HealthCare

August 24, 2023

0.1 Problem statement:

Cardiovascular diseases are the leading cause of death globally. It is therefore necessary to identify the causes and develop a system to predict heart attacks in an effective manner. The data below has the information about the factors that might have an impact on cardiovascular health.

[1]:
```
# import some usefull files.
import pandas as pd
import scipy as sc
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

0.2 Task to be performed:

1. Preliminary analysis:

a. Perform preliminary data inspection and report the findings on the structure of the data, missing values, duplicates, etc.

b. Based on these findings, remove duplicates (if any) and treat missing values using an appropriate strategy

[2]:
```
health =pd.read_excel('1645792390_cep1_dataset.xlsx')
health.head()
```

[2]:
```
   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
```

[3]:
```
health.shape
```

[3]:
```
(303, 14)
```

[4]:
```
health.columns
```

[4]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[5]:
```
health.describe()
```

[5]:
```
              age         sex          cp    trestbps        chol         fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    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  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
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
```

[6]:

```
health.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
```

[7]:

```
# check the null values
health.isna().sum()
```

[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
```

we can see that there is no null values in the health data, lets proceed further.

[8]:

```
#check for the datatypes.
health.dtypes
```

[8]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

[9]:
```
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
```

[10]:
```
# check for the dupicacy and unique value.
health.duplicated().sum()
```

[10]:
```
1
```

[11]:
```
health.nunique().sort_values(ascending=False)
```

[11]:
```
chol        152
thalach      91
trestbps     49
age          41
oldpeak      40
ca            5
thal          4
cp            4
slope         3
restecg       3
target        2
exang         2
fbs           2
sex           2
dtype: int64
```

we can see that 1 duplicate vlaues.

lets find the values and drop it.

[12]:
```
# check the duplicacy in rows
duplicate_rows =health.duplicated()
duplicate_rows.value_counts()
```

[12]:
```
False    302
True       1
dtype: int64
```

[13]:
```
print(health[duplicate_rows])
```

```
     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
```

[14]:
```
#lets drop the duplicate rows.
health.shape
```

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

[15]:
```
health= health.drop_duplicates()
health.shape
```

[15]:
```
(302, 14)
```

[16]:
```
# verifiy the duplicated rows again
health.duplicated().sum()
```

[16]:
```
0
```

1 2. Prepare a report about the data explaining the distribution of the disease and the related factors using the steps listed below:

2 a. Get a preliminary statistical summary of the data and explore the measures of central tendencies and spread of the data

[17]:
```
health.columns
```

[17]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[18]:
```
health.describe()
```

[18]:
```
             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
```

3 b. Identify the data variables which are categorical and describe and explore these variables using the appropriate tools, such as count plot

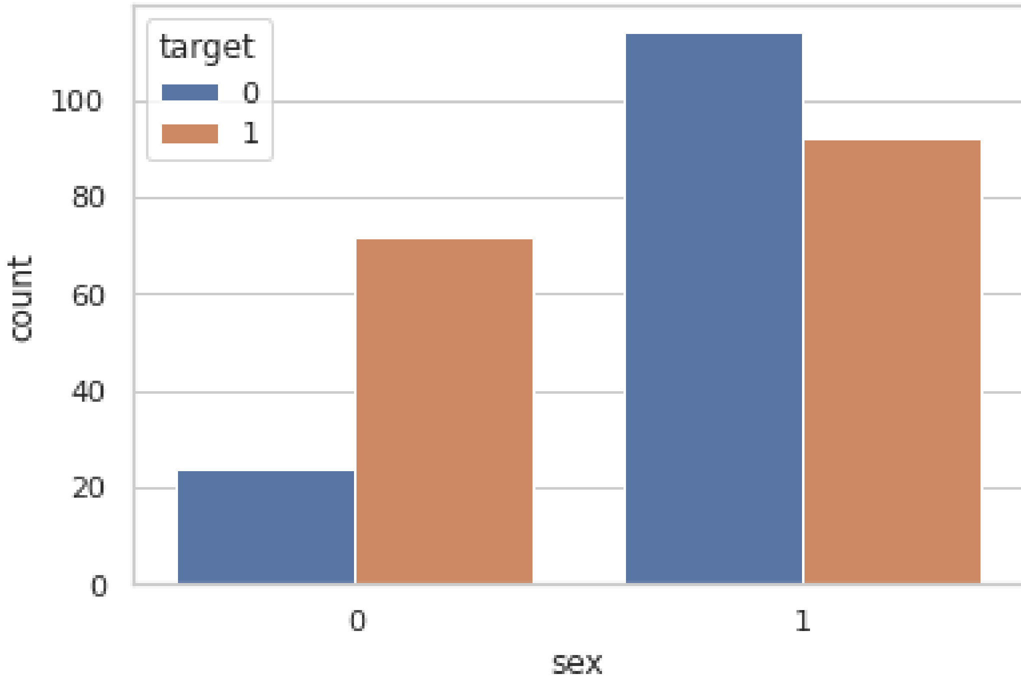
[19]:
```
cat_column= health.select_dtypes(include="object").columns
cat_column
```

[19]:
```
Index([], dtype='object')
```

we can see that there is no categorical variable.lets make the count.

[20]:
```
# lets make the count plot
sns.set_theme(style="whitegrid")
sns.countplot(data=health, x="sex", hue="target")
```

[20]: <AxesSubplot:xlabel='sex', ylabel='count'>



3.1 c. Study the occurrence of CVD across the Age category

[21]:
```
health.columns
```

[21]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[22]:
```
# lets rename the dataset column
health =health.rename(columns={'cp':'Chest_Pain','trstbps':
 ↪'Resting_Blood_Pressure','fbs':'Fasting_Blood_Sugar',
                                'restecg':
 ↪'Electrocardiographic_result','thalach':'Max_Heart_Rate','exang':
 ↪'Exercise_Induced_Angina'
                                ,'ca':'Number_Major_Vessel'})
health.columns
```

[22]:
```
Index(['age', 'sex', 'Chest_Pain', 'trestbps', 'chol', 'Fasting_Blood_Sugar',
       'Electrocardiographic_result', 'Max_Heart_Rate',
       'Exercise_Induced_Angina', 'oldpeak', 'slope', 'Number_Major_Vessel',
       'thal', 'target'],
```

```
      dtype='object')
```

data here we have :

`cp| Chest pain type.`

`trestbps    Resting blood pressure (in mm Hg on admission to the hospital)`

`chol    Serum cholesterol in mg/dl`

`fbs Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)`

`restecg Resting electrocardiographic results`

`thalach Maximum heart rate achieved`

`exang   Exercise induced angina (1 = yes; 0 = no)`

`oldpeak ST depression induced by exercise relative to rest`

`slope   Slope of the peak exercise ST segment`

`ca  Number of major vessels (0-3) colored by fluoroscopy`

`thal    3 = normal; 6 = fixed defect; 7 = reversible defect`

`Target  1 or 0`

[23]:
```
# study the CVD across the age :
disease= health
disease.head()
```

[23]:
```
   age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
0   63    1           3       145   233                    1
1   37    1           2       130   250                    0
2   41    0           1       130   204                    0
3   56    1           1       120   236                    0
4   57    0           0       120   354                    0

   Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
0                            0             150                        0
1                            1             187                        0
2                            0             172                        0
3                            1             178                        0
4                            1             163                        1

   oldpeak  slope  Number_Major_Vessel  thal  target
0      2.3      0                    0     1       1
```

```
1      3.5      0                      0     2       1
2      1.4      2                      0     2       1
3      0.8      2                      0     2       1
4      0.6      2                      0     2       1
```

[24]:
```
 col=['target','oldpeak','thal']
disease =disease.drop(col, axis=1)
```

[25]:
```
disease.head()
```

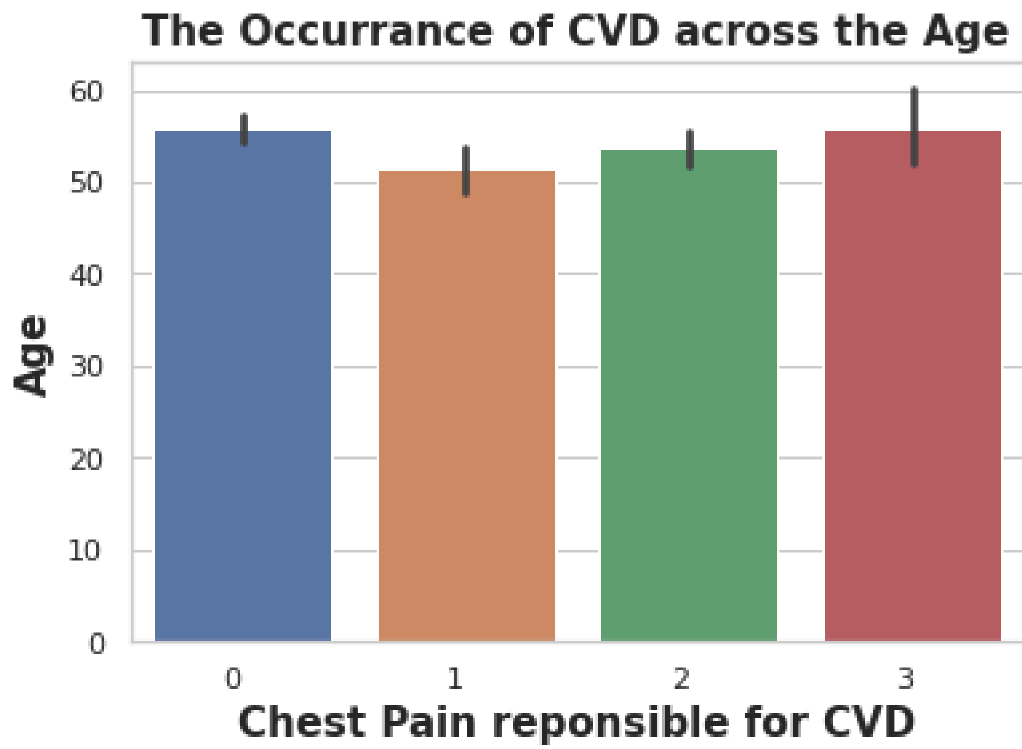
[25]:
```
   age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
0   63    1           3       145   233                    1
1   37    1           2       130   250                    0
2   41    0           1       130   204                    0
3   56    1           1       120   236                    0
4   57    0           0       120   354                    0

   Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
0                            0             150                        0
1                            1             187                        0
2                            0             172                        0
3                            1             178                        0
4                            1             163                        1

   slope  Number_Major_Vessel
0      0                    0
1      0                    0
2      2                    0
3      2                    0
4      2                    0
```

[26]:
```
disease.columns
```

[26]:
```
Index(['age', 'sex', 'Chest_Pain', 'trestbps', 'chol', 'Fasting_Blood_Sugar',
       'Electrocardiographic_result', 'Max_Heart_Rate',
       'Exercise_Induced_Angina', 'slope', 'Number_Major_Vessel'],
      dtype='object')
```

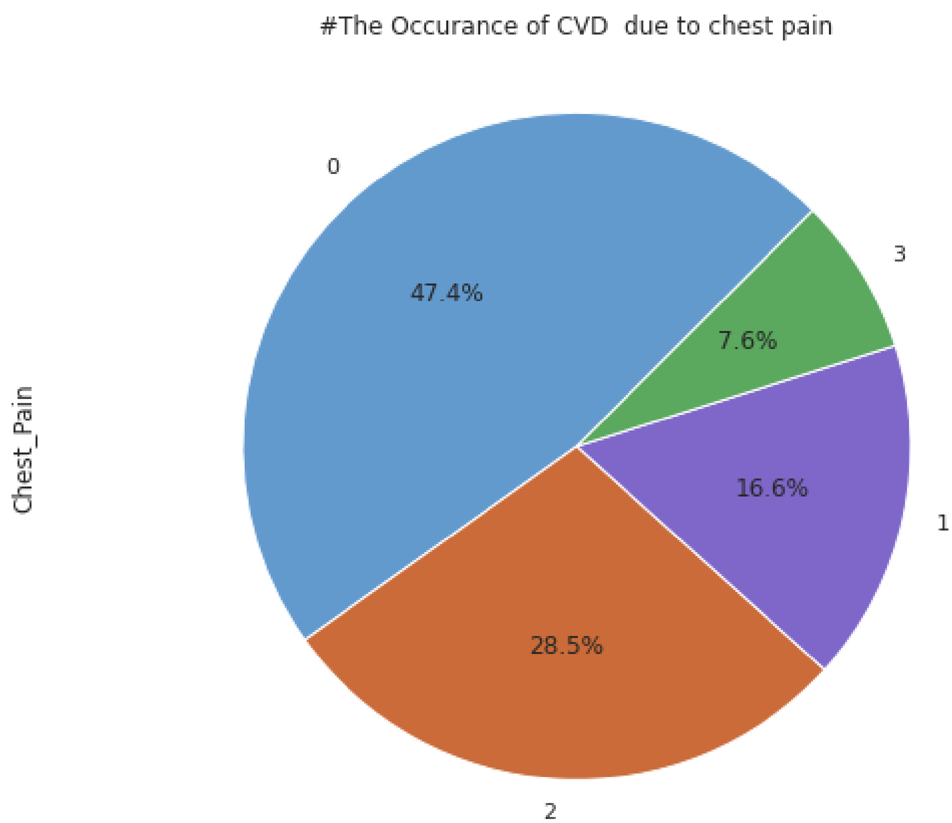
[149]:
```
Chest_pain = sns.barplot(x=disease.Chest_Pain, y=disease.age )
Chest_pain.set_xticklabels(Chest_pain.get_xticklabels(), rotation=0, ha="right")
Chest_pain.set_xlabel('Chest Pain reponsible for CVD', weight='bold', size=15)
Chest_pain.set_ylabel('Age', weight='bold', size=15)
Chest_pain .set_title('The Occurrance of CVD across the Age', weight='bold',␣
 ↪size=15)
```

[149]:
```
Text(0.5, 1.0, 'The Occurrance of CVD across the Age')
```



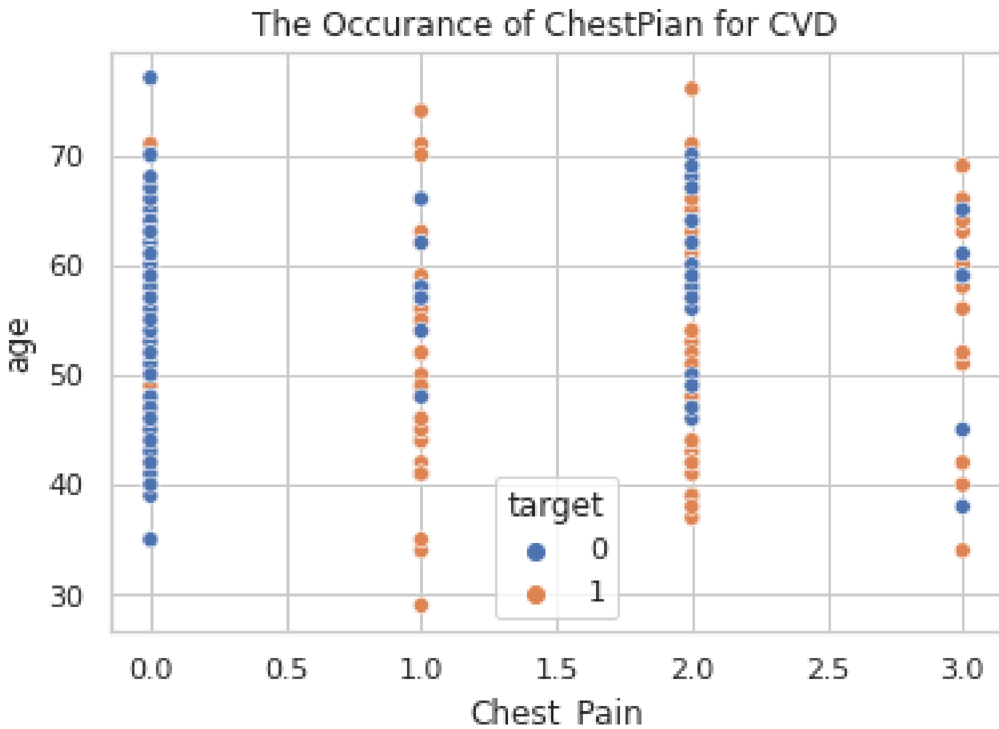
Interpretation: the Chest pain generally occurs around the age of 50 and above that critically responsible for CVD.

[28]:
```
#  lets check fisrt the values counts of chest pain through out the data in␣
↪percentage, before finding the occurance across the age.

colors = ['#639ace','#ca6b39','#7f67ca','#5ba85f','#c360aa','#a7993f','#cc566a']
health['Chest_Pain'].value_counts().plot(kind='pie',autopct='%1.1f%%',
                         startangle=45, shadow=False, colors = colors,
                        figsize = (8,6))
plt.axis('equal')
plt.title('#The Occurance of CVD  due to chest pain\n')
plt.tight_layout()
plt.show()
```




```
[29]: # lets find out the Chest Pain responsible in different age group for CVD.
      sns.scatterplot(x=health['Chest_Pain'], y=health['age'], hue=health['target']).
       ↪set(title='The Occurance of ChestPian for CVD  ')
```

```
[29]: [Text(0.5, 1.0, 'The Occurance of ChestPian for CVD  ')]
```

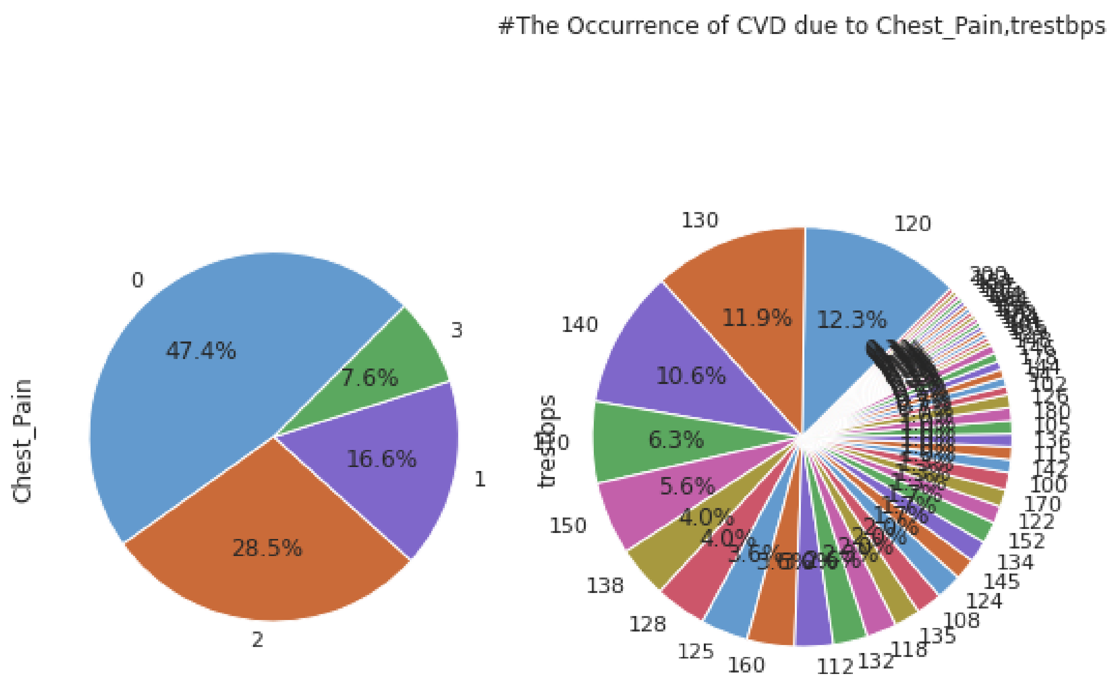


[30]:
```
disease.Chest_Pain.value_counts()
```

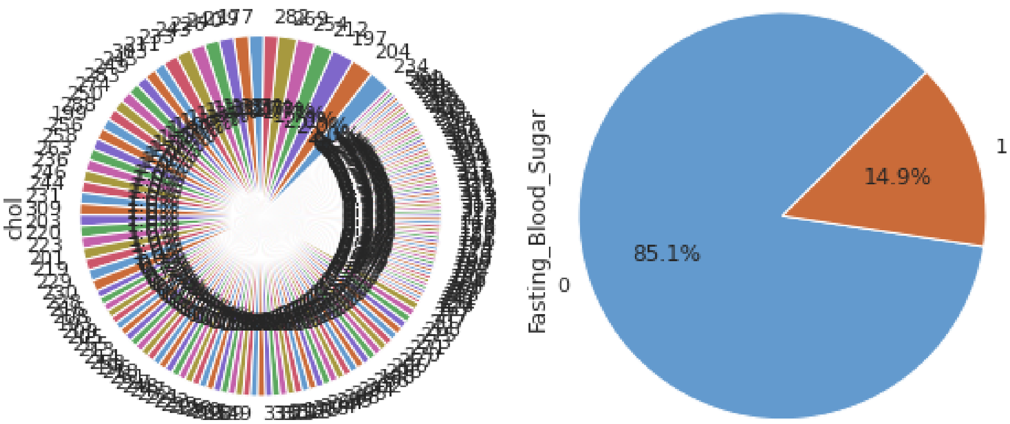
[30]:
```
0    143
2     86
1     50
3     23
Name: Chest_Pain, dtype: int64
```

