Health_Care_Project

December 16, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

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

[2]:
```
df = pd.read_excel("HealthCare_dataset.xlsx")
```

[3]:
```
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]:
```
df.tail()
```

[4]:
```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
```

```
300   68   1   0   144   193   1   1   141   0   3.4
301   57   1   0   130   131   0   1   115   1   1.2
302   57   0   1   130   236   0   0   174   0   0.0

     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

[5]:
```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
```

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

[6]:
```python
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]:
```
df.duplicated().sum()
```

[7]:
```
1
```

[8]:
```
df[df.duplicated()]
```

[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]:
```
data = df.drop([164],axis = 0)
```

[10]:
```
data.duplicated().sum()
```

[10]:
```
0
```

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

[11]:
```
data.describe()
```

[11]:
```
             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
```

[12]:
```
data.skew()
```

[12]:
```
age        -0.203743
sex        -0.786120
cp          0.493022
trestbps    0.716541
chol        1.147332
fbs         1.981201
restecg     0.169467
thalach    -0.532671
exang       0.737281
oldpeak     1.266173
slope      -0.503247
ca          1.295738
thal       -0.481232
target     -0.173691
dtype: float64
```

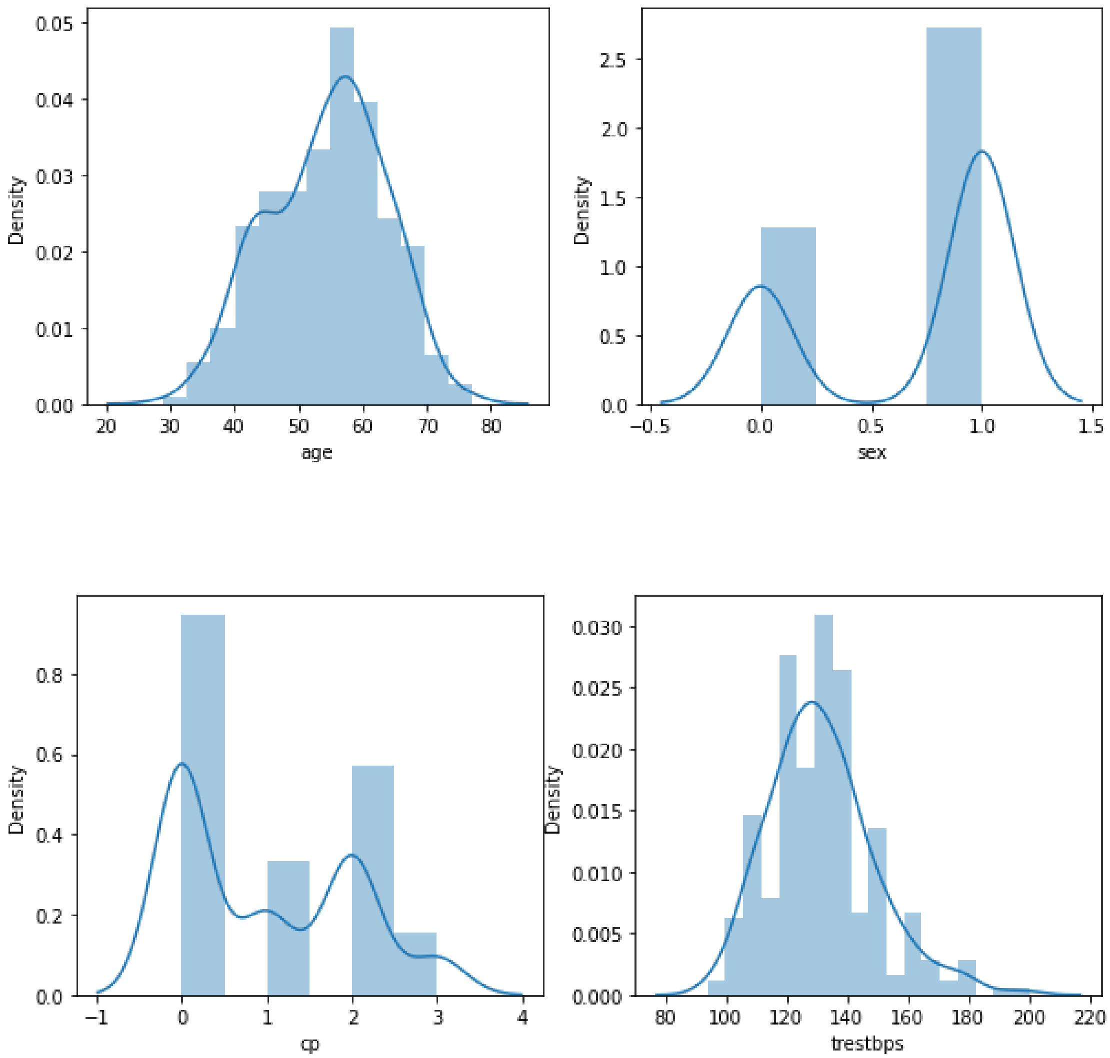
[13]:
```
data.columns
```

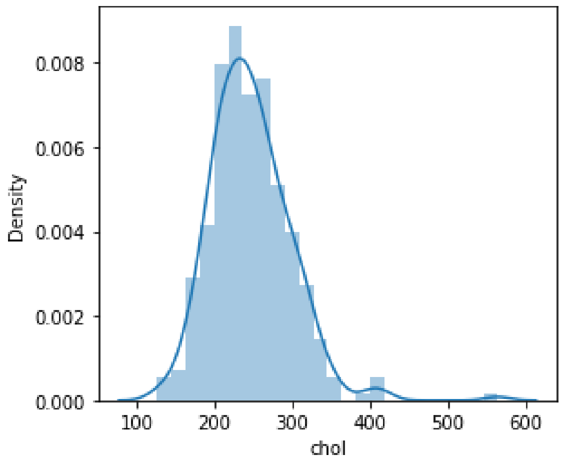
[13]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

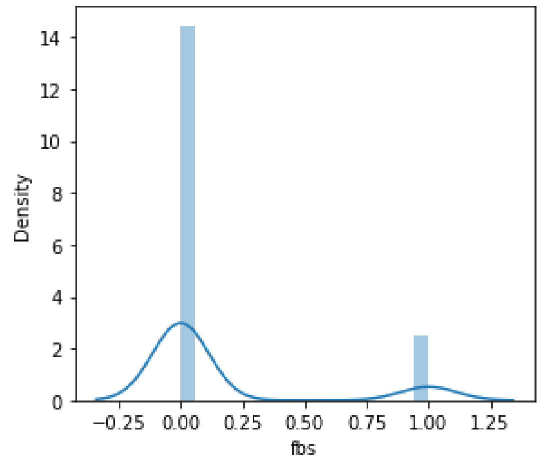
[14]:
```
x= data.columns
```

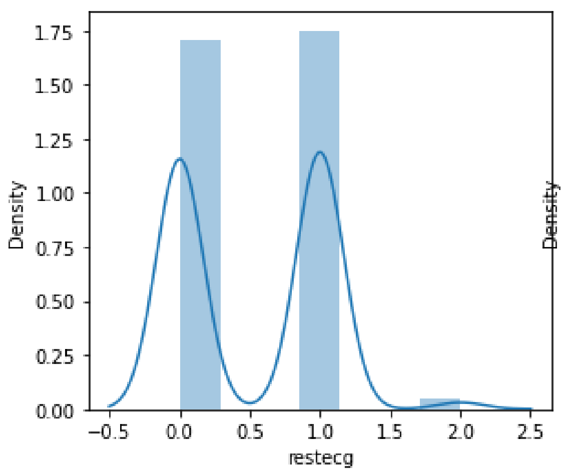
[15]:
```
for i in range(0,len(x)-1,2):
 plt.figure(figsize=(10,4))
```

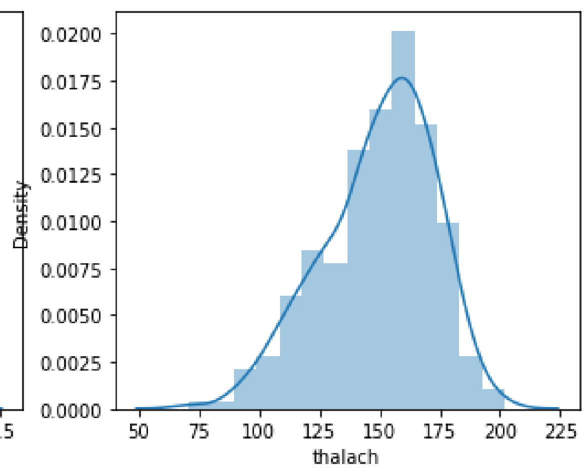
```
