healthcare_capstone

1 1. Preliminary analysis:

1. Perform preliminary data inspection and report the findings as to the structure of the data, missing values, duplicates, etc.
2. Based on the findings from the previous question remove duplicates (if any) , treat missing values using an appropriate strategy.

[1]:
```python
# import the library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
# load the data
df=pd.read_excel("data.xlsx")
```

[3]:
```python
df.head()
```

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

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

[4]:
```python
df.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
```

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

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

[6]:
```
# check missing values
df.isnull().sum()
```

[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
```

[7]:
```
# find duplicate values
duplicate=df[df.duplicated()]
```

[8]:
```
duplicate
```

[8] :
```
     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
```

[9] :
```
df.drop_duplicates(inplace=True)
df.shape
```

[9]:
```
(302, 14)
```

Prepare an informative report about the data explaining the distribution of the disease and the related factors. You could use the below approach to achieve the objective

2.1 Get a preliminary statistical summary of the data. Explore the measures of central tendencies and the spread of the data overall

[10] :
```
df.describe()
```

[10]:

	age	sex	cp	trestbps	chol	fbs
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000

	restecg	thalach	exang	oldpeak	slope	ca
count	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	0.526490	149.569536	0.327815	1.043046	1.397351	0.718543
std	0.526027	22.903527	0.470196	1.161452	0.616274	1.006748
min	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	133.250000	0.000000	0.000000	1.000000	0.000000
50%	1.000000	152.500000	0.000000	0.800000	1.000000	0.000000