[31]:
```
num_cols = ['Chest_Pain', 'trestbps', 'chol', 'Fasting_Blood_Sugar',
            'Electrocardiographic_result', 'Max_Heart_Rate',
            'Exercise_Induced_Angina', 'oldpeak', 'slope',␣
 ↪'Number_Major_Vessel']
colors = ['#639ace', '#ca6b39', '#7f67ca', '#5ba85f', '#c360aa', '#a7993f',␣
 ↪'#cc566a']

for i in range(0, len(num_cols), 2):
    plt.figure(figsize=(10, 4))
    plt.subplot(121)

    health[num_cols[i]].value_counts().plot(kind='pie', autopct='%1.1f%%',
                                            startangle=45, shadow=False,␣
 ↪colors=colors,
                                            figsize=(8, 6))
```

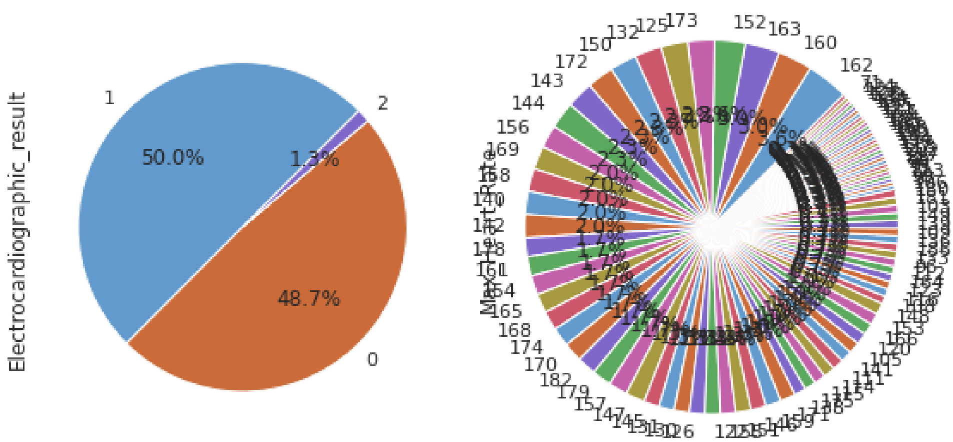
```
    plt.subplot(122)

    health[num_cols[i + 1]].value_counts().plot(kind='pie', autopct='%1.1f%%',
                                                 startangle=45, shadow=False, colors=colors,
                                                 figsize=(8, 6))
    plt.axis('equal')
    plt.title('#The Occurrence of CVD due to {},{}\n'.format(num_cols[i],num_cols[i + 1]))
    plt.tight_layout()
    plt.show()
```

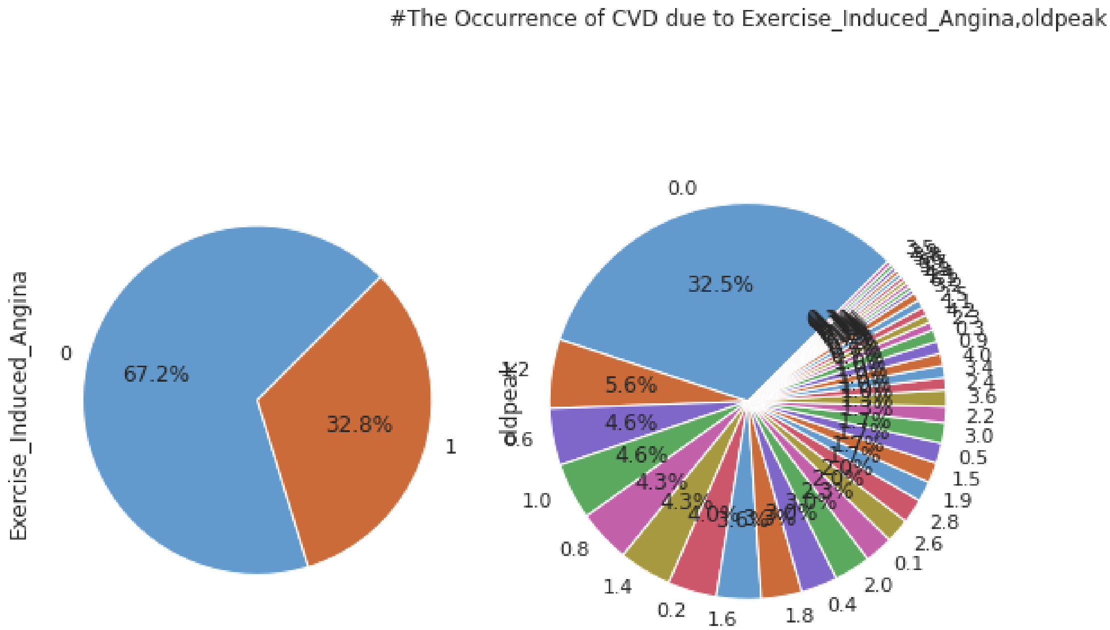


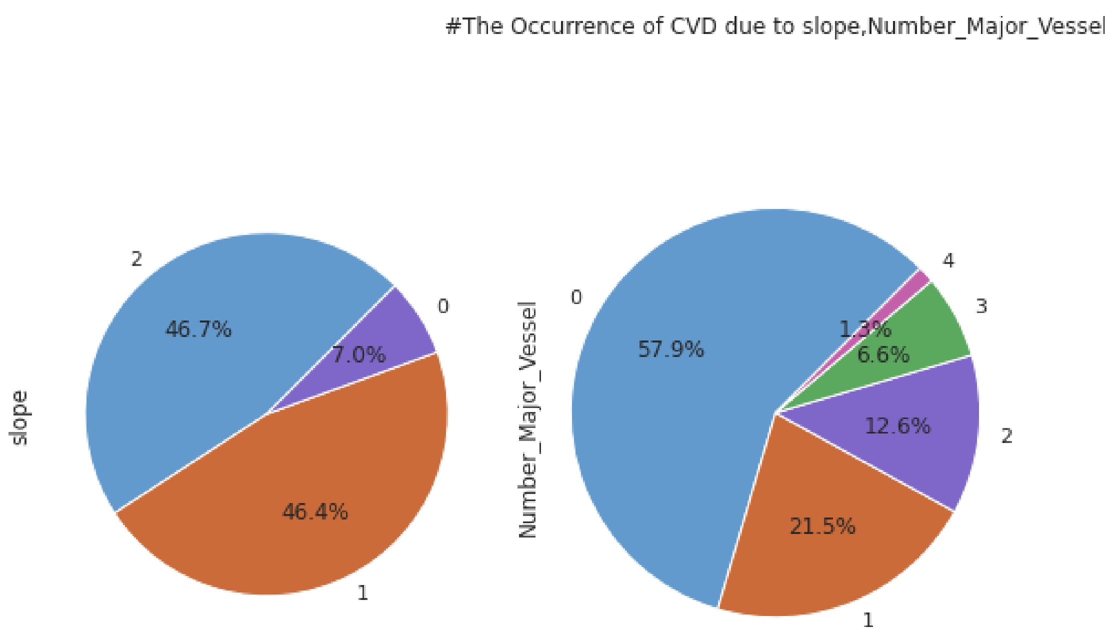
#The Occurrence of CVD due to chol,Fasting_Blood_Sugar



#The Occurrence of CVD due to Electrocardiographic_result,Max_Heart_Rate




#The Occurrence of CVD due to Exercise_Induced_Angina,oldpeak
Exercise_Induced_Angina
0
1
67.2%
32.8%
oldpeak
0.0
32.5%
5.6%
4.6%
4.6%
4.3%
4.3%
0.6
1.0
0.8
1.4
0.2
1.6
1.8
0.4
2.0
0.1
2.6
2.8
1.9
1.5
0.5
3.0
2.2
3.6
2.4
3.4
4.0
0.9
0.3

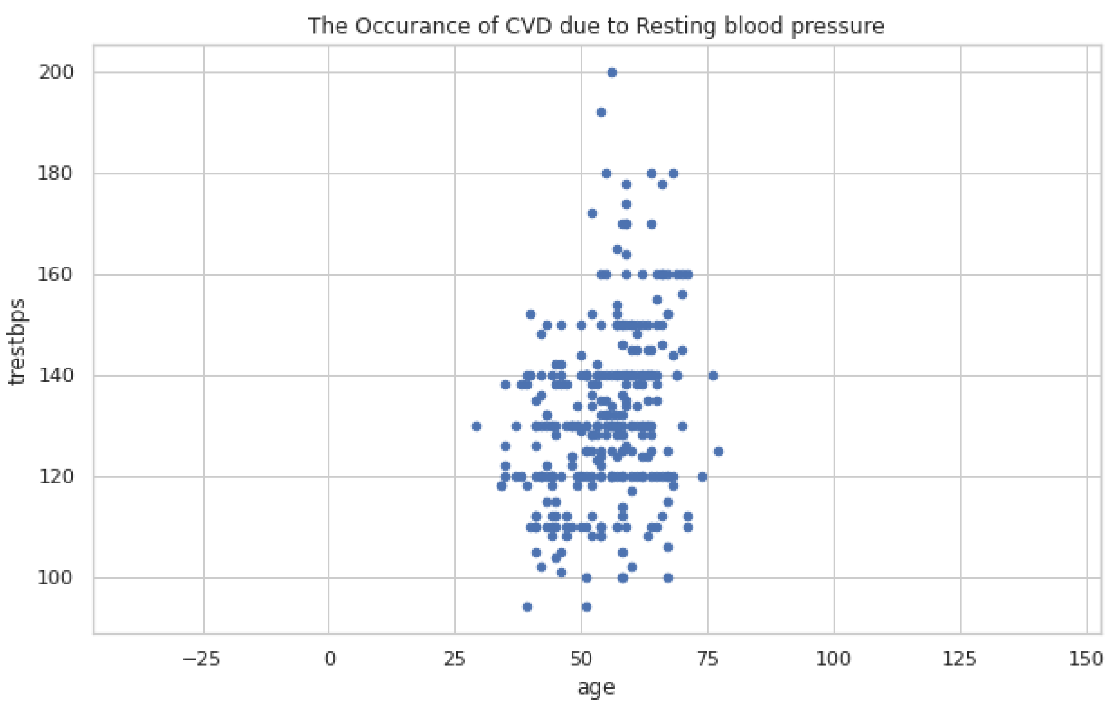

#The Occurrence of CVD due to slope,Number_Major_Vessel
slope
2
0
1
46.7%
7.0%
46.4%
Number_Major_Vessel
0
1
2
3
4
57.9%
21.5%
12.6%
6.6%
1.3%

we can see that some data is not clearly visible, lets try with the line pot.

[32]:
```
# lets find the occurance of cvs due to resting blood pressure across age
health[['age','trestbps']].
 ↪plot(kind='scatter',x='age',y='trestbps',figsize=(10,6),title="The Occurance␣
 ↪of CVD due to Resting blood pressure").axis('equal')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```

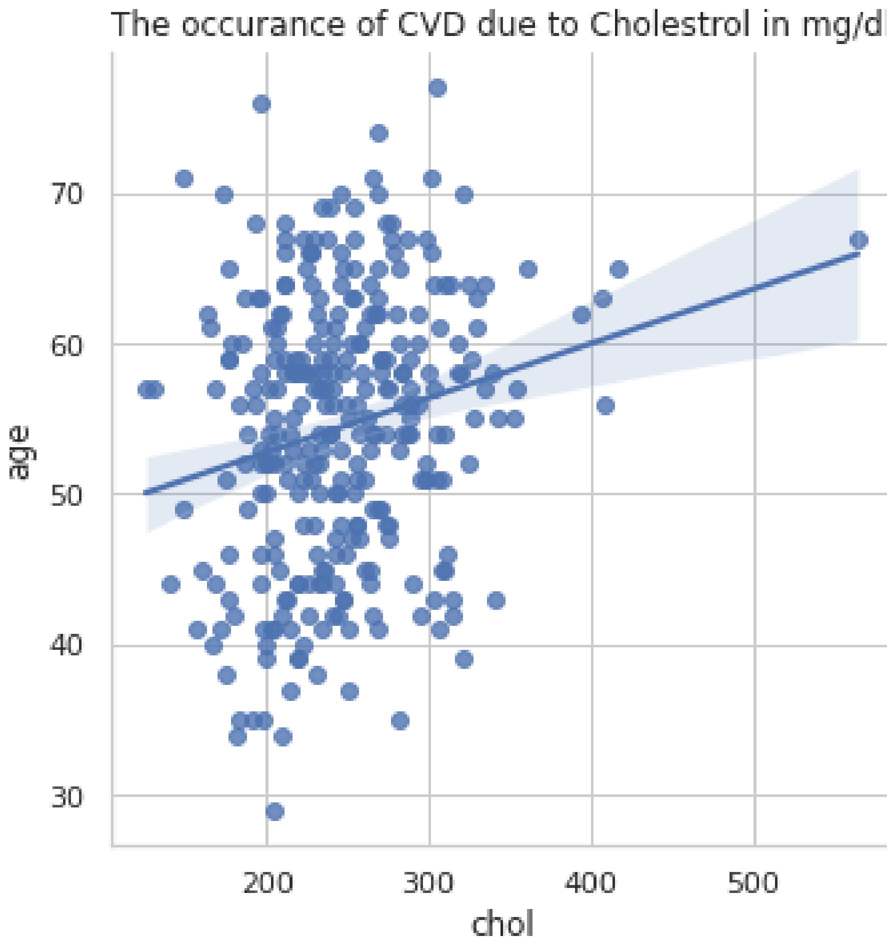
[32]: (26.6, 79.4, 88.7, 205.3)



Interpretation: we can see that age group between 30-70 has more bloodpressure report for CVD.

[33]:
```
#lets find the CVD occurance across ages due to cholesterol
sns.lmplot('chol', 'age', data=health, fit_reg=True).set(title="The occurance␣
 ↪of CVD due to Cholestrol in mg/dl")
```

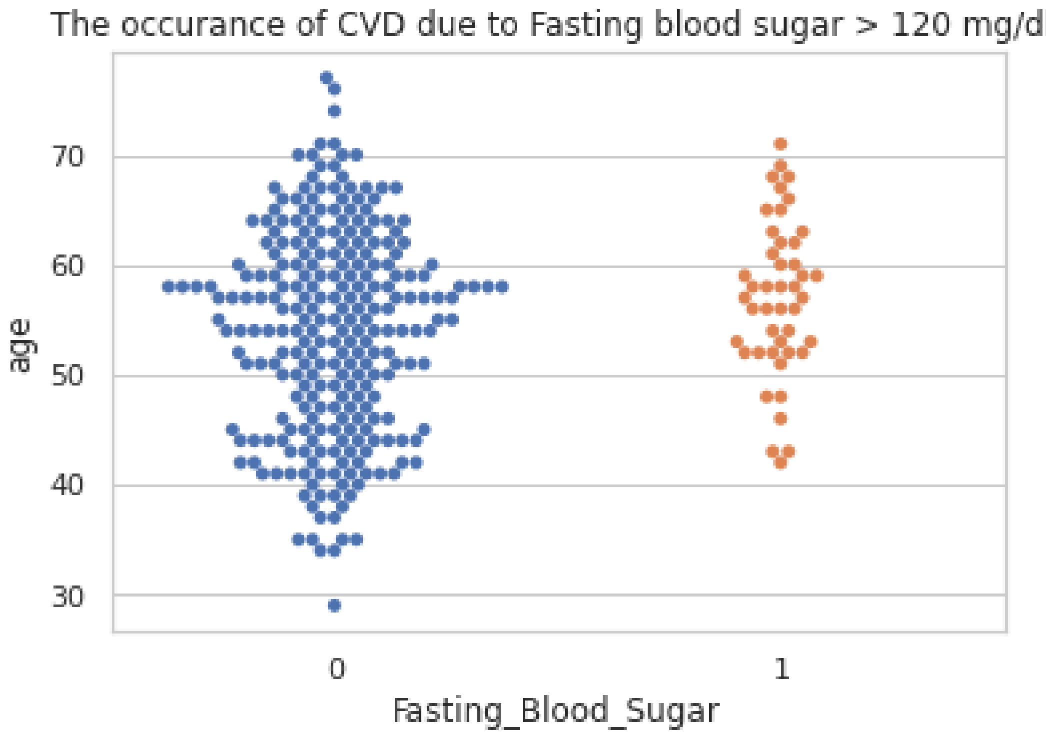
[33]: <seaborn.axisgrid.FacetGrid at 0x7f795286bd90>



Interpretation: we can see that age group between 50 and above has more cholestrol report for CVD.

[34]:
```
#lets find the CVD occurance across ages due to blood sugar
sns.swarmplot(x=health['Fasting_Blood_Sugar'],
              y=health['age']).set(title="The occurance of CVD due to Fasting␣
↪blood sugar > 120 mg/dl ")
```

[34]:
```
[Text(0.5, 1.0, 'The occurance of CVD due to Fasting blood sugar > 120 mg/dl ')]
```



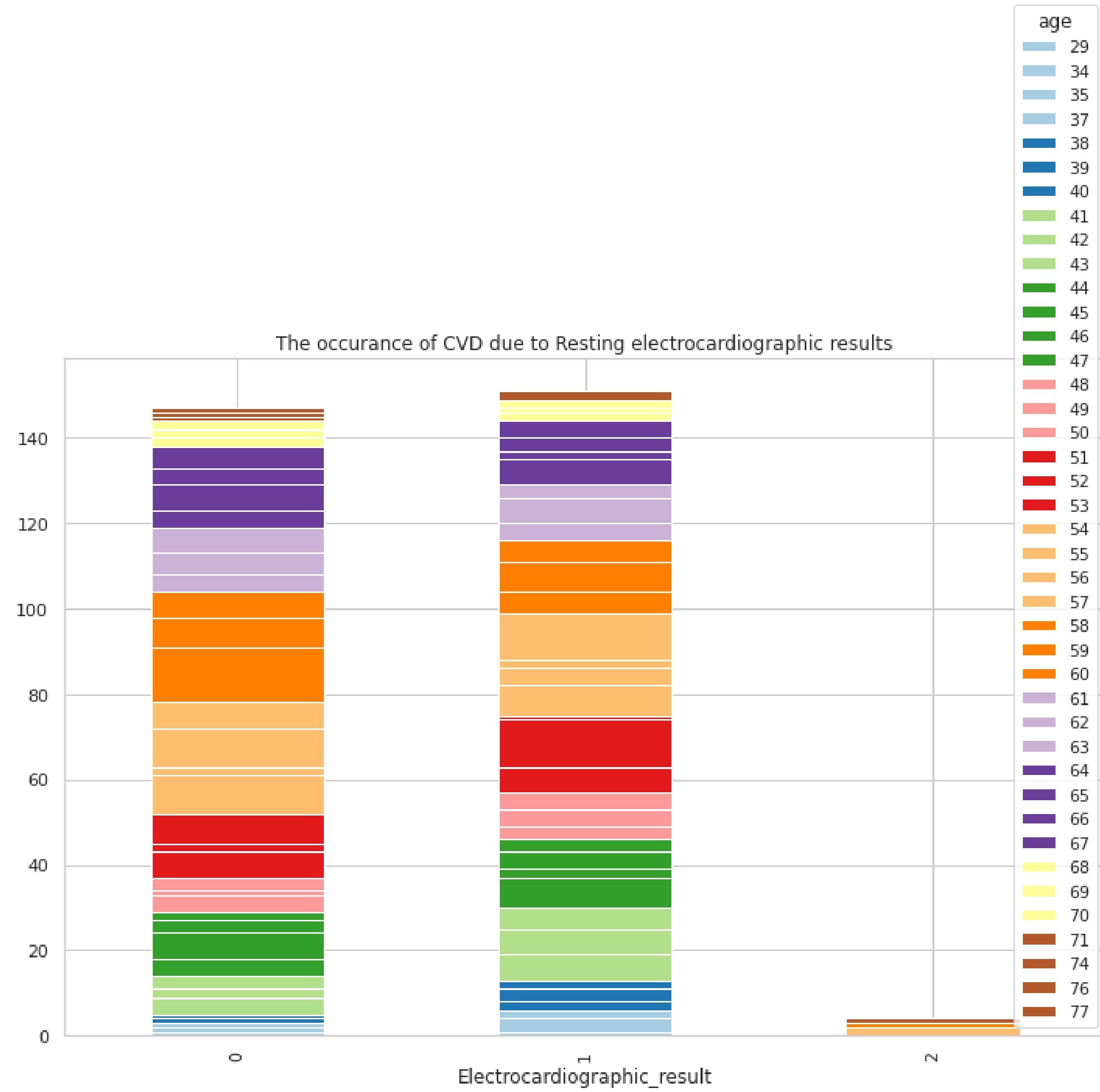
Interpretation: we can see that age group between 40-70 has more Blood Sugar report for CVD.

[35]:
```
##lets find the CVD occurance across ages due to electrocardiographic result
crosstab = pd.crosstab(index=health["Electrocardiographic_result"],␣
↪columns=health["age"])
crosstab
```

[35]:
```
age                           29  34  35  37  38  39  40  41  42  43  …  65  \
Electrocardiographic_result                                           …
0                              1   1   1   0   0   1   1   4   2   3  …   6
1                              0   1   3   2   2   3   2   6   6   5  …   2
2                              0   0   0   0   0   0   0   0   0   0  …   0

age                           66  67  68  69  70  71  74  76  77
Electrocardiographic_result
0                              4   5   2   2   2   1   1   0   1
1                              3   4   2   1   2   2   0   0   0
2                              0   0   0   0   0   0   0   1   0