plt.subplot(121)
sns.distplot(data[x[i]], kde= True)
plt.subplot(122)
sns.distplot(data[x[i+1]], kde= True)
```

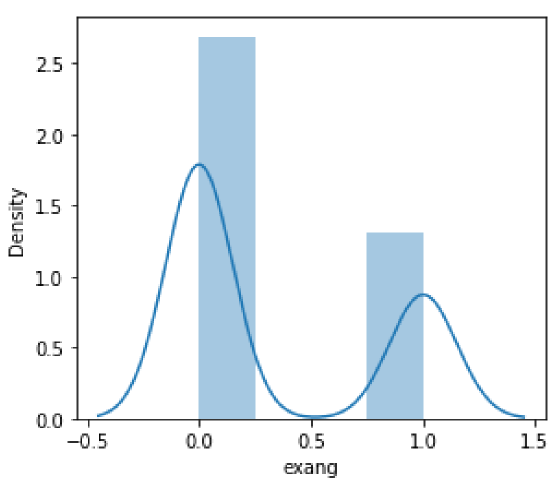


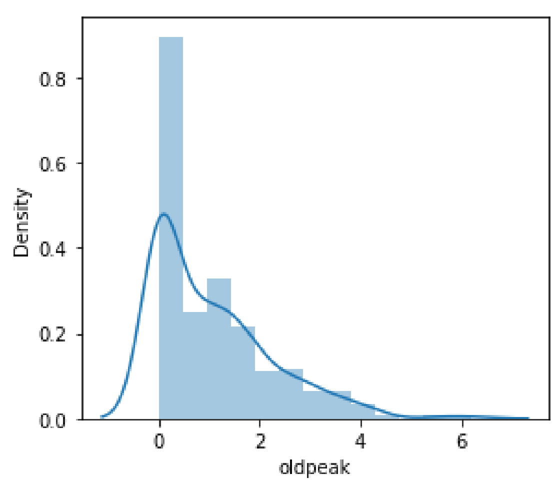

Density
chol

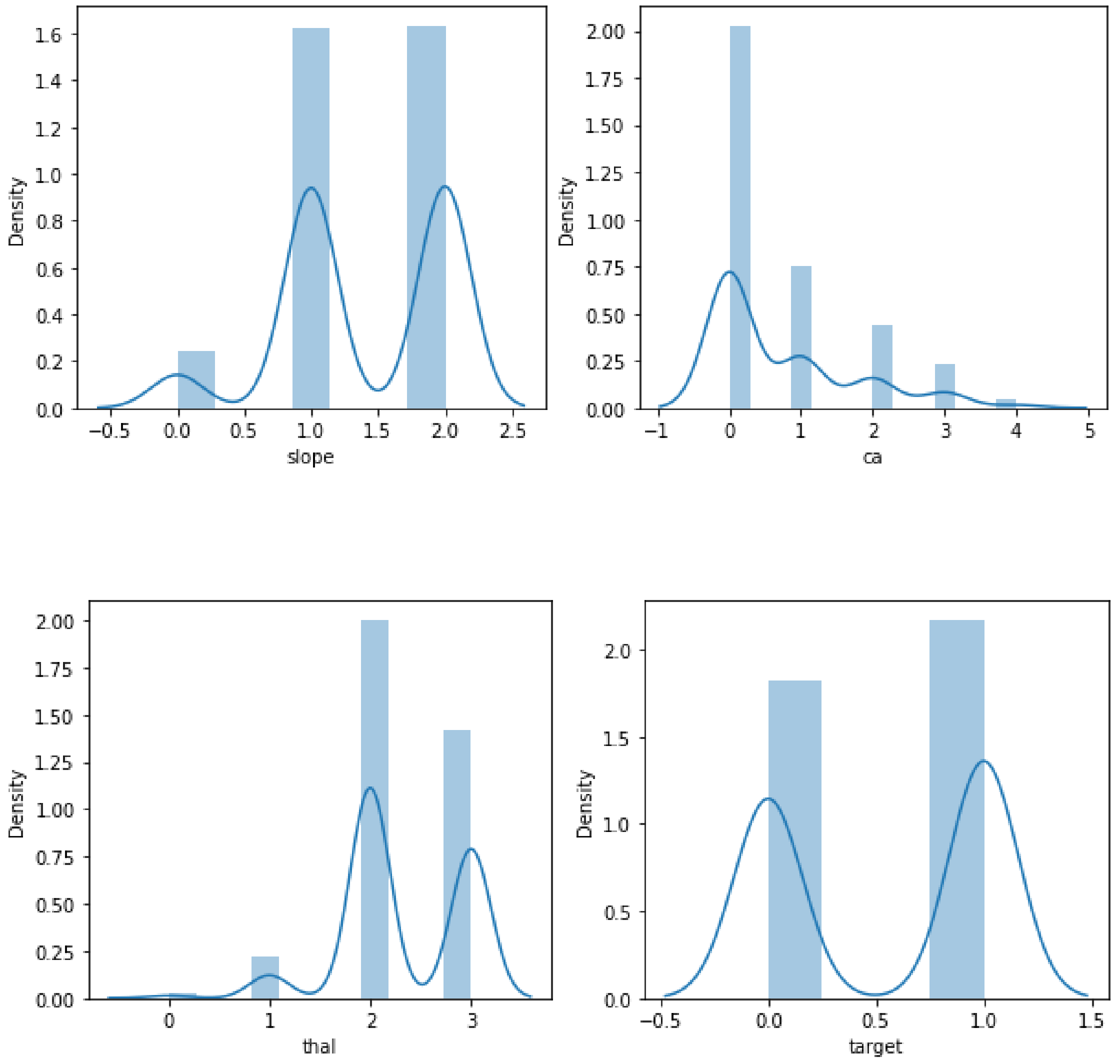

Density
fbs


Density
restecg


Density
thalach


Density
exang


Density
oldpeak



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

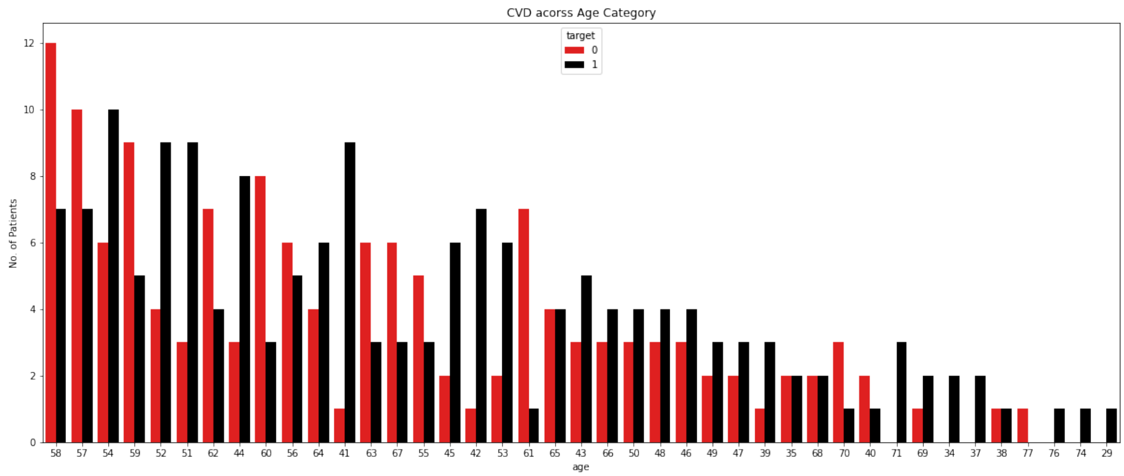
[16]:
```
categorical_data = data.select_dtypes(exclude=[np.number])
```

[17]:
```
categorical_data.sum()
```

[17]:
```
Series([], dtype: float64)
```

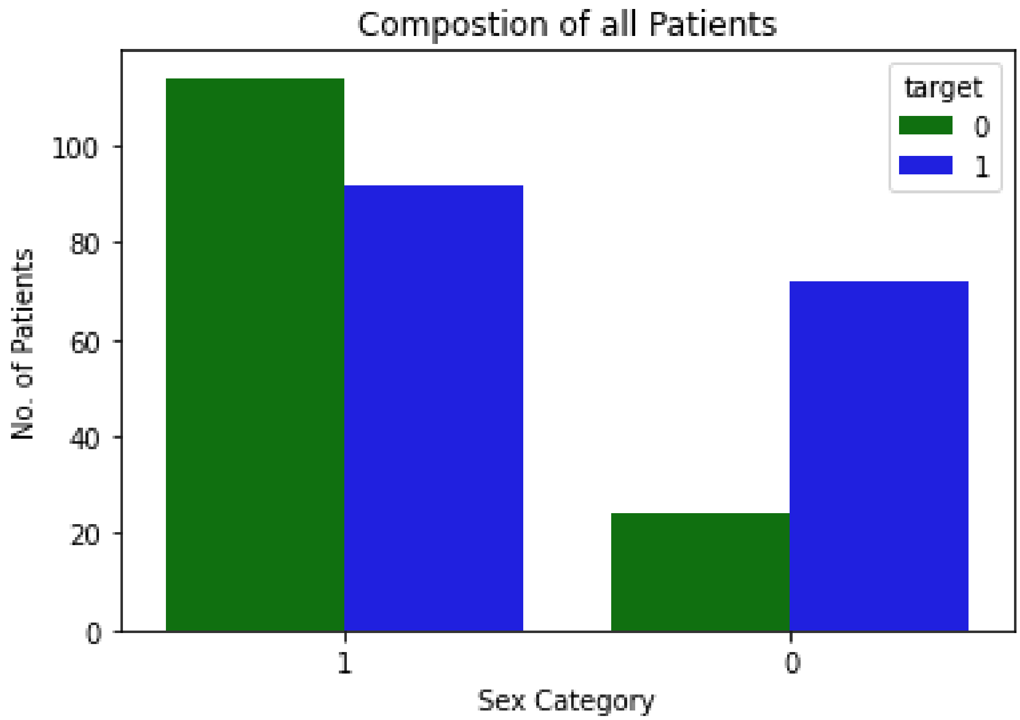
5 Study the occurrence of CVD across the Age category

[18]:
```
plt.figure(figsize=(20,8))
plt.title("CVD acorss Age Category")
sns.
↪countplot(data=data,x="age",hue="target",palette=["red","black"],order=data['age'].
↪value_counts(ascending= False).index)
plt.ylabel("No. of Patients")
plt.show()
```



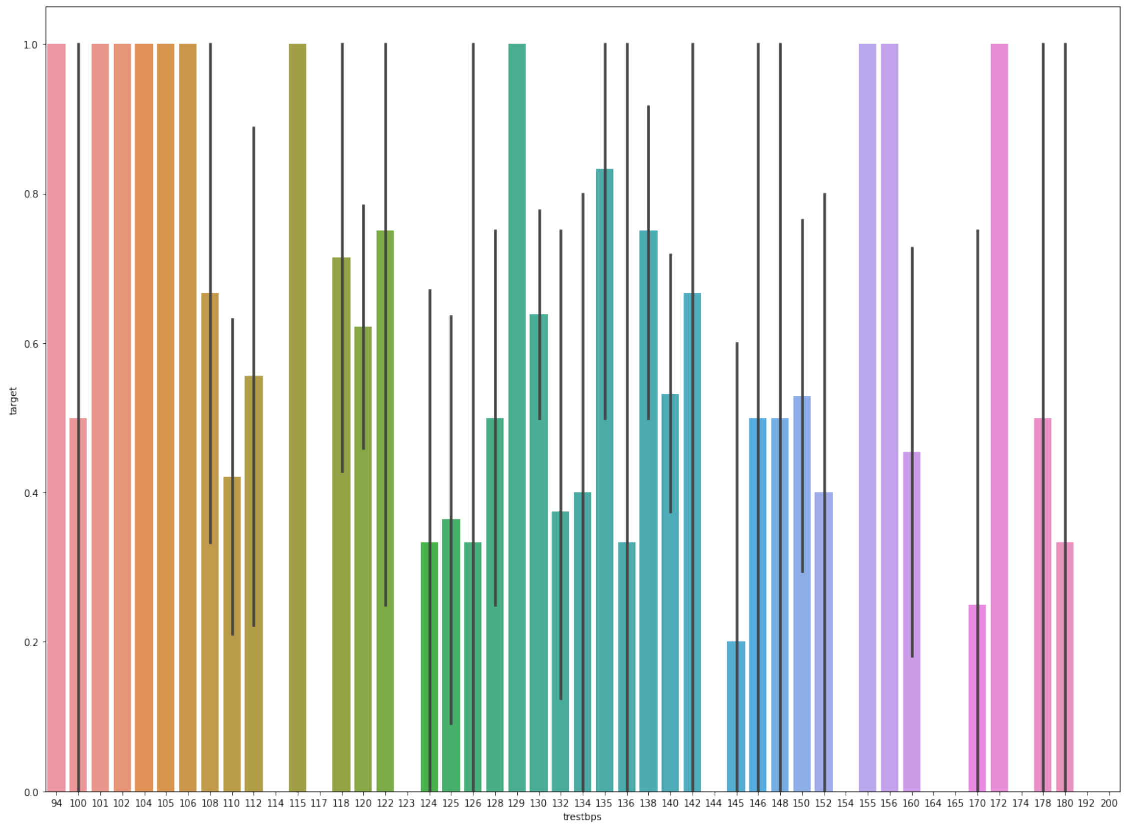