75%	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

	thal	target
count	302.000000	302.000000
mean	2.314570	0.543046
std	0.613026	0.498970
min	0.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	1.000000
max	3.000000	1.000000

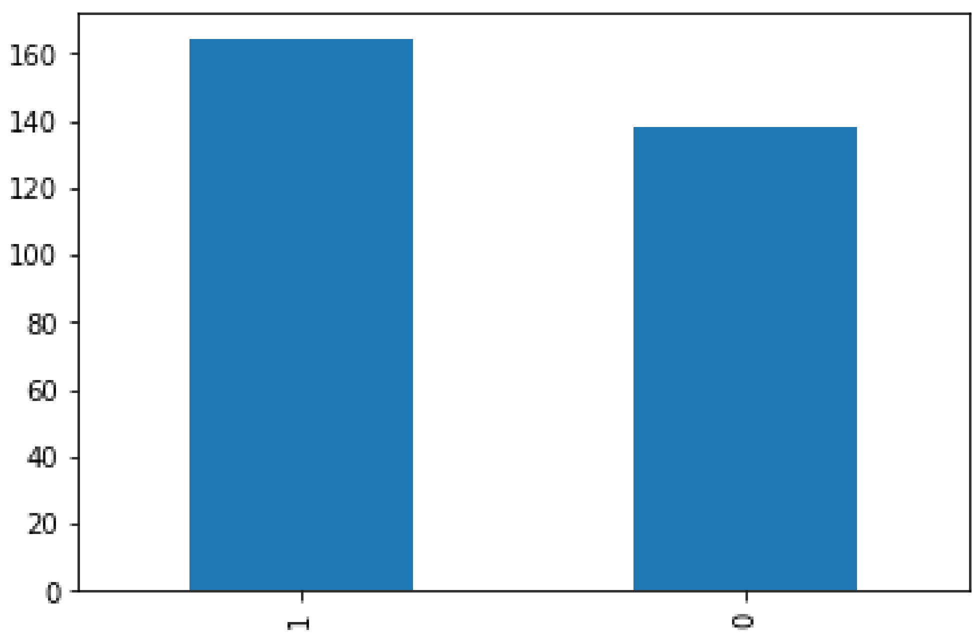
2.2 Identify the data variables which might be categorical in nature. Describe and explore these variables using appropriate tools e.g. count plot

[11] :
```
# check distribution of target variables
df['target'].value_counts()
```

[11]:
```
1    164
0    138
Name: target, dtype: int64
```

[12] :
```
df['target'].value_counts().plot(kind='bar')
```

[12] : <AxesSubplot: >



[13] :
```
df.columns
```

[13] :
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

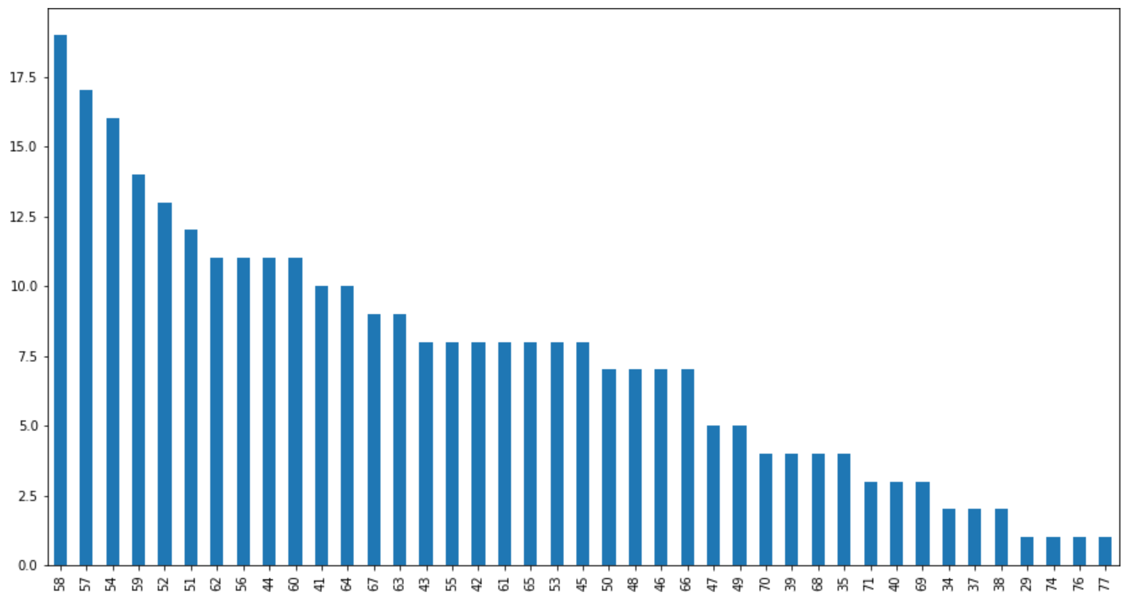
[14] :
```
df['age'].value_counts()
```

[14]:
```
58    19
57    17
54    16
59    14
```

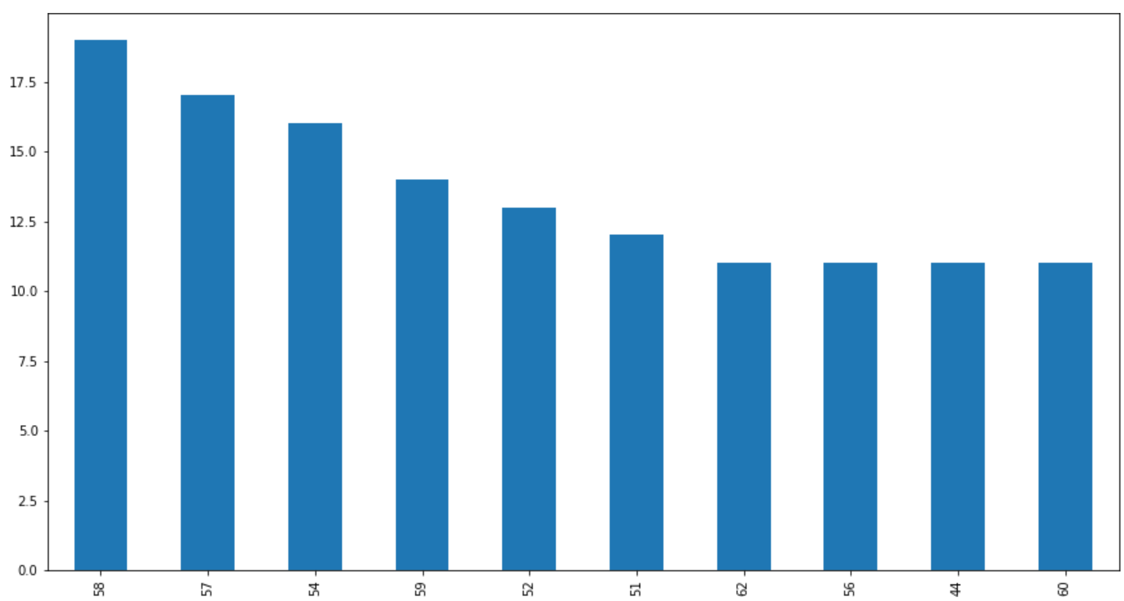
```
52    13
51    12
62    11
56    11
44    11
60    11