[3 rows x 41 columns]
```

[36]:
```
crosstab.plot(kind="bar", figsize=(12,8), stacked=True, colormap='Paired')
plt.title("The occurance of CVD due to Resting electrocardiographic results")
```

[36]: Text(0.5, 1.0, 'The occurance of CVD due to Resting electrocardiographic results')

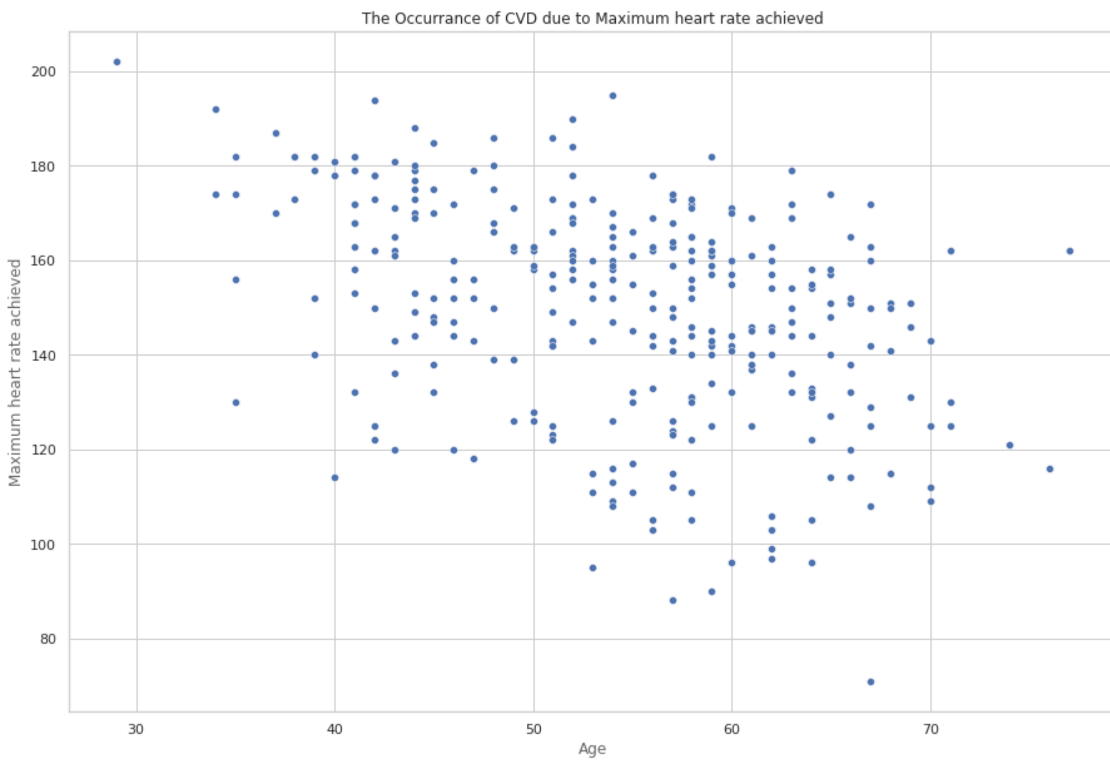


Interpretation: we can see that age group between 40-70 has more Electrocardiographic report for CVD.

[37]:
```
##lets find the CVD occurance across ages due to Maximum Heart Rate
sns.set(style="whitegrid")

f, ax = plt.subplots(figsize=(15, 10))

sns.scatterplot(data=health, x="age", y="Max_Heart_Rate", legend=False,␣
 ↪sizes=(20, 200), ax=ax)
```

```
ax.set_title('The Occurrance of CVD due to Maximum heart rate achieved')
ax.set_xlabel("Age", alpha=0.7)
ax.set_ylabel("Maximum heart rate achieved", alpha=0.7)
```

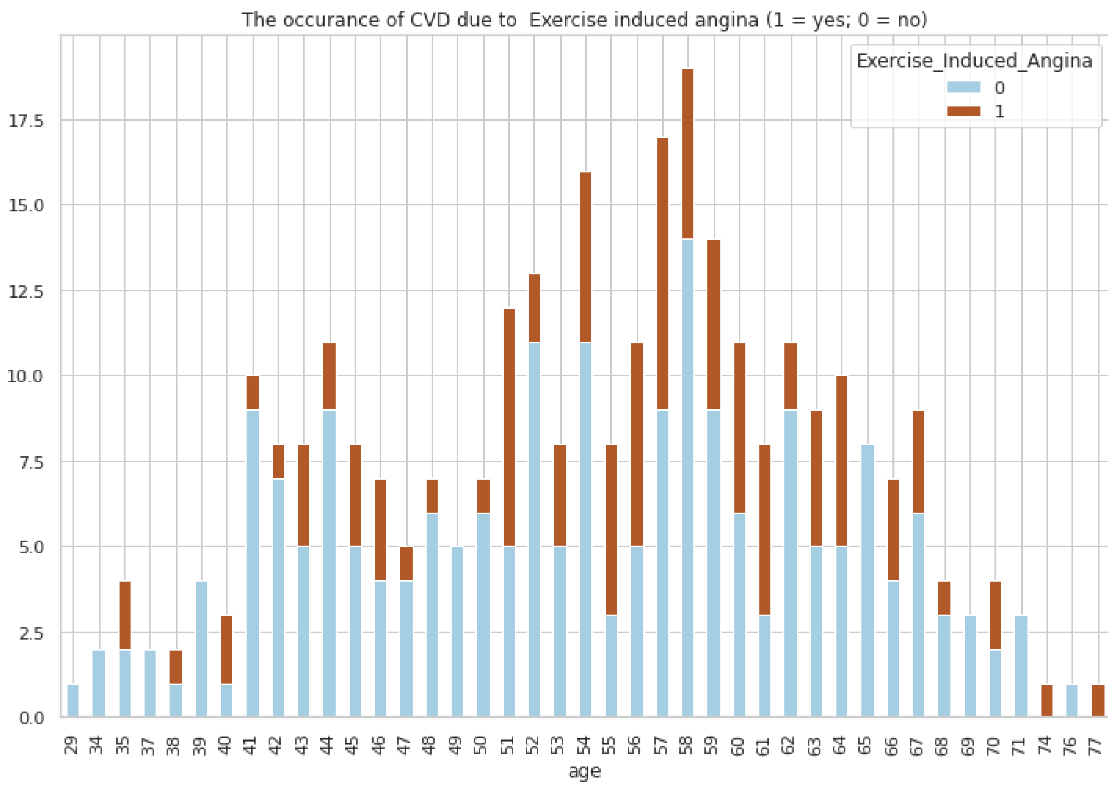
[37]: Text(0, 0.5, 'Maximum heart rate achieved')



Interpretation: we can see that age group between 40-70 has more Maximum Heart Rate report for CVD.

[38]:
```
# #lets find the CVD occurance across ages due to Excersie induced Angina

crosstab =pd.
↪crosstab(index=health['age'],columns=health['Exercise_Induced_Angina'])

crosstab.plot(kind="bar", figsize=(12,8), stacked=True, colormap='Paired')
plt.title("The occurance of CVD due to  Exercise induced angina (1 = yes; 0 =␣
↪no)")
```

[38]: Text(0.5, 1.0, 'The occurance of CVD due to Exercise induced angina (1 = yes; 0 = no)')



Interpretation: we can see that age group between 35-80 has more Exercise Induced Angina report for CVD.

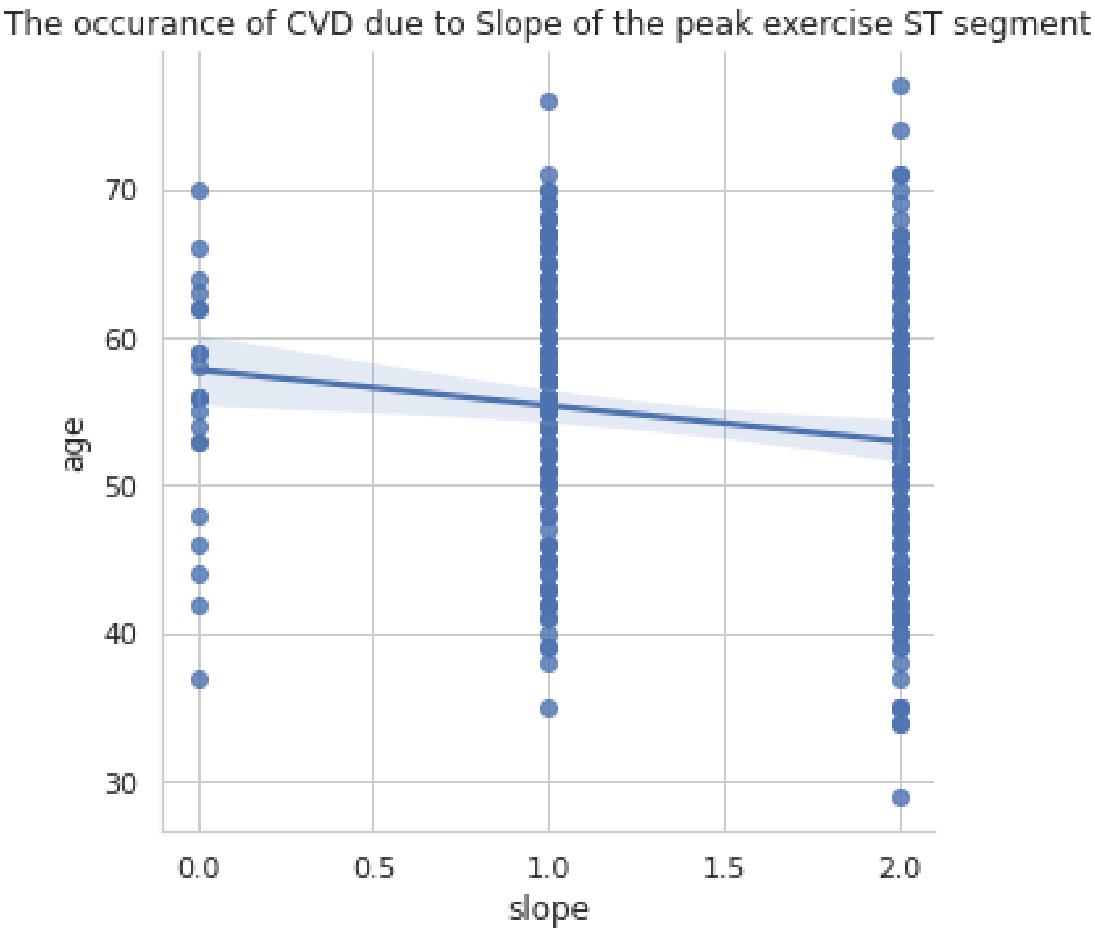
[39]:
```
#lets find the CVD occurance across ages due to oldpeak

sns.lmplot('oldpeak','age',data=health,fit_reg=True).set(title="The occurance␣
↪of CVD due to ST depression induced by exercise relative to rest")
```

[39]: <seaborn.axisgrid.FacetGrid at 0x7f7954d4e310>

Interpretation: we can see that age group between 35-70 has more oldpeak report for CVD.

[40]:
```
#lets find the CVD occurance across ages due to slope

sns.lmplot('slope', 'age', data=health, fit_reg=True).set(title="The occurance␣
↪of CVD due to Slope of the peak exercise ST segment")
```

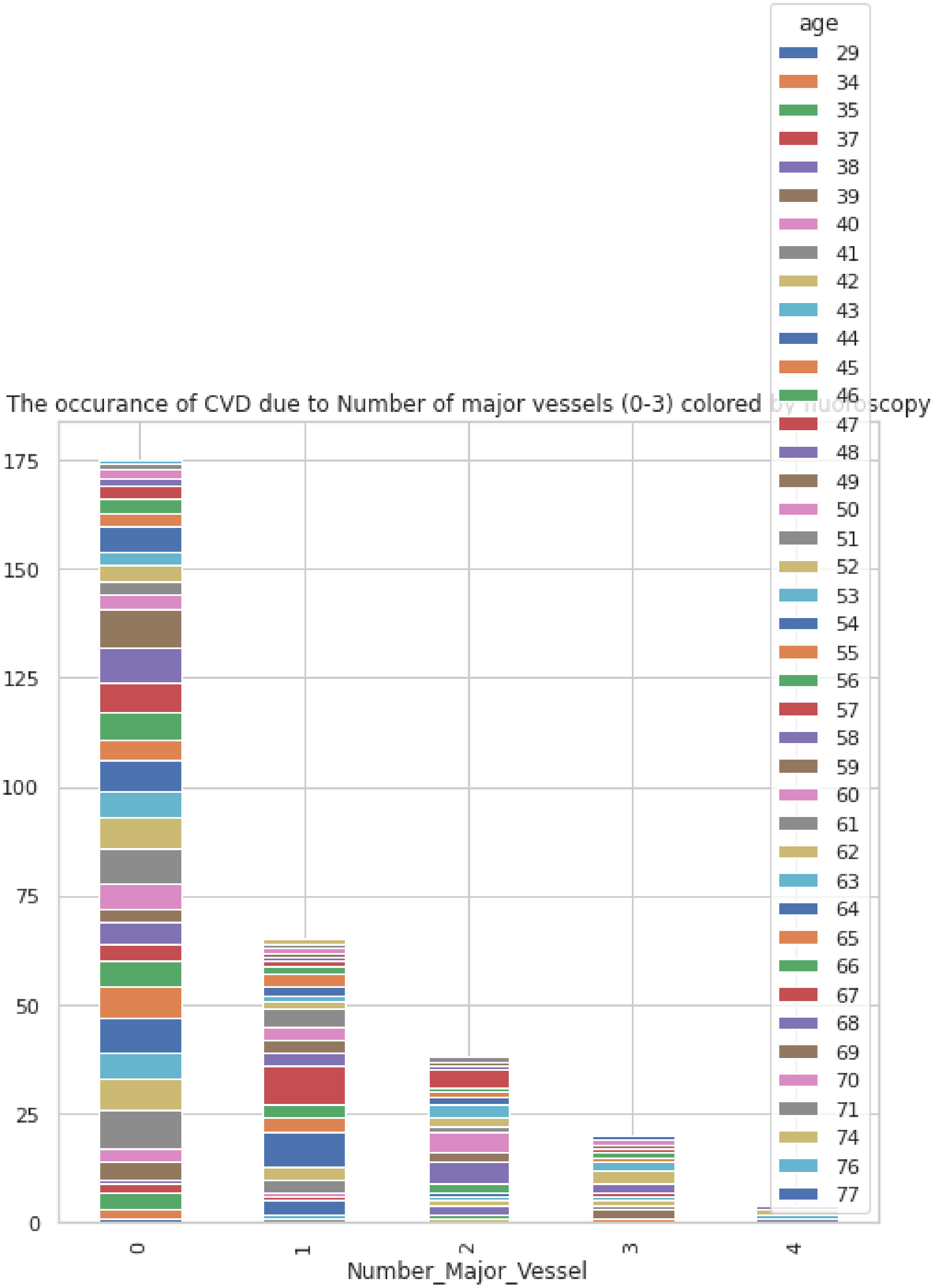
[40]: <seaborn.axisgrid.FacetGrid at 0x7f7952c52790>



Interpretation: we can see that age group between 30-70 has more Exercise ST segment report for CVD.

[41]:
```
#lets find the CVD occurance across ages due to Number Major Vessel
crosstab= pd.crosstab(index=health['Number_Major_Vessel'],columns=health['age'])
crosstab.plot(kind= "bar",figsize=(8,8),stacked =True)
plt.title("The occurance of CVD due to Number of major vessels (0-3) colored by␣
↪fluoroscopy")
```

[41]:
```
Text(0.5, 1.0, 'The occurance of CVD due to Number of major vessels (0-3)
colored by fluoroscopy')
```

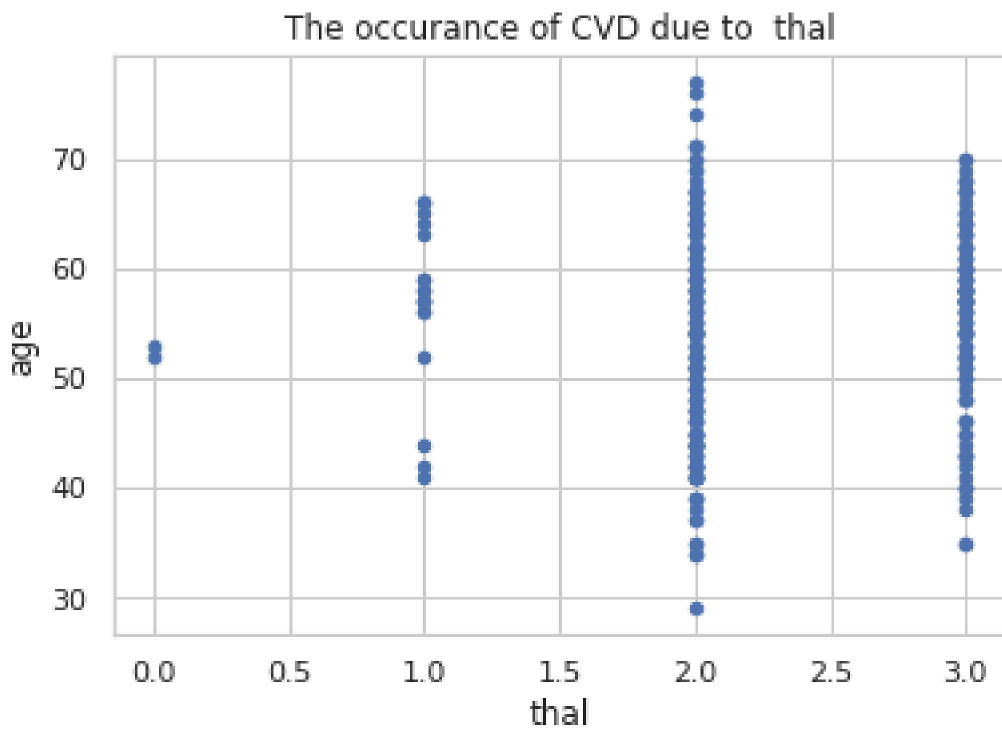


Interpretation: we can see that age group between 25-70 has more number of Major Vessel report for CVD.

[42]:
```
#lets find the CVD occurance across ages due to thal

health.plot.scatter('thal','age')
plt.title("The occurance of CVD due to  thal")
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```

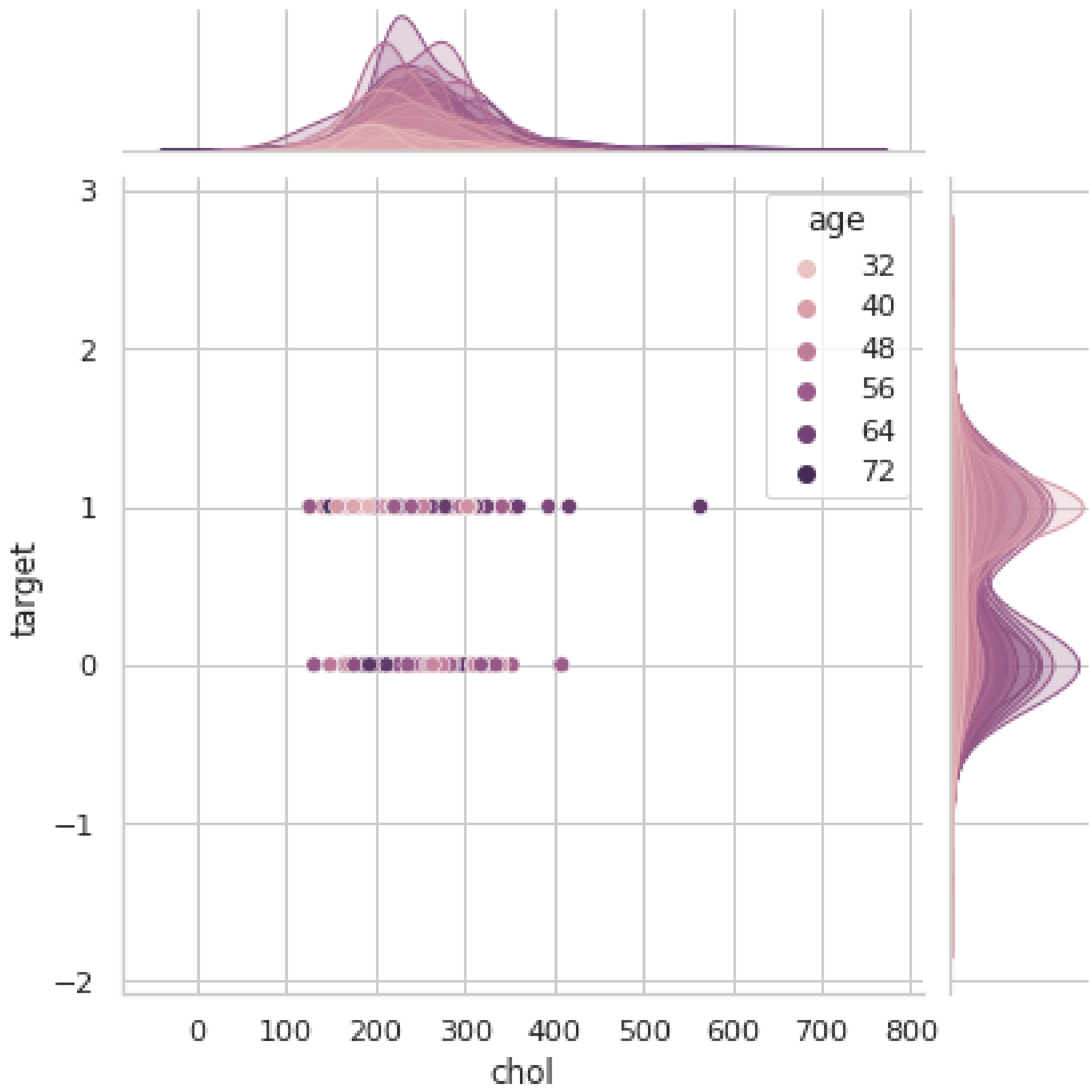
[42]:
```
Text(0.5, 1.0, 'The occurance of CVD due to  thal')
```



Interpretation: we can see that age group between 30-70 has more thal report for CVD.