6 Study the composition of all patients with respect to the Sex category

[19]:
```
ax=sns.countplot(data=data,x= data["sex"],hue="target",order=data["sex"].
↪value_counts(ascending= False).index, palette=["green","blue"]);
plt.title("Compostion of all Patients")
plt.xlabel("Sex Category")
plt.ylabel("No. of Patients")
plt.show()
```



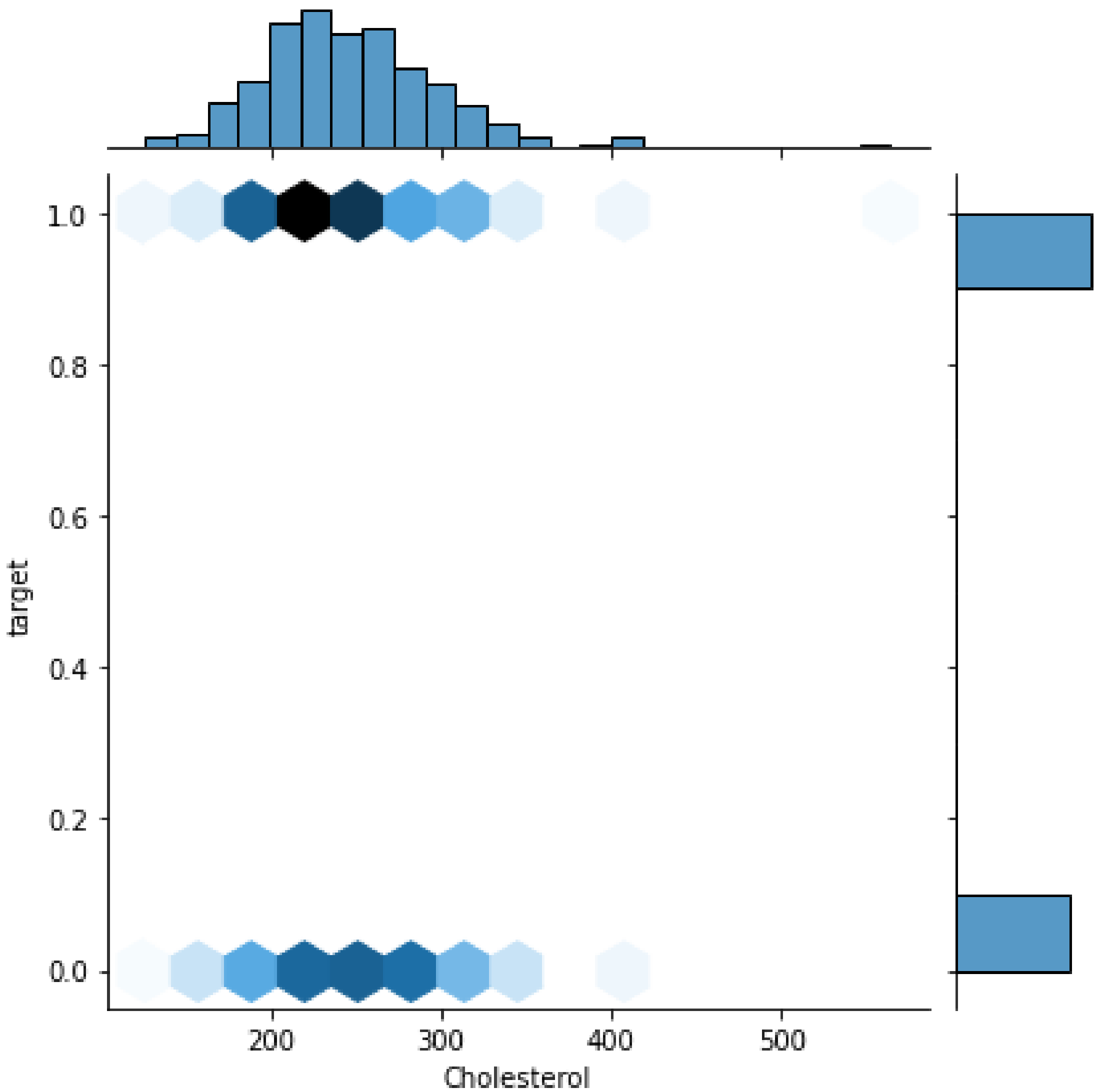
7 Study if one can detect heart attacks based on anomalies in the resting blood pressure (trestbps) of a patient

[20]:
```
plt.figure(figsize=(20,15))
sns.barplot(x="trestbps",y="target",data=data)
plt.show()
```



8 Describe the relationship between cholesterol levels and a target variable

[21]:
```
sns.jointplot(x="chol",y="target",data=data,kind="hex")
plt.xlabel("Cholesterol")
plt.show()
```



[22]:
```
data[data["sex"]==1]["target"]
```

[22]:
```
0      1
1      1
3      1
5      1
7      1
      ..
295    0
297    0
299    0
300    0
301    0
```