41    10
64    10
67     9
63     9
43     8
55     8
42     8
61     8
65     8
53     8
45     8
50     7
48     7
46     7
66     7
47     5
49     5
70     4
39     4
68     4
35     4
71     3
40     3
69     3
34     2
37     2
38     2
29     1
74     1
76     1
77     1
Name: age, dtype: int64
```

[15] :

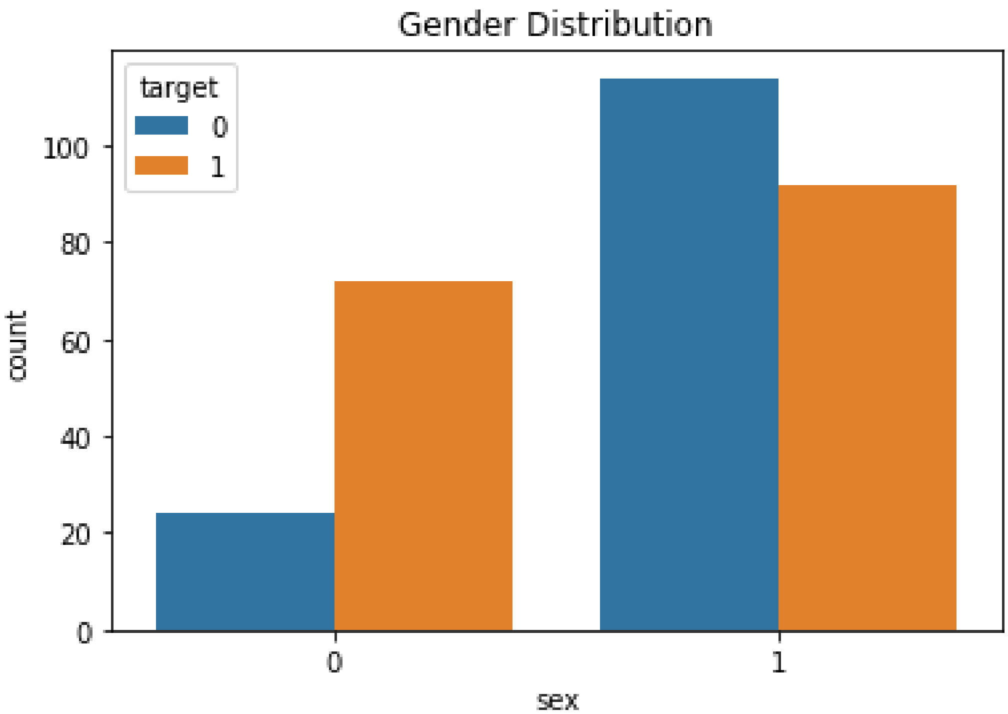
```
plt.figure(figsize=(15,8))
df['age'].value_counts().plot(kind='bar')
plt.show()
```



[16] :
```python
# Analyze distribution of age in range of 10
plt.figure(figsize=(15,8))
df['age'].value_counts()[:10].plot(kind='bar')
plt.show()
```



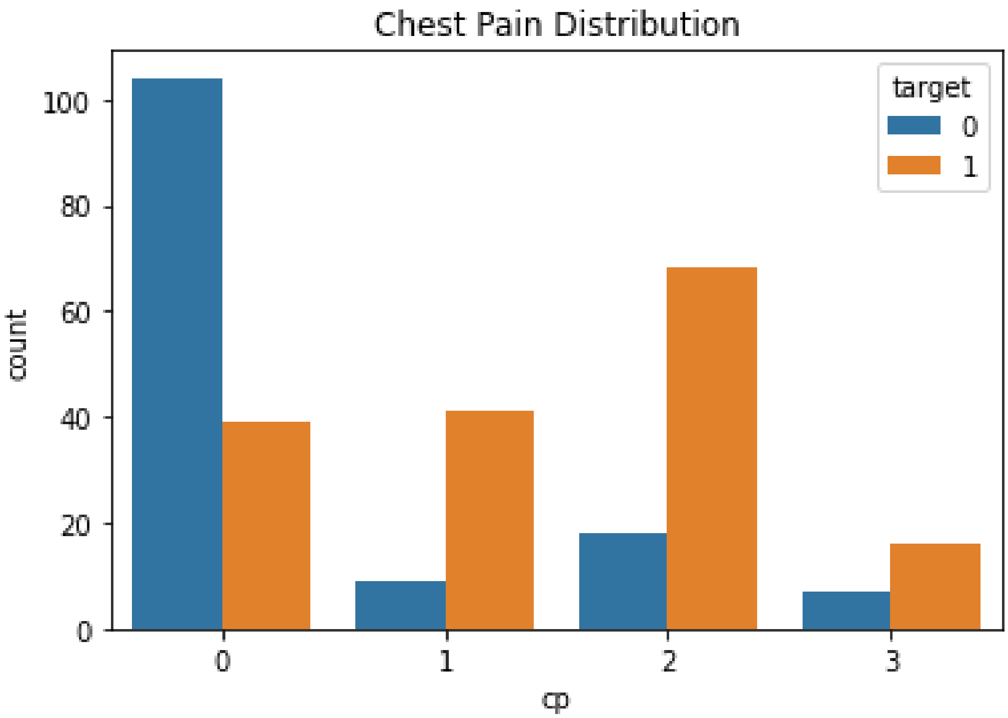
[17] :
```python
sns.countplot(x=df['sex'],hue='target',data=df)
plt.title('Gender Distribution')
plt.show()
```



[18] :
```
df['cp'].value_counts()
```

[18]:
```
0    143
2     86
1     50
3     23
Name: cp, dtype: int64
```

[19] :
```
sns.countplot(x=df['cp'],hue='target',data=df)
plt.title('Chest Pain Distribution')
