Cardio_Health_Project

November 11, 2023

[1]:

```python
#necessary imports:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

[2]:

```python
#importing the data
data=pd.read_excel('cardio_health.xlsx')
```

[8]:

```python
data.head()
```

[8]:

```
   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
```

[9]:
```
data.shape
```

[9]: (303, 14)

[10]:
```
#missing value check:
data.isna().sum()
```

[10]:
```
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
```

[3]:
```
# Checking duplicates in the dataset
data[data.duplicated()]
```

[3]:
```
     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
```

[4]:
```
 # Dropping duplicate entries
data.drop_duplicates(inplace=True)
 # rechecking for duplicates
data[data.duplicated()].shape
```

[4]: (0, 14)

[36]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 302 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       302 non-null    int64
 1   sex       302 non-null    int64
```

```
 2   cp        302 non-null    int64
 3   trestbps  302 non-null    int64
 4   chol      302 non-null    int64
 5   fbs       302 non-null    int64
 6   restecg   302 non-null    int64
 7   thalach   302 non-null    int64
 8   exang     302 non-null    int64
 9   oldpeak   302 non-null    float64
 10  slope     302 non-null    int64
 11  ca        302 non-null    int64
 12  thal      302 non-null    int64
 13  target    302 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 43.5 KB
```

[22]:
```
#checking the dataset for imbalance:
data['target'].value_counts()
```

[22]:
```
1    164
0    138
Name: target, dtype: int64
```

[ ]:
```
# 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
```

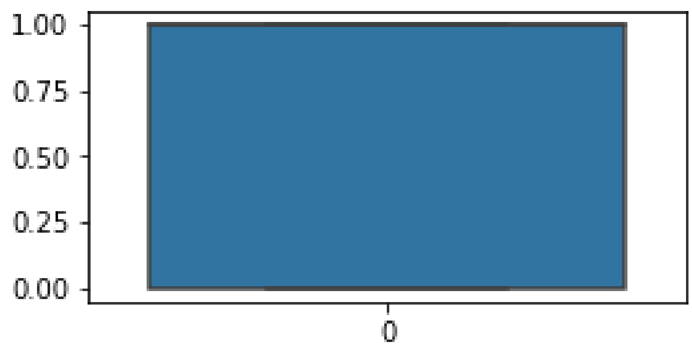
[32]:
```
data.describe().T
```

[32]:

	count	mean	std	min	25%	50%	75%	max
age	302.0	54.420530	9.047970	29.0	48.00	55.5	61.00	77.0
sex	302.0	0.682119	0.466426	0.0	0.00	1.0	1.00	1.0
cp	302.0	0.963576	1.032044	0.0	0.00	1.0	2.00	3.0
trestbps	302.0	131.602649	17.563394	94.0	120.00	130.0	140.00	200.0
chol	302.0	246.500000	51.753489	126.0	211.00	240.5	274.75	564.0
fbs	302.0	0.149007	0.356686	0.0	0.00	0.0	0.00	1.0
restecg	302.0	0.526490	0.526027	0.0	0.00	1.0	1.00	2.0
thalach	302.0	149.569536	22.903527	71.0	133.25	152.5	166.00	202.0
exang	302.0	0.327815	0.470196	0.0	0.00	0.0	1.00	1.0
oldpeak	302.0	1.043046	1.161452	0.0	0.00	0.8	1.60	6.2
slope	302.0	1.397351	0.616274	0.0	1.00	1.0	2.00	2.0
ca	302.0	0.718543	1.006748	0.0	0.00	0.0	1.00	4.0
thal	302.0	2.314570	0.613026	0.0	2.00	2.0	3.00	3.0
target	302.0	0.543046	0.498970	0.0	0.00	1.0	1.00	1.0

[33]:
```
plt.figure(figsize=(4,2))
sns.boxplot(data.target)          # No Outliers in Target
```

[33]: <AxesSubplot: >

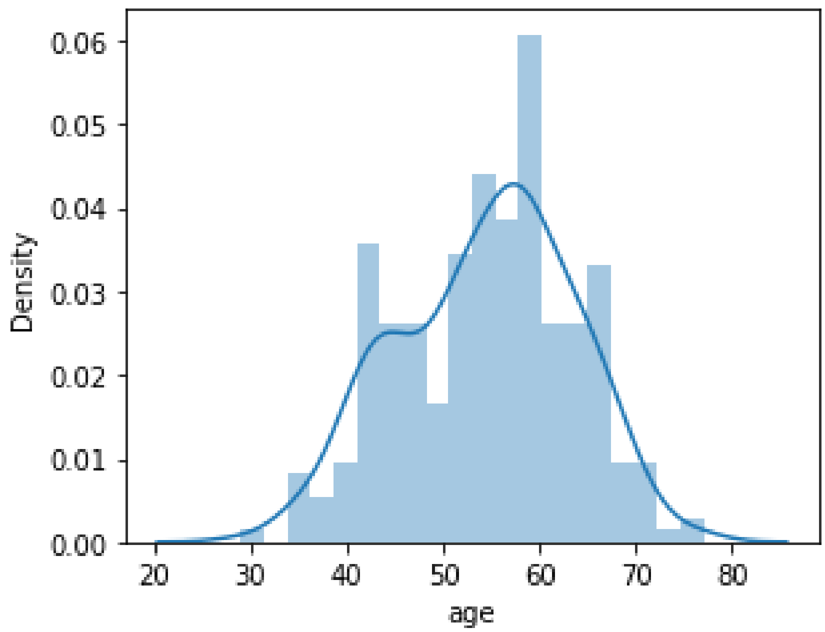


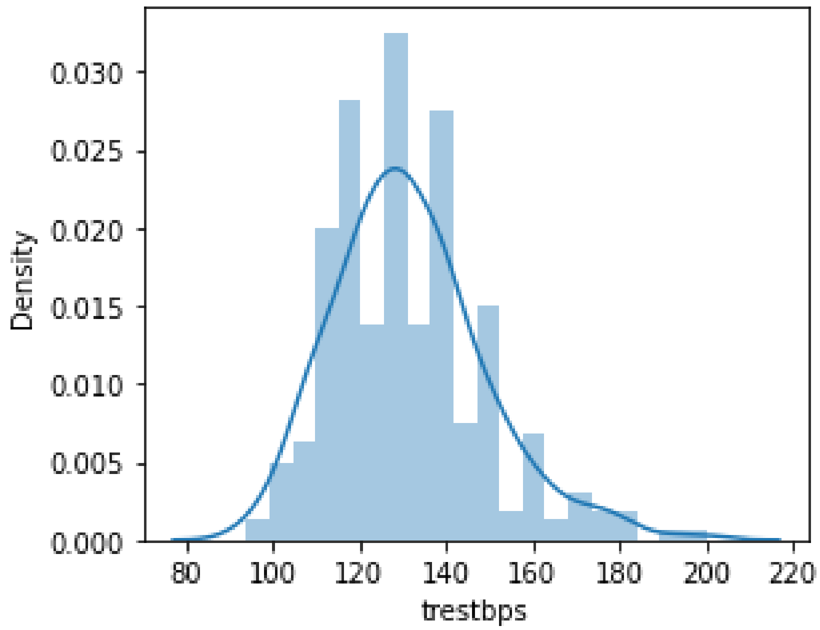
[35]:
```
data.nunique()

# here we identify that the variables with few unique values are categorical␣
↪and the variables with high unique values are numeric
```