```
Name: target, Length: 206, dtype: int64
```

[23]:
```python
from scipy.stats import pearsonr
data["chol"].corr(data["target"])
```

[23]:
```
-0.08143720051844144
```

[24]:
```python
corr=pearsonr(data["chol"],data["target"])
```

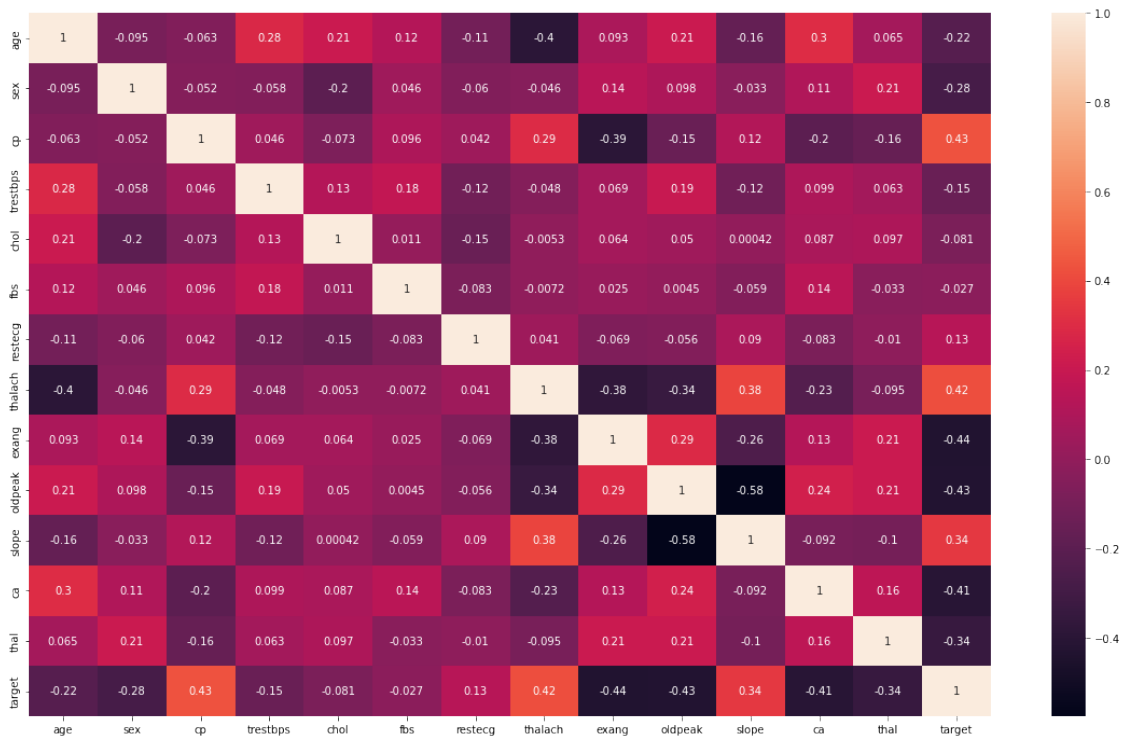
[25]:
```python
corr
```

[25]:
```
(-0.08143720051844144, 0.15803697464249133)
```

[26]:
```python
k=4
data.corr().nlargest(k,"chol")["chol"]
```

[26]:
```
chol        1.000000
age         0.207216
trestbps    0.125256
thal        0.096810
Name: chol, dtype: float64
```

[27]:
```python
plt.figure(figsize=(20,12))
sns.heatmap(data.corr(),annot=True)
plt.show()
```

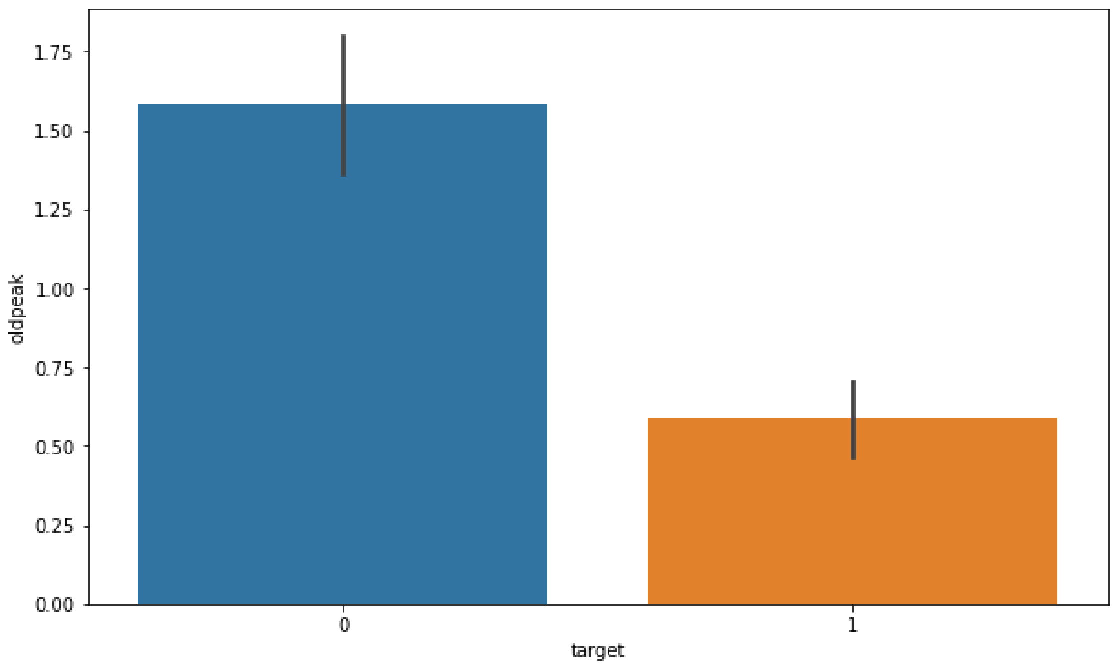


9 State what relationship exists between peak exercising and the occurrence of a heart attack

[28]:
```
data[["oldpeak","target"]].corr()
```

[28]:
```
          oldpeak    target
oldpeak  1.000000 -0.429146
target  -0.429146  1.000000
```

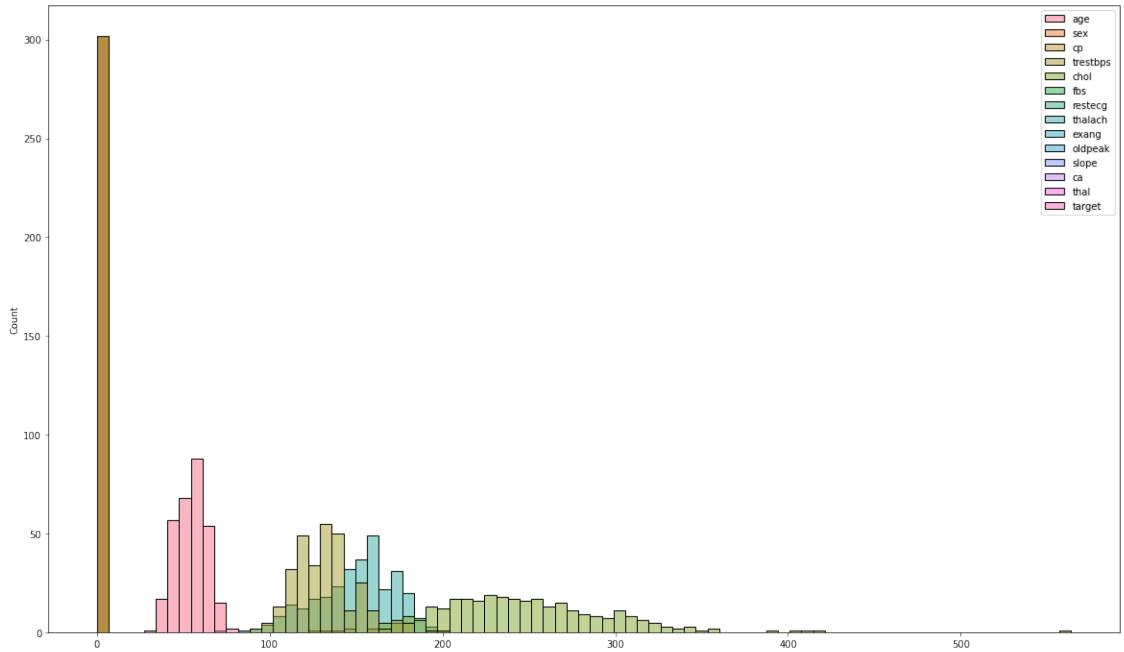
[29]:
```
plt.figure(figsize=(10,6))
sns.barplot(data=data,y="oldpeak",x="target")
plt.show()