plt.show()
```

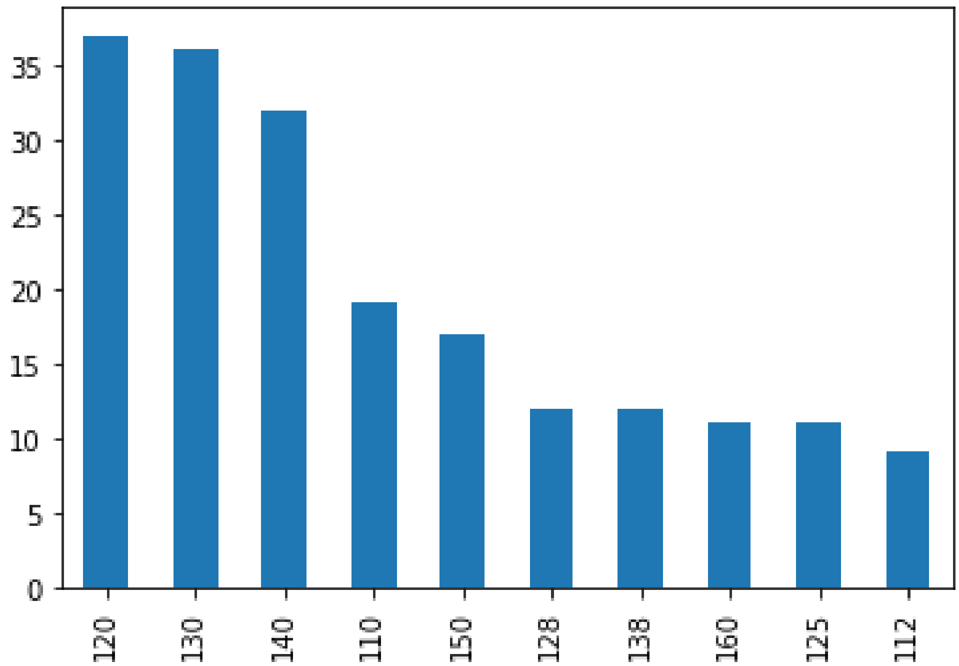


[20] :
```
df['trestbps'].value_counts()[:10]
```

[20]:
```
120    37
130    36
140    32
110    19
150    17
128    12
138    12
160    11
125    11
112     9
Name: trestbps, dtype: int64
```

[21] :
```
df['trestbps'].value_counts()[:10].plot(kind='bar')
```

[21] : <AxesSubplot: >

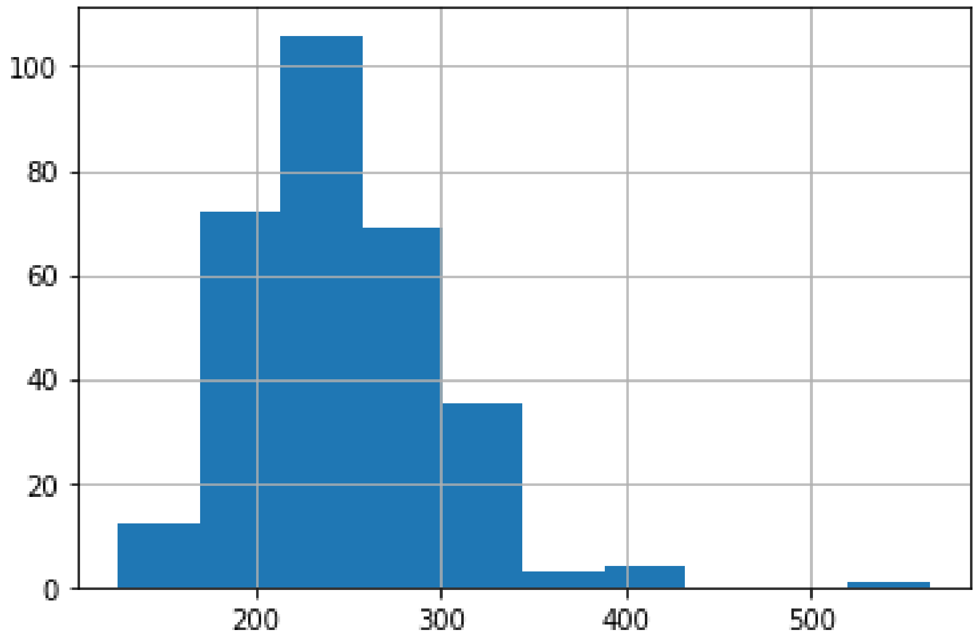


[22] :
```
df['chol'].value_counts()
```

[22]:
```
204    6
197    6
234    6
212    5
254    5
      ..
284    1
224    1
167    1
276    1
131    1
Name: chol, Length: 152, dtype: int64
```

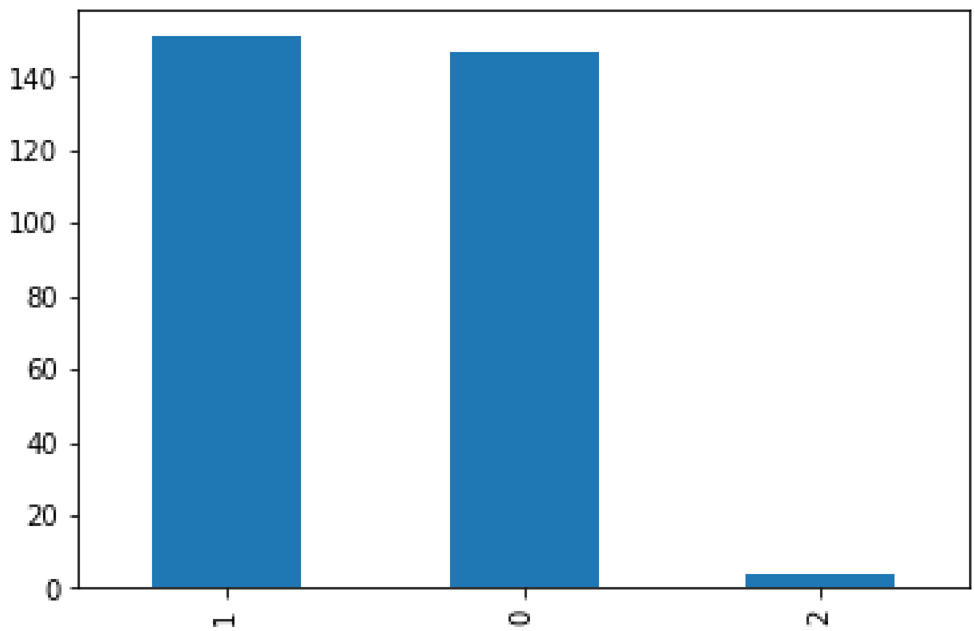
[24]:
```
df['chol'].hist()
```

[24] : <AxesSubplot: >



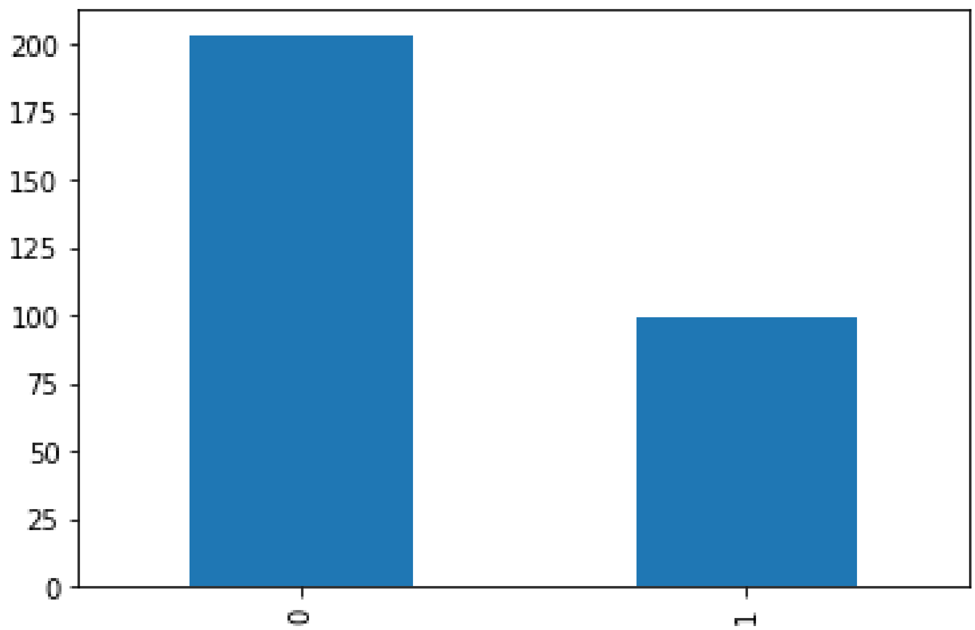
[25] :
```
df['restecg'].value_counts().plot(kind='bar')
```

[25] : <AxesSubplot: >



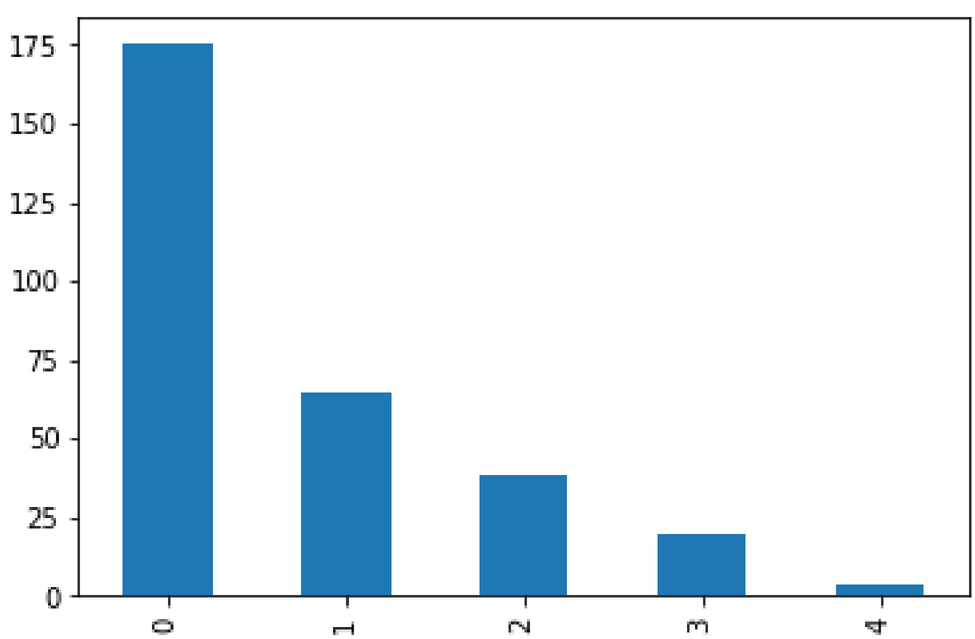
[26] :
```
df['exang'].value_counts().plot(kind='bar')
```

[26] : <AxesSubplot: >



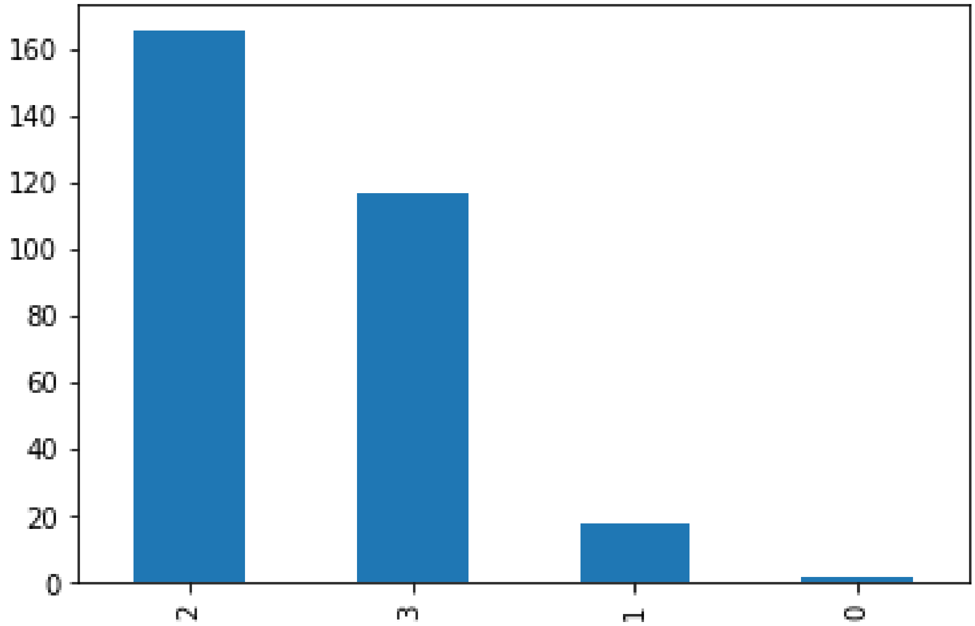
[27] :
```
df['ca'].value_counts().plot(kind='bar')
```

[27] : <AxesSubplot: >



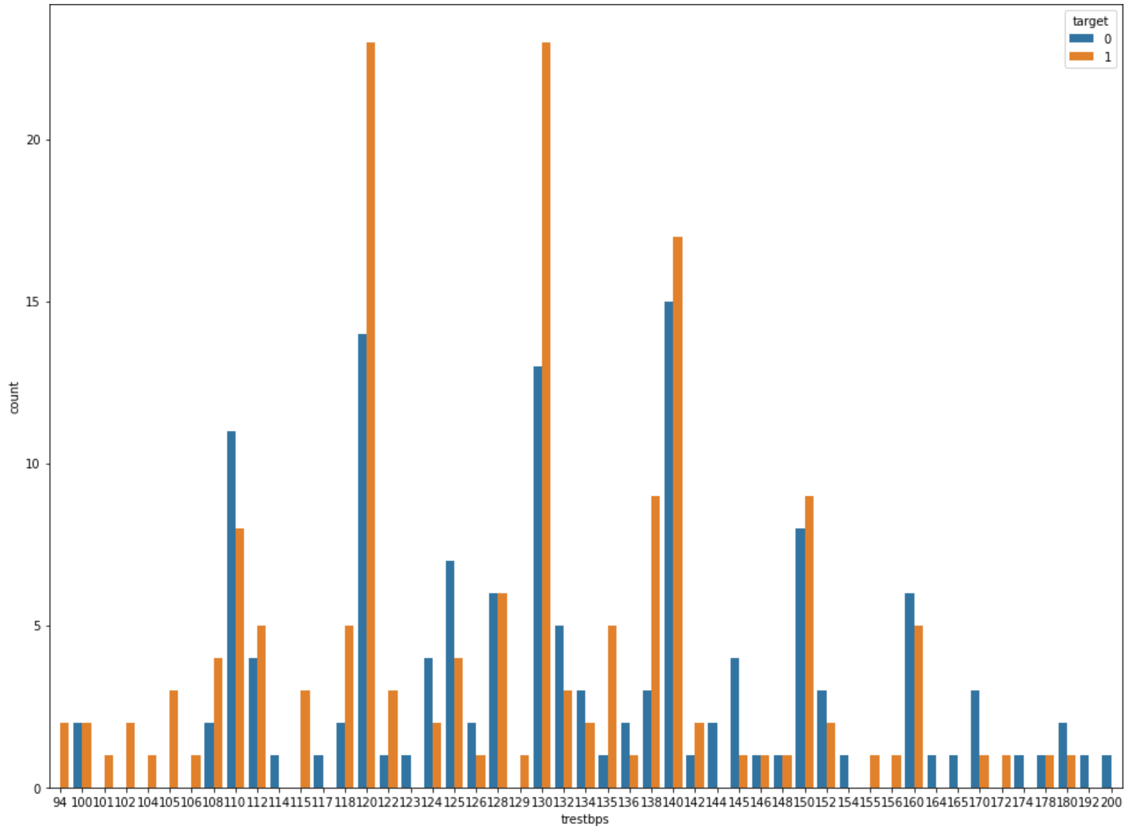
[28] :
```
df['thal'].value_counts().plot(kind='bar')
```

[28] : <AxesSubplot: >



2.4 Can we detect a heart attack based on anomalies in the Resting Blood Pressure of the patient?

[29] :
```
# bivariate analysis
plt.figure(figsize=(16,12))
sns.countplot(x=df['trestbps'],hue='target',data=df)
plt.show()
```



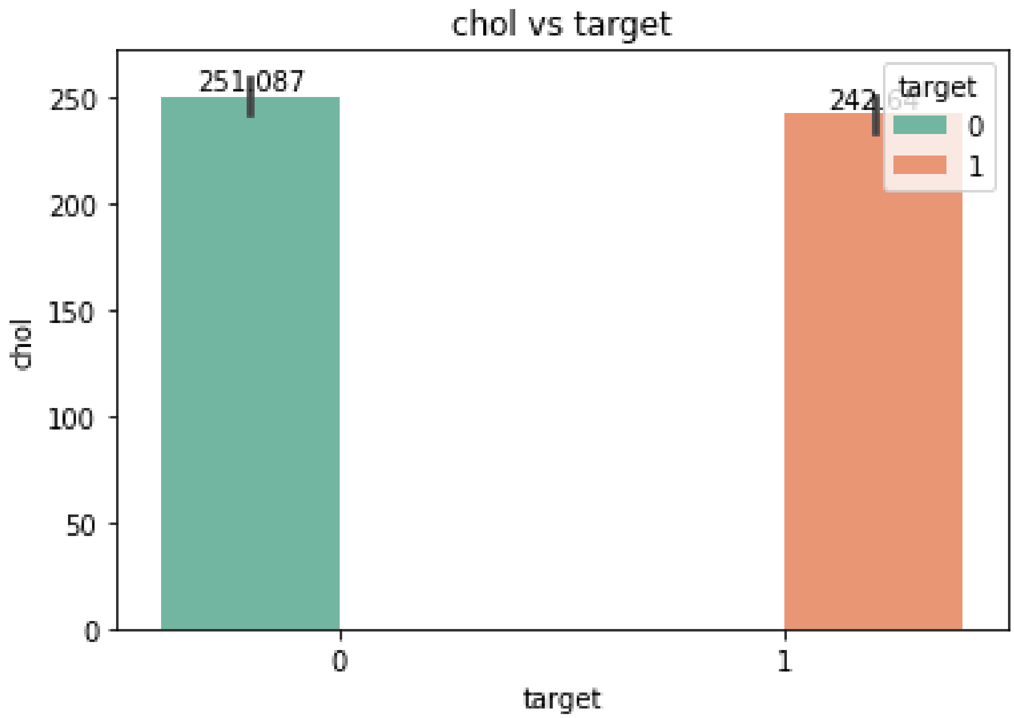
[30] :
```
df['chol']
```

[30]:
```
0      233
1      250
2      204
3      236
4      354
      ...
298    241
299    264
300    193
301    131
302    236
Name: chol, Length: 302, dtype: int64
```

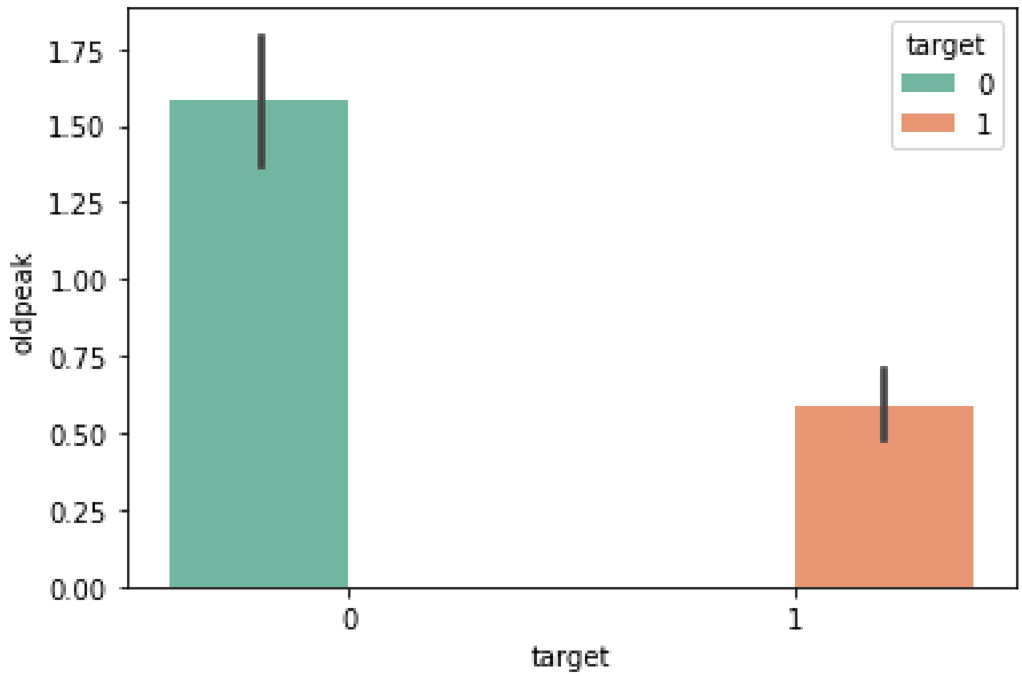
Describe the relationship between Cholesterol levels and our target variable

[31] :
```
ax=sns.barplot(x=df['target'],y=df['chol'],hue='target',data=df,palette='Set2')
for label in ax.containers:
    ax.bar_label(label)
plt.title('chol vs target')
```