[43]:
```
#lets find the CVD occurance across ages due to 'target'
sns.jointplot(data=health, x="chol", y="target", hue="age")
```

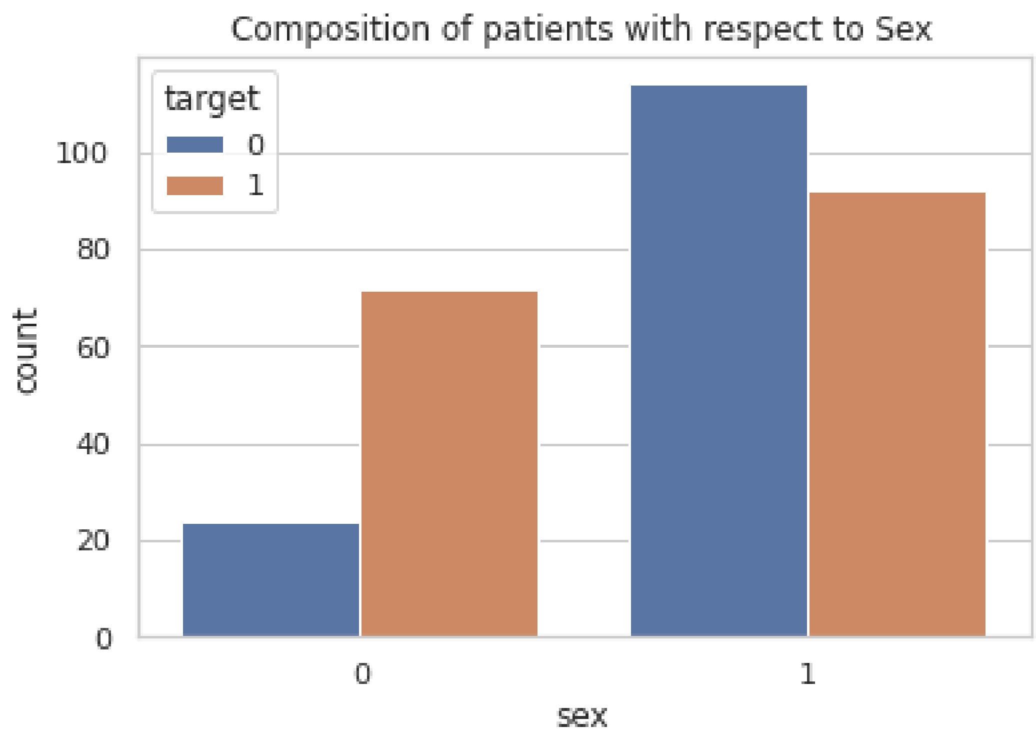
[43]:
```
<seaborn.axisgrid.JointGrid at 0x7f7954912610>
```



Interpretation: Overall, we can see that age group between 35-70 has more occurance of CVD due to output result.

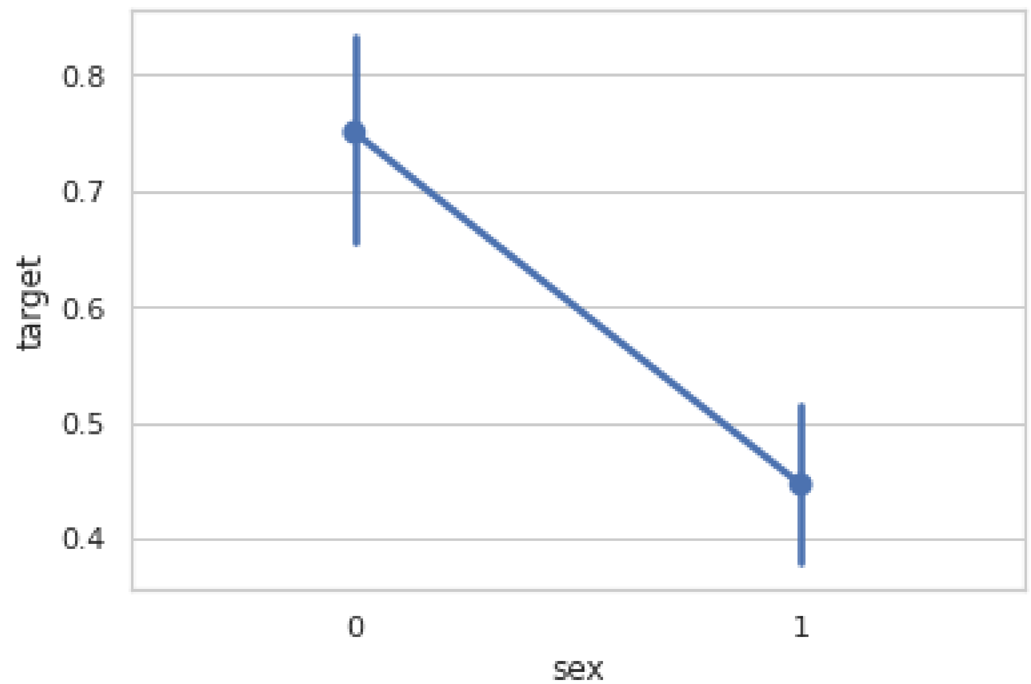
3.2 d. Study the composition of all patients with respect to the Sex category

[44]:
```
# lets plot the countplot for all patient (target) detects heart diseases again␣
↪their gender(sex).
sns.countplot(data=health, x='sex', hue='target')
plt.title('Composition of patients with respect to Sex')
plt.show()
```



[45]:
```
sns.pointplot(data=health, x='sex',y='target')
```

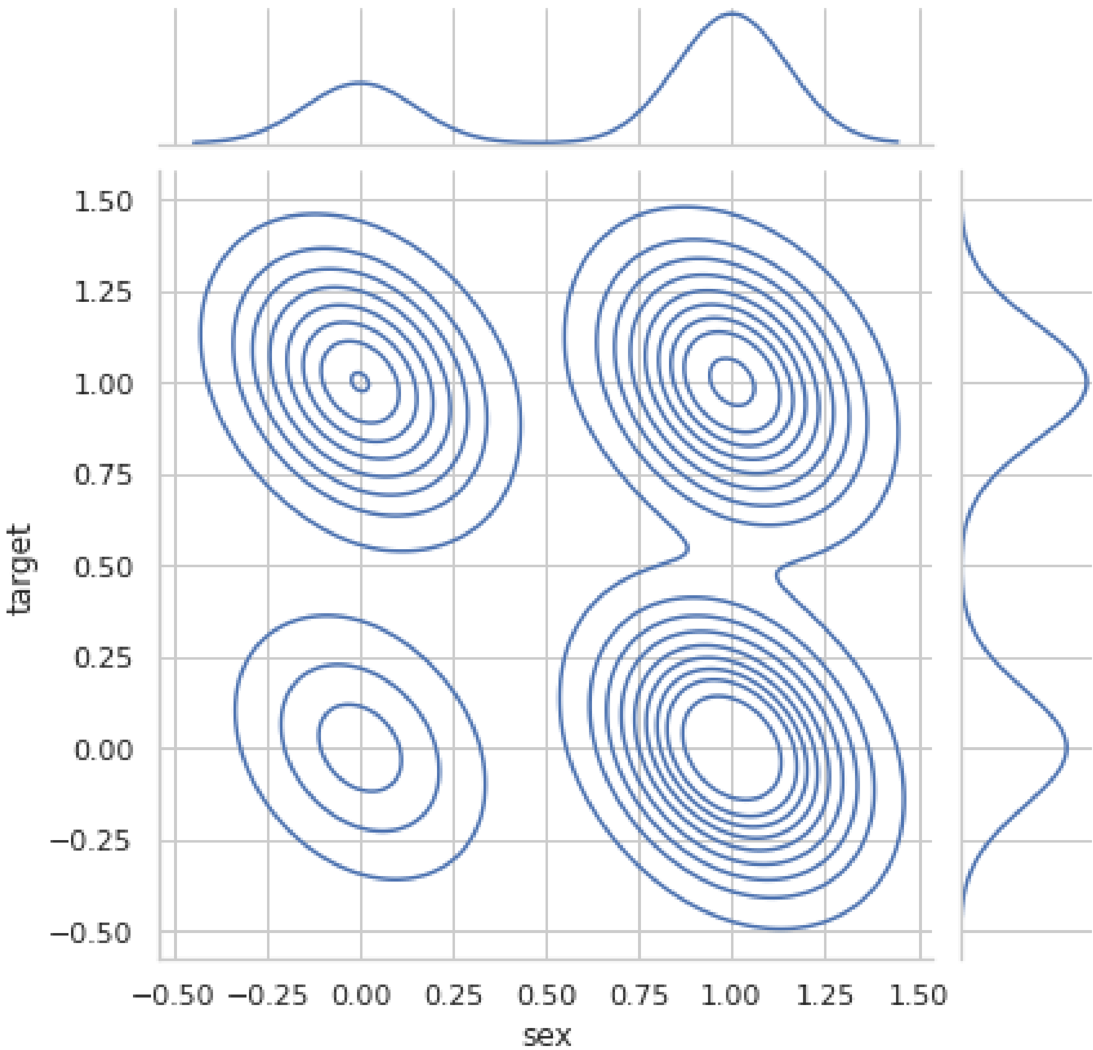
[45]: <AxesSubplot:xlabel='sex', ylabel='target'>



[46]:

```python
# make KDE plot for concentrated gender impact of heart disease
sns.jointplot(data=health, x="sex", y="target", kind="kde")
```

[46]: <seaborn.axisgrid.JointGrid at 0x7f795039ea10>



Interpretation: we can see that Male, has more diseases in the age (50-80) and Female ,has more heart diseases in the age (35-60).

[47]:

```python
# check the skewness of the data
health.skew()
```

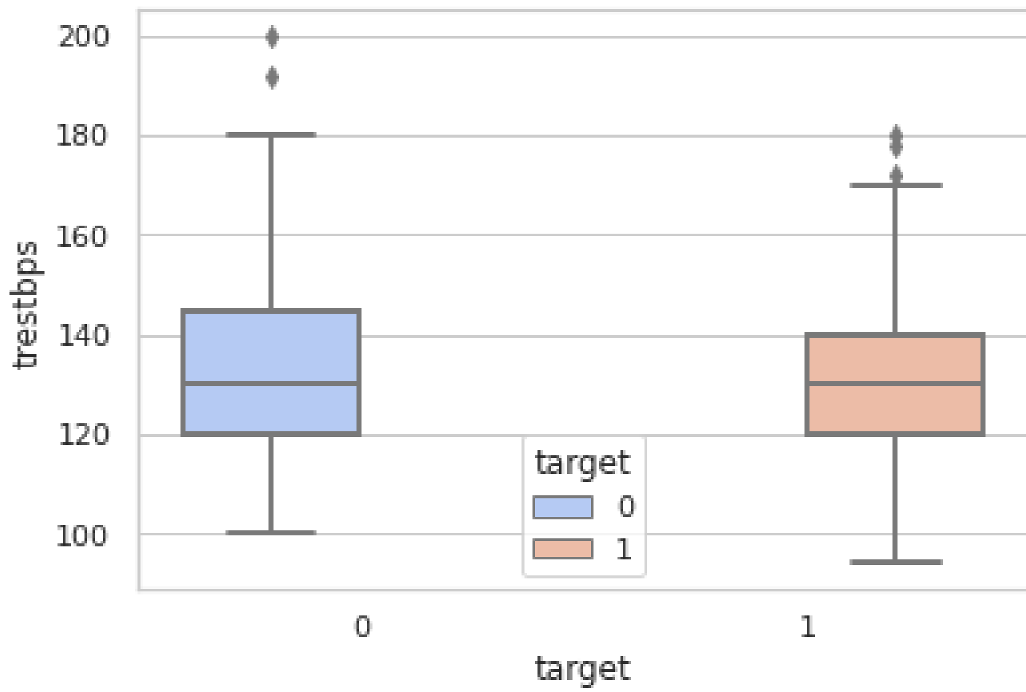
```
[47]: age                             -0.203743
      sex                             -0.786120
      Chest_Pain                       0.493022
      trestbps                         0.716541
      chol                             1.147332
      Fasting_Blood_Sugar              1.981201
      Electrocardiographic_result      0.169467
      Max_Heart_Rate                  -0.532671
      Exercise_Induced_Angina          0.737281
      oldpeak                          1.266173
      slope                           -0.503247
      Number_Major_Vessel              1.295738
      thal                            -0.481232
      target                          -0.173691
      dtype: float64
```

we can see that age,sex,Max_Heart_Rate,Slope,Thal,Target are negative Skew and rest are positive skewed.

3.2.1 Study if one can detect heart attacks based on anomalies in the resting blood pressure(tretbps) of a patient.

```
[48]: sns.boxplot(data=health, y='trestbps', x='target', hue='target',␣
       ↪color='coolwarm', palette='coolwarm',
                      saturation=0.75, width=0.8, dodge=True, fliersize=5,␣
       ↪linewidth=2, whis=1.5)
```

```
[48]: <AxesSubplot:xlabel='target', ylabel='trestbps'>
```



we can see there is an outliers lets find out which index row has an outliers in trestbps(Resting blood pressure (in mm Hg on admission to the hospital).

[49]:
```
# lets find the number of outliers
def outliers(col):
    sorted(col)
    Q1,Q3 =np.percentile(col,[25,75])
    IQR =Q3-Q1
    lower_range =Q1-(1.5*IQR)
    upper_range=Q3+(1.5*IQR)
    return lower_range,upper_range
```

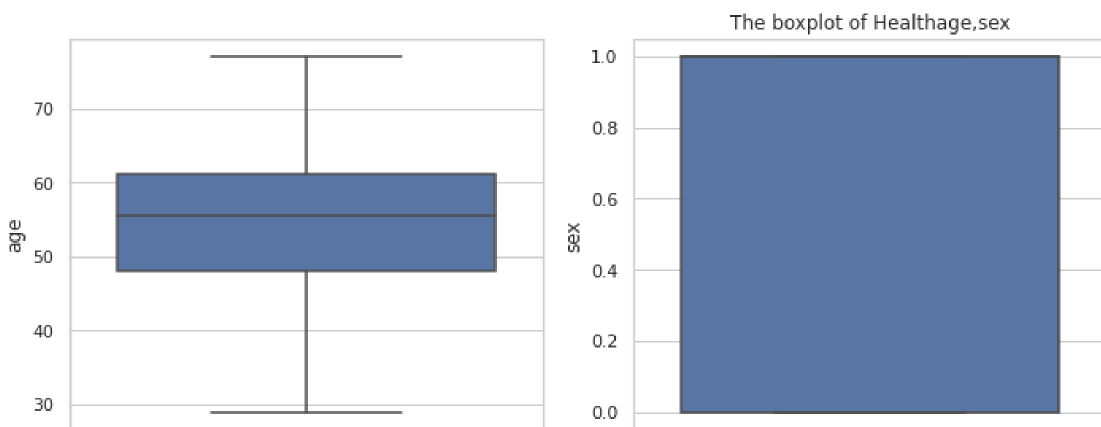
[50]:
```
# check for outliers in the column name= trestbps
lower_range,upper_range =outliers(health['trestbps'])
low_val =health[health['trestbps'].values<lower_range]
print(low_val)
print('--'*30)
up_val =health[health['trestbps'].values>upper_range]
print(up_val)
lower_outlier= low_val.value_counts().sum(axis=0)
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in Resting blood pressure (in mm Hg on admission to the␣
 ↪hospital)",Total_outliers)
```

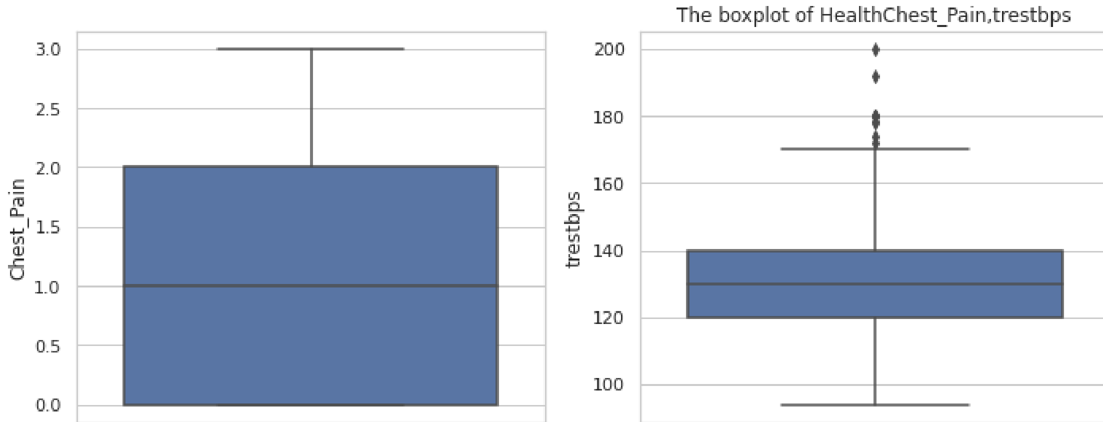
```
low_ind =list(health[health['trestbps']<lower_range].index)
up_ind =list(health[health['trestbps']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

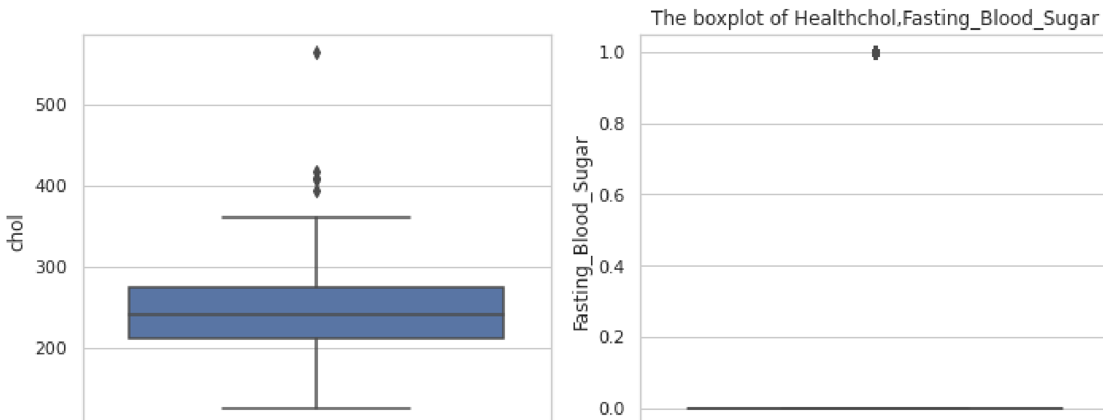
```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
------------------------------------------------------------
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
8     52    1           2       172   199                    1
101   59    1           3       178   270                    0
110   64    0           0       180   325                    0
203   68    1           2       180   274                    1
223   56    0           0       200   288                    1
241   59    0           0       174   249                    0
248   54    1           1       192   283                    0
260   66    0           0       178   228                    1
266   55    0           0       180   327                    0

     Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
8                              1             162                        0
101                            0             145                        0
110                            1             154                        1
203                            0             150                        1
223                            0             133                        1
241                            1             143                        1
248                            0             195                        0
260                            1             165                        1
266                            2             117                        1

     oldpeak  slope  Number_Major_Vessel  thal  target
8        0.5      2                    0     3       1
101      4.2      0                    0     3       1
110      0.0      2                    0     2       1
203      1.6      1                    0     3       0
223      4.0      0                    2     3       0
241      0.0      1                    0     2       0
248      0.0      2                    1     3       0
260      1.0      1                    2     3       0
266      3.4      1                    0     2       0
Total Outliers in Resting blood pressure (in mm Hg on admission to the hospital)
9
Total index that has outliers [8, 101, 110, 203, 223, 241, 248, 260, 266]
```

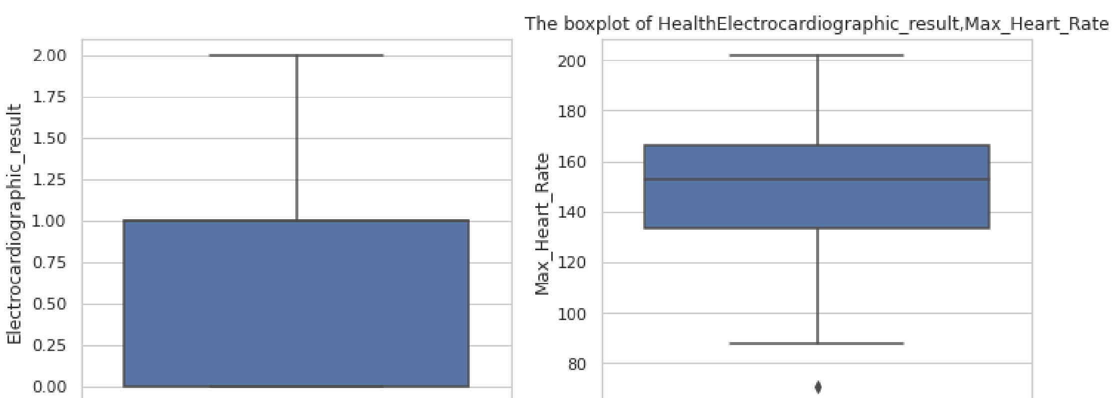
there are total 9 patient name list (rows) whose heart attack based on anomalies in the resting blood pressure.

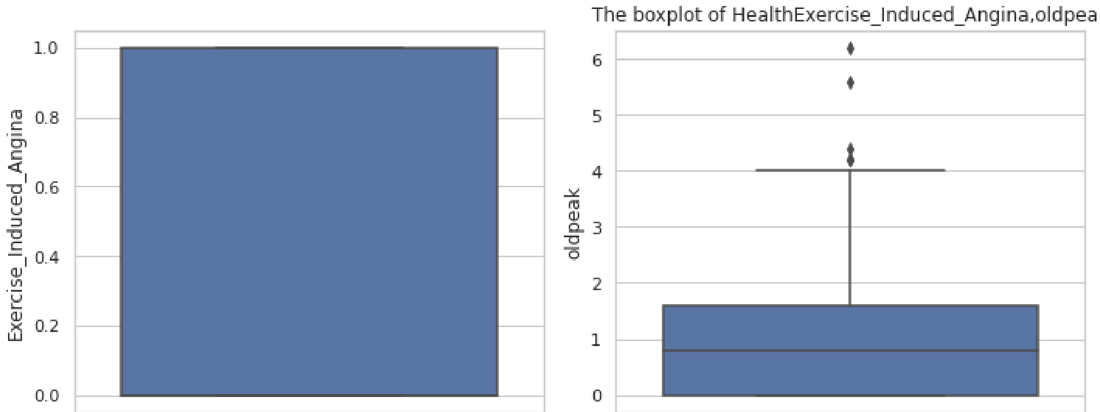
[51]:
```
# lets check for other outliers as well with the help of  box plot.
num_cols =['age', 'sex', 'Chest_Pain', 'trestbps', 'chol',␣
↪'Fasting_Blood_Sugar',
       'Electrocardiographic_result', 'Max_Heart_Rate',
       'Exercise_Induced_Angina', 'oldpeak', 'slope', 'Number_Major_Vessel',
       'thal', 'target'];
facet= None

for i in range(0,len(num_cols),2):
    plt.figure(figsize=(10,4))
    plt.subplot(121)
    sns.boxplot(facet,num_cols[i],data=health)
    plt.subplot(122)
    sns.boxplot(facet,num_cols[i+1],data=health)
    plt.title("The boxplot of Health{},{}".format(num_cols[i],num_cols[i+1]))
    plt.tight_layout()
    plt.show()
```

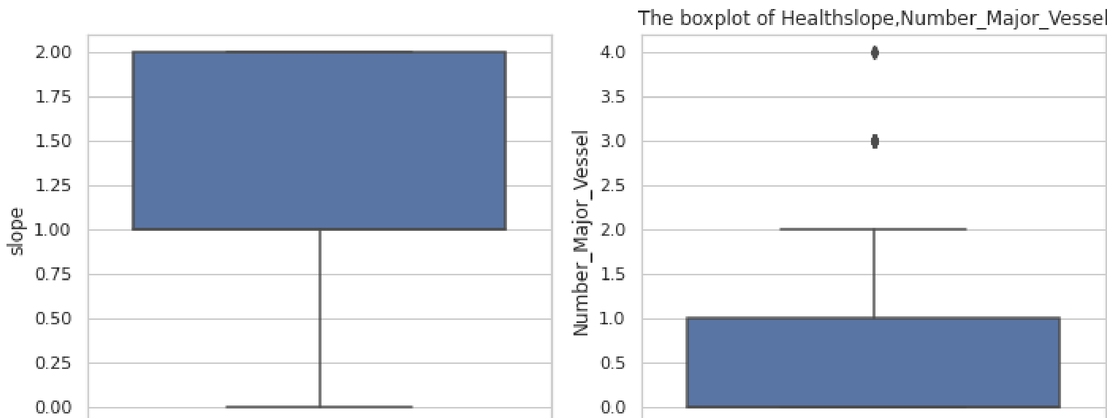


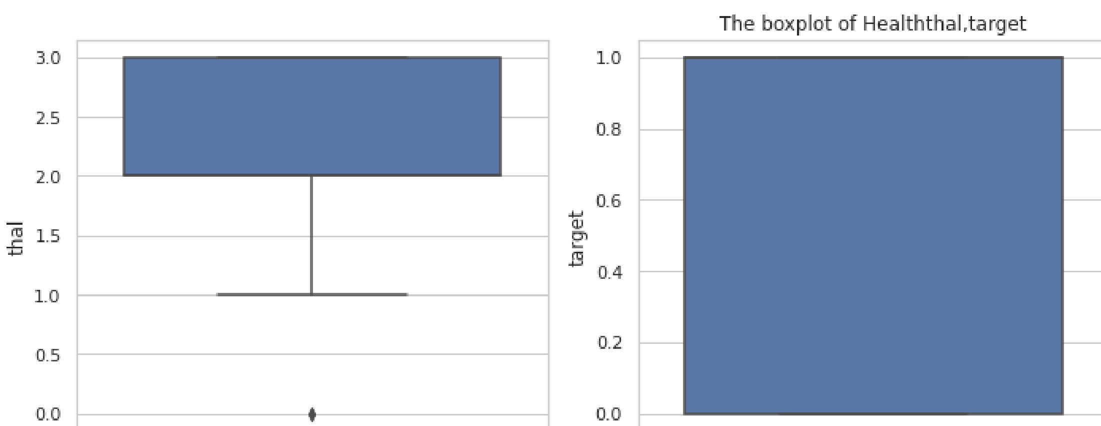

The boxplot of HealthChest_Pain,trestbps
Chest_Pain
trestbps


The boxplot of Healthchol,Fasting_Blood_Sugar
chol
Fasting_Blood_Sugar


The boxplot of HealthElectrocardiographic_result,Max_Heart_Rate
Electrocardiographic_result
Max_Heart_Rate


The boxplot of HealthExercise_Induced_Angina,oldpeak
Exercise_Induced_Angina
0.0
0.2
0.4
0.6
0.8
1.0
oldpeak
0
1
2
3
4
5
6

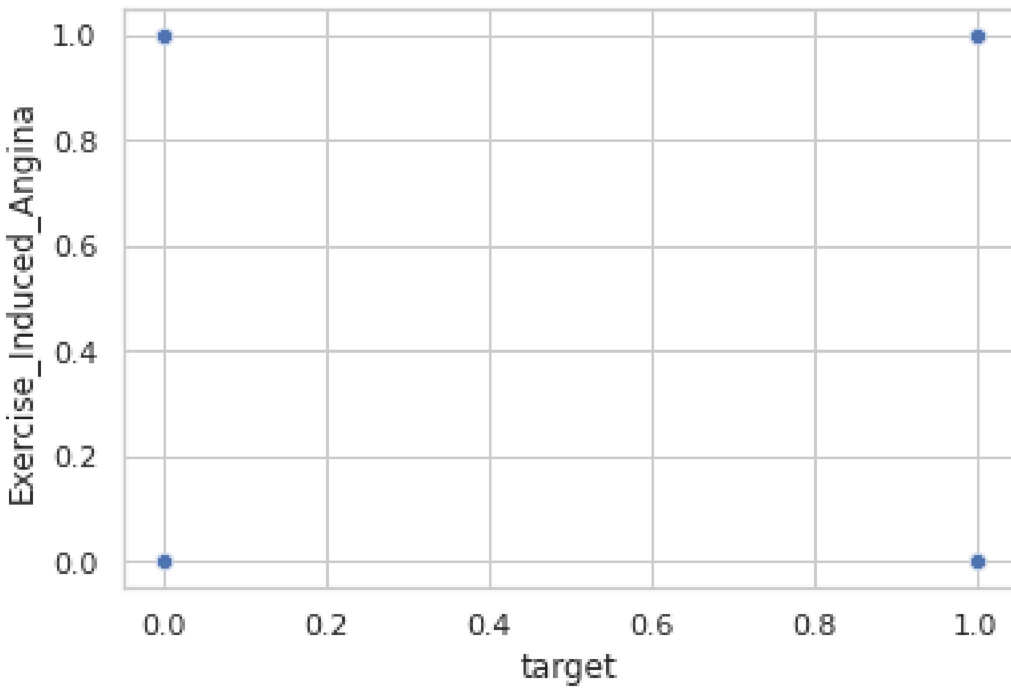

The boxplot of Healthslope,Number_Major_Vessel
slope
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Number_Major_Vessel
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


The boxplot of Healththal,target
thal
0.0
0.5
1.0
1.5
2.0
2.5
3.0
target
0.0
0.2
0.4
0.6
0.8
1.0

4 State what relationship exists between peak exercising and the occurrence of a heart attack

[52]:
```
sns.scatterplot(data=health,y='Exercise_Induced_Angina',x='target')
```

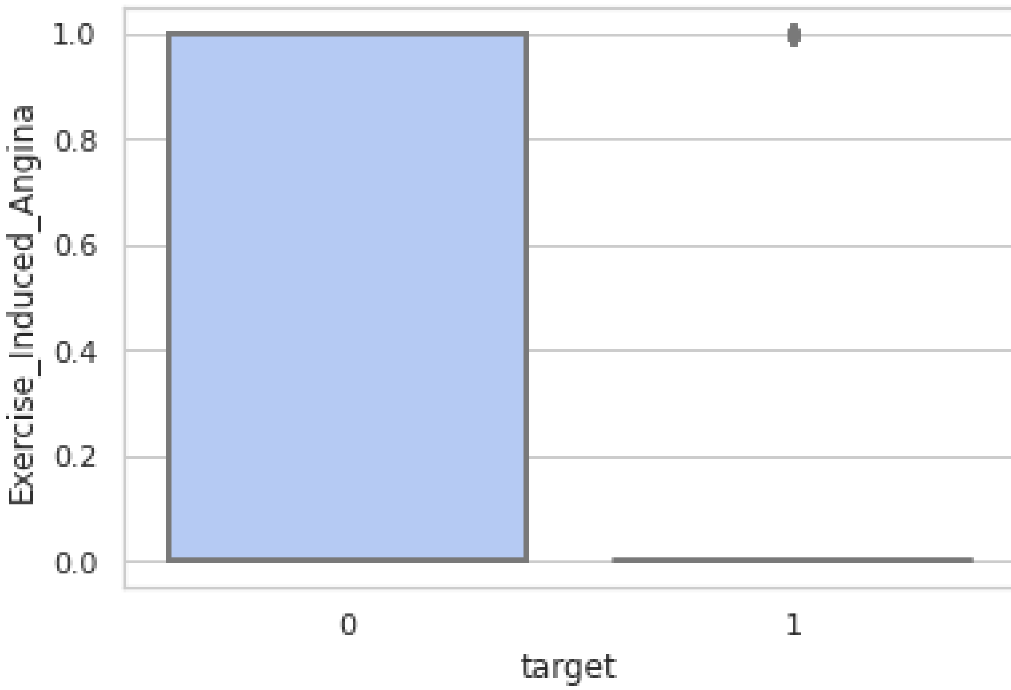
[52]: <AxesSubplot:xlabel='target', ylabel='Exercise_Induced_Angina'>



Not clear with the above graphs let us try the boxplot.

[53]:
```
sns.boxplot(data=health,␣
↪y='Exercise_Induced_Angina',x='target',color='coolwarm',palette='coolwarm',
            saturation=0.75,width=0.8,dodge=True,linewidth=2,fliersize=5,whis=1.
↪5)
```

[53]: <AxesSubplot:xlabel='target', ylabel='Exercise_Induced_Angina'>



we can see their is an outlier, lets find out which are these outliers.

[54]:
```
# lets find the number of outliers
def outliers(col):
    sorted(col)
    Q1,Q3 =np.percentile(col,[25,75])
    IQR =Q3-Q1
    lower_range =Q1-(1.5*IQR)
    upper_range=Q3+(1.5*IQR)
    return lower_range,upper_range
```

[55]:
```
# Check for Max_Heart_Rate
lower_range,upper_range =outliers(health['Exercise_Induced_Angina'])
low_val =health[health['Exercise_Induced_Angina'].values<lower_range]
print(low_val)

up_val =health[health['Exercise_Induced_Angina'].values>upper_range]
print(up_val)
lower_outlier= low_val.value_counts().sum(axis=0)
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in Exercise_Induced_Angina",Total_outliers)