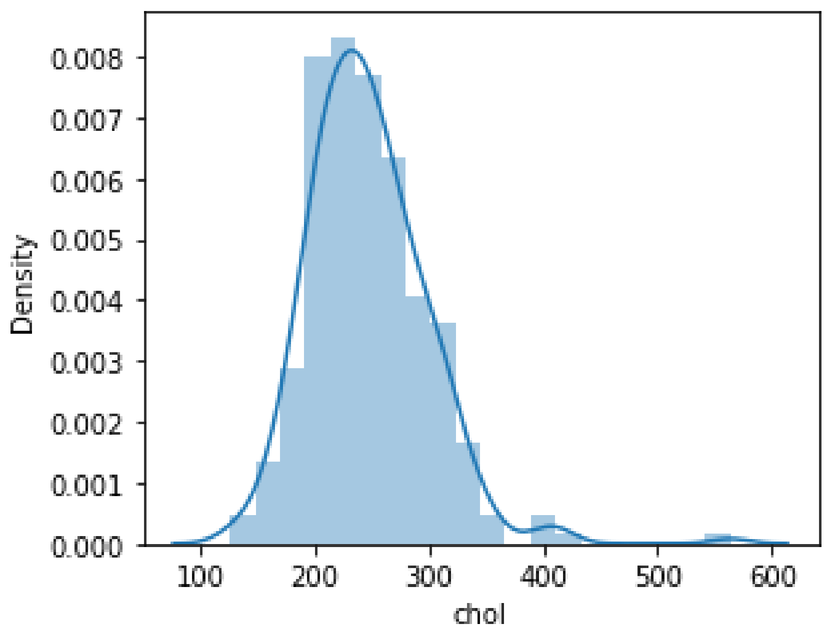
[35]:
```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

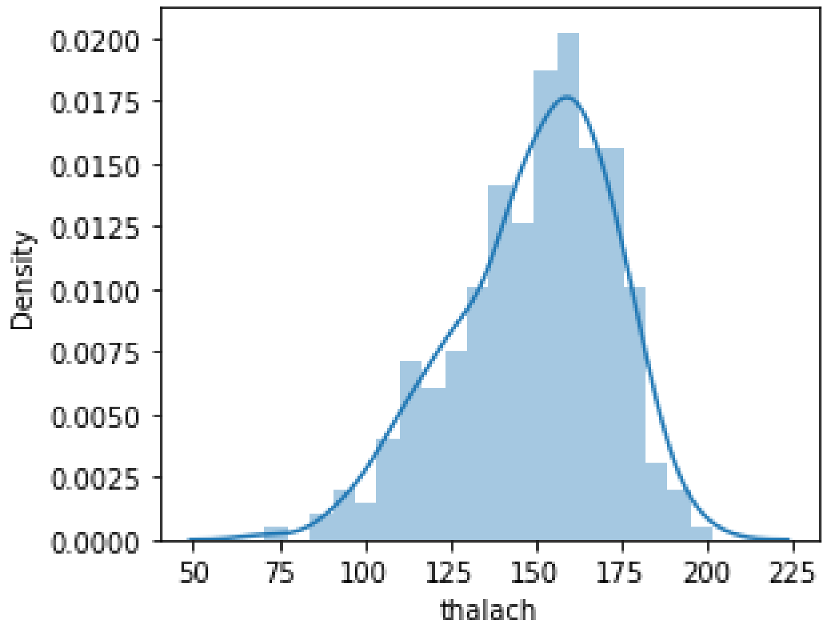
[37]:
```
numeric_cols=['age','trestbps','chol','thalach','oldpeak']

categorical_cols=['sex','cp','fbs','restecg','exang','slope','ca','thal','target']

#separating numeric and categorical columns
```

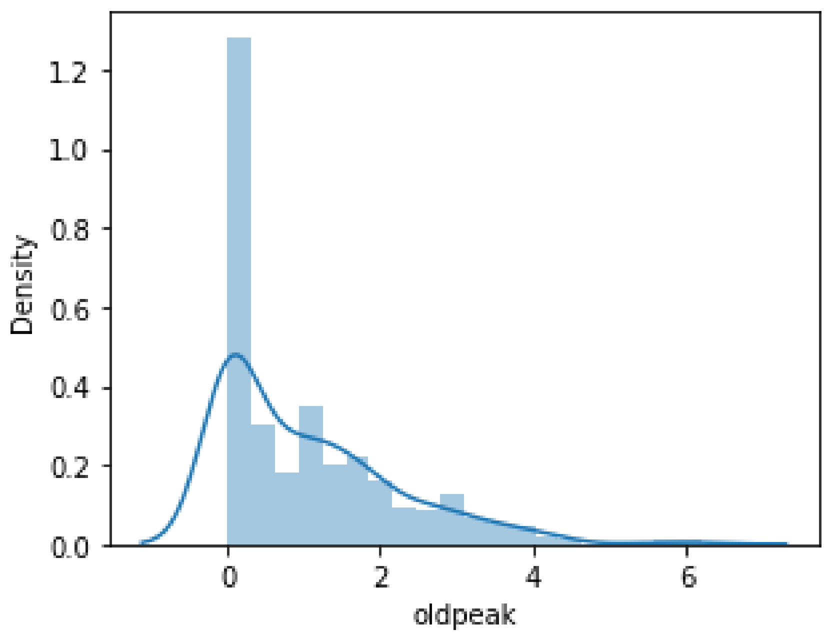
[38]:
```
#Exploring Numerical data

for i in numeric_cols:
    plt.figure(figsize=(4.5,3.5))
    sns.distplot(data[i],bins=20)

    plt.tight_layout()
    plt.show()
```




0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
80
100
120
140
160
180
200
220
trestbps


0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
100
200
300
400
500
600
chol





[ ]:
```
#Analysis:
-Age : The majority of patients are between 50 and 60 years age. Also there are␣
↪less patients in the age range 45 to 50.
```

```
-Trestbps : The resting blood pressure for most patients is between 110 and 140.
 ↪ Also patient traffic peaks at values around 115, 130 and 140.

-Chol : Cholesterol values for most patients are between 200 to 300.

-Thalach : The maximum heart rate achieved in most patients are between 150 to␣
 ↪160.

-Oldpeak : Majority of patients are in the range 0 to 1.5.
```

[ ]:
```
# 2.b.Identify the data variables which are categorical and describe and␣
 ↪explore these variables using the appropriate tools,
such as count plot
```

[39]:
```
#Statistics for Categorical variables

