm)

[[28  7]
 [16 25]]

In [141… #printing metrics
print(classification_report(y_test,ypred1))

              precision    recall  f1-score   support

           0       0.64      0.80      0.71        35
           1       0.78      0.61      0.68        41

    accuracy                           0.70        76
   macro avg       0.71      0.70      0.70        76
weighted avg       0.71      0.70      0.70        76

In [142… #fitting the lgr model
model2=lgr.fit(X_train,y_train)
model2

C:\Users\sujoydutta\anaconda3\lib\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A colu
mn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example
using ravel().
  y = column_or_1d(y, warn=True)

Out[142]: ▾ LogisticRegression
          LogisticRegression()

In [143… #predict
ypred2= model2.predict(X_test)
ypred2

Out[143]: array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 1])

In [144… #accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred2)*100))

Accuracy of the model: 76.31578947368422%

In [145… #confusion matrix
cm=confusion_matrix(y_test,ypred2)
print(cm)

[[24 11]
 [ 7 34]]

In [146… #printing metrics
print(classification_report(y_test,ypred2))

              precision    recall  f1-score   support

           0       0.77      0.69      0.73        35
           1       0.76      0.83      0.79        41

    accuracy                           0.76        76
   macro avg       0.76      0.76      0.76        76
weighted avg       0.76      0.76      0.76        76

In [148… #fitting the gn model
model3=gn.fit(X_train,y_train)
model3

C:\Users\sujoydutta\anaconda3\lib\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A colu
mn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example
using ravel().
  y = column_or_1d(y, warn=True)

Out[148]: ▾ GaussianNB
          GaussianNB()

In [149… #predict
ypred3= model3.predict(X_test)
ypred3

Out[149]: array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1])

In [150… #accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred3)*100))

Accuracy of the model: 76.31578947368422%

In [151… #confusion matrix
cm=confusion_matrix(y_test,ypred3)
print(cm)

[[26  9]
 [ 9 32]]

In [152… #printing metrics
print(classification_report(y_test,ypred3))

              precision    recall  f1-score   support

           0       0.74      0.74      0.74        35
           1       0.78      0.78      0.78        41

    accuracy                           0.76        76
   macro avg       0.76      0.76      0.76        76
weighted avg       0.76      0.76      0.76        76

In [153… #fitting the svc model
model4=svc.fit(X_train,y_train)
model4

C:\Users\sujoydutta\anaconda3\lib\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A colu
mn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example
using ravel().
  y = column_or_1d(y, warn=True)

Out[153]: ▾ SVC
          SVC()

In [154… #predict
ypred4= model4.predict(X_test)
ypred4

Out[154]: array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 1])

In [155… #accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred4)*100))

Accuracy of the model: 80.26315789473685%
```

In [156…
```
#confusion matrix
cm=confusion_matrix(y_test,ypred4)
print(cm)
```
```
[[25 10]
 [ 5 36]]
```

In [157…
```
#printing metrics
print(classification_report(y_test,ypred4))
```
```
              precision    recall  f1-score   support

           0       0.83      0.71      0.77        35
           1       0.78      0.88      0.83        41

    accuracy                           0.80        76
   macro avg       0.81      0.80      0.80        76
weighted avg       0.81      0.80      0.80        76
```

In [161…
```
#fitting the knc model
model5=knc.fit(X_train,y_train)
model5
```
```
C:\Users\sujoydutta\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:207: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```
Out[161]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [162…
```
#predict
ypred5= model5.predict(X_test)
ypred5
```
Out[162]:
```
array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 0])
```

In [163…
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred5)*100))
```
```
Accuracy of the model: 80.26315789473685%
```

In [164…
```
#confusion matrix
cm=confusion_matrix(y_test,ypred5)
print(cm)
```
```
[[26  9]
 [ 6 35]]
```

In [165…
```
#printing metrics
print(classification_report(y_test,ypred5))
```
```
              precision    recall  f1-score   support

           0       0.81      0.74      0.78        35
           1       0.80      0.85      0.82        41

    accuracy                           0.80        76
   macro avg       0.80      0.80      0.80        76
weighted avg       0.80      0.80      0.80        76
```

In [166…
```
#fitting the rfc model
model6=rfc.fit(X_train,y_train)
model6
```
```
C:\Users\sujoydutta\AppData\Local\Temp\ipykernel_9012\1772640938.py:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  model6=rfc.fit(X_train,y_train)
```
Out[166]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [167…
```
#predict
ypred6= model6.predict(X_test)
ypred6
```
Out[167]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1])
```