```
plt.show()
```



[32]:
```
ax=sns.barplot(x=df['target'],y=df['oldpeak'],hue='target',data=df,palette='Set2')
plt.show()
```

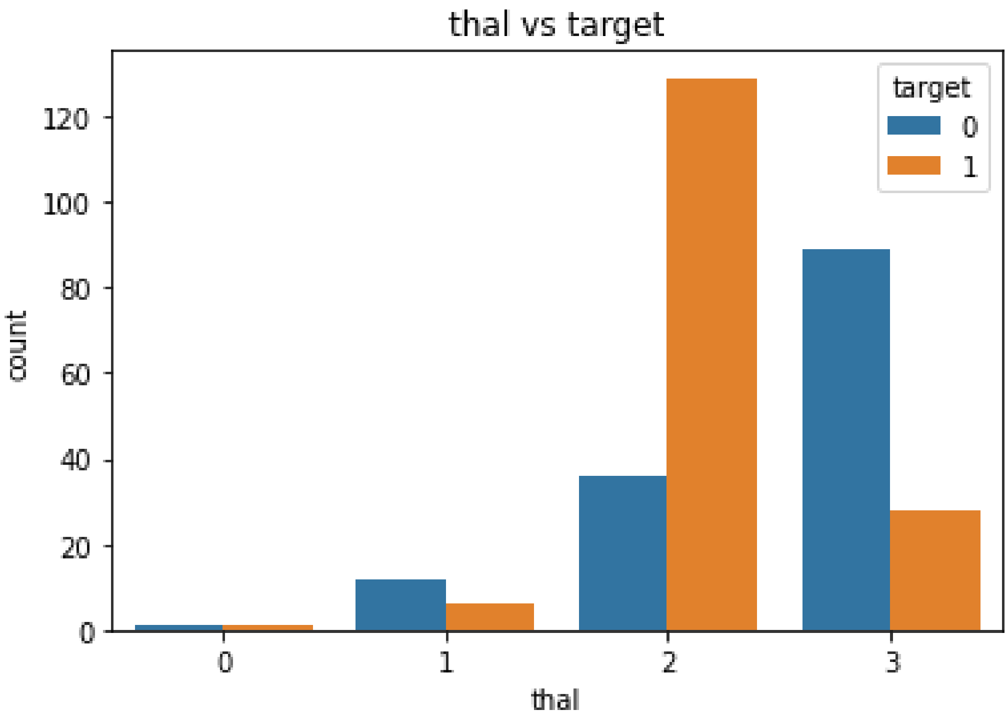


Is thalassemia a major cause of CVD

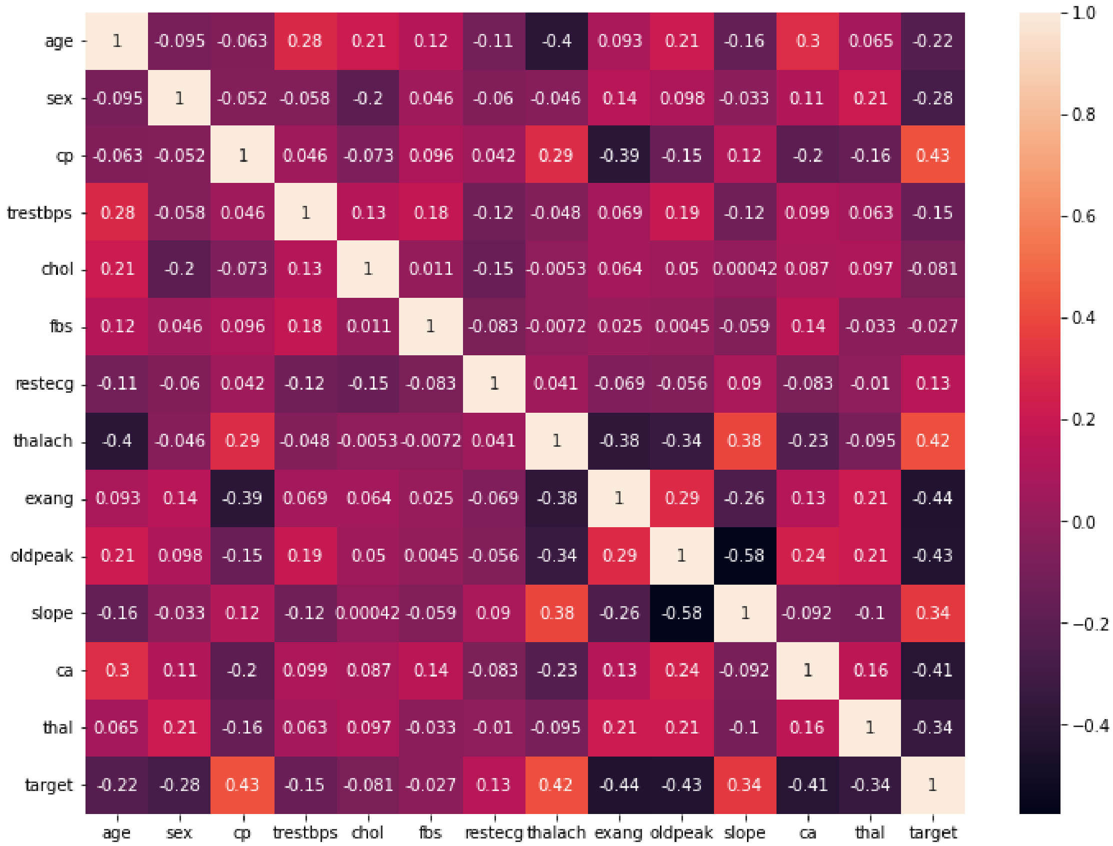
[33] :
```
df['thal'].value_counts()
```

[33]:
```
2    165
3    117
1     18
0      2
Name: thal, dtype: int64
```

[34] :
```
sns.countplot(x=df['thal'],hue='target',data=df)
plt.title('thal vs target')
plt.show()
```

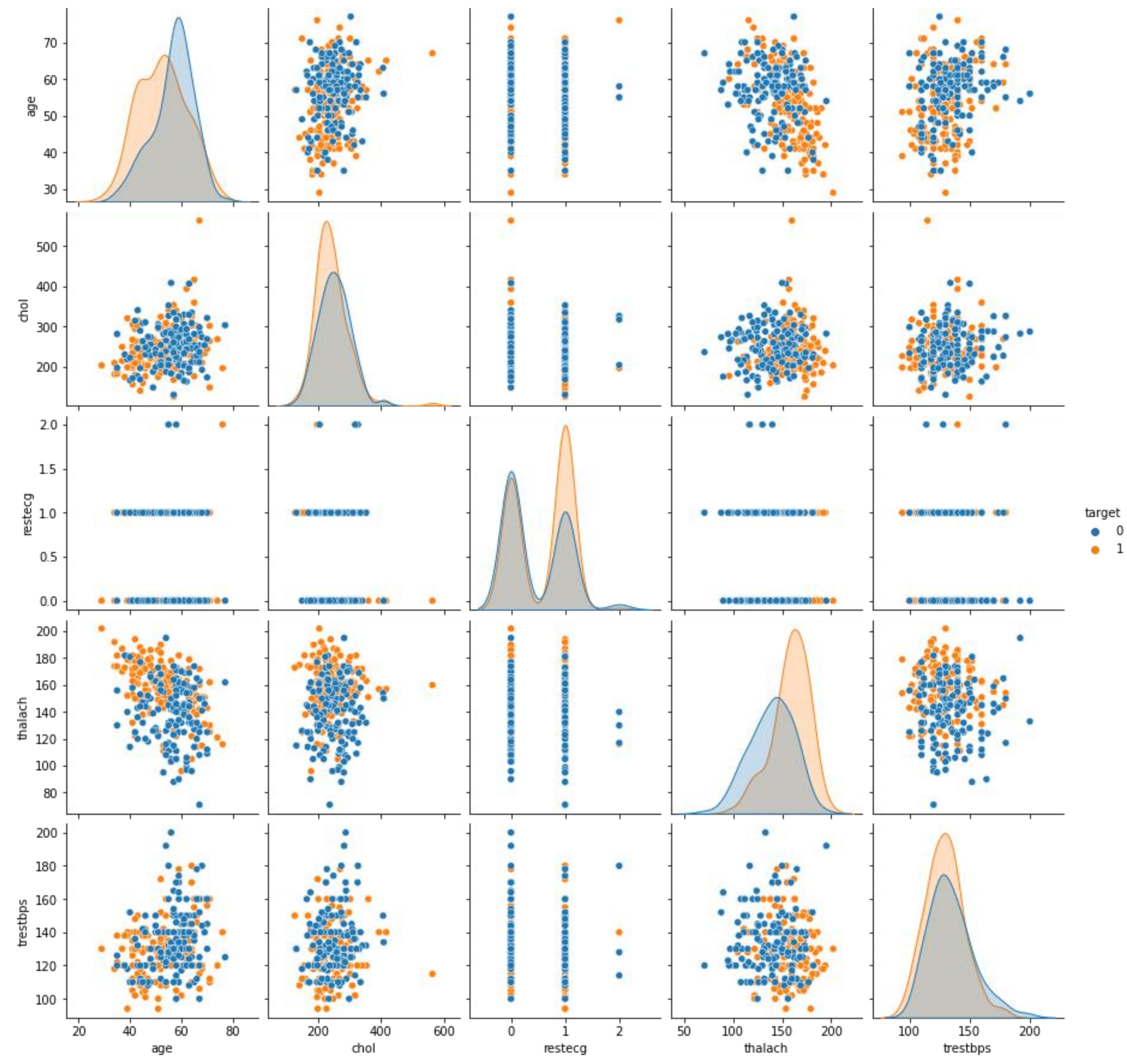


[35] :
```
# multivariate analysis
plt.figure(figsize=(12,9))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



[41]:
```
cont_feature=['age','chol','restecg','thalach','trestbps']
plt.figure(figsize=(12,8))
sns.pairplot(df[cont_feature+['target']],hue='target')
plt.show()
```

<Figure size 864x576 with 0 Axes>



3. Build a baseline model to predict using a Logistic Regression and explore the results

[42]:
```
# extract dep and indep var
X=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

[43]:
```
X
```

[43]:
```
array([[63.,  1.,  3., ...,  0.,  0.,  1.],
       [37.,  1.,  2., ...,  0.,  0.,  2.],
       [41.,  0.,  1., ...,  2.,  0.,  2.],
       ...,
       [68.,  1.,  0., ...,  1.,  2.,  3.],
       [57.,  1.,  0., ...,  1.,  1.,  3.],
       [57.,  0.,  1., ...,  1.,  1.,  2.]])
```

[44]:
```
y
```

[44]:
```
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, 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])
```

[45]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 3,random_state=42)
```

[46]:
```
X_train
```

[46]:
```
array([[39.,  0.,  2., ...,  2.,  0.,  2.],
       [29.,  1.,  1., ...,  2.,  0.,  2.],
       [50.,  0.,  2., ...,  1.,  0.,  2.],
       ...,
       [69.,  1.,  3., ...,  1.,  1.,  2.],
       [61.,  1.,  3., ...,  1.,  2.,  2.],
       [63.,  0.,  1., ...,  2.,  2.,  2.]])
```

[47]:
```
X_train.shape
```

[47]:
```
(211, 13)
```

[48]:
```
X_test.shape
```

[48]:
```
(91, 13)
```

[49]:
```
# scale the data
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

[51]:
```
X_train=sc.fit_transform(X_train)
```

[52]:
```
X_test=sc.transform(X_test)
```

[53]:
```
X_train
```

[53]:
```
array([[-1.69312171, -1.34660066,  0.93980295, ...,  0.93912285,
        -0.67862717, -0.53004604],
       [-2.80086956,  0.74261066, -0.01816044, ...,  0.93912285,
        -0.67862717, -0.53004604],
       [-0.47459908, -1.34660066,  0.93980295, ..., -0.67189277,
        -0.67862717, -0.53004604],
       ...,
       [ 1.63012184,  0.74261066,  1.89776634, ..., -0.67189277,
         0.37424292, -0.53004604],
       [ 0.74392356,  0.74261066,  1.89776634, ..., -0.67189277,
         1.42711302, -0.53004604],
       [ 0.96547313, -1.34660066, -0.01816044, ...,  0.93912285,
         1.42711302, -0.53004604]])
```

[54]:
```
# Apply Logistic Regression on Data
from sklearn.linear_model import LogisticRegression
log_reg=LogisticRegression()
```

[55]:
```
log_reg.fit(X_train,y_train)
```

[55]:
```
LogisticRegression()
```

[56]:
```
y_pred=log_reg.predict(X_test)
```

[57]:
```
y_pred
```

[57]:
```
array([0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 1])
```

[58]:
```
from sklearn.metrics import accuracy_score, classification_report
```

[60]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.8131868131868132
```

[61]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80        42
           1       0.83      0.82      0.82        49

    accuracy                           0.81        91
```

macro avg	0.81	0.81	0.81	91
weighted avg	0.81	0.81	0.81	91