data[categorical_cols].describe().T
```

[39]:

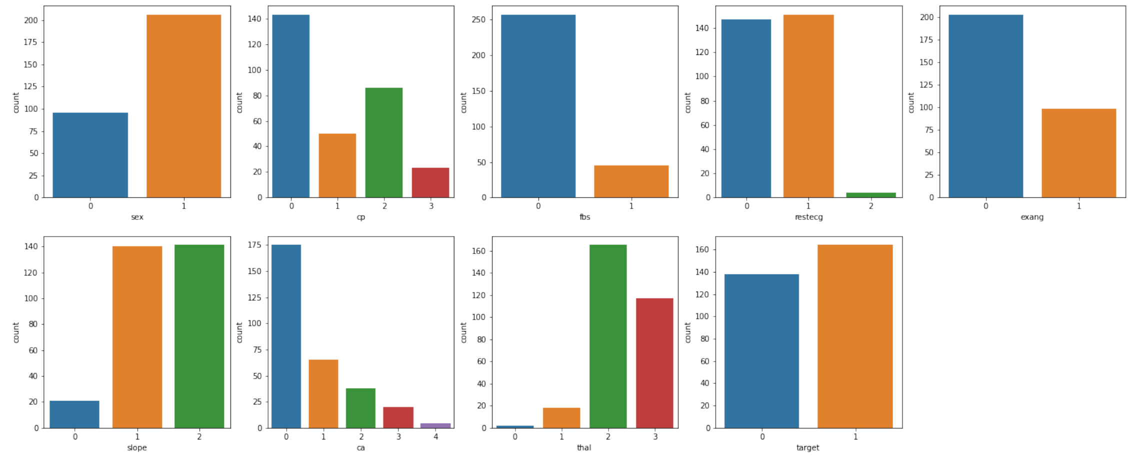
	count	mean	std	min	25%	50%	75%	max
sex	302.0	0.682119	0.466426	0.0	0.0	1.0	1.0	1.0
cp	302.0	0.963576	1.032044	0.0	0.0	1.0	2.0	3.0
fbs	302.0	0.149007	0.356686	0.0	0.0	0.0	0.0	1.0
restecg	302.0	0.526490	0.526027	0.0	0.0	1.0	1.0	2.0
exang	302.0	0.327815	0.470196	0.0	0.0	0.0	1.0	1.0
slope	302.0	1.397351	0.616274	0.0	1.0	1.0	2.0	2.0
ca	302.0	0.718543	1.006748	0.0	0.0	0.0	1.0	4.0
thal	302.0	2.314570	0.613026	0.0	2.0	2.0	3.0	3.0
target	302.0	0.543046	0.498970	0.0	0.0	1.0	1.0	1.0

[40]:
```
categorical=data[categorical_cols]
categorical.head()
```

[40]:

	sex	cp	fbs	restecg	exang	slope	ca	thal	target
0	1	3	1	0	0	0	0	1	1
1	1	2	0	1	0	0	0	2	1
2	0	1	0	0	0	2	0	2	1
3	1	1	0	1	0	2	0	2	1
4	0	0	0	1	1	2	0	2	1

[41]:
```
# count plot for categorical variables

plt.figure(figsize=(25,10))
for i in range(9):
    plt.subplot(2,5,i+1)
    sns.countplot(x= categorical_cols[i], data=categorical)
```

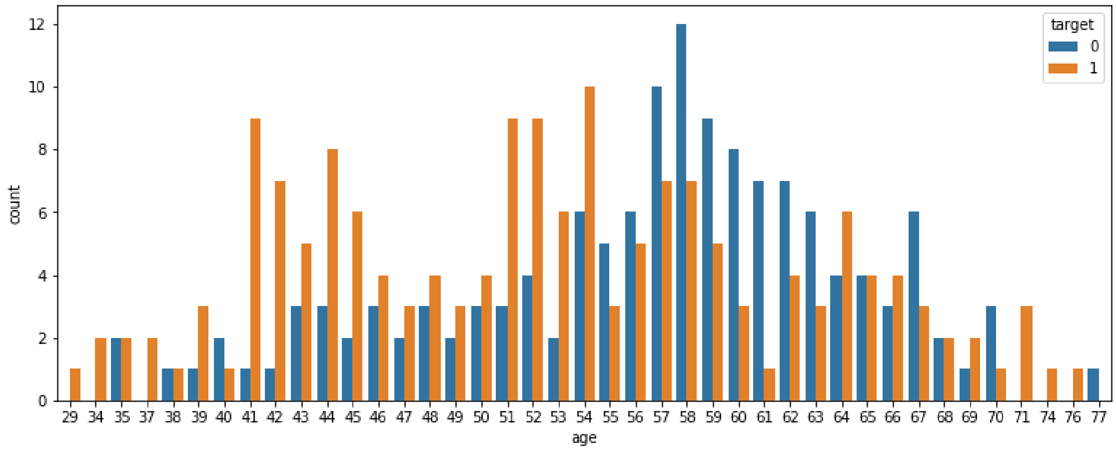


[ ]:
```
#Analysis from the above Count plot
-Sex (1 = male; 0 = female):The count of Male patient is almost double that of Females.

-cp (Chest pain type): Chest pain of Type 0 is highest observation value in patients, followed by type 2.

-fbs (Fasting blood sugar > 120 mg/dl (1 = true; 0 = false): Majority of patients have fasting Blood Sugar <120 mg/dl.

-restecg (Resting electrocardiographic results): Most common observations are 0 and 1 while there are very less patients with values 2.

-exang (Exercise induced angina (1 = yes, 0 = no)): Almost half of the patients have Exercise induced angina.

-slope (Slope of the peak exercise ST segment): The minimum observation value is 0 and other two observations are almost equal

-ca (Number of major vessels (0-3) colored by fluoroscopy): Mostly the number of large vessels colored by fluoroscopy is
absent.

-thai (3 = normal; 6 = fixed defect; 7 = reversible defect): Majority of patients are in observations 2 followed by 3 which is normal.

-target(1 or 0): More than half of the patients have a risk of heart attack.
```

[42]:
```
#2.c. Study the occurrence of CVD across the Age category
#Occurrence of CVD across the Age category
```

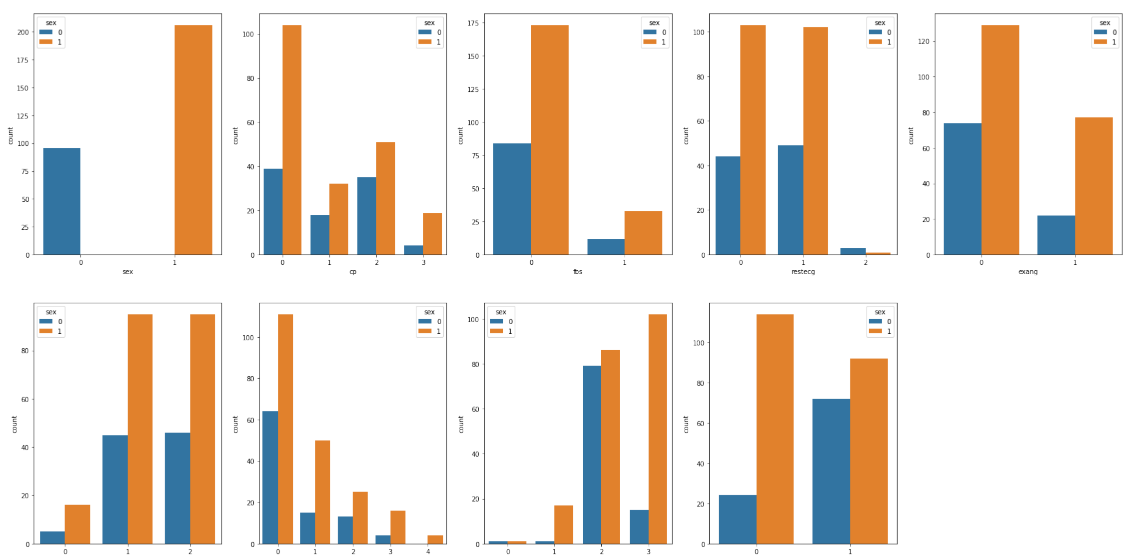
```
plt.figure(figsize=(13,5))
sns.countplot(x='age', data=data, hue='target')
```

[42]: <AxesSubplot: xlabel='age', ylabel='count'>



[ ]:
```
#It can be observed that people between age 41-45 and 51-54 are more exposed to
↪CVD (target=1)
```

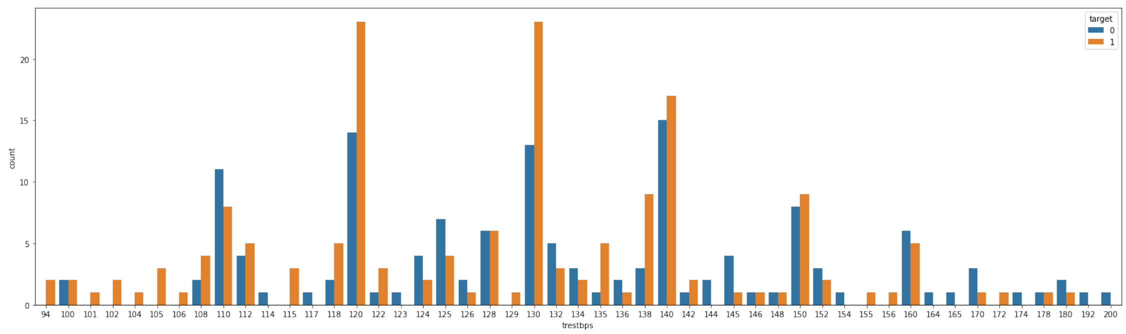
[44]:
```
#2.d. Study the composition of all patients with respect to the Sex category
#composition of patients with respect to Sex category