```



When peak exercising is less than 0.75 then the person has Occurance of Heart Attack

10 Check if thalassemia is a major cause of CVD

[30]:
```
plt.figure(figsize=(20,12))
sns.histplot(data)
plt.show()
```

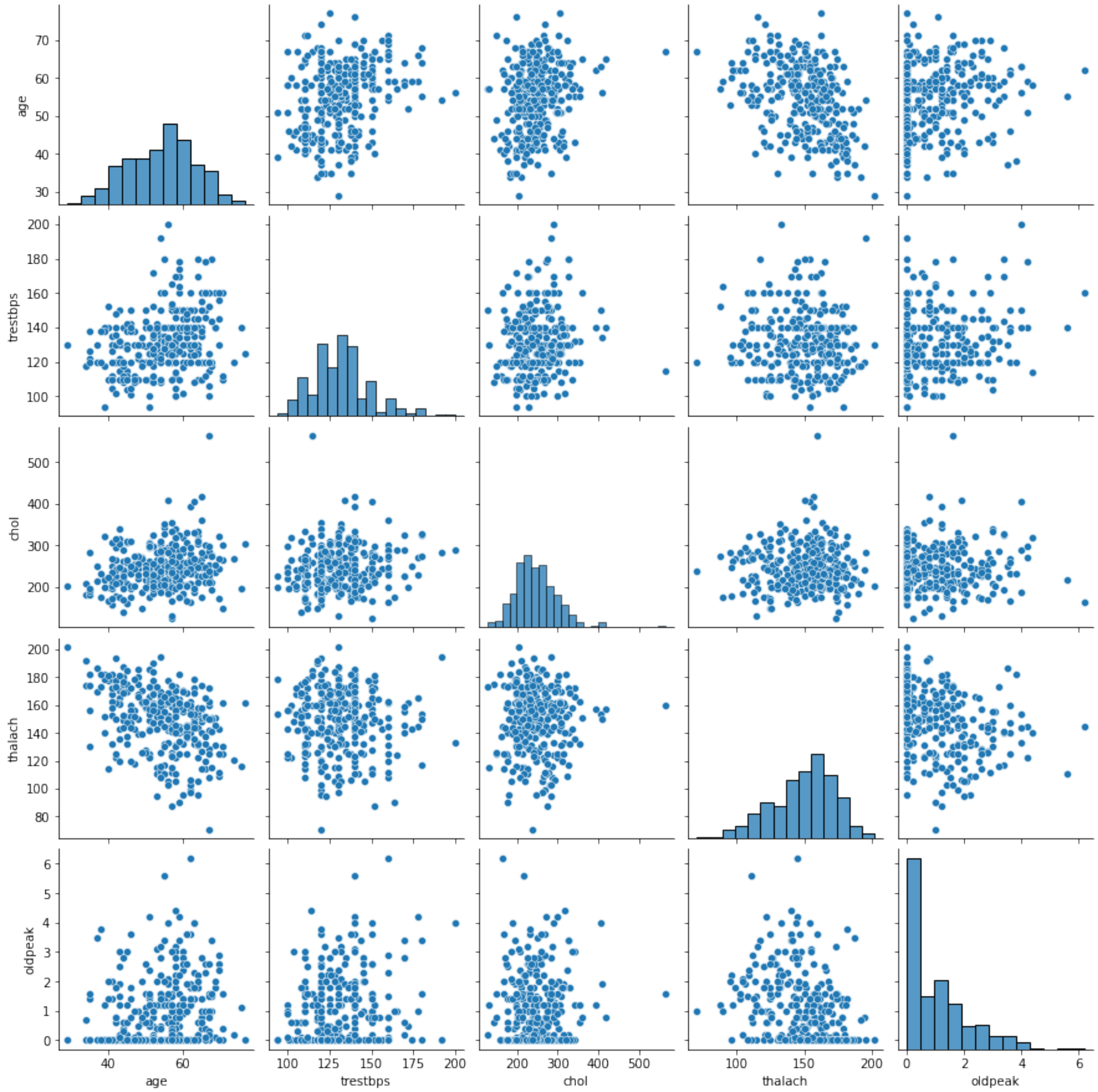


thalassemia is not the major cause of CVD

11 List how the other factors determine the occurrence of CVD

[31]:
```
subdata=data[["age","trestbps","chol","thalach","oldpeak"]]
sns.pairplot(subdata)
```

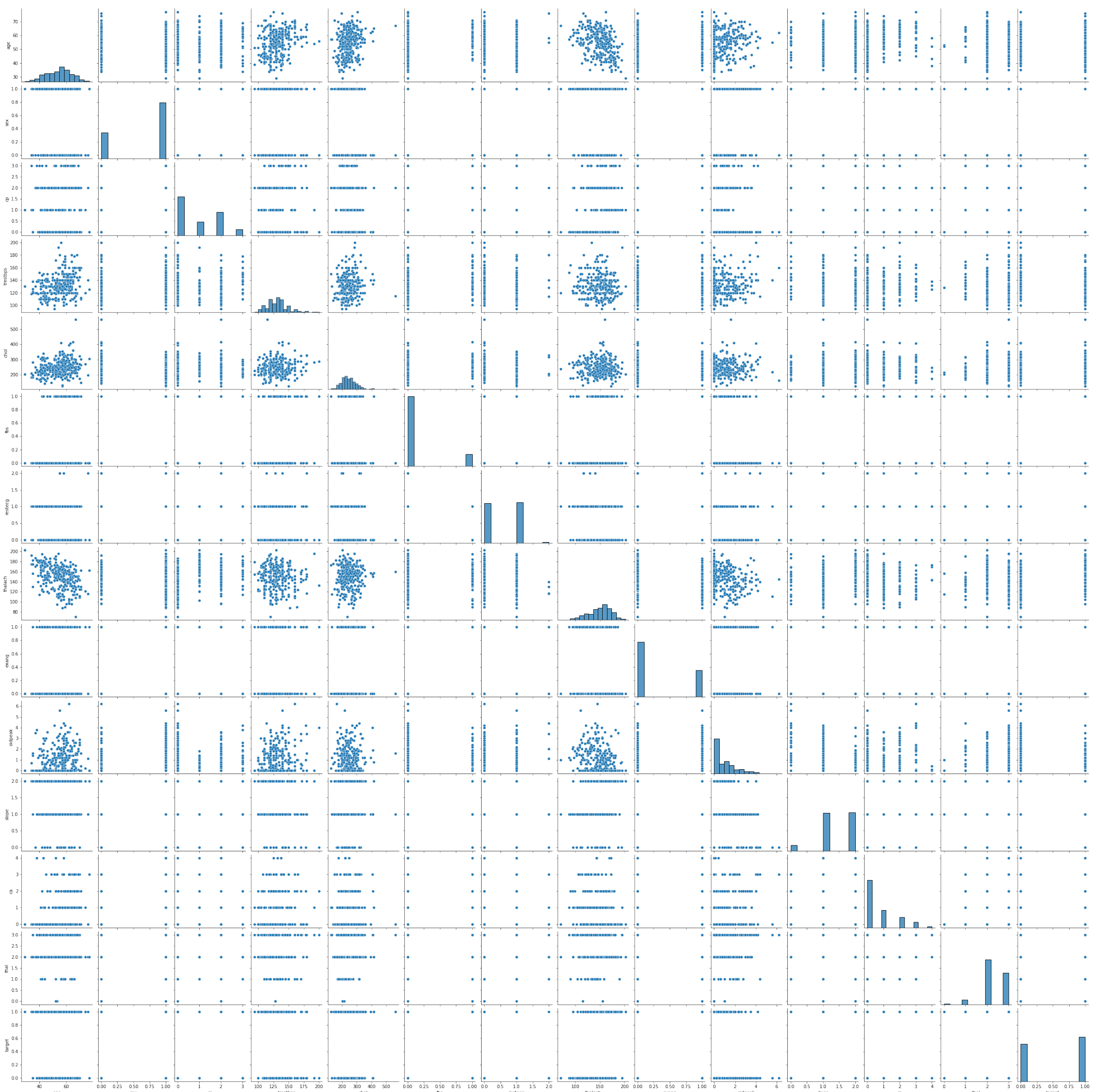
[31]: <seaborn.axisgrid.PairGrid at 0x7f5ec0b0e090>



12 Use a pair plot to understand the relationship between all the given variables

[32]:
```
sns.pairplot(data)
```

[32]: <seaborn.axisgrid.PairGrid at 0x7f5ec1067ad0>



13 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

[33]:
```
upper_limit_chol=data['chol'].mean()+3*data['chol'].std()
lower_limit_chol=data['chol'].mean()-3*data['chol'].std()
```

[34]:
```
data['chol']=np.where(
 data['chol']>upper_limit_chol,
 upper_limit_chol,
 np.where(
 data['chol']<lower_limit_chol,
 lower_limit_chol,