low_ind =list(health[health['Exercise_Induced_Angina']<lower_range].index)
```

```
up_ind =list(health[health['Exercise_Induced_Angina']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
Total Outliers in Exercise_Induced_Angina 0
Total index that has outliers []
```

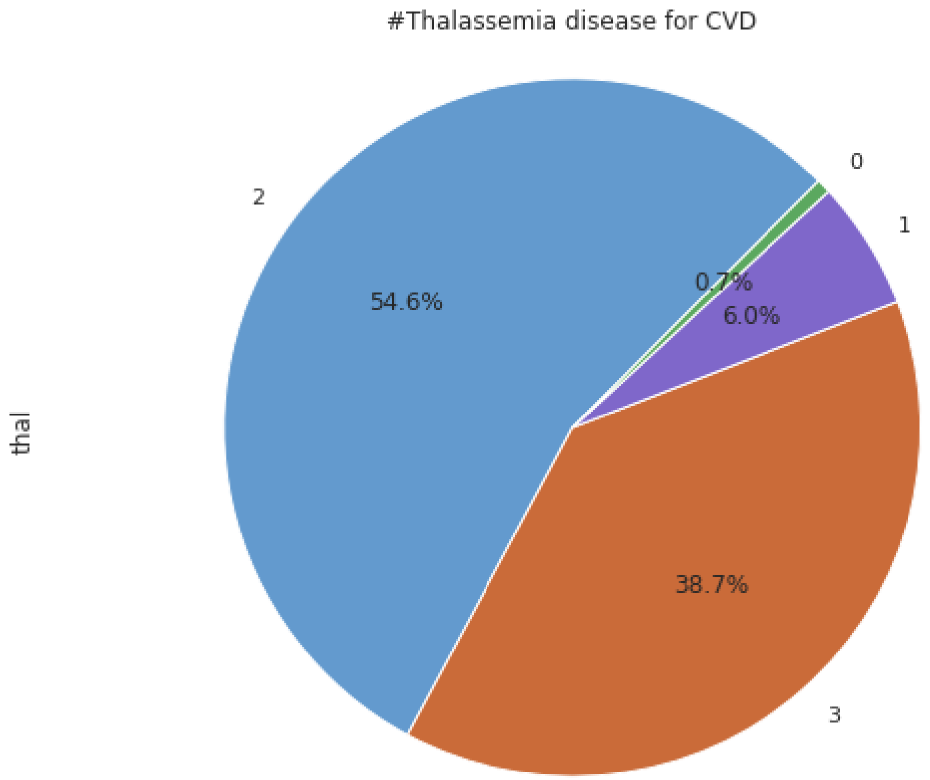
5 --> Interpretation: we can see that there are no outliers in the data.

6 h. Check if thalassemia is a major cause of CVD

[56]:
```
health.thal.value_counts()
```

[56]:
```
2    165
3    117
1     18
0      2
Name: thal, dtype: int64
```

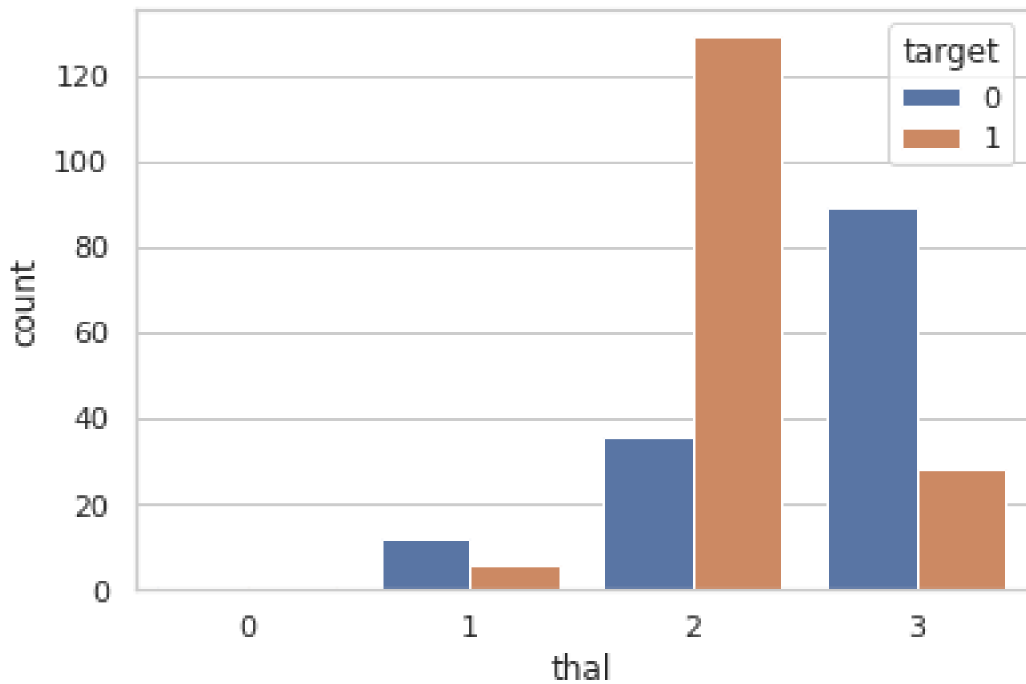
[57]:
```
# lets count ht percentage of thal as disease
colors = ['#639ace','#ca6b39','#7f67ca','#5ba85f','#c360aa','#a7993f','#cc566a']
health['thal'].value_counts().plot(kind='pie',autopct='%1.1f%%',
                          startangle=45, shadow=False, colors = colors,
                        figsize = (8,6))
plt.axis('equal')
plt.title('#Thalassemia disease for CVD ')
plt.tight_layout()
plt.show()
```



thalassemia is a major cause of CVD: we can see that: 38.7% [3] = normal; 6 = fixed defect; 6.0%[1], 7 = reversible defect,

[58]:
```
# Now lets plot a count plot for positive values.
sns.countplot(data=health, x='thal',hue='target')
```

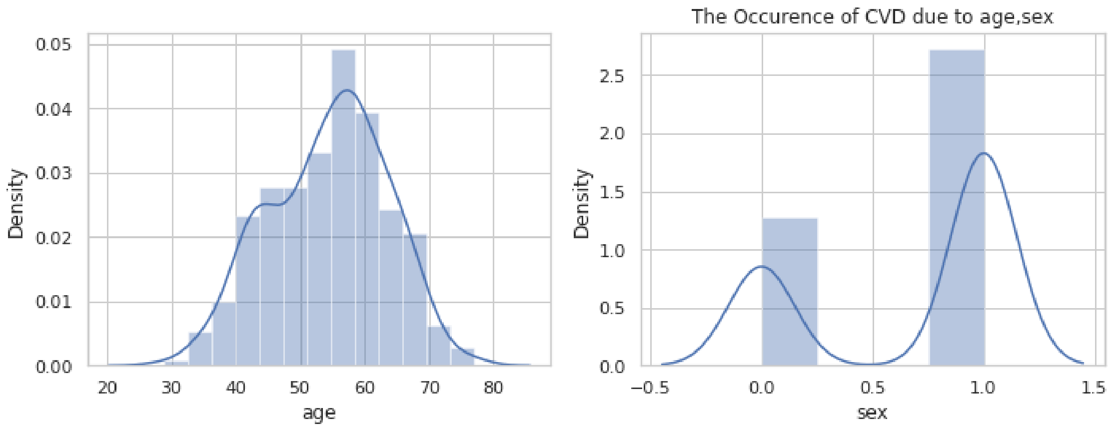
[58]: <AxesSubplot:xlabel='thal', ylabel='count'>

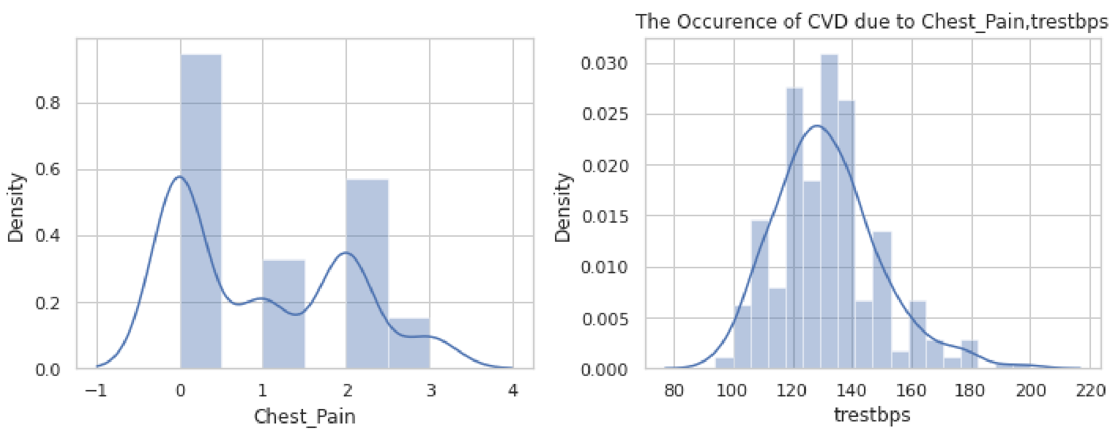


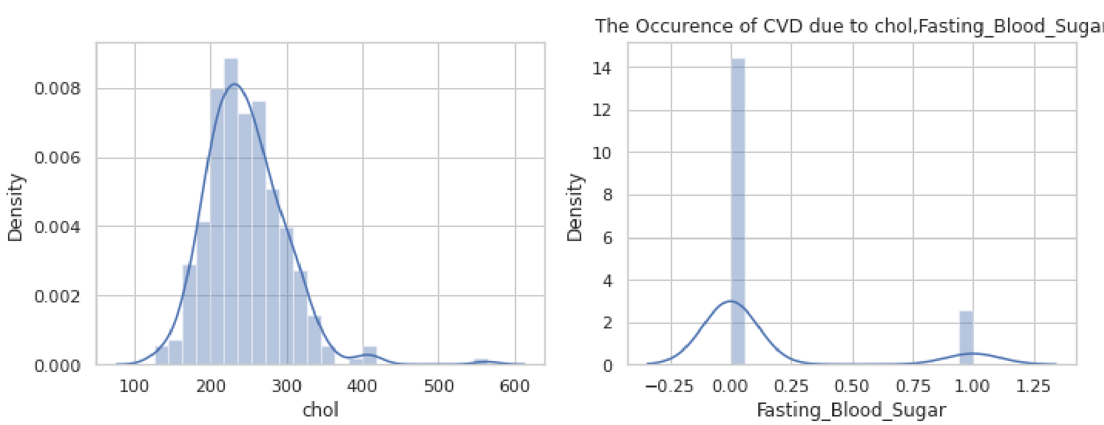
we can verify with our data that thal with number_2 has most impact of Cardiovarscular Diseases value count of Thal 2- 165 which is equivelent of 54.6%

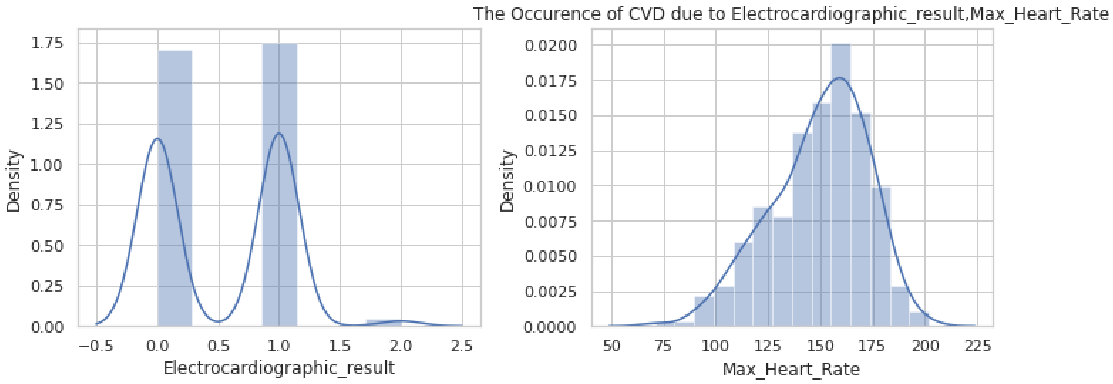
7 List how many other factors determine the occurence of CVD.

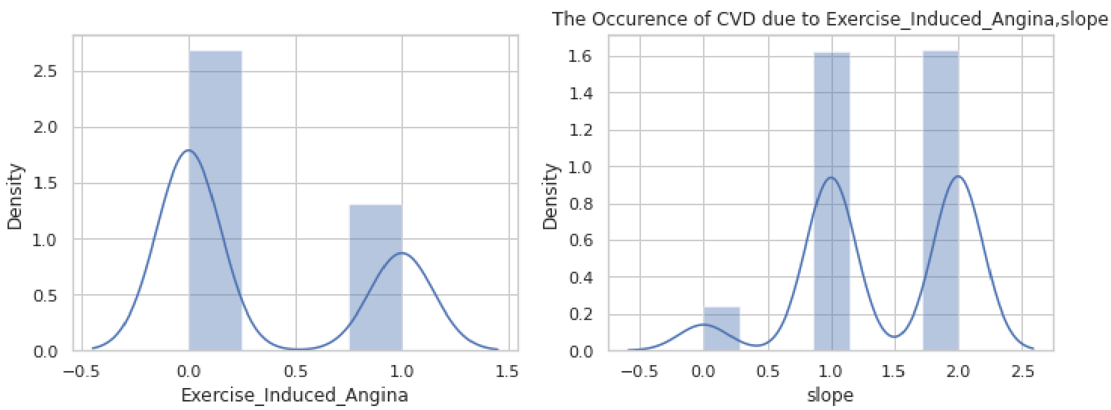
[59]:
```
num_cols = ['age', 'sex', 'Chest_Pain', 'trestbps', 'chol',␣
↪'Fasting_Blood_Sugar',
       'Electrocardiographic_result', 'Max_Heart_Rate',
       'Exercise_Induced_Angina', 'slope', 'Number_Major_Vessel','target']



for i in range(0,len(num_cols),2):
        plt.figure(figsize=(10,4))
        plt.subplot(121)
        sns.distplot(health[num_cols[i]],hist= True,kde=True)
        plt.subplot(122)
        sns.distplot(health[num_cols[i+1]],hist=True,kde=True)
        plt.title("The Occurence of CVD due to {},{}".
↪format(num_cols[i],num_cols[i+1]))
        plt.tight_layout()
        plt.show()
```

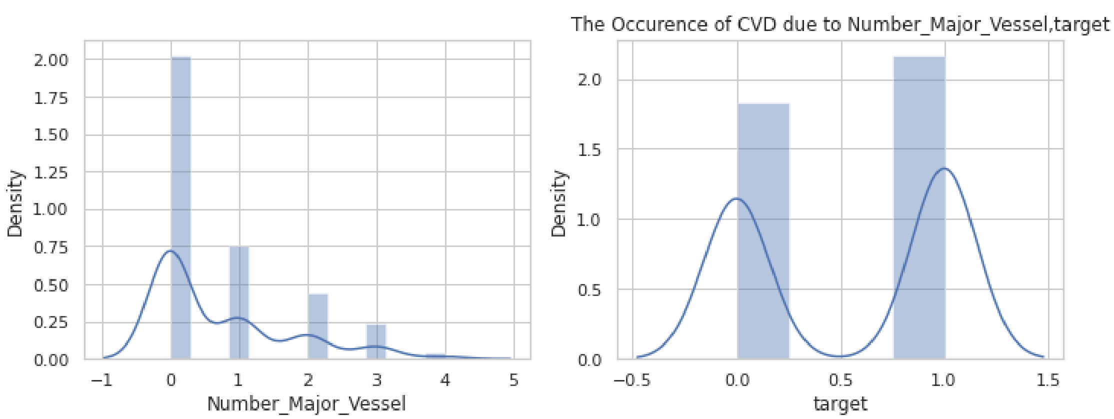

The Occurence of CVD due to age,sex
Density
age
Density
sex


The Occurence of CVD due to Chest_Pain,trestbps
Density
Chest_Pain
Density
trestbps


The Occurence of CVD due to chol,Fasting_Blood_Sugar
Density
chol
Density
Fasting_Blood_Sugar


The Occurence of CVD due to Electrocardiographic_result,Max_Heart_Rate
Density
Electrocardiographic_result
Density
Max_Heart_Rate

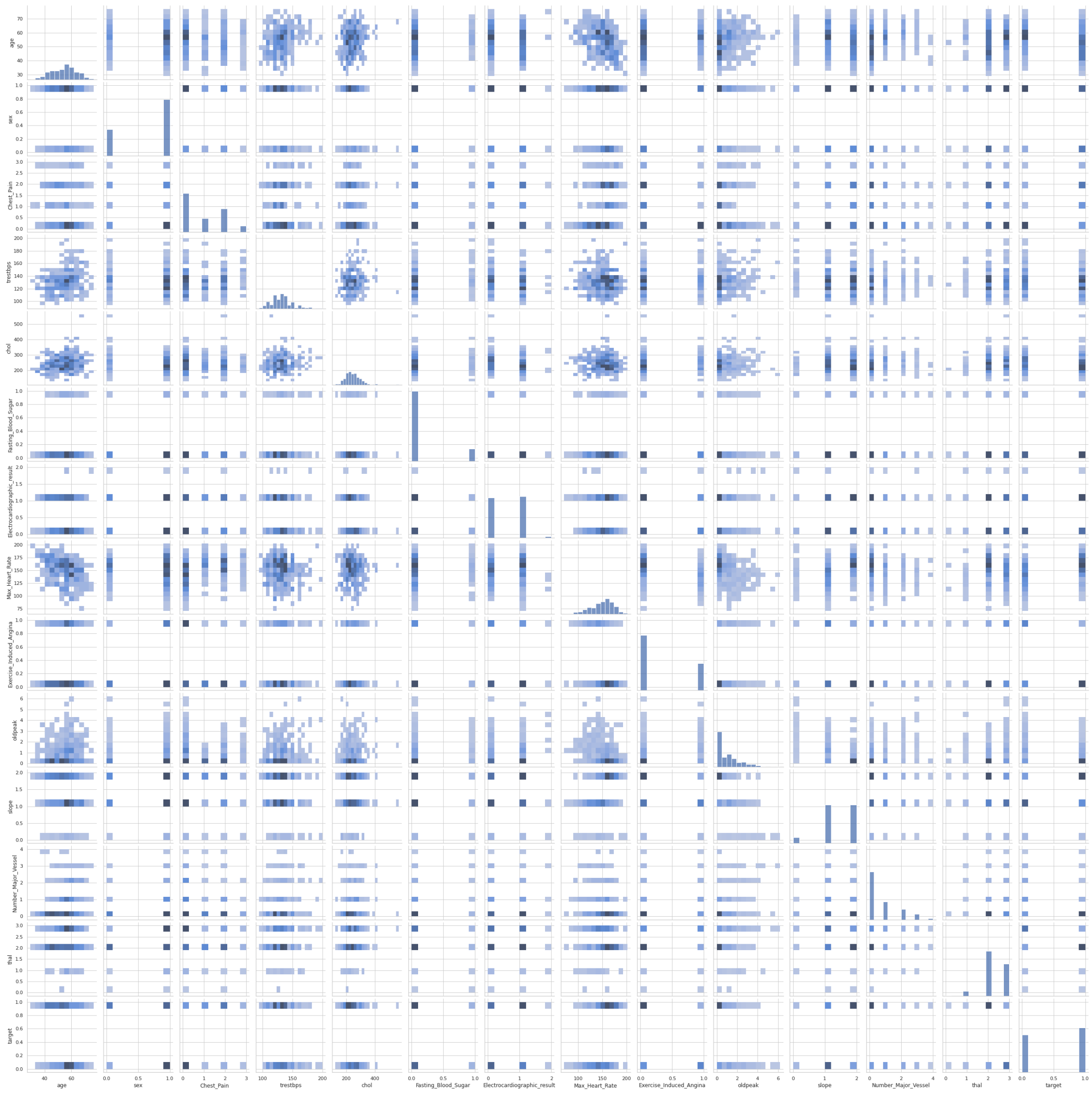

The Occurence of CVD due to Exercise_Induced_Angina,slope
Density
Exercise_Induced_Angina
Density
slope


The Occurence of CVD due to Number_Major_Vessel,target
Density
Number_Major_Vessel
Density
target

8 Use a Pairplot to understand the relationship between all the given variables.

[60]:
```
sns.pairplot(health,kind='hist')
```

[60]: <seaborn.axisgrid.PairGrid at 0x7f794b67b2d0>



9 Lets check all outliers on by one.

[61]:
```
# lets find the number of outliers
def outliers(col):
    sorted(col)
    Q1,Q3 =np.percentile(col,[25,75])
    IQR =Q3-Q1
    lower_range =Q1-(1.5*IQR)
    upper_range=Q3+(1.5*IQR)
    return lower_range,upper_range
```

[62]:
```
lower_range,upper_range =outliers(health['thal'])
```

[63]:
```
low_val =health[health['thal'].values<lower_range]
low_val
```

[63]:
```
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
48    53    0           2       128   216                    0
281   52    1           0       128   204                    1

     Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
48                             0             115                        0
281                            1             156                        1

     oldpeak  slope  Number_Major_Vessel  thal  target
48       0.0      2                    0     0       1
281      1.0      1                    0     0       0
```

[64]:
```
up_val =health[health['thal'].values>upper_range]
up_val
```

[64]:
```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
```

[65]:
```
lower_outlier= low_val.value_counts().sum(axis=0)
```

[66]:
```
upper_outlier =up_val.value_counts().sum(axis=0)
```

[67]:
```
Total_outliers =lower_outlier+upper_outlier
Total_outliers
```

[67]:
```
2
```

[68]:
```python
low_ind =list(health[health['thal']<lower_range].index)
up_ind =list(health[health['thal']>upper_range].index)
total_ind= list(low_ind+up_ind)
print(total_ind)
```

```
[48, 281]
```

[69]:
```python
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (302, 14)
Shape After Dropping Outlier Rows: (300, 14)
```

[70]:
```python
# check for Number_ Major_Vessel
lower_range,upper_range =outliers(health['Number_Major_Vessel'])

low_val =health[health['Number_Major_Vessel'].values<lower_range]
print(low_val)
print('--'*30)

up_val =health[health['Number_Major_Vessel'].values>upper_range]
print(up_val)

lower_outlier= low_val.value_counts().sum(axis=0)
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in Number of Major Vessels",Total_outliers)

low_ind =list(health[health['Number_Major_Vessel']<lower_range].index)
up_ind =list(health[health['Number_Major_Vessel']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
------------------------------------------------------------
    age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
52   62    1           2       130   231                    0
92   52    1           2       138   223                    0
97   52    1           0       108   233                    1
99   53    1           2       130   246                    1
```

158	58	1	1	125	220	0
163	38	1	2	138	175	0
165	67	1	0	160	286	0
181	65	0	0	150	225	0
191	58	1	0	128	216	0
204	62	0	0	160	164	0
208	49	1	2	120	188	0
217	63	1	0	130	330	1
220	63	0	0	150	407	0
231	57	1	0	165	289	1
234	70	1	0	130	322	0
238	77	1	0	125	304	0
247	66	1	1	160	246	0
249	69	1	2	140	254	0
250	51	1	0	140	298	0
251	43	1	0	132	247	1
252	62	0	0	138	294	1
255	45	1	0	142	309	0
267	49	1	2	118	149	0
291	58	1	0	114	318	0

	Electrocardiographic_result	Max_Heart_Rate	Exercise_Induced_Angina \
52	1	146	0
92	1	169	0
97	1	147	0
99	0	173	0
158	1	144	0
163	1	173	0
165	0	108	1
181	0	114	0
191	0	131	1
204	0	145	0
208	1	139	0
217	0	132	1
220	0	154	0
231	0	124	0
234	0	109	0
238	0	162	1
247	1	120	1
249	0	146	0
250	1	122	1
251	0	143	1
252	1	106	0
255	0	147	1
267	0	126	0
291	2	140	0

	oldpeak	slope	Number_Major_Vessel	thal	target

```
52        1.8     1                    3     3       1
92        0.0     2                    4     2       1
97        0.1     2                    3     3       1
99        0.0     2                    3     2       1
158       0.4     1                    4     3       1
163       0.0     2                    4     2       1
165       1.5     1                    3     2       0
181       1.0     1                    3     3       0
191       2.2     1                    3     3       0
204       6.2     0                    3     3       0
208       2.0     1                    3     3       0
217       1.8     2                    3     3       0
220       4.0     1                    3     3       0
231       1.0     1                    3     3       0
234       2.4     1                    3     2       0
238       0.0     2                    3     2       0
247       0.0     1                    3     1       0
249       2.0     1                    3     3       0
250       4.2     1                    3     3       0
251       0.1     1                    4     3       0
252       1.9     1                    3     2       0
255       0.0     1                    3     3       0
267       0.8     2                    3     2       0
291       4.4     0                    3     1       0
Total Outliers in Number of Major Vessels 24
Total index that has outliers [52, 92, 97, 99, 158, 163, 165, 181, 191, 204,
208, 217, 220, 231, 234, 238, 247, 249, 250, 251, 252, 255, 267, 291]
```

[71]:

```python
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (300, 14)
Shape After Dropping Outlier Rows: (276, 14)
```

[72]:

```python
# check for oldpeak
lower_range,upper_range =outliers(health['oldpeak'])
low_val= health[health['oldpeak'].values<lower_range]
print(low_val)
print('-'*30)
up_val=health[health['oldpeak'].values>upper_range]
print(up_val)

lower_outlier= low_val.value_counts().sum(axis=0)
```

```
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in oldpeak",Total_outliers)

low_ind =list(health[health['oldpeak']<lower_range].index)
up_ind =list(health[health['oldpeak']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
------------------------------
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
101   59    1           3       178   270                    0
221   55    1           0       140   217                    0

     Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
101                            0             145                        0
221                            1             111                        1

     oldpeak  slope  Number_Major_Vessel  thal  target
101      4.2      0                    0     3       1
221      5.6      0                    0     3       0
Total Outliers in oldpeak 2
Total index that has outliers [101, 221]
```

[73]:
```
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (276, 14)
Shape After Dropping Outlier Rows: (274, 14)
```

[74]:
```
# check for chol
lower_range, upper_range =outliers(health['chol'])
low_val =health[health['chol'].values<lower_range]
print(low_val)
print('--'*30)
up_val =health[health['chol'].values>upper_range]
print(up_val)
lower_outlier= low_val.value_counts().sum(axis=0)
```

```
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in chol",Total_outliers)

low_ind =list(health[health['chol']<lower_range].index)
up_ind =list(health[health['chol']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
------------------------------------------------------------
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
28    65    0           2       140   417                    1
85    67    0           2       115   564                    0
96    62    0           0       140   394                    0
246   56    0           0       134   409                    0

     Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
28                             0             157                        0
85                             0             160                        0
96                             0             157                        0
246                            0             150                        1

     oldpeak  slope  Number_Major_Vessel  thal  target
28       0.8      2                    1     2       1
85       1.6      1                    0     3       1
96       1.2      1                    0     2       1
246      1.9      1                    2     3       0
Total Outliers in chol 4
Total index that has outliers [28, 85, 96, 246]
```

[75]:
```
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (274, 14)
Shape After Dropping Outlier Rows: (270, 14)
```

[76]:
```
# check for Fasting_Blood_Sugar
lower_range,upper_range =outliers(health['Fasting_Blood_Sugar'])
```

```
low_val =health[health['Fasting_Blood_Sugar'].values<lower_range]
print(low_val)
print('--'*30)
up_val =health[health['Fasting_Blood_Sugar'].values>upper_range]
print(up_val)
lower_outlier= low_val.value_counts().sum(axis=0)
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in Fasting_Blood_Sugar",Total_outliers)

low_ind =list(health[health['Fasting_Blood_Sugar']<lower_range].index)
up_ind =list(health[health['Fasting_Blood_Sugar']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
------------------------------------------------------------
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
0     63    1           3       145   233                    1
8     52    1           2       172   199                    1
14    58    0           3       150   283                    1
23    61    1           2       150   243                    1
26    59    1           2       150   212                    1
29    53    1           2       130   197                    1
36    54    0           2       135   304                    1
60    71    0           2       110   265                    1
64    58    1           2       140   211                    1
76    51    1           2       125   245                    1
78    52    1           1       128   205                    1
83    52    1           3       152   298                    1
87    46    1           1       101   197                    1
90    48    1           2       124   255                    1
93    54    0           1       132   288                    1
103   42    1           2       120   240                    1
106   69    1           3       160   234                    1
111   57    1           2       150   126                    1
136   60    0           2       120   178                    1
137   62    1           1       128   208                    1
169   53    1           0       140   203                    1
170   56    1           2       130   256                    1
176   60    1           0       117   230                    1
197   67    1           0       125   254                    1
203   68    1           2       180   274                    1
```

214	56	1	0	125	249	1
215	43	0	0	132	341	1
219	48	1	0	130	256	1
222	65	1	3	138	282	1
223	56	0	0	200	288	1
260	66	0	0	178	228	1
269	56	1	0	130	283	1
278	58	0	1	136	319	1
282	59	1	2	126	218	1
292	58	0	0	170	225	1
297	59	1	0	164	176	1
300	68	1	0	144	193	1

	Electrocardiographic_result	Max_Heart_Rate	Exercise_Induced_Angina \
0	0	150	0
8	1	162	0
14	0	162	0
23	1	137	1
26	1	157	0
29	0	152	0
36	1	170	0
60	0	130	0
64	0	165	0
76	0	166	0
78	1	184	0
83	1	178	0
87	1	156	0
90	1	175	0
93	0	159	1
103	1	194	0
106	0	131	0
111	1	173	0
136	1	96	0
137	0	140	0
169	0	155	1
170	0	142	1
176	1	160	1
197	1	163	0
203	0	150	1
214	0	144	1
215	0	136	1
219	0	150	1
222	0	174	0
223	0	133	1
260	1	165	1
269	0	103	1
278	0	152	0
282	1	134	0

```
292                           0             146                        1
297                           0              90                        0
300                           1             141                        0

     oldpeak  slope  Number_Major_Vessel  thal  target
0        2.3      0                    0     1       1
8        0.5      2                    0     3       1
14       1.0      2                    0     2       1
23       1.0      1                    0     2       1
26       1.6      2                    0     2       1
29       1.2      0                    0     2       1
36       0.0      2                    0     2       1
60       0.0      2                    1     2       1
64       0.0      2                    0     2       1
76       2.4      1                    0     2       1
78       0.0      2                    0     2       1
83       1.2      1                    0     3       1
87       0.0      2                    0     3       1
90       0.0      2                    2     2       1
93       0.0      2                    1     2       1
103      0.8      0                    0     3       1
106      0.1      1                    1     2       1
111      0.2      2                    1     3       1
136      0.0      2                    0     2       1
137      0.0      2                    0     2       1
169      3.1      0                    0     3       0
170      0.6      1                    1     1       0
176      1.4      2                    2     3       0
197      0.2      1                    2     3       0
203      1.6      1                    0     3       0
214      1.2      1                    1     2       0
215      3.0      1                    0     3       0
219      0.0      2                    2     3       0
222      1.4      1                    1     2       0
223      4.0      0                    2     3       0
260      1.0      1                    2     3       0
269      1.6      0                    0     3       0
278      0.0      2                    2     2       0
282      2.2      1                    1     1       0
292      2.8      1                    2     1       0
297      1.0      1                    2     1       0
300      3.4      1                    2     3       0
Total Outliers in Fasting_Blood_Sugar 37
Total index that has outliers [0, 8, 14, 23, 26, 29, 36, 60, 64, 76, 78, 83, 87,
90, 93, 103, 106, 111, 136, 137, 169, 170, 176, 197, 203, 214, 215, 219, 222,
223, 260, 269, 278, 282, 292, 297, 300]
```

[77]:
```python
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (270, 14)
Shape After Dropping Outlier Rows: (233, 14)
```

[78]:
```python
# Check for trestbps
lower_range,upper_range =outliers(health['trestbps'])
low_val =health[health['trestbps'].values<lower_range]
print(low_val)

up_val =health[health['trestbps'].values>upper_range]
print(up_val)
lower_outlier= low_val.value_counts().sum(axis=0)
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in trestbps",Total_outliers)

low_ind =list(health[health['trestbps']<lower_range].index)
up_ind =list(health[health['trestbps']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
110   64    0           0       180   325                    0
241   59    0           0       174   249                    0
248   54    1           1       192   283                    0
266   55    0           0       180   327                    0

     Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
110                            1             154                        1
241                            1             143                        1
248                            0             195                        0
266                            2             117                        1

     oldpeak  slope  Number_Major_Vessel  thal  target
110      0.0      2                    0     2       1
241      0.0      1                    0     2       0
248      0.0      2                    1     3       0
```

```
266      3.4      1                     0     2       0
Total Outliers in trestbps 4
Total index that has outliers [110, 241, 248, 266]
```

[79]:
```
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (233, 14)
Shape After Dropping Outlier Rows: (229, 14)
```

[80]:
```
# Check for Max_Heart_Rate
lower_range,upper_range =outliers(health['Max_Heart_Rate'])
low_val =health[health['Max_Heart_Rate'].values<lower_range]
print(low_val)

up_val =health[health['Max_Heart_Rate'].values>upper_range]
print(up_val)
lower_outlier= low_val.value_counts().sum(axis=0)
upper_outlier =up_val.value_counts().sum(axis=0)
Total_outliers =lower_outlier+upper_outlier
print("Total Outliers in Max_Heart_Rate",Total_outliers)

low_ind =list(health[health['Max_Heart_Rate']<lower_range].index)
up_ind =list(health[health['Max_Heart_Rate']>upper_range].index)
total_ind= list(low_ind+up_ind)
print("Total index that has outliers",total_ind)
```

```
     age  sex  Chest_Pain  trestbps  chol  Fasting_Blood_Sugar  \
272   67    1           0       120   237                    0

     Electrocardiographic_result  Max_Heart_Rate  Exercise_Induced_Angina  \
272                            1              71                        0

     oldpeak  slope  Number_Major_Vessel  thal  target
272      1.0      1                    0     2       0
Empty DataFrame
Columns: [age, sex, Chest_Pain, trestbps, chol, Fasting_Blood_Sugar,
Electrocardiographic_result, Max_Heart_Rate, Exercise_Induced_Angina, oldpeak,
slope, Number_Major_Vessel, thal, target]
Index: []
Total Outliers in Max_Heart_Rate 1
Total index that has outliers [272]
```

[81]:
```
# lets drop the outliers for better prediction and accuracy of the data
print("Shape Before Dropping Outlier Rows:", health.shape)

health.drop(total_ind, inplace = True)

print("Shape After Dropping Outlier Rows:", health.shape)
```

```
Shape Before Dropping Outlier Rows: (229, 14)
Shape After Dropping Outlier Rows: (228, 14)
```

[82]:
```
correlation = health['chol'].corr(health['target'])
correlation
```

[82]:
```
-0.0673214602790367
```

[83]:
```
health.columns
```

[83]:
```
Index(['age', 'sex', 'Chest_Pain', 'trestbps', 'chol', 'Fasting_Blood_Sugar',
       'Electrocardiographic_result', 'Max_Heart_Rate',
       'Exercise_Induced_Angina', 'oldpeak', 'slope', 'Number_Major_Vessel',
       'thal', 'target'],
      dtype='object')
```

10 f. Describe the relationship between cholesterol levels and a target variable

[84]:
```
#lets find the correlation between cholestrol level, target varible by using␣
↪correlation function.
```

[85]:
```
corr_1 =health['chol'].corr(health['target'])
corr_1
```

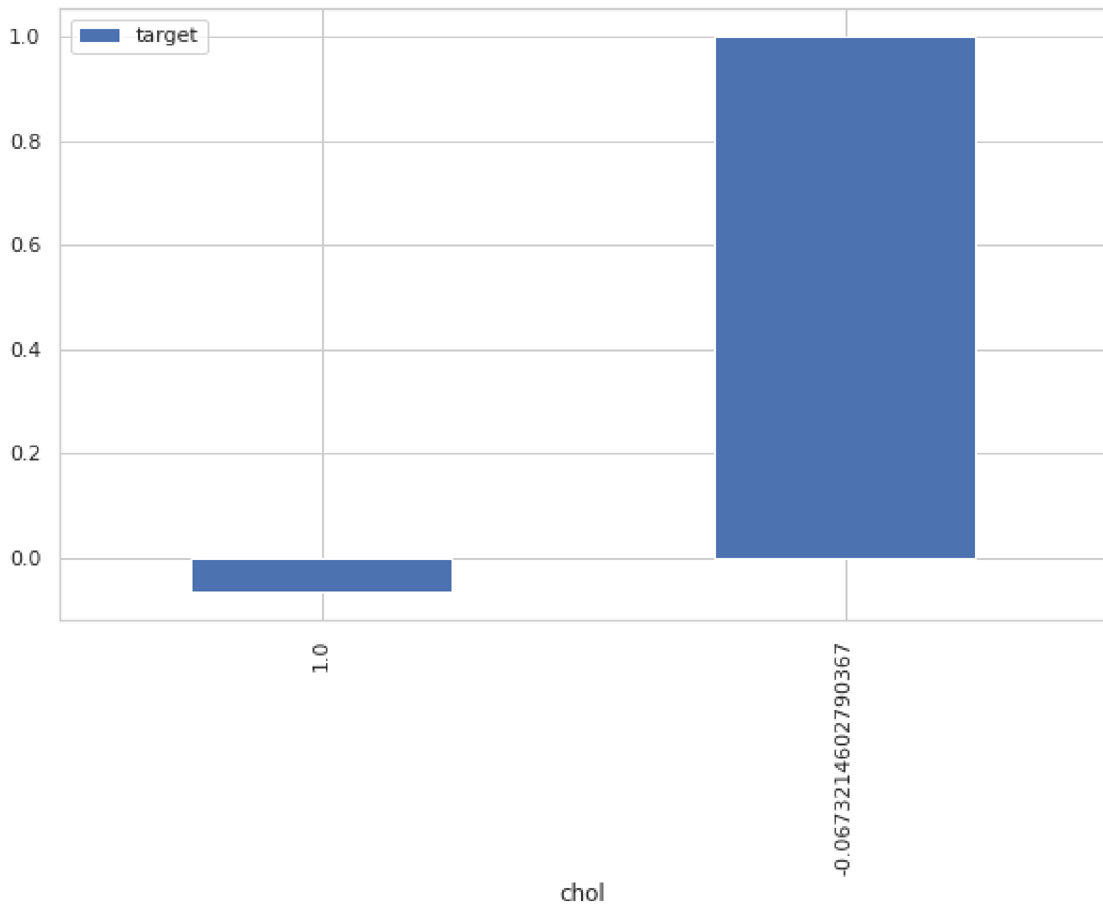
[85]:
```
-0.0673214602790367
```

[86]:
```
corr=health[["chol","target"]].corr()
print(corr)
```

```
            chol    target
chol    1.000000 -0.067321
target -0.067321  1.000000
```

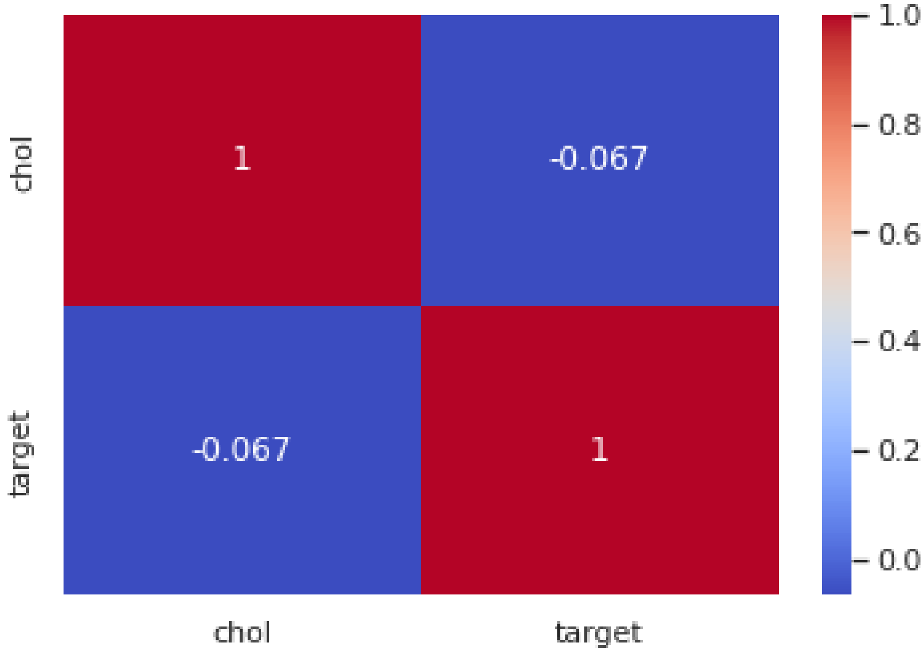
[87]:
```
# lets plot the coorelation bar plot and heat map
corr.plot(kind='bar', x='chol',y='target',figsize=(10,6))
```

[87]:
```
<AxesSubplot:xlabel='chol'>
```




```
[88]: # lets plot the heat map
      sns.heatmap(corr, annot =True,cmap='coolwarm')
```

```
[88]: <AxesSubplot:>
```



interpretation: it has a correlation of 81% betwen the output target and cholestrol for the occurance of heart attack.

11 More Accurate relationship between cholestral and target varial let us use auto correlation factors and plot the graph.

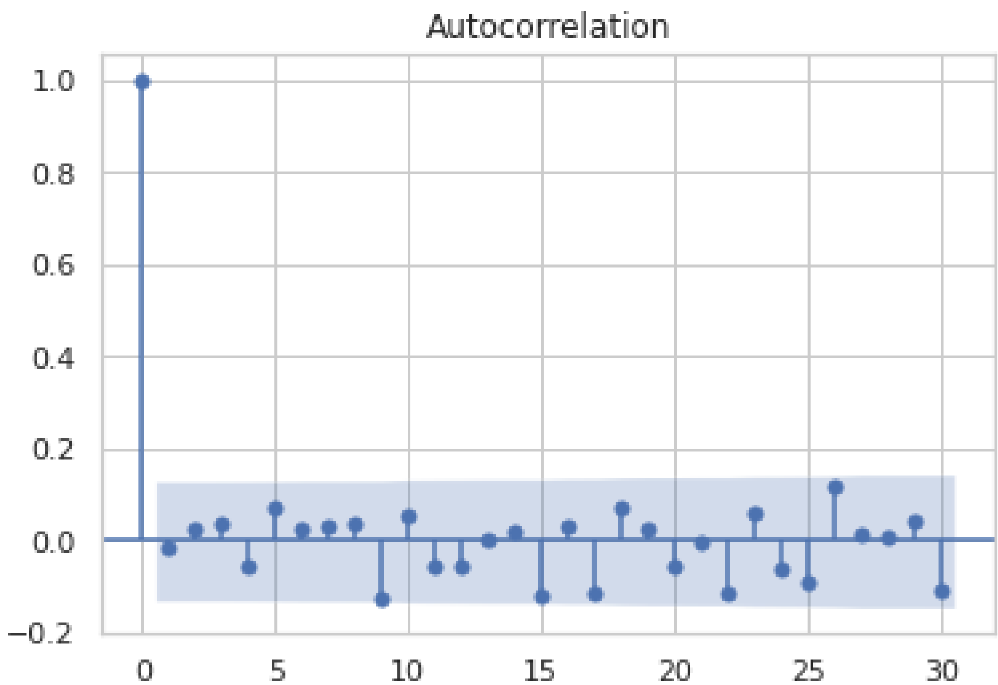
[89]:
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.api import qqplot
import statsmodels.api as sm
from scipy import stats
import statsmodels.tsa.api as tsa
import statsmodels.api as sm
from scipy import stats
```

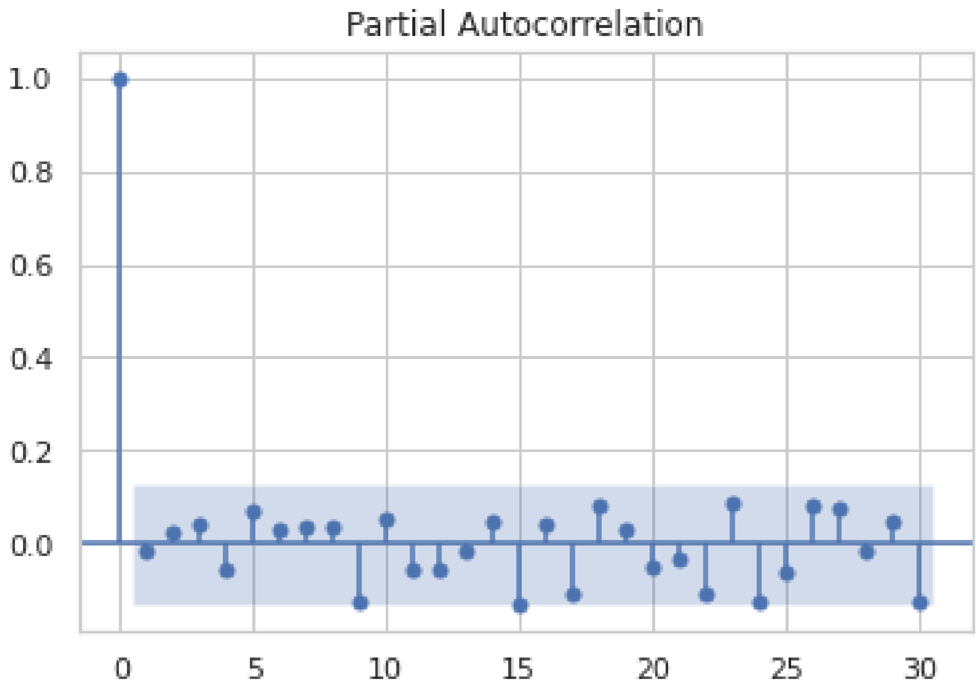
[90]:
```
crosstab =pd.crosstab(index=health['target'],columns=health['chol'] )
crosstab
```

[90]:
```
chol     131  141  149  157  160  166  167  168  169  172  …  321  325  326  \
target                                                     …
0          1    0    0    0    0    1    1    0    1    1  …    0    0    1
1          0    1    1    1    1    0    0    1    0    0  …    1    1    0
```