In [170…
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred6)*100))
```
```
Accuracy of the model: 75.0%
```

In [171…
```
#printing metrics
print(classification_report(y_test,ypred6))
```
```
              precision    recall  f1-score   support

           0       0.75      0.69      0.72        35
           1       0.75      0.80      0.78        41

    accuracy                           0.75        76
   macro avg       0.75      0.75      0.75        76
weighted avg       0.75      0.75      0.75        76
```

In [172…
```
#fitting the rfc model
model7=gbc.fit(X_train,y_train)
model7
```
```
C:\Users\sujoydutta\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:570: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```
Out[172]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [173…
```
#predict
ypred7= model7.predict(X_test)
ypred7
```
Out[173]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0])
```

In [174…
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred7)*100))
```
```
Accuracy of the model: 67.10526315789474%
```

In [175…
```
#printing metrics
print(classification_report(y_test,ypred7))
```
```
              precision    recall  f1-score   support

           0       0.64      0.66      0.65        35
           1       0.70      0.68      0.69        41

    accuracy                           0.67        76
   macro avg       0.67      0.67      0.67        76
weighted avg       0.67      0.67      0.67        76
```

Conclusion

KNC and SVC are the models having the most accuracy of 80% and they should be used.

Hyper-Parameter Tuning

Let us find the best hyperparamters for the models that will increase the accuracy.

In [256…
```
# Generate some random data for classification
X, y = make_classification(n_samples=303, n_features=13, random_state=42)
```

In [257…
```
#setting the hyper parameters for KNC
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
```

In [258…
```
# Define the evaluation metric
scorer = make_scorer(accuracy_score)
```

In [259…
```
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [260…
```
# Create the KNeighborsClassifier  model
pknc = KNeighborsClassifier()
```

In [261…
```
# Perform grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(pknc, param_grid, scoring=scorer, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
```
Out[261]:
```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

In [262…
```
# Print the best hyperparameters and the corresponding evaluation score
print("Best hyperparameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)

# Test the final model on a separate test set
y_pred = grid_search.predict(X_val)
print("Test accuracy: ", accuracy_score(y_val, y_pred))
```
```
Best hyperparameters:  {'n_neighbors': 7, 'p': 1, 'weights': 'uniform'}
Best score:  0.9093537414965986
Test accuracy:  0.8688524590163934
```

In [305…
```
# Generate some random data for classification for SVC
X, y = make_classification(n_samples=303, n_features=13, random_state=42)
```

In [306…
```
# Define the search space for hyperparameters for SVC
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'degree': [2, 3, 4],
    'gamma': ['scale', 'auto'],
    'class_weight': [None, 'balanced']
}
```

In [307…
```
# Define the evaluation metric
scorer = make_scorer(accuracy_score)
```

In [308…
```
# Create the SVC  model
psvc = SVC()
```

In [309…
```
# Perform grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(psvc, param_grid, scoring=scorer, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
```
Out[309]:
```
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

In [310…
```
# Print the best hyperparameters and the corresponding evaluation score
print("Best hyperparameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)

# Test the final model on a separate test set
y_pred = grid_search.predict(X_val)
print("Test accuracy: ", accuracy_score(y_val, y_pred))
```
```
Best hyperparameters:  {'C': 0.1, 'class_weight': None, 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
Best score:  0.9545918367346939
Test accuracy:  0.8852459016393442
```

Observation

We find SVC can be the model with the highest accuracy as it's score 95% compared to 91% for KNC model.

In [311…
```
#creating the final model
fm = SVC(C=0.1, class_weight=None,degree= 2, gamma= 'scale', kernel= 'linear')
fm
```
Out[311]:
```
▾ SVC
SVC(C=0.1, degree=2, kernel='linear')
```

In [312…
```
#fitting the model
fm.fit(X_train,y_train)
```
Out[312]:
```
▾ SVC
SVC(C=0.1, degree=2, kernel='linear')
```

In [313…
```
#predict
ypredf= fm.predict(X_test)
ypredf
```
Out[313]:
```
array([1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1])
```

In [314…
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypredf)*100))
```
```
Accuracy of the model: 32.89473684210527%
```

In [315…
```
#printing metrics
print(classification_report(y_test,ypredf))
```
```
              precision    recall  f1-score   support

           0       0.29      0.31      0.30        35
           1       0.37      0.34      0.35        41

    accuracy                           0.33        76
   macro avg       0.33      0.33      0.33        76
weighted avg       0.33      0.33      0.33        76
```

Final statement

Hyperparameter tuning didnt improve the accuracy but rather reduced the accuracy score which is rather dissapointing.