 data['chol']
 )
)
```

[35]:
```
upper_limit_trestbps=data['trestbps'].mean()+3*data['trestbps'].std()
lower_limit_trestbps=data['trestbps'].mean()-3*data['trestbps'].std()
```

[36]:
```
data['trestbps']=np.where(
 data['trestbps']>upper_limit_trestbps,
 upper_limit_trestbps,
 np.where(
 data['trestbps']<lower_limit_trestbps,
 lower_limit_trestbps,
 data['trestbps']
 )
)
```

[37]:
```
upper_limit_thalach=data['thalach'].mean()+3*data['thalach'].std()
lower_limit_thalach=data['thalach'].mean()-3*data['thalach'].std()
```

[38]:
```
data['thalach']=np.where(
 data['thalach']>upper_limit_thalach,
 upper_limit_thalach,
 np.where(
 data['thalach']<lower_limit_thalach,
 lower_limit_thalach,
 data['thalach']
 )
)
```

[39]:
```
upper_limit_oldpeak=data['oldpeak'].mean()+3*data['oldpeak'].std()
lower_limit_oldpeak=data['oldpeak'].mean()-3*data['oldpeak'].std()
```

[40]:
```
data['oldpeak']=np.where(
 data['oldpeak']>upper_limit_oldpeak,
 upper_limit_oldpeak,
 np.where(
 data['oldpeak']<lower_limit_oldpeak,
 lower_limit_oldpeak,
 data['oldpeak']
 )
```