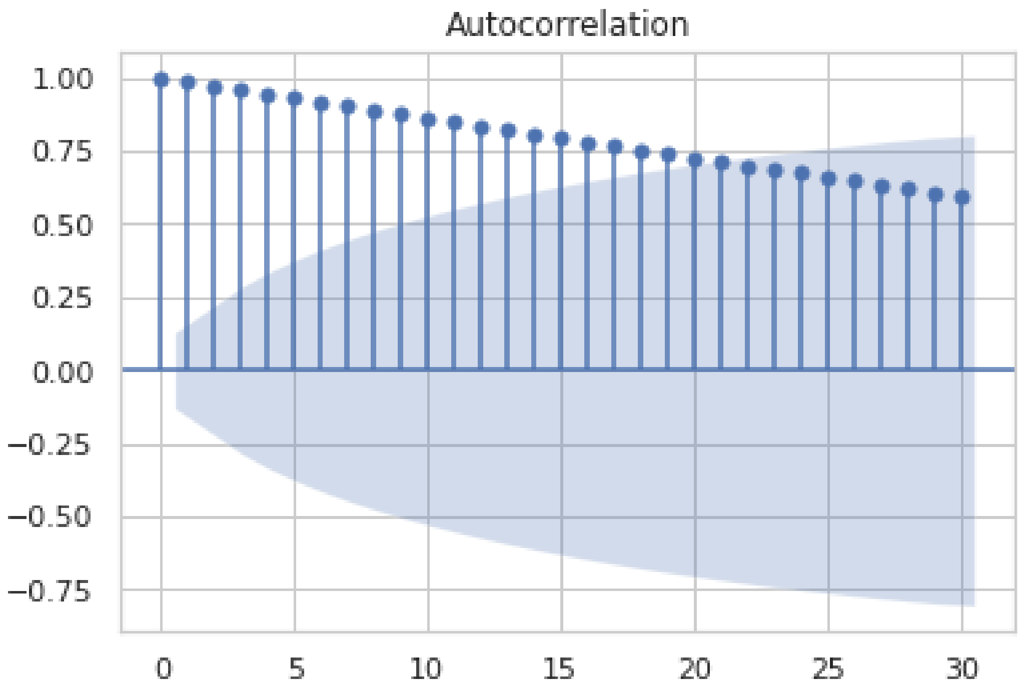
```
chol     330  335  340  342  353  354  360
target
0          1    2    0    0    1    0    0
1          0    0    1    1    0    1    1

[2 rows x 132 columns]
```

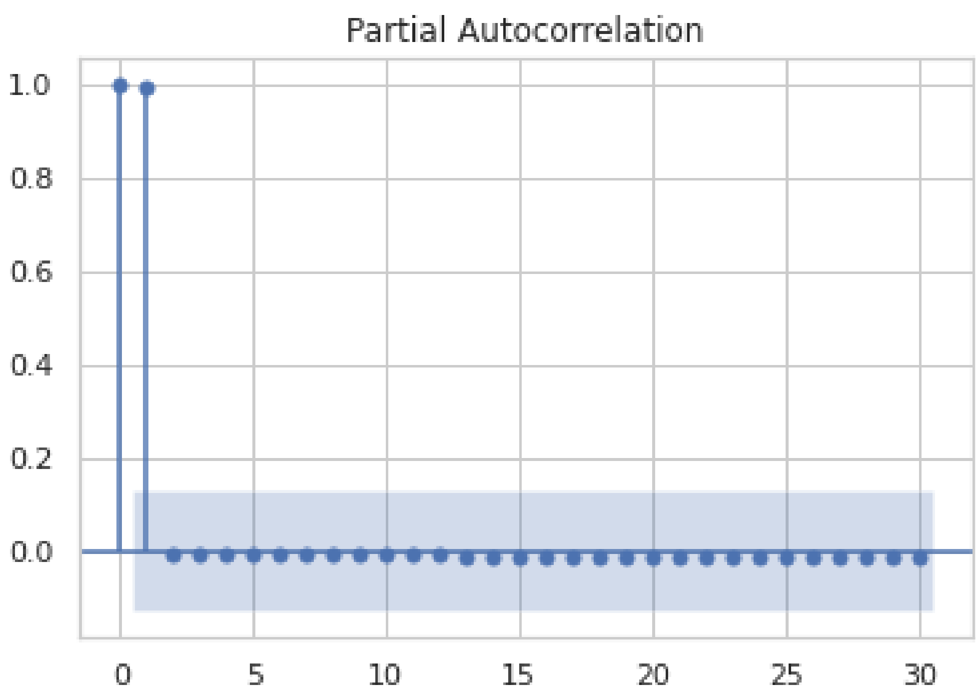
[91]:
```
plot_acf(health.chol,lags=30)
plt.show()

plot_pacf(health.chol,lags=30)
plt.show()
```





[92]:
```
plot_acf(health.target,lags=30)
plt.show()

plot_pacf(health.target,lags=30)
plt.show()
```

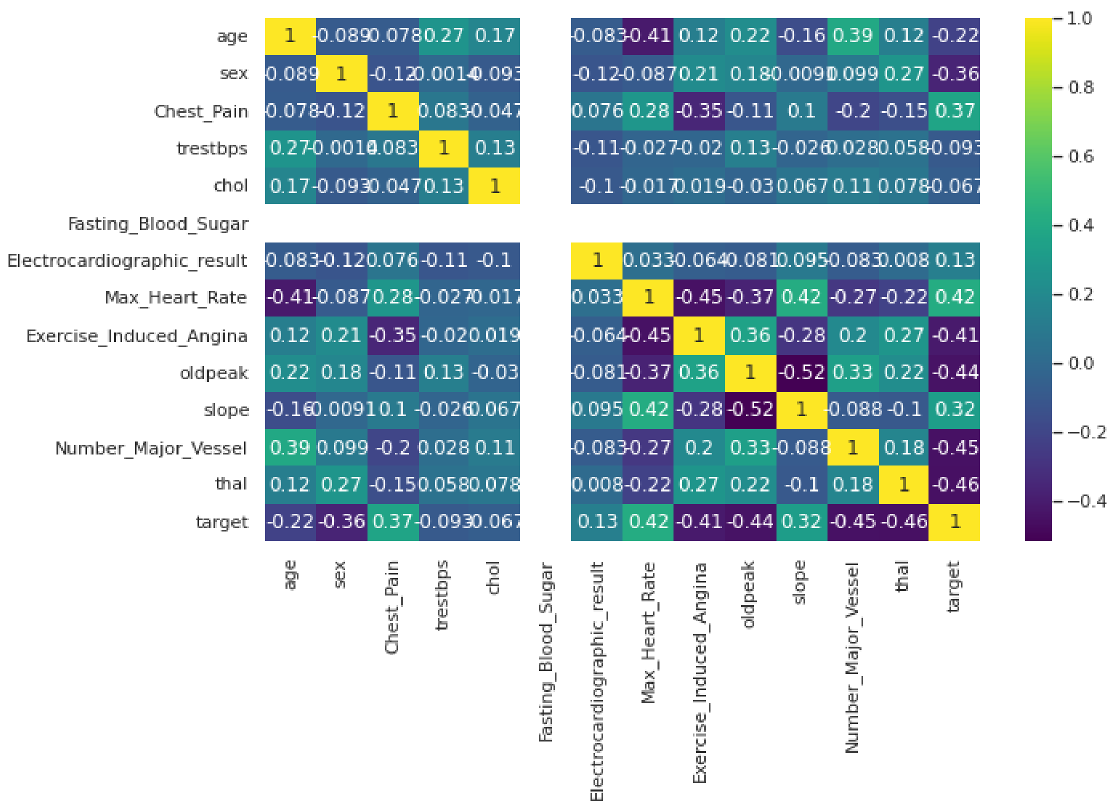




We can see that ACF plot decreasing exponentially and PACF plot has just the spike on lag 1. Hence, this is a ARMA(1,0) model.(AutoRegression Moving Average)

```
[93]: # lets plot heat map for entire health dataset
       #number of variables for heatmap
      cols = health.corr()
      plt.figure(figsize=(10,6))
      sns.heatmap(cols, annot=True, cmap = 'viridis')
```

[93]: <AxesSubplot:>



12 3. Build a baseline model to predict the risk of a heart attack using a logistic regression and random forest and explore the results while using correlation analysis and logistic regression (leveraging standard error and p-values from statsmodels) for feature selection

```
[102]: # lets find the p value.
       from scipy.stats import chi2_contingency
       from scipy.stats import ttest_ind
       from statsmodels.formula.api import ols
```

```
from statsmodels.stats import weightstats as stests
from scipy import stats
import statsmodels.api as sm
```

[103]:
```
health.columns
```

[103]:
```
Index(['age', 'sex', 'Chest_Pain', 'trestbps', 'chol', 'Fasting_Blood_Sugar',
       'Electrocardiographic_result', 'Max_Heart_Rate',
       'Exercise_Induced_Angina', 'oldpeak', 'slope', 'Number_Major_Vessel',
       'thal', 'target'],
      dtype='object')
```

[148]:
```
# Create the formula string
formula = "target ~ age + sex + Chest_Pain + trestbps + chol +␣
↪Fasting_Blood_Sugar + Electrocardiographic_result + Max_Heart_Rate +␣
↪Exercise_Induced_Angina + oldpeak + slope + Number_Major_Vessel + thal"

# Fit the OLS model
model = ols(formula, data=health).fit()

# Print the overall model statistics
print(f"Overall model F({model.df_model:.0f}, {model.df_resid:.0f}) = {model.
↪fvalue:.3f}, p = {model.f_pvalue:.4f}")

# Perform ANOVA
res = sm.stats.anova_lm(model, typ=2)
print(res)
```

```
Overall model F(12, 215) = 21.058, p = 0.0000
                                sum_sq     df          F    PR(>F)
age                           0.068365    1.0   0.575286  0.448997
sex                           1.844672    1.0  15.522754  0.000110
Chest_Pain                    1.519225    1.0  12.784146  0.000432
trestbps                      0.244397    1.0   2.056580  0.153003
chol                          0.056249    1.0   0.473329  0.492201
Fasting_Blood_Sugar           1.126735    1.0   9.481373  0.002346
Electrocardiographic_result   0.072670    1.0   0.611512  0.435080
Max_Heart_Rate                0.399642    1.0   3.362950  0.068061
Exercise_Induced_Angina       0.162281    1.0   1.365581  0.243867
oldpeak                       0.328765    1.0   2.766530  0.097712
slope                         0.709848    1.0   5.973308  0.015330
Number_Major_Vessel           2.895271    1.0  24.363453  0.000002
thal                          2.826315    1.0  23.783189  0.000002
Residual                     25.549880  215.0        NaN       NaN
```

Interpretation: we have seen that pvalue is 0 therefore there are correlation between the target variable nd dependent other variables.

13 lets use the Machine Learning Function for Predicting the Accuracy score.

[142]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import␣
↪accuracy_score,precision_score,f1_score,confusion_matrix,mean_squared_error,r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score, KFold
```

[99]:
```
# lets split the data into trin and test set, eliminate the target column from␣
↪x and put into y.
x_data= health.drop('target',axis=1)
y_data= health['target']
```

[104]:
```
sc =StandardScaler()
```

[106]:
```
x_data= sc.fit_transform(x_data)
x_data
```

[106]:
```
array([[-1.77367443,  0.69319512,  1.03837841, …, -2.47463235,
        -0.68550748, -0.56484923],
       [-1.33930518, -1.44259526,  0.05601211, …,  0.93452883,
        -0.68550748, -0.56484923],
       [ 0.2895795 ,  0.69319512,  0.05601211, …,  0.93452883,
        -0.68550748, -0.56484923],
       …,
       [-0.90493593,  0.69319512,  2.0207447 , …, -0.77005176,
        -0.68550748,  1.22384   ],
       [ 0.39817181,  0.69319512, -0.92635418, …, -0.77005176,
         0.74839807,  1.22384   ],
       [ 0.39817181, -1.44259526,  0.05601211, …, -0.77005176,
         0.74839807, -0.56484923]])
```

[107]:
```
x_train,x_test,y_train,y_test␣
↪=train_test_split(x_data,y_data,random_state=30,test_size=0.3)
```

[108]:
```
print("the train set for x_data",x_train.shape)
print("the test set for x_data",x_test.shape)
print("the train set for y_data",y_train.shape)
print("the test set for y_data",y_test.shape)
```

```
the train set for x_data (159, 13)
```

```
the test set for x_data (69, 13)
the train set for y_data (159,)
the test set for y_data (69,)
```

14 find the model prediction using RandomForest Classifier

[112]:
```
#lets use GridSearchCv for best estimator prediction.
RFC= RandomForestClassifier(random_state=30)
parm={'n_estimators':[10,15,25,100], 'max_depth':[3,5,7,10]}
grid= zip([RFC],[parm])
best= None

for i,j in grid:
    a= GridSearchCV(i,param_grid=j,cv=3, n_jobs=1)
    a.fit(x_train,y_train)
    if best is None:
        best=a
print("Best CV score",best.best_score_)
print("Model Parameter",best.best_params_)
print("Best Estimator",best.best_estimator_)
```

```
Best CV score 0.8742138364779874
Model Parameter {'max_depth': 3, 'n_estimators': 100}
Best Estimator RandomForestClassifier(max_depth=3, random_state=30)
```

[114]:
```
rfc = best.best_estimator_
Model =rfc.fit(x_train,y_train)
Model
```

[114]:
```
RandomForestClassifier(max_depth=3, random_state=30)
```

[118]:
```
y_pred =rfc.predict(x_test)
y_pred
```

[118]:
```
array([0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 0])
```

[143]:
```
# Using the cross validation value to predict the score.
seed=7
kfold=KFold(n_splits=5,random_state=7,shuffle=True)
print(cross_val_score(rfc,x_data,y_data,cv=kfold,scoring='accuracy'))
result=cross_val_score(rfc,x_data,y_data,cv=kfold,scoring='accuracy')
print(result*100)
```

```
[0.7826087  0.86956522 0.80434783 0.82222222 0.82222222]
[78.26086957 86.95652174 80.43478261 82.22222222 82.22222222]
```

[119]:
```python
#Check the Accuracy score and confussion matrix

print("The Accuracy Score by Using Random Forest␣
 ↪Classifier",accuracy_score(y_test,y_pred))
```

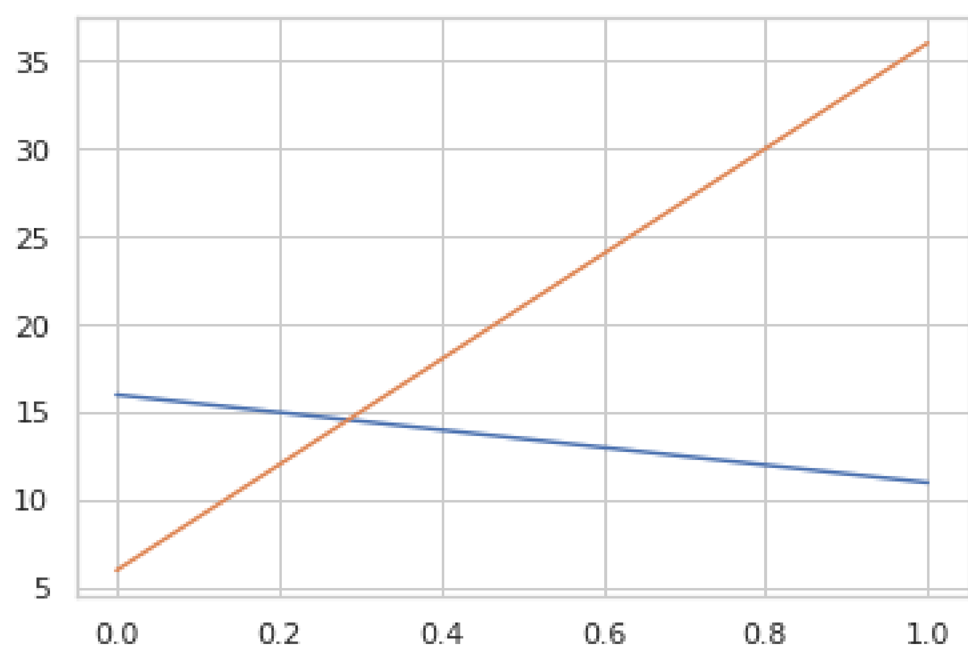
```
The Accuracy Score by Using Random Forest Classifier 0.7536231884057971
```

[122]:
```python
Confusion_Matrix =confusion_matrix(y_pred,y_test)
Confusion_Matrix
```

[122]:
```
array([[16,  6],
       [11, 36]])
```

[123]:
```python
# plot the graph for confusion matrix
plt.plot(Confusion_Matrix)
```

[123]:
```
[<matplotlib.lines.Line2D at 0x7f793d04c390>,
 <matplotlib.lines.Line2D at 0x7f793d03f650>]
```



[124]:
```python
# lets calculate the precision score, f1 score.
```

```
print("The Precision Score by Using Random Forest␣
↪Classifier",precision_score(y_test,y_pred))
print()
print("The F1 Score by Using Random Forest Classifier",f1_score(y_test,y_pred))
```

```
The Precision Score by Using Random Forest Classifier 0.7659574468085106

The F1 Score by Using Random Forest Classifier 0.8089887640449439
```

[128]:
```
# lets calulate the accuracy score by using logistic regression
lr =LogisticRegression(random_state=30)
lr
```

[128]:
```
LogisticRegression(random_state=30)
```

[129]:
```
lr.fit(x_train,y_train)
```

[129]:
```
LogisticRegression(random_state=30)
```

[131]:
```
y_pred_1 =lr.predict(x_test)
y_pred_1
```

[131]:
```
array([0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 0])
```

[133]:
```
#Check the Accuracy score and confussion matrix

print("The Accuracy Score by Using Logistic␣
↪Regression",accuracy_score(y_test,y_pred_1))
```

```
The Accuracy Score by Using Logistic Regression 0.782608695652174
```

[134]:
```
# lets calculate the precision score, f1 score.



print("The Precision Score by Using Logistic␣
↪Regression",precision_score(y_test,y_pred_1))
print()
print("The F1 Score by Using Logistic Regression",f1_score(y_test,y_pred_1))
```

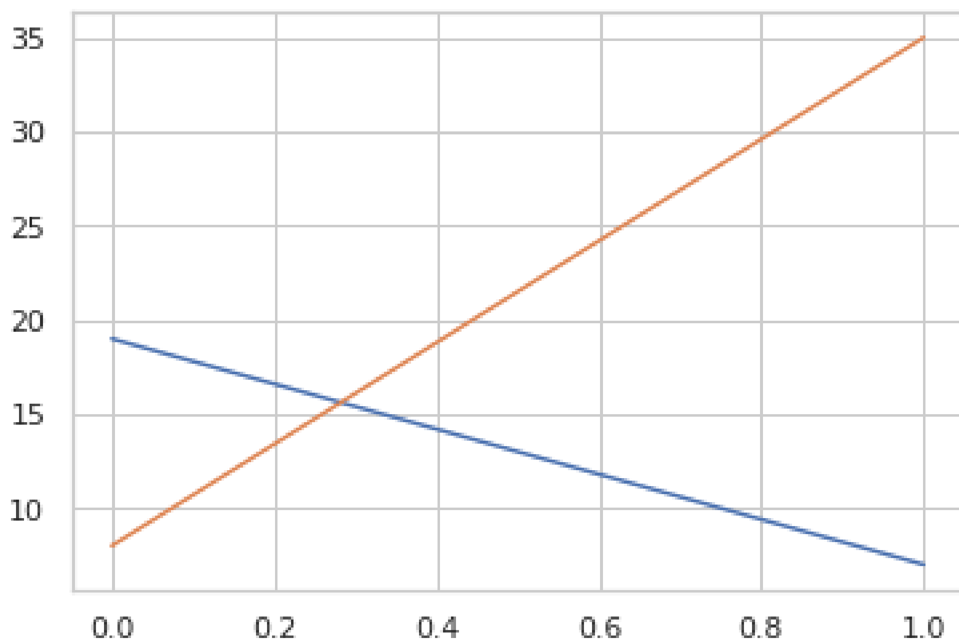
```
The Precision Score by Using Logistic Regression 0.813953488372093

The F1 Score by Using Logistic Regression 0.8235294117647058
```

[136]:
```
Confusion_Matrix_lr =confusion_matrix(y_test,y_pred_1)
Confusion_Matrix_lr
```

[136]:
```
array([[19,  8],
       [ 7, 35]])
```

[137]:
```
plt.plot(Confusion_Matrix_lr)
```

[137]:
```
[<matplotlib.lines.Line2D at 0x7f793cfd2050>,
 <matplotlib.lines.Line2D at 0x7f793d074690>]
```



[139]:
```
print("the mean squared error",mean_squared_error(y_test,y_pred_1))
```

```
the mean squared error 0.21739130434782608
```

[140]:
```
print("the r2 score",r2_score(y_test,y_pred_1))
```

```
the r2 score 0.08730158730158732
```

15 Conclusion:

hence, by using Logistic Regression the accuracy score for having CardioVascular diseases denpending on the fators like Thalseminia, BloodSugar, ECG ,gender and age is 78% where as by using random forest classifier and GridSearch CV we have noticed the accuracy percentage is 75%

```
[ ]:
```

```
[ ]:
```