plt.figure(figsize=(30,15))
for i in range(9):
    plt.subplot(2,5,i+1)
    sns.countplot(x= categorical_cols[i], data=categorical, hue='sex')
```



[ ]:
```
#From Observation, We can notice that Males are more prone to CVD than Females.
```

[45]:
```
# 2.e. Study if one can detect heart attacks based on anomalies in the resting␣
↪blood pressure (trestbps) of a patient
plt.figure(figsize=(25,7))
sns.countplot(x= 'trestbps', data= data, hue='target')
```

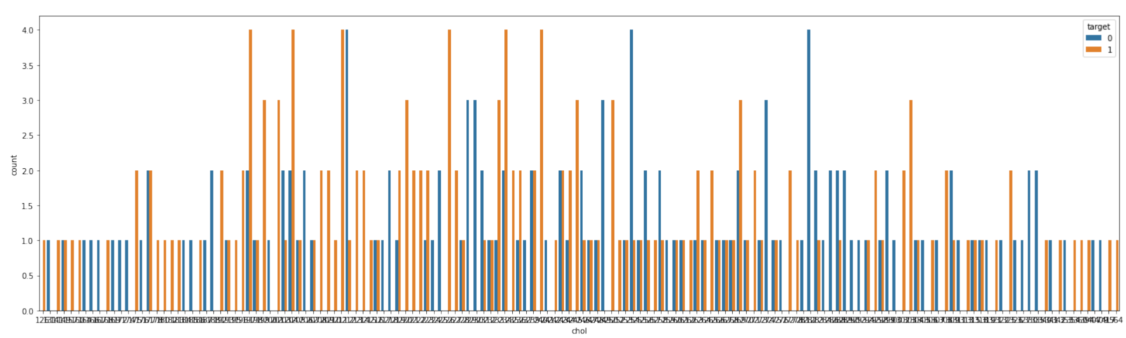
[45]:
```
<AxesSubplot: xlabel='trestbps', ylabel='count'>
```



[ ]:
```
# we can observe that  when resting blood pressure (trestbps) values are 120,␣
↪130 and 140 risk of heart attacks increases.
```

[ ]:
```
#2.f. Describe the relationship between cholesterol levels and a target variable
plt.figure(figsize=(25,7))
sns.countplot(x= 'chol', data= data, hue='target')
```

[ ]:
```
<AxesSubplot: xlabel='chol', ylabel='count'>
```

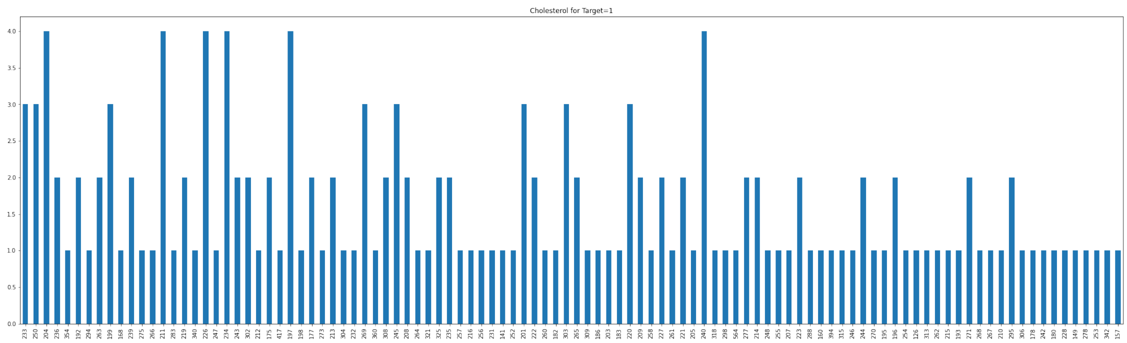


[47]:
```
df=data.groupby('target')['chol']
#cholesterol graph for Target=1
```

```python
df.get_group(1).value_counts(sort=False).plot(kind='bar',title="Cholesterol for Target=1", figsize=(35,10))
```

[47]:

```
<AxesSubplot: title={'center': 'Cholesterol for Target=1'}>
```

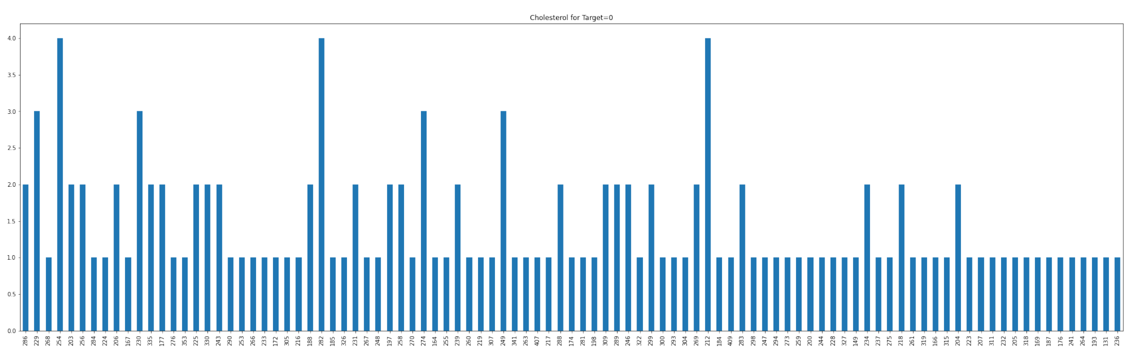


[48]:

```python
#cholesterol graph for Target=0

df.get_group(0).value_counts(sort=False).plot(kind='bar',title="Cholesterol for Target=0",figsize=(35,10))
```

[48]:

```
<AxesSubplot: title={'center': 'Cholesterol for Target=0'}>
```



[50]:

```python
#Correlation between Cholesterol value and Target
data[['chol', 'target']].corr()
```

[50]:

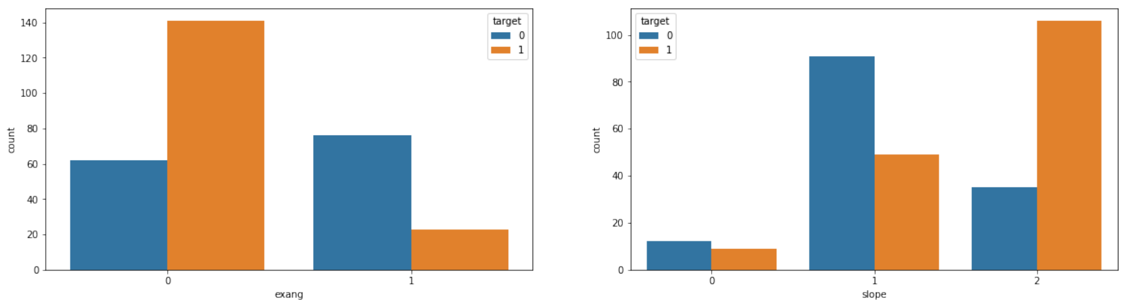
	chol	target
chol	1.000000	-0.081437
target	-0.081437	1.000000

[ ]:

```python
#From the above graphs, we can say that it is difficult to predict patients having a heart attack using cholesterol values.
```

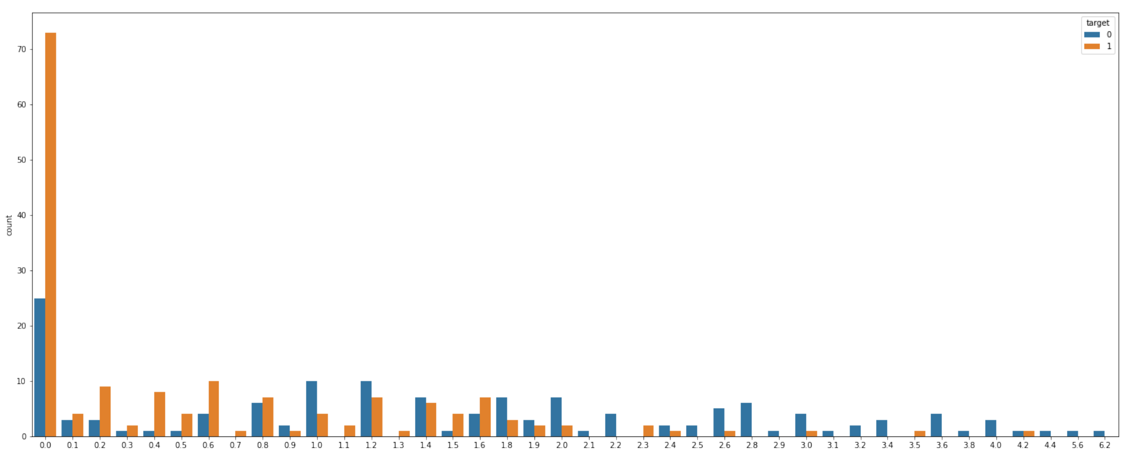
```python
#The correlation between the two variables is also negative.

#We can also say that there are chances of having a heart attack for Cholestrol values between 190 to 250.
```

[5]:
```python
# 2.g. State what relationship exists between peak exercising and the occurrence of a heart attack
cols=['exang','slope']
plt.figure(figsize=(20,5))
for i in range(len(cols)):
    plt.subplot(1,2,i+1)
    sns.countplot(x= cols[i],hue='target', data=data)
```



[58]:
```python
plt.figure(figsize=(25,10))
sns.countplot(x= data['oldpeak'],hue='target', data=data)
```

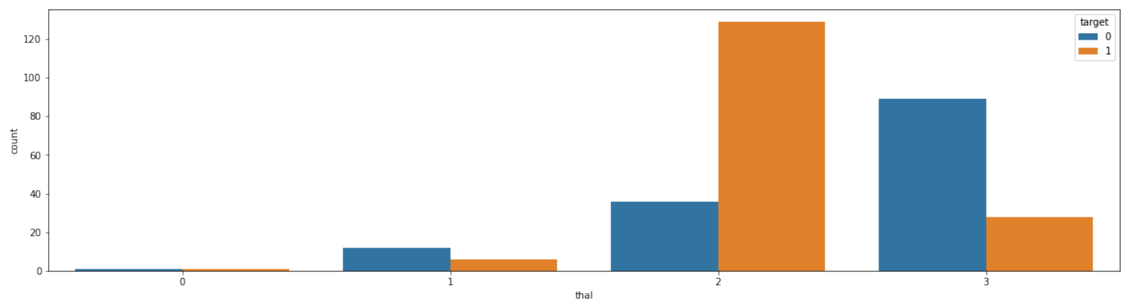
[58]: <AxesSubplot: xlabel='oldpeak', ylabel='count'>



[ ]:
```
#exang: Occurance of heart attacks in Exercise induced angina is less and it can be seen that patients with no exercise induced angina suffers from heart attacks.

#slope: occurance of heart attack is highest where Slope of the peak exercise ST segment value is 2.

#oldpeak: Occurance of heart attack is highest where oldpeak value is 0
```

[60]:
```
#2.h. Check if thalassemia is a major cause of CVD.
plt.figure(figsize=(20,5))
sns.countplot(x= data['thal'],hue='target', data=data)
```

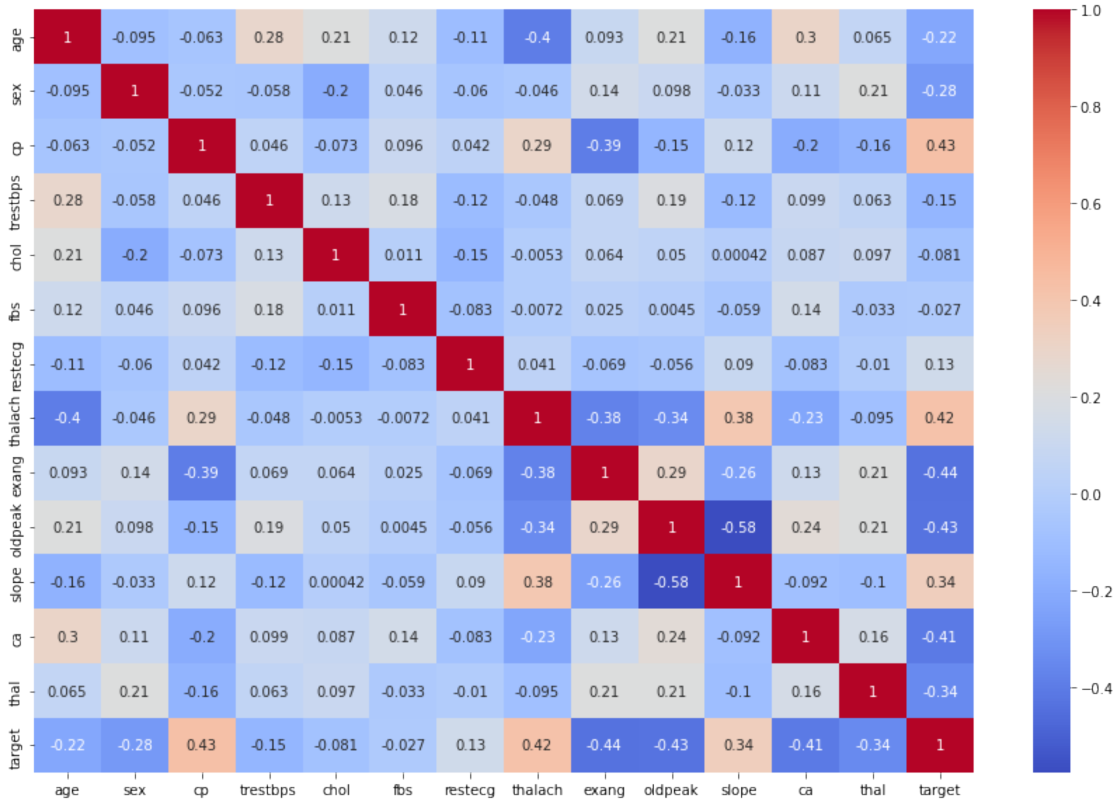
[60]: <AxesSubplot: xlabel='thal', ylabel='count'>



[ ]:
```
# Patients having thal value as 2 have high risk of CVD
```

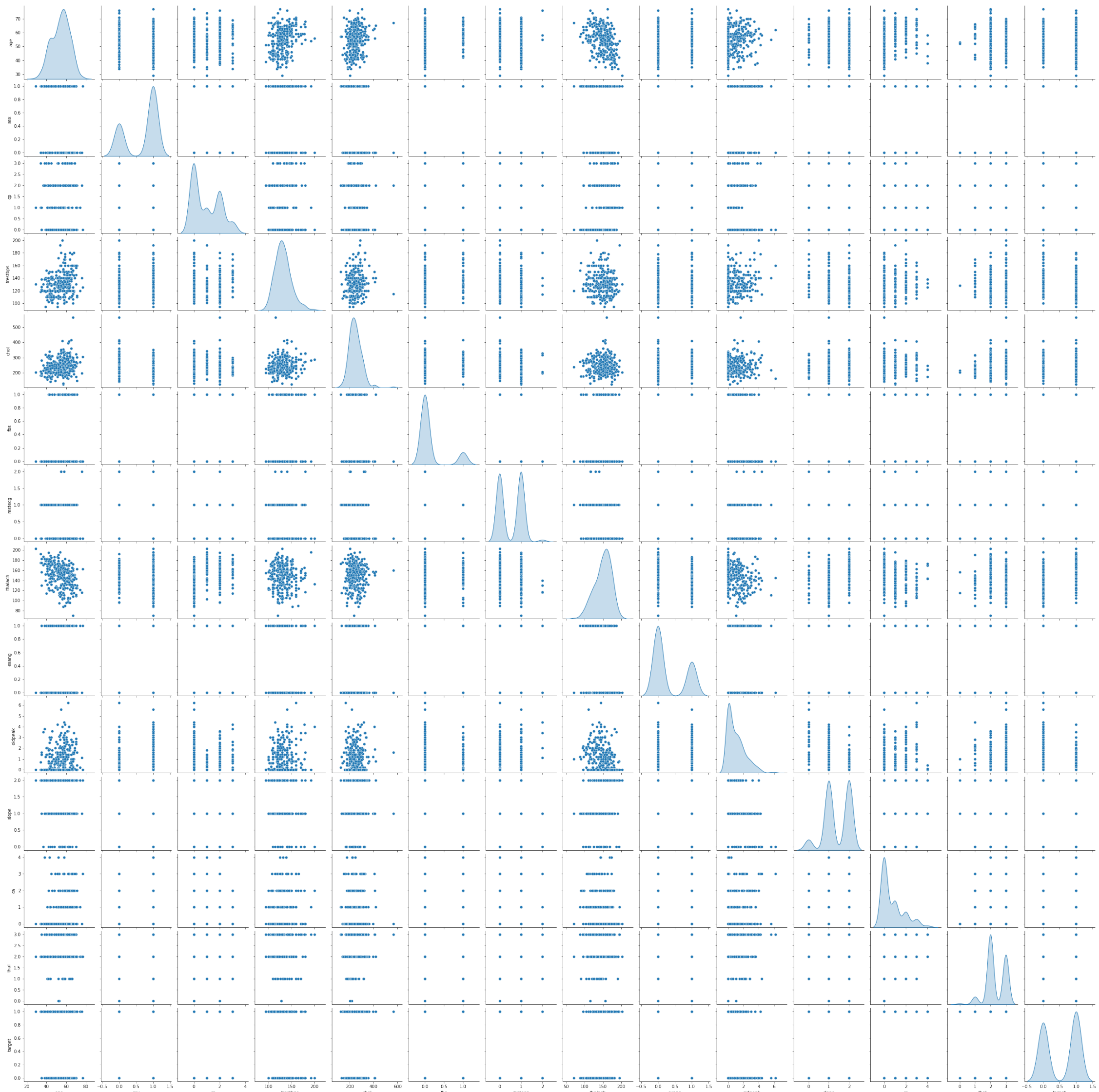
[62]:
```
#2.i. List how the other factors determine the occurrence of CVD
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(),cmap='coolwarm',annot=True)
```

[62]: <AxesSubplot: >



[ ]:
```
#From the above heat map we can conclude that 'Chest pain(cp)' and
#'Maximum heart rate(thalach)' are the main triggers for occurance of CVD with correlation values 0.43 and 0.42 respectively.

-'Slope of the peak exercise ST segment(slope)' is also moderately correlated with 'target' variable(correlation value 0.34)
 and so is also a cause of CVD.

- We can observe that 'thalach' variable is also highly correlated with 'cp' and 'slope' variables.

- In general we can say that the "target" variable correlates with more than one variables.
  So there are multiple causes that can trigger CVD in patients.
```

[65]:
```
#2.j. Use a pair plot to understand the relationship between all the given variables
sns.pairplot(data,diag_kind='kde')
plt.show()
```



[ ]:

```
#We can see that Pairplot is not of much help instead Heatmap provides better insights of relationship between all the variables.

-We have already noted "target" variable correlates maximum with "cp", "thalach" and "slope"

-"age' variable is highly correlated to "trestbps" and "ca"

-"thalach" variable is highly correlated with "cp" and "slope" variables

-"exang' variable is highly correlated to "oldpeak"

-"chol" and "fbs" have least correlation with "target" variable
```

[ ]:
```
#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.
```

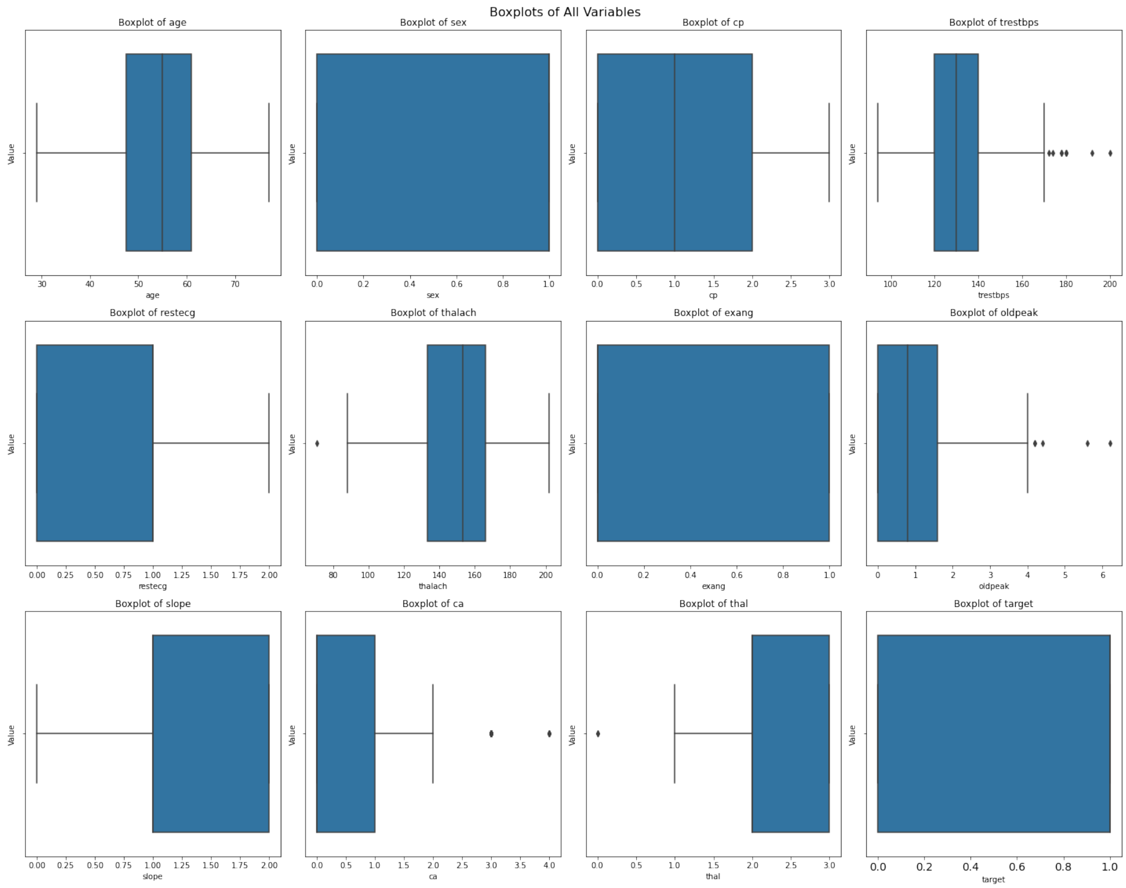
[6]:
```
#dropping columns "chol" and "fbs" as they have very low correlation with target

data.drop(['chol','fbs'], axis=1, inplace = True)
```

[7]:
```
data.head()
```

[7]:
```
   age  sex  cp  trestbps  restecg  thalach  exang  oldpeak  slope  ca  thal  \
0   63    1   3       145        0      150      0      2.3      0   0     1
1   37    1   2       130        1      187      0      3.5      0   0     2
2   41    0   1       130        0      172      0      1.4      2   0     2
3   56    1   1       120        1      178      0      0.8      2   0     2
4   57    0   0       120        1      163      1      0.6      2   0     2

   target
0       1
1       1
2       1
3       1
4       1
```

[17]:
```
# Create a figure with 12 subplots
fig, axes = plt.subplots(3, 4, figsize=(20, 16))

# Iterate over the data columns and create a boxplot for each column
for column_index, column in enumerate(data.columns):
    # Create a subplot
    subplot = axes[column_index // 4, column_index % 4]

    # Create a boxplot of the column data
    sns.boxplot(x=column, data=data, ax=subplot)

    # Set the subplot title and axis labels
    subplot.set_title(f"Boxplot of {column}")
    subplot.set_xlabel(column)
    subplot.set_ylabel("Value")

# Increase the font size of the axis labels and title
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.suptitle("Boxplots of All Variables", fontsize=16)
```

```python
# Adjust the subplot layout to make the plots more readable
fig.tight_layout()

# Display the plot
plt.show()
```



[19]:
```python
#Treating Outliers for "trestbps"

Q3 = data.trestbps.quantile(0.75)
Q1 = data.trestbps.quantile(0.25)
IQR = Q3-Q1
upper = Q3 + 1.5 * (IQR)
```

[20]:
```python
#number of outliers in "trestbps"
data[data.trestbps > upper]
```

[20]:

	age	sex	cp	trestbps	restecg	thalach	exang	oldpeak	slope	ca	\
8	52	1	2	172	1	162	0	0.5	2	0	
101	59	1	3	178	0	145	0	4.2	0	0	

```
110  64  0  0  180  1  154  1  0.0  2  0
203  68  1  2  180  0  150  1  1.6  1  0
223  56  0  0  200  0  133  1  4.0  0  2
241  59  0  0  174  1  143  1  0.0  1  0
248  54  1  1  192  0  195  0  0.0  2  1
260  66  0  0  178  1  165  1  1.0  1  2
266  55  0  0  180  2  117  1  3.4  1  0

     thal  target
8       3       1
101     3       1
110     2       1
203     3       0
223     3       0
241     2       0
248     3       0
260     3       0
266     2       0
```

[21]:
```python
data[data.trestbps > upper].shape
```

[21]: (9, 12)

[22]:
```python
#percentage of outliers in "trestbps"
data[data.trestbps > upper].shape[0]/data.shape[0]*100
```

[22]: 2.9702970297029703

[23]:
```python
 #indexes of outliers in "trestbps"
trestbps_index= data[data.trestbps > upper].index
# since our dataset is small we shall not remove the outliers and treat it␣
 ↪using capping method
data.loc[trestbps_index,'trestbps']=upper
#outliers capped to upper
data.loc[trestbps_index,'trestbps']
```

[23]:
```
8      170
101    170
110    170
203    170
223    170
241    170
248    170
260    170
266    170
Name: trestbps, dtype: int64
```

[25]:
```
#Treating Outliers for "oldpeak" using capping method

Q3 = data.oldpeak.quantile(0.75)
Q1 = data.oldpeak.quantile(0.25)
IQR = Q3-Q1
upper_oldpeak = Q3 + 1.5 * (IQR)
```

[26]:
```
#number of outliers in "oldpeak"
data[data.oldpeak > upper_oldpeak].shape
```

[26]:
```
(5, 12)
```

[27]:
```
#indexes of outliers in "oldpeak"
oldpeak_index = data[data.oldpeak > upper_oldpeak].index
```

[29]:
```
#assigning upper value to outliers
data.loc[oldpeak_index,'oldpeak'] = upper_oldpeak

#capped outliers
data.loc[oldpeak_index,'oldpeak']
```

[29]:
```
101    4.0
204    4.0
221    4.0
250    4.0
291    4.0
Name: oldpeak, dtype: float64
```

[48]:
```
# Treat outliers in the `thalach` column using the capping method

# Calculate the lower outlier bound
lower = data.thalach.quantile(0.85) - 1.5 * (data.thalach.quantile(0.85) - data.
 ↪thalach.quantile(0.15))

# Identify the outliers
outlier_index = data[data.thalach < lower].index

# Cap the outliers
data.loc[outlier_index, 'thalach'] = lower

# Print the capped outliers
print(data.loc[outlier_index, 'thalach'])
```

```
136    101
198    101
216    101
233    101
```

```
243    101
262    101
272    101
297    101
Name: thalach, dtype: int64
```

[ ]:
```
#Encoding and Scaling of data
```

[17]:
```
#creating a copy of dataset to apply Encoding and Scaling
data1= data.copy()
```

[18]:
```
data1.head()
```

[18]:
```
   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
```

[20]:
```
#separating numerical and Categorical columns

numeric=['age','trestbps','thalach','oldpeak']

categorical=['sex','cp','restecg','exang','slope','ca','thal']
```

[21]:
```
data1[numeric].head()
```

[21]:
```
   age  trestbps  thalach  oldpeak
0   63       145      150      2.3
1   37       130      187      3.5
2   41       130      172      1.4
3   56       120      178      0.8
4   57       120      163      0.6
```

[22]:
```
#scaling numeric columns

ss = StandardScaler()

data1[numeric] = ss.fit_transform(data1[numeric])
```

[23]:
```
data1.head()
```

[23]:
```
        age  sex  cp  trestbps  chol  fbs  restecg   thalach  exang   oldpeak  \
0  0.952197    1   3  0.763956   233    1        0  0.015443      0  1.087338
1 -1.915313    1   2 -0.092738   250    0        1  1.633471      0  2.122573
2 -1.474158    0   1 -0.092738   204    0        0  0.977514      0  0.310912
3  0.180175    1   1 -0.663867   236    0        1  1.239897      0 -0.206705
4  0.290464    0   0 -0.663867   354    0        1  0.583939      1 -0.379244

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

[24]:
```
#Encoding Categorical Columns

data_dummies = pd.get_dummies(data1, columns=categorical, drop_first=True)
```

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

[25]:
```
        age  trestbps  chol  fbs   thalach   oldpeak  target  sex_1  cp_1  \
0  0.952197  0.763956   233    1  0.015443  1.087338       1      1     0
1 -1.915313 -0.092738   250    0  1.633471  2.122573       1      1     0
2 -1.474158 -0.092738   204    0  0.977514  0.310912       1      0     1
3  0.180175 -0.663867   236    0  1.239897 -0.206705       1      1     1
4  0.290464 -0.663867   354    0  0.583939 -0.379244       1      0     0

   cp_2  …  exang_1  slope_1  slope_2  ca_1  ca_2  ca_3  ca_4  thal_1  \
0     0  …        0        0        0     0     0     0     0       1
1     1  …        0        0        0     0     0     0     0       0
2     0  …        0        0        1     0     0     0     0       0
3     0  …        0        0        1     0     0     0     0       0
4     0  …        1        0        1     0     0     0     0       0

   thal_2  thal_3
0       0       0
1       1       0
2       1       0
3       1       0
4       1       0

[5 rows x 23 columns]
```

[ ]:
```
#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.
```

[26]:
```
#Defining our X and y

X = data_dummies.drop('target', axis=1)
y = data_dummies['target']
```

[72]:
```
# Logistic Regression Model

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.25)
```

[64]:
```
log_reg = LogisticRegression()

log_reg.fit(train_X, train_y)
```

[64]:
```
LogisticRegression()
```

[73]:
```
print('Train Score: {}'.format(log_reg.score(train_X, train_y)))

print('Test Score: {}'.format(log_reg.score(test_X, test_y)))
```

```
Train Score: 0.8502202643171806
Test Score: 0.868421052631579
```

[ ]:
```
# Using LDA and Logistic Regression
```

[3]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
```

[27]:
```
#we can see that dimention is reduced to 1 column
X_ld = lda.fit_transform(X, y)
X_ld.shape
```

[27]:
```
(303, 1)
```

[32]:
```
train_ld_X, test_ld_X, train_ld_y, test_ld_y = train_test_split(X_ld, y,␣
↪test_size=0.2)

log_reg = LogisticRegression()

log_reg.fit(train_ld_X, train_ld_y)
```

[32]:
```
LogisticRegression()
```

[33]:
```python
print('Train Score: {}'.format(log_reg.score(train_ld_X, train_ld_y)))

print('Test Score: {}'.format(log_reg.score(test_ld_X, test_ld_y)))
```

```
Train Score: 0.8677685950413223
Test Score: 0.9180327868852459
```

[ ]:
```python
 # We can see improvement in the scores after applying LDA.
```

[35]:
```python
#Report for Train set
predict_ld_y=log_reg.predict(test_ld_X)
print(metrics.classification_report(train_ld_y, log_reg.predict(train_ld_X)))
```

```
              precision    recall  f1-score   support

           0       0.88      0.83      0.86       115
           1       0.86      0.90      0.88       127

    accuracy                           0.87       242
   macro avg       0.87      0.87      0.87       242
weighted avg       0.87      0.87      0.87       242
```

[ ]:
```python
#Classification Report for Test set
```

[36]:
```python
print(metrics.classification_report(test_ld_y, predict_ld_y))
```

```
              precision    recall  f1-score   support

           0       0.95      0.83      0.88        23
           1       0.90      0.97      0.94        38

    accuracy                           0.92        61
   macro avg       0.93      0.90      0.91        61
weighted avg       0.92      0.92      0.92        61
```

[37]:
```python
#Accuracy Score

accuracy = accuracy_score(test_ld_y,predict_ld_y)

print("Test Accuracy of Logistic Regression with LDA : {}".format(accuracy*100))
```

```
Test Accuracy of Logistic Regression with LDA : 91.80327868852459
```

[38]:
```python
#Cross Validation Score
```

```
scores = cross_val_score(log_reg, test_ld_X,test_ld_y, cv=5).mean()     #Model␣
↪Performance

print("Cross-Validation Accuracy Scores: ", scores*100)
```

```
Cross-Validation Accuracy Scores:  91.66666666666667
```

[39]:
```
## Random Forest Model
```

[40]:
```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3)
rfc=RandomForestClassifier()

rfc.fit(train_X,train_y)
```

[40]:
```
RandomForestClassifier()
```

[41]:
```
RandomForestClassifier()
pred_y = rfc.predict(test_X)
```

[42]:
```
print('Test accuracy of Random Forest : ', accuracy_score(test_y, pred_y)*100)
```

```
Test accuracy of Random Forest :  82.41758241758241
```

[43]:
```
cross_val_score(rfc, train_X,train_y, cv=5)
```

[43]:
```
array([0.76744186, 0.74418605, 0.83333333, 0.73809524, 0.73809524])
```

[44]:
```
#training score
cross_val_score(rfc, train_X,train_y, cv=5).mean()
```

[44]:
```
0.7687707641196013
```

[45]:
```
#Cross Validation Score #testing score

cross_val_score(rfc, test_X, test_y, cv=5).mean()
```

[45]:
```
0.8350877192982455
```

[46]:
```
#Applying grid search CV
param_grid = {
    'n_estimators': [20, 50, 100, 150,200],
    'max_depth': [3, 5, 7, None],
    'min_samples_leaf': [3, 5, 7, 9]
}

gscv = GridSearchCV(rfc, param_grid, cv=5, verbose=1)
gscv.fit(train_X,train_y)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
```

[46]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [3, 5, 7, None],
                         'min_samples_leaf': [3, 5, 7, 9],
                         'n_estimators': [20, 50, 100, 150, 200]},
             verbose=1)
```

[47]:
```python
#training score
cross_val_score(gscv.best_estimator_, train_X,train_y, cv=5).mean()
```

[47]:
```
0.7830564784053158
```

[48]:
```python
#Cross Validation Score after Grid Search CV

cvs=cross_val_score(gscv.best_estimator_, test_X, test_y, cv=5).mean()

print("Cross-Validation Accuracy Scores: ", cvs*100)
```

```
Cross-Validation Accuracy Scores:  81.28654970760235
```

[49]:
```python
y_pred = gscv.best_estimator_.predict(test_X)
print('Test accuracy of Random Forest after Grid Search CV : ',␣
 ↪accuracy_score(test_y, y_pred)*100)
```

```
Test accuracy of Random Forest after Grid Search CV :  82.41758241758241
```

[ ]:
```python
#Conclusion
We prefer the Model created with "Logistic Regression using LDA Algorithm",
which gives the best results as compared to Random Forest Algorithm.
```