```
)
```

[41]:
```
x=pd.DataFrame(data.iloc[:,:-1])
y=pd.DataFrame(data.iloc[:,-1])
```

[42]:
```
from sklearn.model_selection import train_test_split
```

[43]:
```
from sklearn.preprocessing import StandardScaler
```

[44]:
```
ss=StandardScaler()
```

[45]:
```
ss.fit_transform(x)
```

[45]:
```
array([[ 0.94979429,  0.68265615,  1.97647049, …, -2.27118179,
        -0.71491124, -2.1479552 ],
       [-1.92854796,  0.68265615,  1.005911  , …, -2.27118179,
        -0.71491124, -0.51399432],
       [-1.48572607, -1.46486632,  0.0353515 , …,  0.97951442,
        -0.71491124, -0.51399432],
       …,
       [ 1.50332164,  0.68265615, -0.93520799, …, -0.64583368,
         1.27497996,  1.11996657],
       [ 0.28556146,  0.68265615, -0.93520799, …, -0.64583368,
         0.28003436,  1.11996657],
       [ 0.28556146, -1.46486632,  0.0353515 , …, -0.64583368,
         0.28003436, -0.51399432]])
```

[46]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

[47]:
```
from sklearn.linear_model import LogisticRegression
```

[48]:
```
lr=LogisticRegression()
```

[49]:
```
lr.fit(x_train,y_train)
```

[49]:
```
LogisticRegression()
```

[50]:
```
y_pred=lr.predict(x_test)
y_pred
```

[50]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[51]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

[51]: 0.8524590163934426

[52]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
```

[53]:
```
rfc.fit(x_train,y_train)
```

[53]: RandomForestClassifier()

[54]:
```
y_pred1=rfc.predict(x_test)
y_pred1
```

[54]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[55]:
```
from sklearn.metrics import accuracy_score
```

[56]:
```
accuracy_score(y_test,y_pred1)
```

[56]: 0.8852459016393442

[57]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

[57]: 0.4019607843137255

[58]:
```
from scipy.stats import chi2_contingency
#difining the table
stat, p, dof, expected = chi2_contingency(data)
# interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
 print('Dependent (reject H0)')
else:
 print('Independent (H0 holds true)')
```

```
p value is 3.125482768461386e-51
Dependent (reject H0)
```

[ ]: