Heart Attack

May 14, 2023

[1]:
```
#Importing Libraries
```

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import statsmodels.api as sm
%matplotlib inline
```

[3]:
```
from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

[4]:
```
#1. Preliminary Analysis
```

[5]:
```
#Importing the Dataset
```

[6]:
```
cwd = os.getcwd()
dataset_dir = os.path.join(cwd,'Dataset')
dataset_path = os.path.join(dataset_dir,'1645792390_cep1_dataset.xlsx')
dataset = pd.read_excel(dataset_path)
```

[7]:
```
#Understanding the dataset
```

[8]:
```
#   •   age: The person's age in years
#   •   sex: The person's sex (1 = male, 0 = female)
#
#   •   cp: chest pain type
#           -- Value 0: asymptomatic
#           -- Value 1: atypical angina
#           -- Value 2: non-anginal pain
#           -- Value 3: typical angina
#
```

```
#   •   trestbps: The person's resting blood pressure (mm Hg on admission to␣
 ↪the hospital)
#   •   chol: The person's cholesterol measurement in mg/dl
#   •   fbs: The person's fasting blood sugar (> 120 mg/dl, 1 = true; 0 = false)
#
#   •   restecg: resting electrocardiographic results
#           -- Value 0: showing probable or definite left ventricular␣
 ↪        hypertrophy by Estes' criteria
#           -- Value 1: normal
#           -- Value 2: having ST-T wave abnormality (T wave inversions and/or␣
 ↪ST elevation or depression of > 0.05 mV)
#
#   •   thalach: The person's maximum heart rate achieved
#   •   exang: Exercise induced angina (1 = yes; 0 = no)
#   •   oldpeak: ST depression induced by exercise relative to rest ('ST'␣
 ↪relates to positions on the ECG plot.)
#   •   slope: the slope of the peak exercise ST segment (0: downsloping; 1:␣
 ↪flat; 2: upsloping)
#   •   ca: The number of major vessels (0-3)
#   •   thal: Results of the blood flow observed via the radioactive dye.
#
#           Value 1: fixed defect (no blood flow in some part of the heart)
#           Value 2: normal blood flow
#           Value 3: reversible defect (a blood flow is observed but it is not␣
 ↪normal)
#
#   •   target : 0 = disease, 1 = no disease
```

[9]:
```
dataset.shape
```

[9]:
```
(303, 14)
```

[10]:
```
dataset.columns
```

[10]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[11]:
```
dataset.head()
```

[11]:

	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
```

[12]:
```
dataset.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
```

[13]:
```
#1.a. Finding out null values in dataset. If 0 means no missing value.
```

[14]:
```
dataset.isna().sum()
```

[14]:
```
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
```

[15]:
```
#Checking if dataset is imbalanced or not.
```

[16]:
```
dataset.target.value_counts()
```

[16]:
```
1    165
0    138
Name: target, dtype: int64
```

[17]:
```
#1.b. Checking for duplicates
```

[18]:
```
dataset[dataset.duplicated()==True]
```

[18]:
```
     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
```

[19]:
```
#Removing duplicates
```

[20]:
```
dataset.drop_duplicates(keep='last',inplace=True)
dataset.shape
```

[20]:
```
(302, 14)
```

[21]:
```
#2. Understanding distribution of disease and related factors
```

[22]:
```
#2.a. Preliminary statistical summary of the data
```

[23]:
```
dataset.describe()
```

[23]:
```
              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
```

[24]:
```
#For ca max should be 3 as it lies between 0-3
#For thal min should be 1 as it lies between 1-3
#So dropping those rows
```

[25]:
```
dataset = dataset[(dataset['ca'] >= 0) & (dataset['ca'] <= 3)]
dataset = dataset[(dataset['thal'] >= 1) & (dataset['thal'] <= 3)]
dataset.describe()
```

[25]:
```
              age         sex          cp   trestbps        chol         fbs  \
count  296.000000  296.000000  296.000000  296.00000  296.000000  296.00000
mean    54.523649    0.679054    0.959459  131.60473  247.155405    0.14527
std      9.059471    0.467631    1.034184   17.72662   51.977011    0.35297
min     29.000000    0.000000    0.000000   94.00000  126.000000    0.00000
25%     48.000000    0.000000    0.000000  120.00000  211.000000    0.00000
50%     56.000000    1.000000    1.000000  130.00000  242.500000    0.00000
75%     61.000000    1.000000    2.000000  140.00000  275.250000    0.00000
max     77.000000    1.000000    3.000000  200.00000  564.000000    1.00000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  296.000000  296.000000  296.000000  296.000000  296.000000  296.000000
mean     0.523649  149.560811    0.327703    1.059122    1.395270    0.679054
std      0.526692   22.970792    0.470171    1.166474    0.618235    0.939726
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.000000    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    1.650000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000    3.000000

             thal      target
count  296.000000  296.000000
mean     2.327703    0.540541
```

```
std      0.585743    0.499198
min      1.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

[26]:
```
#Exploring the measures of central tendencies and spread of the data
```

[27]:
```
# Measures of central tendency
print("Mean:")
print(dict(dataset.mean().map('{:,.4f}'.format)))
print("\nMedian:")
print(dict(dataset.median()))
print("\nMode:")
print(dict(dataset.mode().iloc[0]))

# Measures of spread
print("\nStandard Deviation:")
print(dict(dataset.std().map('{:,.4f}'.format)))
print("\nRange:")
print(dict(dataset.max() - dataset.min()))
print("\nInterquartile Range:")
print(dict(dataset.quantile(0.75) - dataset.quantile(0.25)))
```

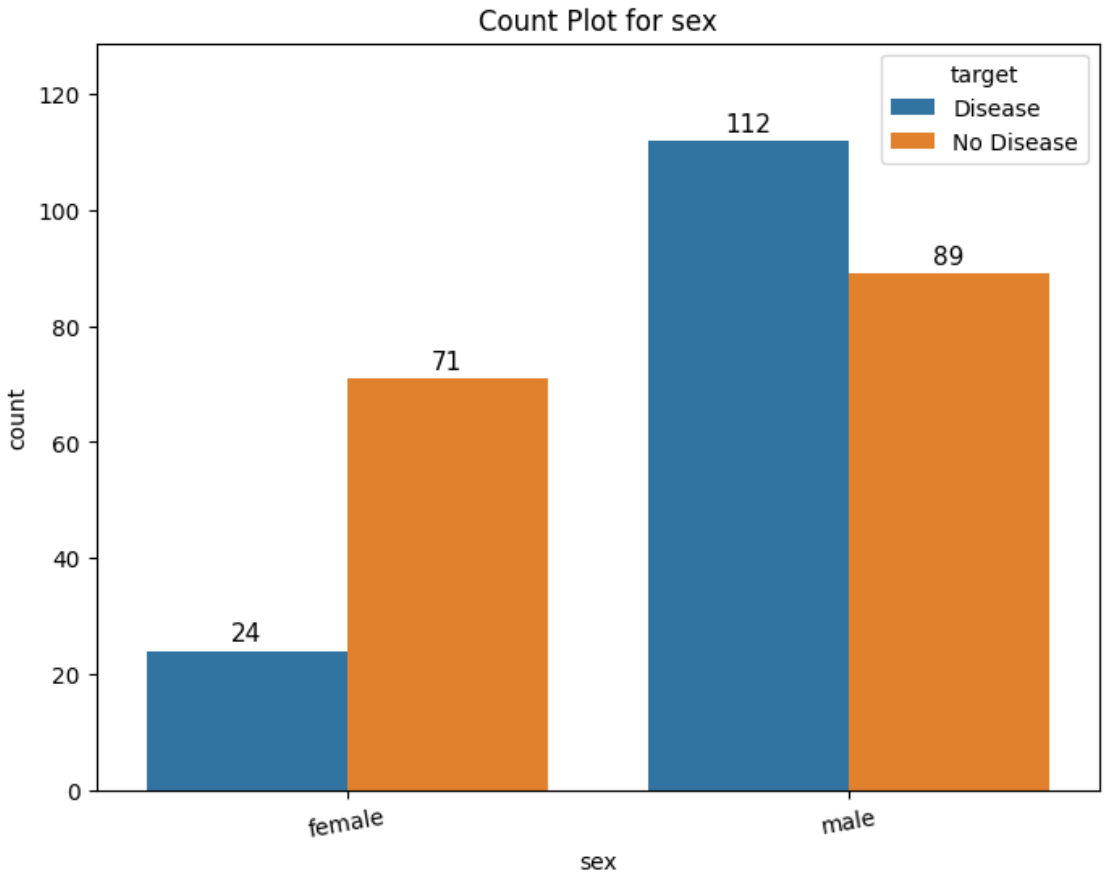
```
Mean:
{'age': '54.5236', 'sex': '0.6791', 'cp': '0.9595', 'trestbps': '131.6047',
'chol': '247.1554', 'fbs': '0.1453', 'restecg': '0.5236', 'thalach': '149.5608',
'exang': '0.3277', 'oldpeak': '1.0591', 'slope': '1.3953', 'ca': '0.6791',
'thal': '2.3277', 'target': '0.5405'}

Median:
{'age': 56.0, 'sex': 1.0, 'cp': 1.0, 'trestbps': 130.0, 'chol': 242.5, 'fbs':
0.0, 'restecg': 1.0, 'thalach': 152.5, 'exang': 0.0, 'oldpeak': 0.8, 'slope':
1.0, 'ca': 0.0, 'thal': 2.0, 'target': 1.0}

Mode:
{'age': 58.0, 'sex': 1.0, 'cp': 0.0, 'trestbps': 120.0, 'chol': 197.0, 'fbs':
0.0, 'restecg': 1.0, 'thalach': 162.0, 'exang': 0.0, 'oldpeak': 0.0, 'slope':
2.0, 'ca': 0.0, 'thal': 2.0, 'target': 1.0}

Standard Deviation:
{'age': '9.0595', 'sex': '0.4676', 'cp': '1.0342', 'trestbps': '17.7266',
'chol': '51.9770', 'fbs': '0.3530', 'restecg': '0.5267', 'thalach': '22.9708',
'exang': '0.4702', 'oldpeak': '1.1665', 'slope': '0.6182', 'ca': '0.9397',
'thal': '0.5857', 'target': '0.4992'}

Range:
```

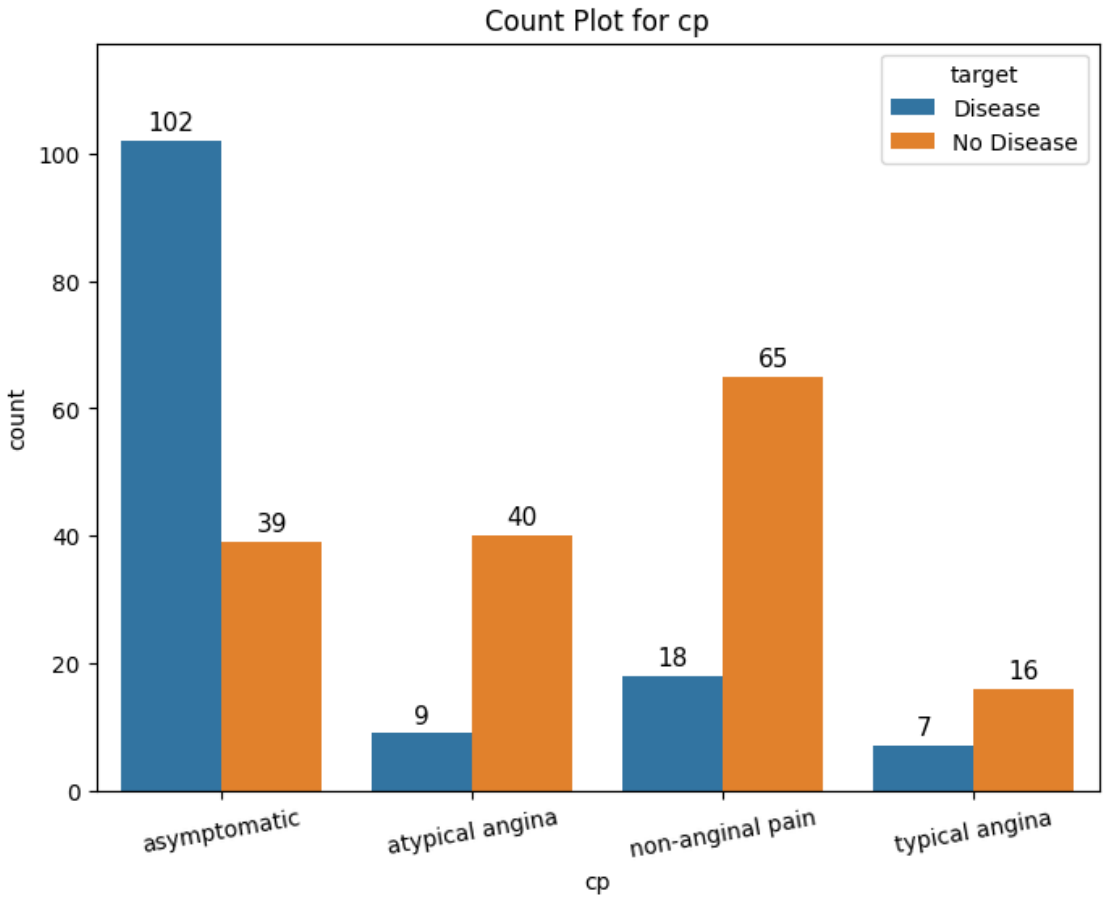
```
{'age': 48.0, 'sex': 1.0, 'cp': 3.0, 'trestbps': 106.0, 'chol': 438.0, 'fbs':
1.0, 'restecg': 2.0, 'thalach': 131.0, 'exang': 1.0, 'oldpeak': 6.2, 'slope':
2.0, 'ca': 3.0, 'thal': 2.0, 'target': 1.0}

Interquartile Range:
{'age': 13.0, 'sex': 1.0, 'cp': 2.0, 'trestbps': 20.0, 'chol': 64.25, 'fbs':
0.0, 'restecg': 1.0, 'thalach': 33.0, 'exang': 1.0, 'oldpeak':
1.6500000000000001, 'slope': 1.0, 'ca': 1.0, 'thal': 1.0, 'target': 1.0}
```

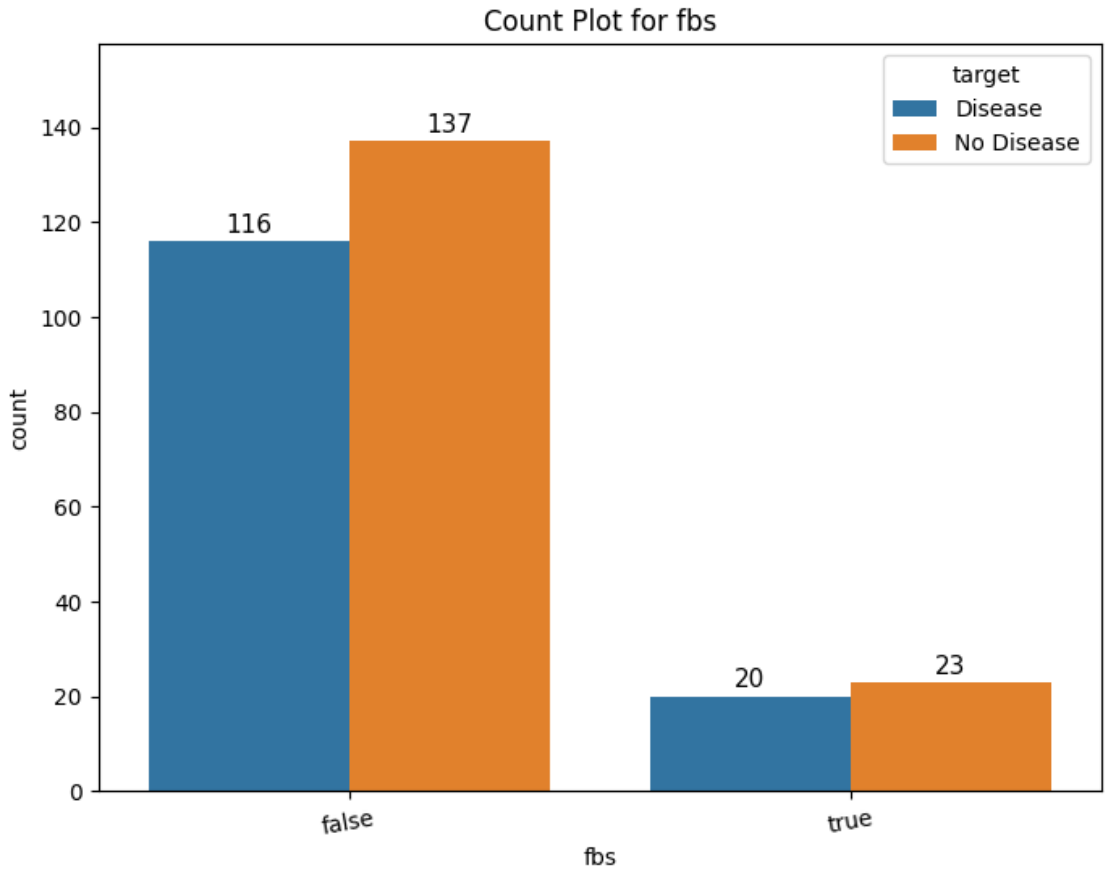
[28]:
```
#2.b. Identifying Categorical data variables and plotting them for visualization
```

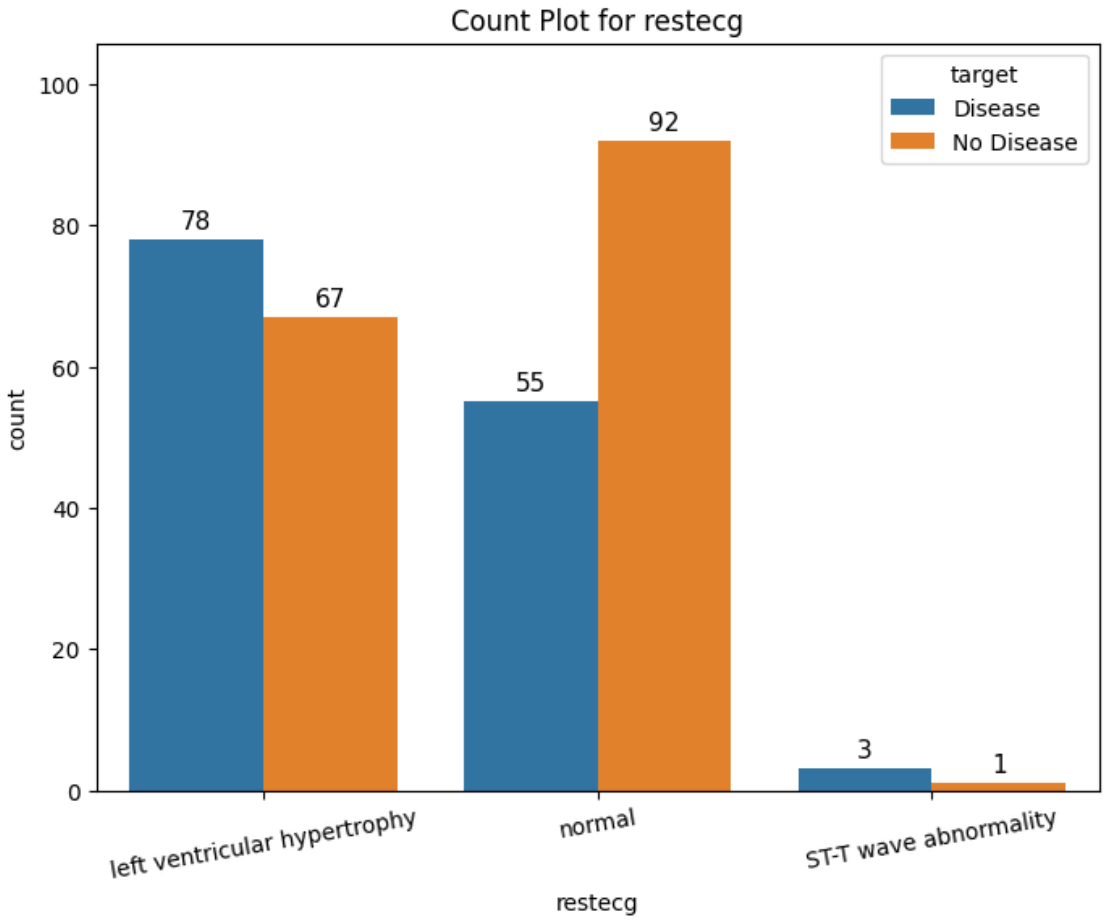
[29]:
```
graph_dir = os.path.join(cwd,'Graph')
```

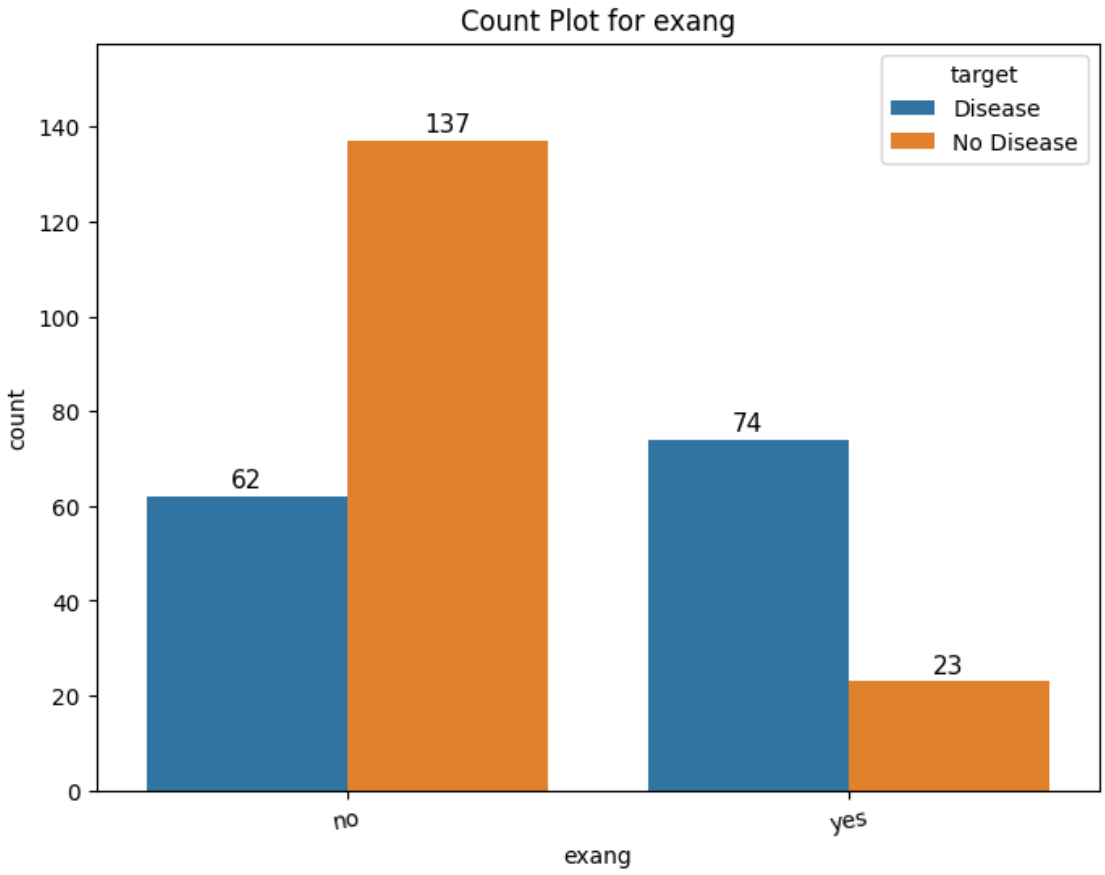
[30]:
```
categorical_vars = {'sex':['female','male'],
                    'cp':['asymptomatic','atypical angina','non-anginal␣
 ↪pain','typical angina'],
                    'fbs':['false','true'],
                    'restecg':['left ventricular hypertrophy','normal','ST-T␣
 ↪wave abnormality'],
                    'exang':['no','yes'],
                    'slope':['downsloping','flat','upsloping'],
                    'ca':['0 vessel','1 vessel','2 vessels','3 vessels'],
                    'thal':['fixed defect','normal blood flow','reversible␣
 ↪defect']}

for var in list(categorical_vars.keys()):
    plt.figure(figsize=(8,6))
    ax = sns.countplot(x=var, data=dataset, hue='target')
    title = 'Count Plot for {}'.format(var)
    plt.xticks(np.
 ↪arange(len(list(categorical_vars[var]))),list(categorical_vars[var]),␣
 ↪rotation=10)
    plt.legend(title='target', labels=['Disease', 'No Disease'])
    plt.margins(None,0.15)
    plt.title(title)
    for p in ax.patches:
        ax.annotate(int(p.get_height()), (p.get_x() + p.get_width() / 2. , p.
 ↪get_height()+1),
                    ha='center', va='center', fontsize=11, color='black',␣
 ↪xytext=(0, 5),
                    textcoords='offset points')
    plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
    plt.show()
    plt.close()
```

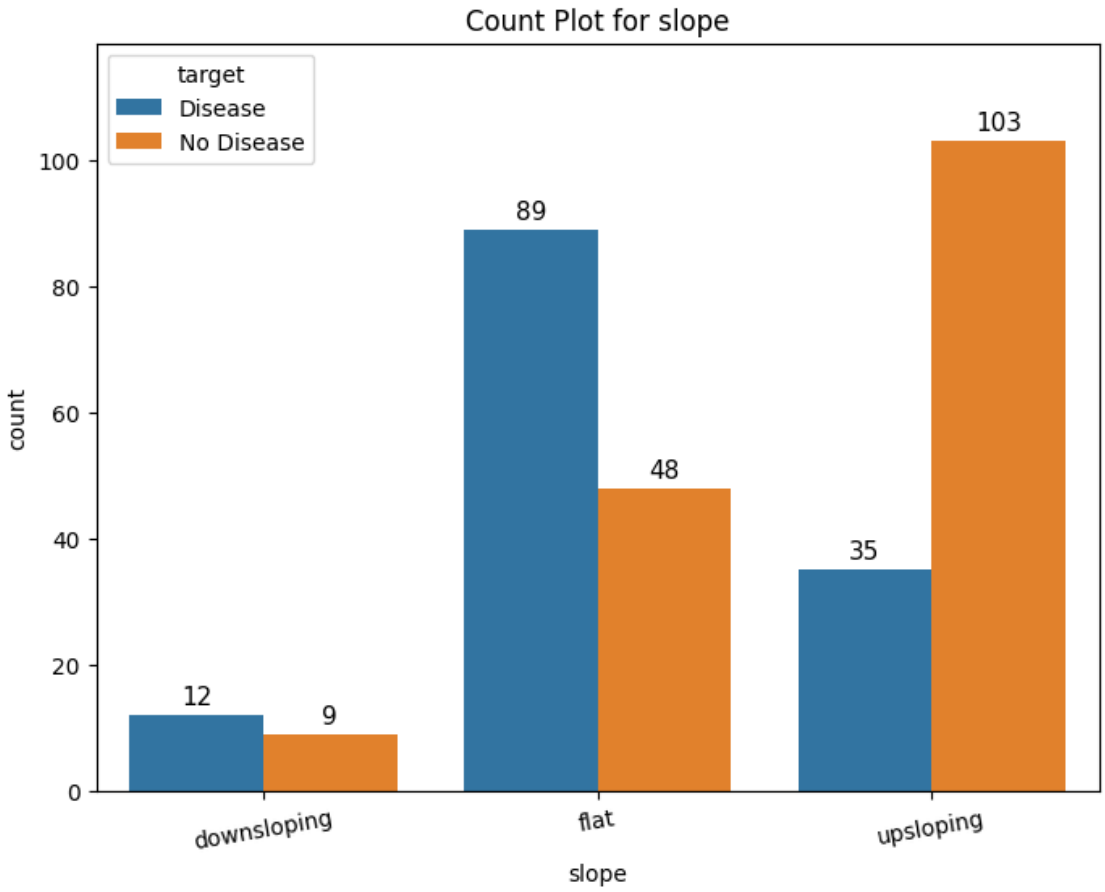

Count Plot for sex
target
Disease
No Disease
120
100
80
60
40
20
0
count
24
71
112
89
female
male
sex

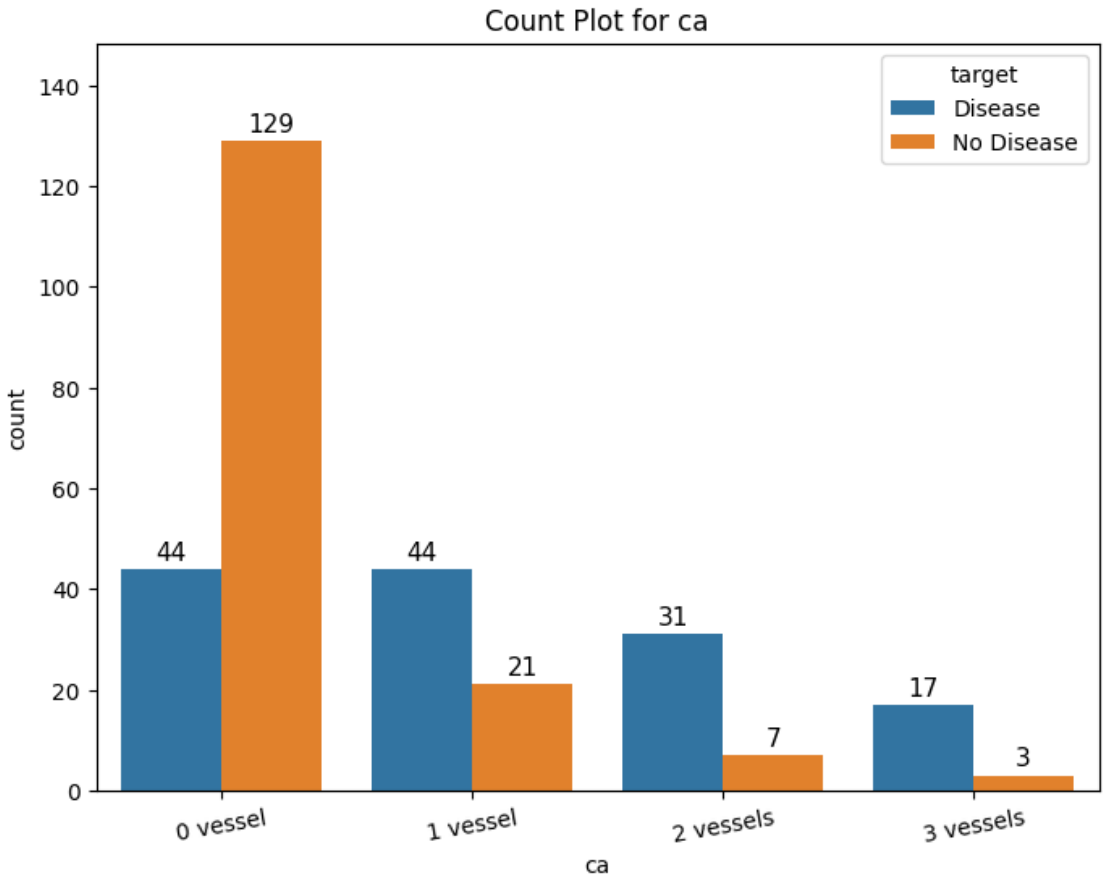

Count Plot for cp
target
Disease
No Disease
count
cp
asymptomatic
atypical angina
non-anginal pain
typical angina
102
39
9
40
18
65
7
16
0
20
40
60
80
100

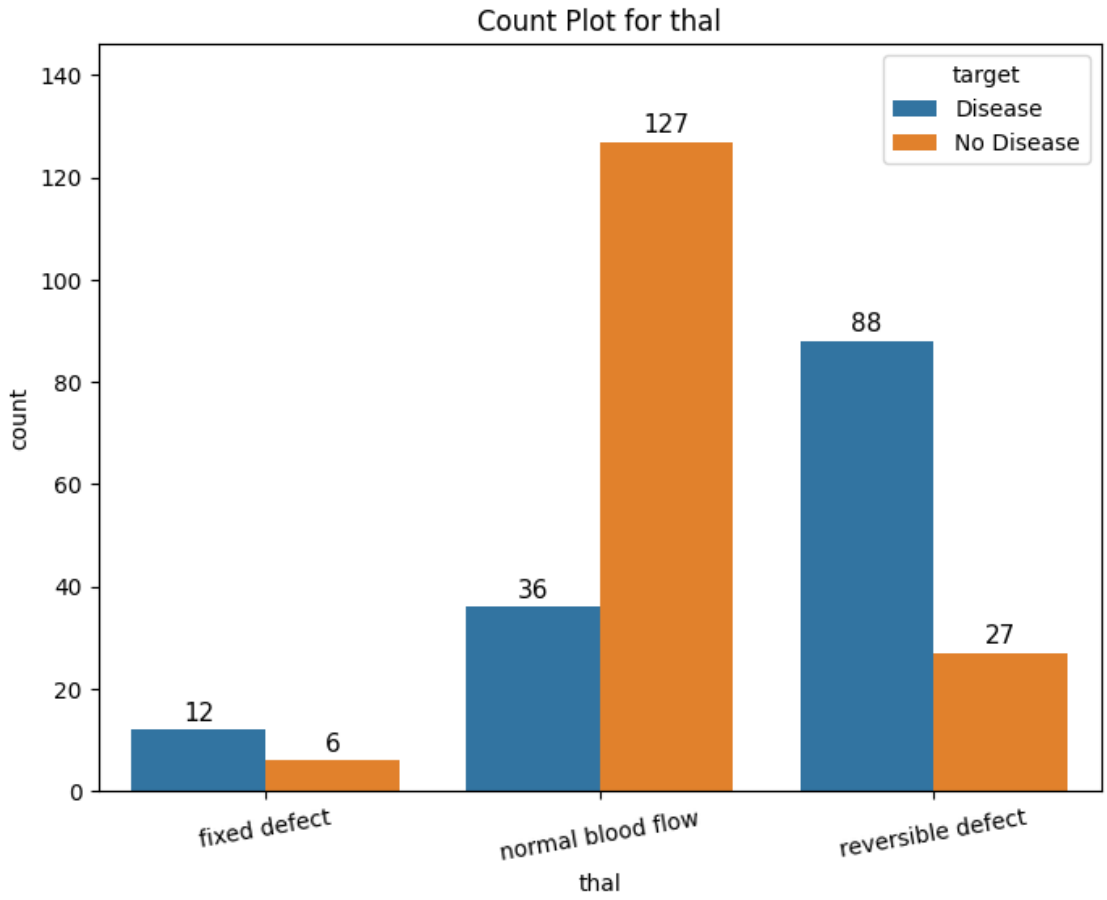

Count Plot for fbs
target
Disease
No Disease
count
fbs
false
true
116
137
20
23
0
20
40
60
80
100
120
140


Count Plot for restecg
target
Disease
No Disease
count
100
80
60
40
20
0
78
67
55
92
3
1
left ventricular hypertrophy
normal
ST-T wave abnormality
restecg


Count Plot for exang
target
Disease
No Disease
count
140
120
100
80
60
40
20
0
137
62
74
23
no
yes
exang


Count Plot for slope
target
Disease
No Disease
count
0
20
40
60
80
100
12
9
89
48
35
103
downsloping
flat
upsloping
slope

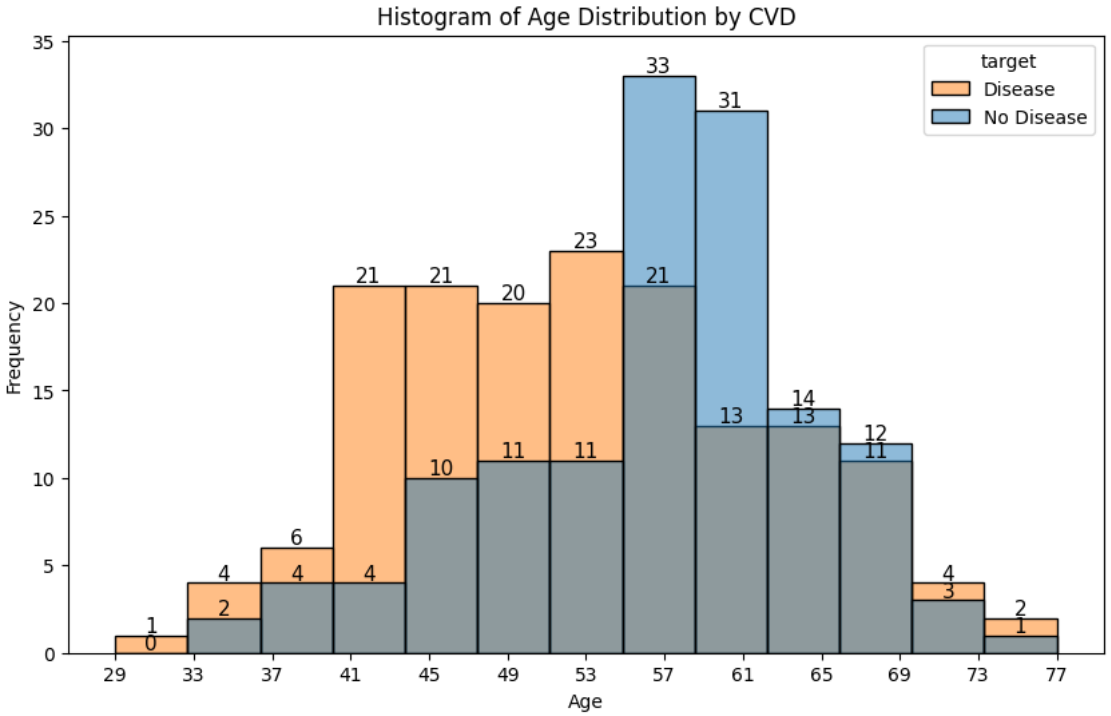

Count Plot for ca
target
Disease
No Disease
count
ca
0 vessel
1 vessel
2 vessels
3 vessels
129
44
44
21
31
7
17
3
0
20
40
60
80
100
120
140



[31]:
```
#2.c. Study the occurrence of CVD across the Age category
```

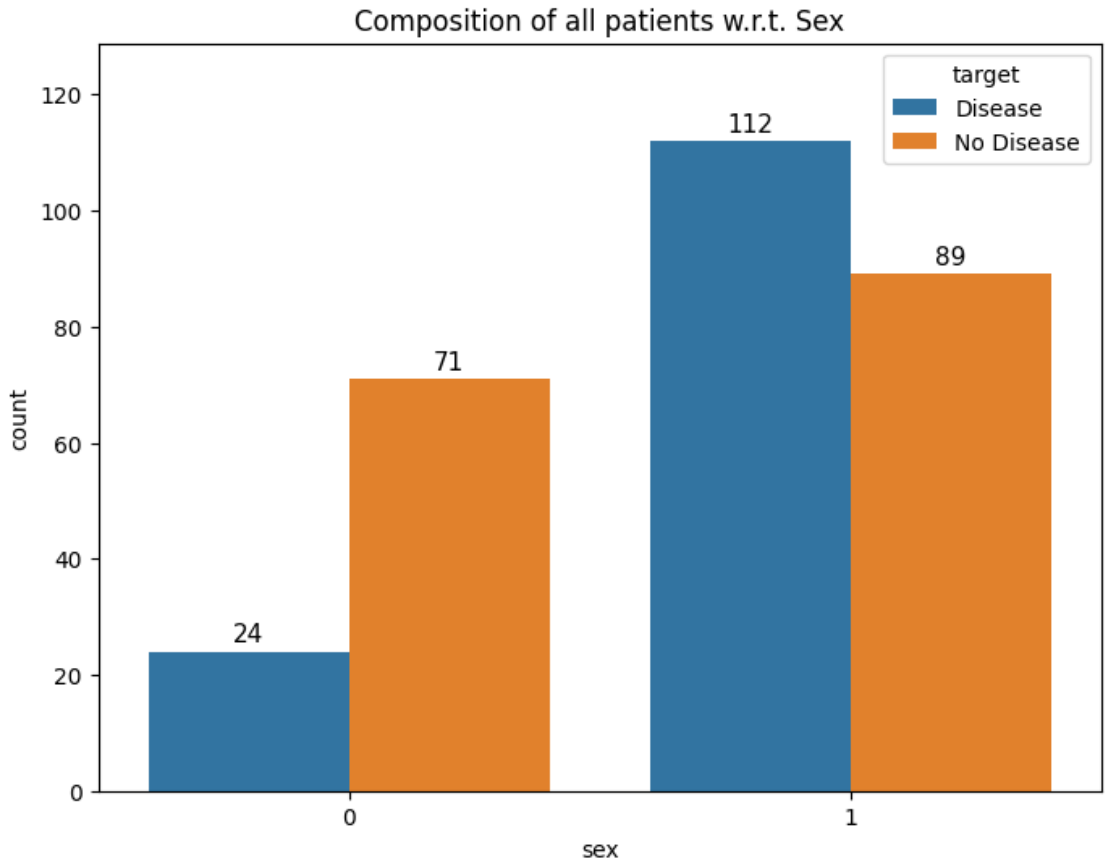
[32]:
```
plt.figure(figsize=(10,6))
ax = sns.histplot(x='age',data=dataset,hue='target',legend=False)
for p in ax.patches:
    ax.annotate(int(p.get_height()), (p.get_x() + p.get_width() / 2. , p.
 ↪get_height()),
                ha='center', va='center', fontsize=11, color='black',␣
 ↪xytext=(0, 5),
                textcoords='offset points')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.margins(None,0.07)
plt.xticks(np.arange(min(dataset['age']), max(dataset['age'])+1, 4))
title = 'Histogram of Age Distribution by CVD'
plt.title(title)
plt.legend(title='target', labels=['Disease', 'No Disease'])
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
```

```
plt.show()
plt.close()
```



[33]:
```
#2.d. Study the composition of all patients with respect to the Sex category
```

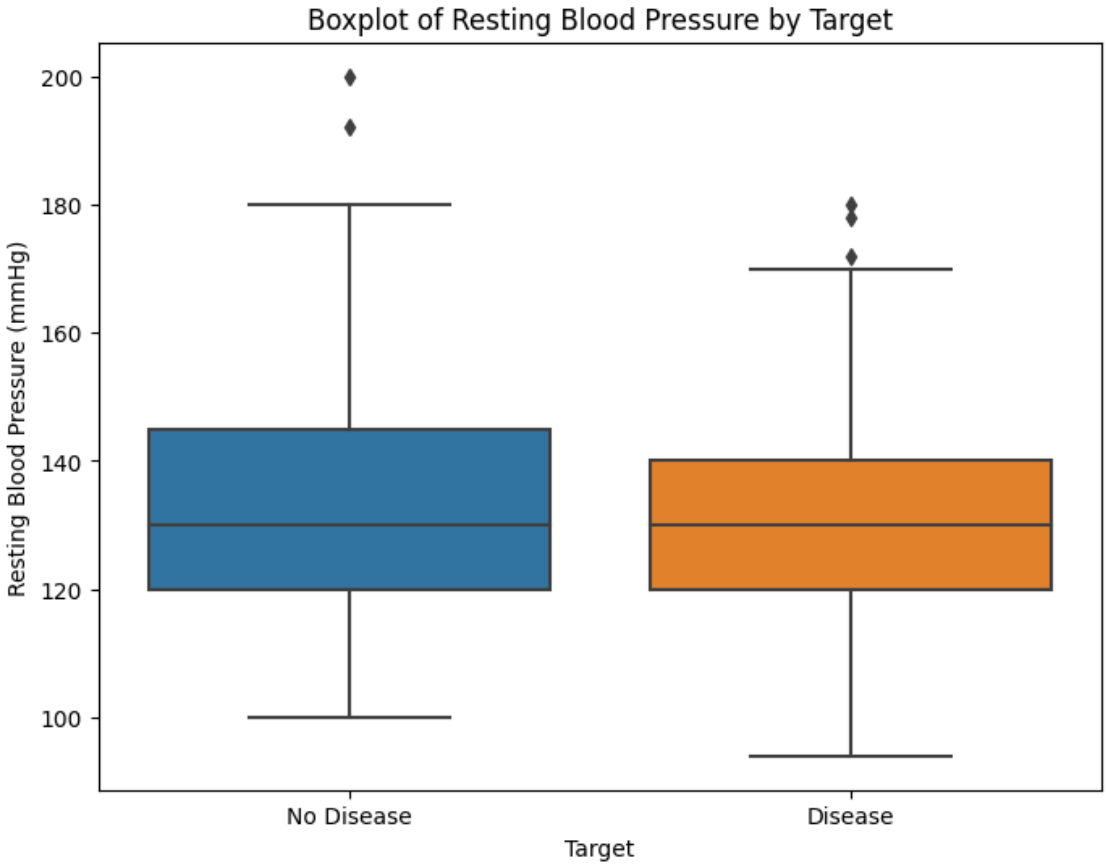
[34]:
```
plt.figure(figsize=(8,6))
ax = sns.countplot(x='sex', hue='target', data=dataset)
title = 'Composition of all patients w.r.t. Sex'
plt.title(title)
plt.legend(title='target', labels=['Disease', 'No Disease'])
plt.margins(None,0.15)
for p in ax.patches:
    ax.annotate(int(p.get_height()), (p.get_x() + p.get_width() / 2. , p.
 ↪get_height()+1),
                ha='center', va='center', fontsize=11, color='black',␣
 ↪xytext=(0, 5),
                textcoords='offset points')
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
plt.show()
plt.close()
```

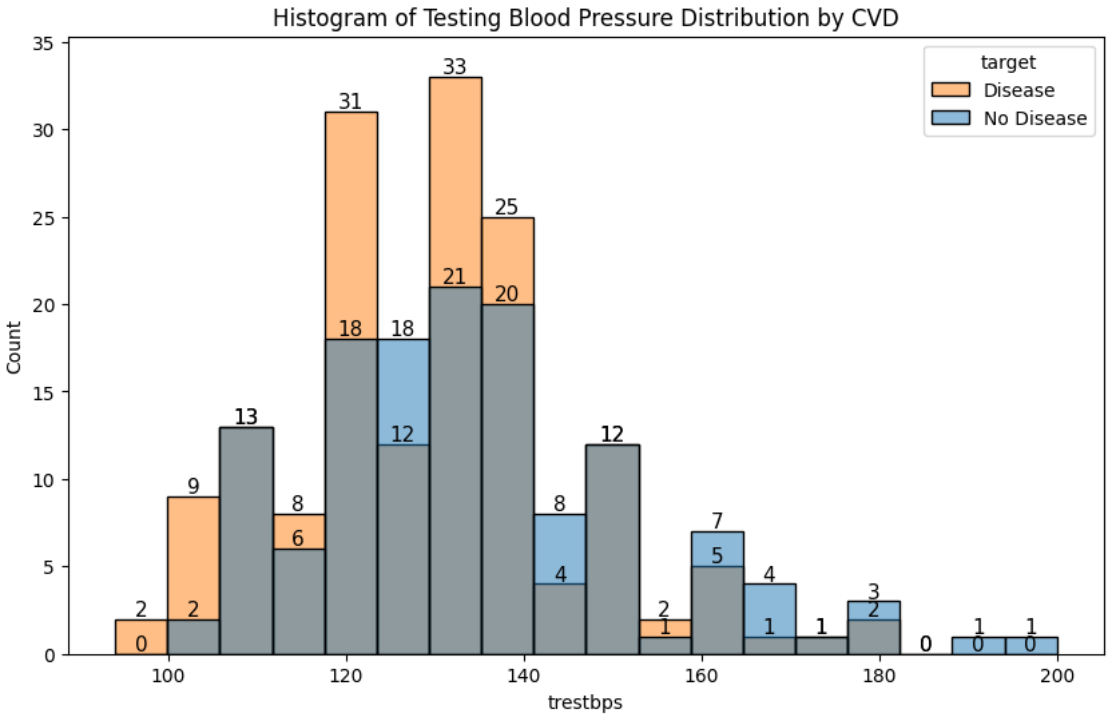


[35]:
```
#2.e. Study if one can detect heart attacks based on anomalies in the resting␣
↪blood pressure (trestbps) of a patient
```

[36]:
```
#Box Plot for this
plt.figure(figsize=(8,6))
sns.boxplot(x='target', y='trestbps', data=dataset)
plt.xticks([0,1], ['No Disease', 'Disease'])
plt.xlabel('Target')
plt.ylabel('Resting Blood Pressure (mmHg)')
title = 'Boxplot of Resting Blood Pressure by Target'
plt.title(title)
plt.savefig(os.path.join(graph_dir, title+'.png'), bbox_inches='tight')
plt.show()
plt.close()

#Histogram for the same
plt.figure(figsize=(10,6))
ax = sns.histplot(x='trestbps',data=dataset,hue='target',legend=False)
for p in ax.patches:
```

```
    ax.annotate(int(p.get_height()), (p.get_x() + p.get_width() / 2. , p.
 ↪get_height()),
                ha='center', va='center', fontsize=11, color='black',␣
 ↪xytext=(0, 5),
                textcoords='offset points')
plt.xlabel('trestbps')
plt.ylabel('Count')
plt.margins(None,0.07)
title = 'Histogram of Testing Blood Pressure Distribution by CVD'
plt.title(title)
plt.legend(title='target', labels=['Disease', 'No Disease'])
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
plt.show()
plt.close()
```

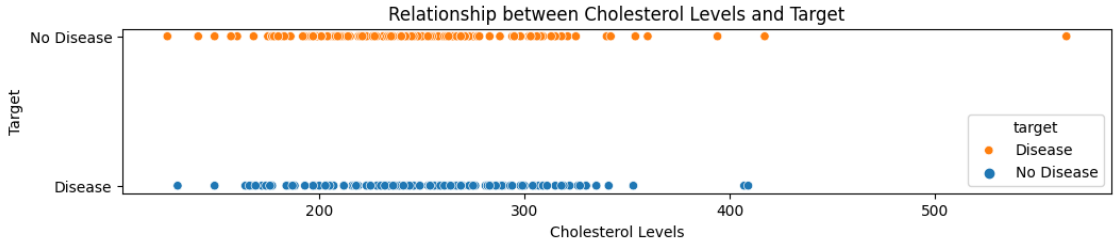


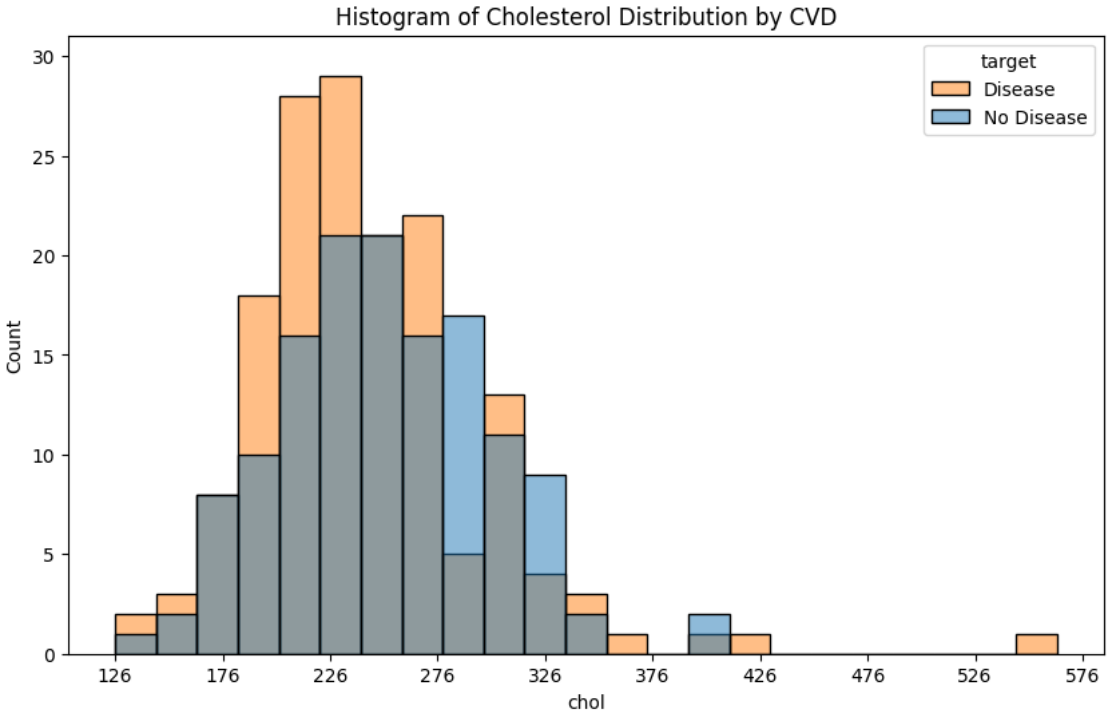


[37]:
```
#2.f. Describe the relationship between cholesterol levels and a target variable
```

[38]:
```
# Create scatter plot of cholesterol levels and target variable
plt.figure(figsize=(12,2))
sns.scatterplot(x='chol', y='target', data=dataset, hue='target')
plt.xlabel('Cholesterol Levels')
plt.ylabel('Target')
plt.yticks([0,1], ['Disease','No Disease'])
title = 'Relationship between Cholesterol Levels and Target'
plt.title(title)
plt.legend(title='target', labels=['Disease', 'No Disease'])
plt.savefig(os.path.join(graph_dir, title+'.png'), bbox_inches='tight')
plt.show()
plt.close()

#Histogram for the same
plt.figure(figsize=(10,6))
ax = sns.histplot(x='chol',data=dataset,hue='target',legend=False)
plt.xlabel('chol')
plt.ylabel('Count')
plt.margins(None,0.07)
plt.xticks(np.arange(min(dataset['chol']), max(dataset['chol'])+20, 50))
title = 'Histogram of Cholesterol Distribution by CVD'
```

```
plt.title(title)
plt.legend(title='target', labels=['Disease', 'No Disease'])
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
plt.show()
plt.close()
```

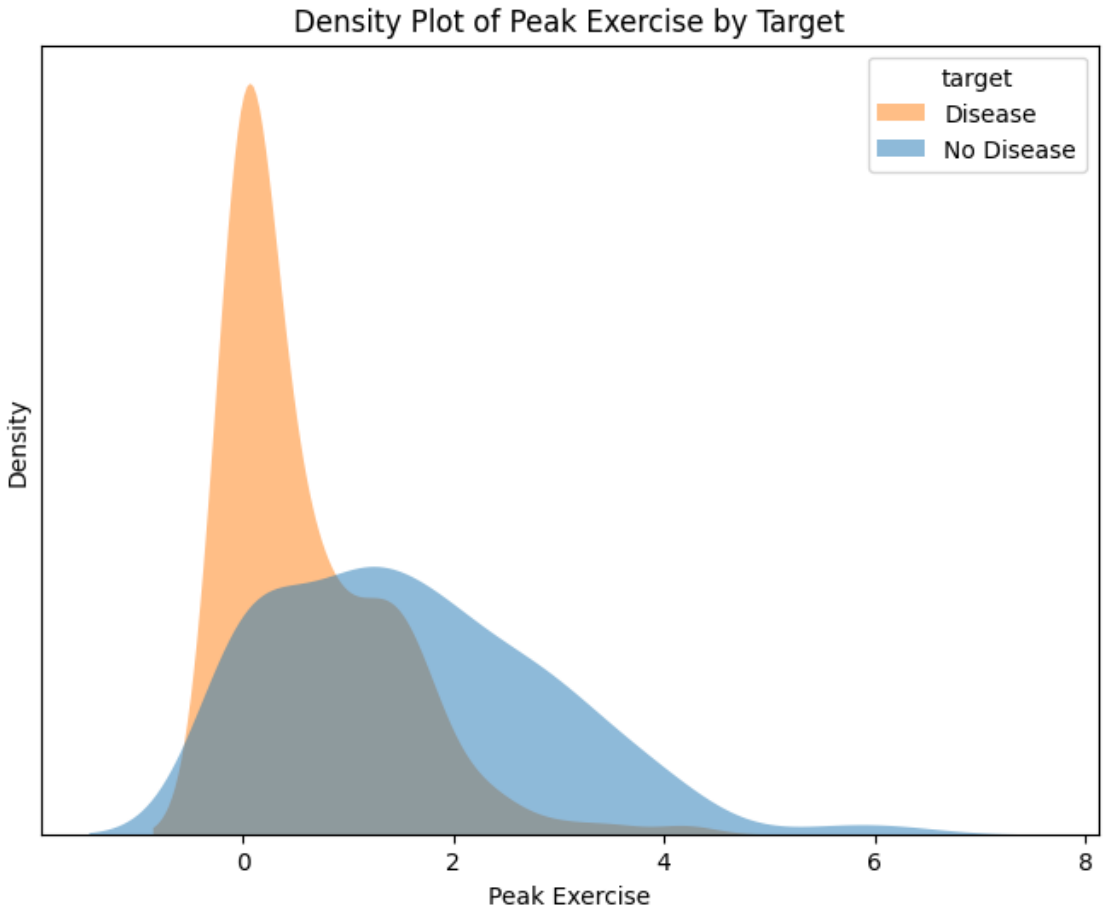




[39]:
```
#2.g. State what relationship exists between peak exercising and the occurrence␣
↪of a heart attack
```

[40]:
```
plt.figure(figsize=(8,6))
sns.kdeplot(x='oldpeak', hue='target', data=dataset, fill=True,␣
↪common_norm=False, alpha=.5, linewidth=0)
plt.xlabel('Peak Exercise')
```

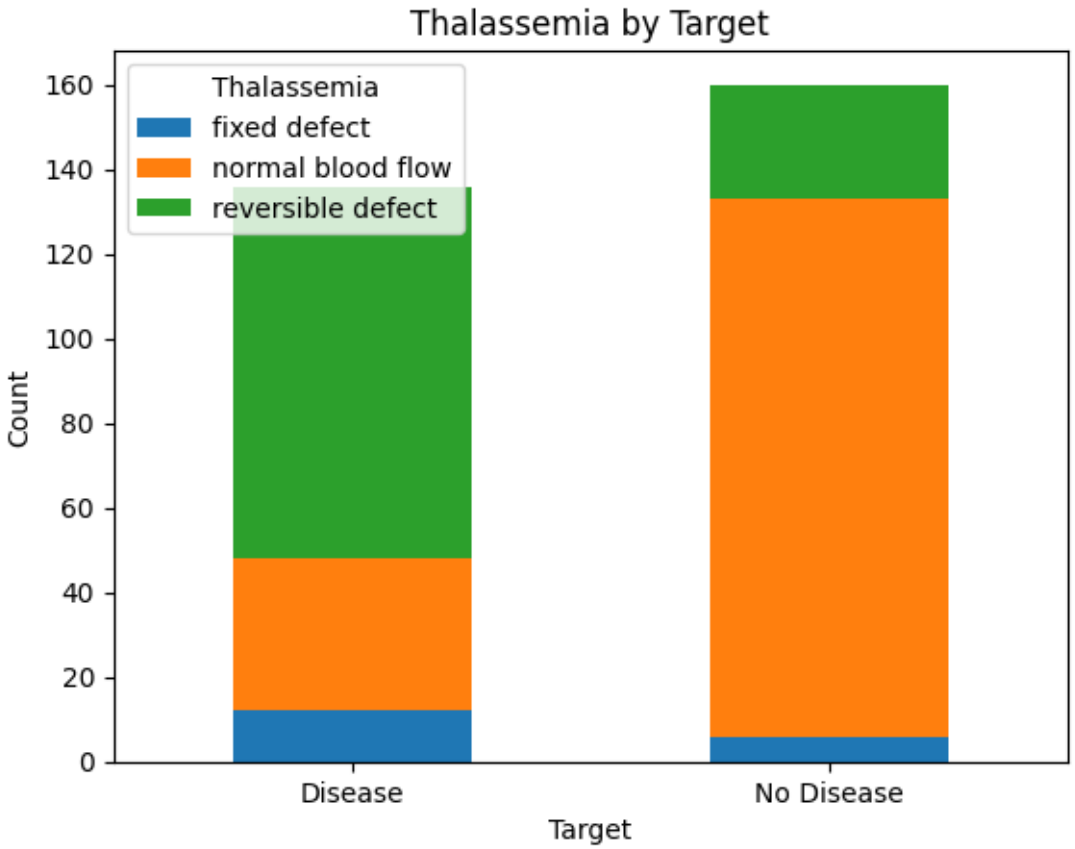
```
plt.ylabel('Density')
plt.yticks([])
title = 'Density Plot of Peak Exercise by Target'
plt.title(title)
plt.legend(title='target', labels=['Disease', 'No Disease'])
plt.savefig(os.path.join(graph_dir, title+'.png'), bbox_inches='tight')
plt.show()
plt.close()
```



[41]:
```
#2.h. Check if thalassemia is a major cause of CVD
```

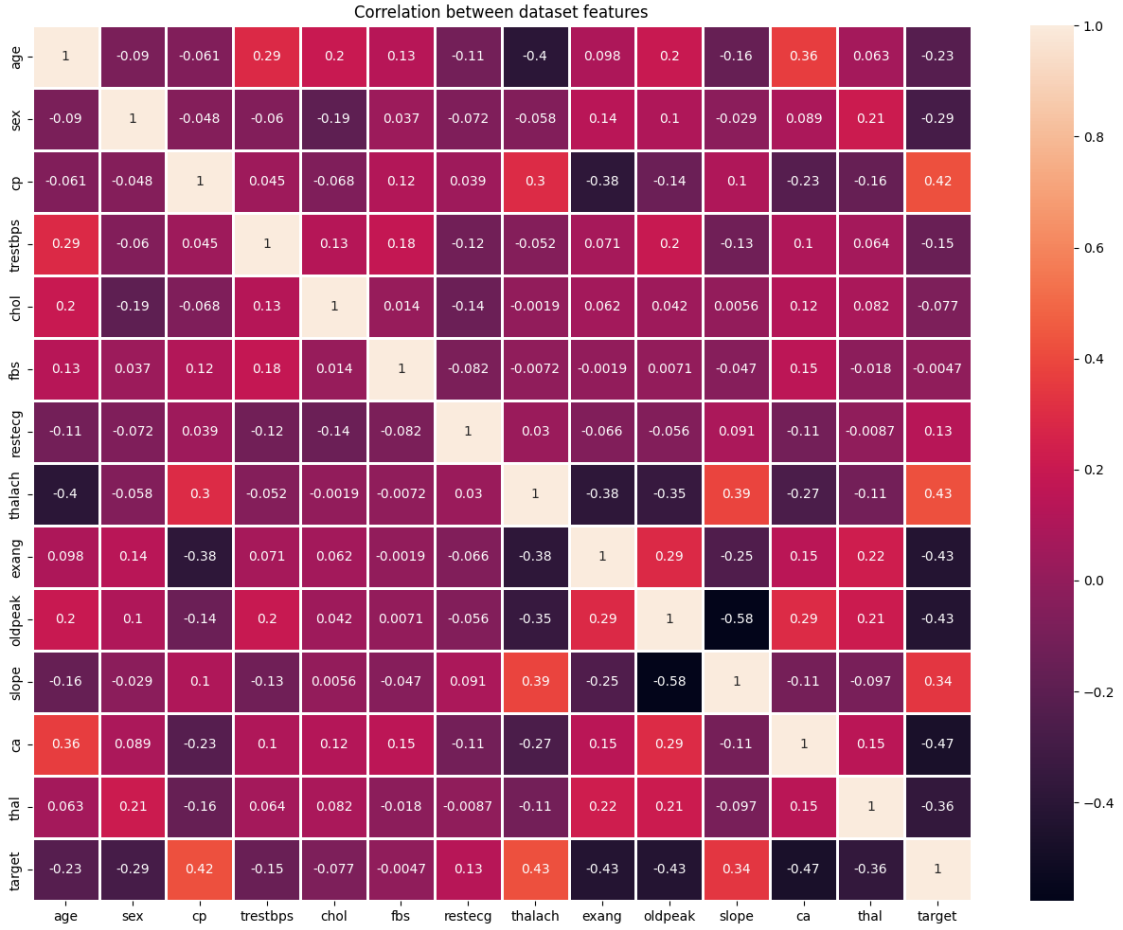
[42]:
```
thalassemia_counts = dataset.groupby(['target', 'thal']).size().unstack()
thalassemia_counts.plot(kind='bar', stacked=True, color=['#1f77b4', '#ff7f0e',␣
↪'#2ca02c'])
plt.xticks([0,1], ['Disease','No Disease'], rotation=0)
plt.xlabel('Target')
plt.ylabel('Count')
title = 'Thalassemia by Target'
```

```
plt.title(title)
plt.legend(title='Thalassemia', labels=list(categorical_vars['thal']))
plt.savefig(os.path.join(graph_dir, title+'.png'), bbox_inches='tight')
plt.show()
plt.close()
```

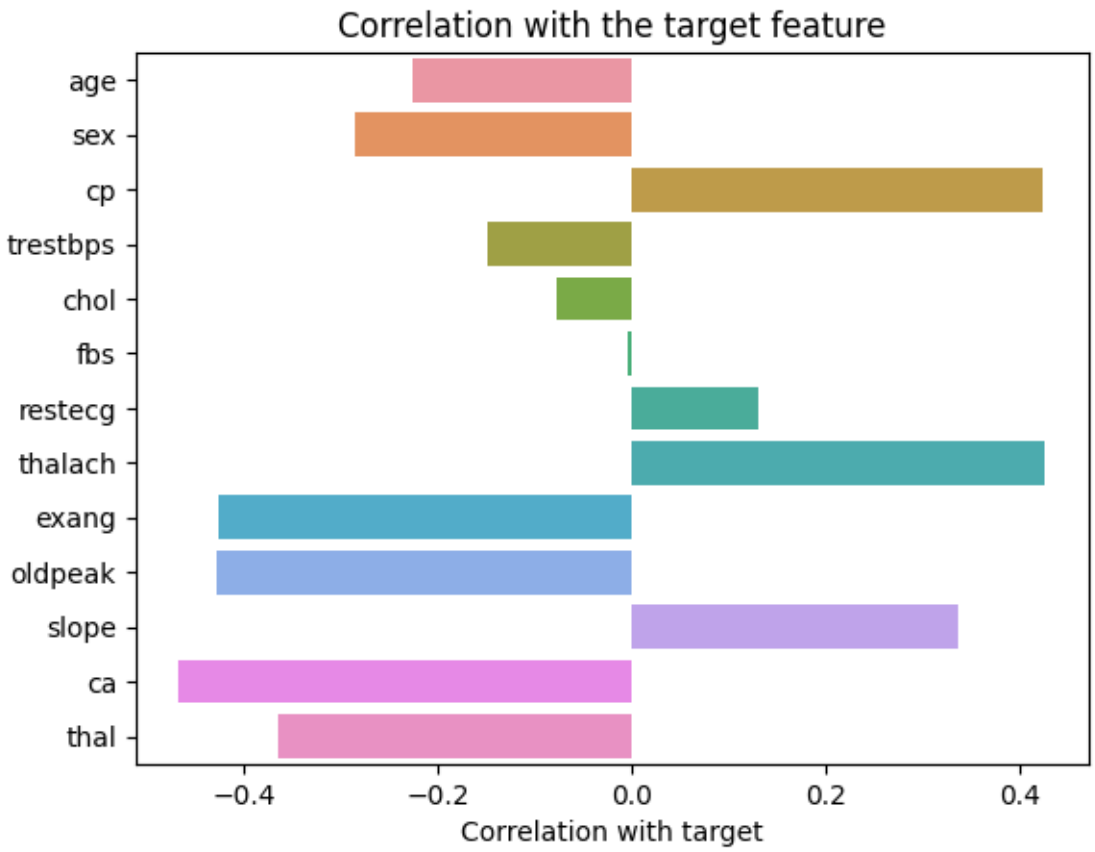


[43]:
```
#2.i. List how the other factors determine the occurrence of CVD
```

[44]:
```
#Correlation between the dataset features

plt.figure(figsize=(16,12))
sns.heatmap(dataset.corr(),annot=True,linewidth =2)
title = 'Correlation between dataset features'
plt.title(title)
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
plt.show()
plt.close()
```



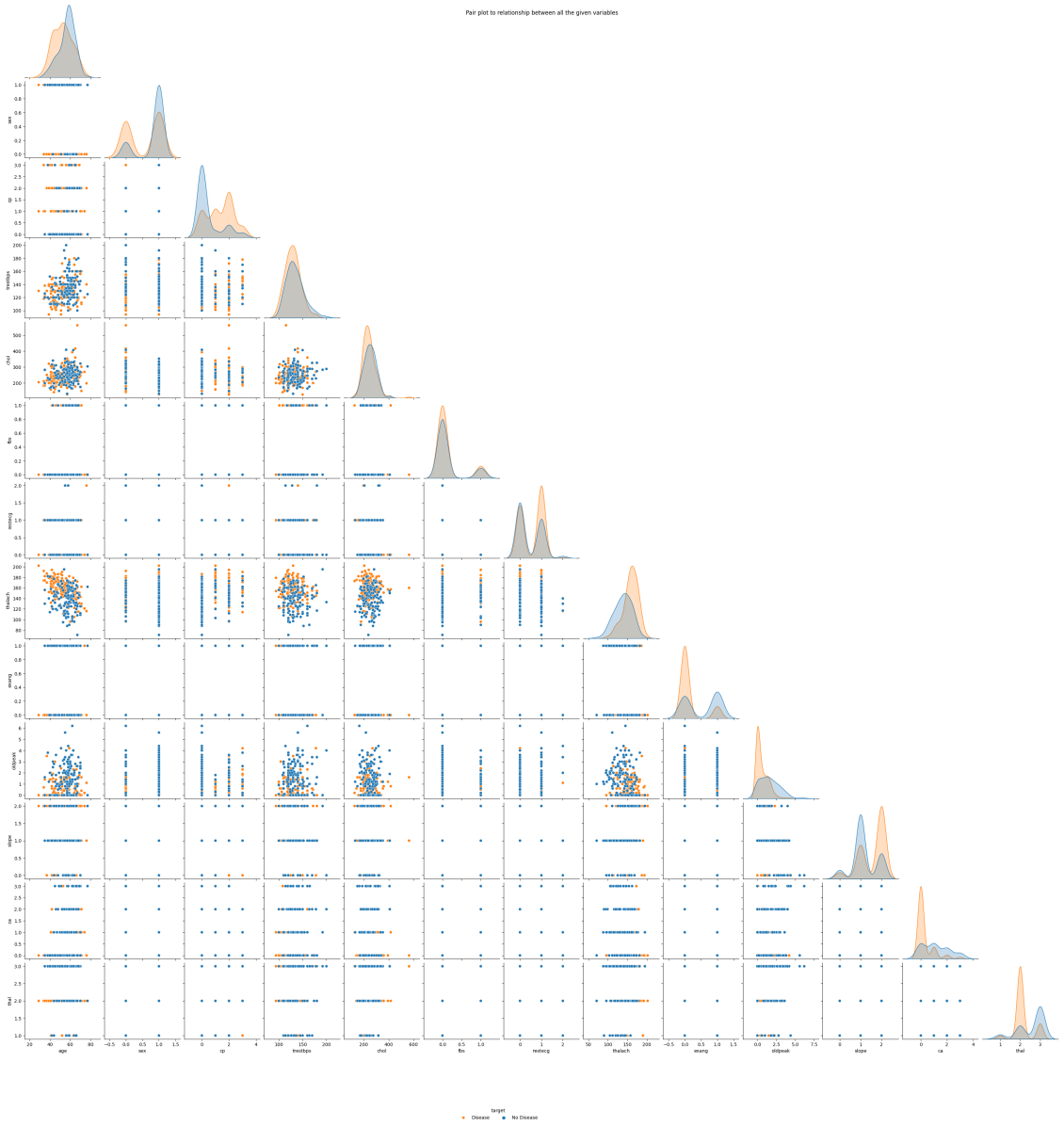
[45]:
```
rem_target = dataset.drop('target', axis=1)
title = 'Correlation with the target feature'
sns.barplot(y=rem_target.columns,x=rem_target.corrwith(dataset['target']))
plt.xlabel('Correlation with target')
plt.title(title)
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
plt.show()
plt.close()
```



[46]:
```
#2.j. Use a pair plot to understand the relationship between all the given␣
↪variables
```

[47]:
```
title = 'Pair plot to relationship between all the given variables'
g = sns.pairplot(data=dataset,hue='target',corner=True)
sns.move_legend(g,1,bbox_to_anchor=(0.5, -0.035), labels=['Disease', 'No␣
↪Disease'], ncol=5, title='target', frameon=False)
plt.suptitle(title)
plt.savefig(os.path.join(graph_dir, title + '.png'), bbox_inches='tight')
plt.show()
plt.close()
```

```
C:\Python311\Lib\site-packages\seaborn\utils.py:482: UserWarning: You have mixed
positional and keyword arguments, some input may be discarded.
  new_legend = legend_func(handles, labels, loc=loc, **props)
```



[48]:
```
#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
```

[49]:
```
corr = dataset.corr()

# Define the features and target variable
X = dataset.drop(['target'], axis=1)
```

```
y = dataset['target']
```

[50]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[51]:
```
# Baseline Logistic Regression Model
pipe = make_pipeline(StandardScaler(), LogisticRegression(solver='liblinear'))
pipe.fit(X_train, y_train)
y_pred_lr = pipe.predict(X_test)
acc_lr = accuracy_score(y_test, y_pred_lr)
cm_lr = confusion_matrix(y_test, y_pred_lr)
print('Logistic Regression Accuracy: {:.3f}'.format(acc_lr*100))
print('Confusion Matrix:\n', cm_lr)
del pipe
```

```
Logistic Regression Accuracy: 85.000
Confusion Matrix:
 [[24  4]
 [ 5 27]]
```

[52]:
```
# Baseline Random Forest Model
pipe = make_pipeline(StandardScaler(), RandomForestClassifier(n_estimators=500,␣
↪random_state=124))
pipe.fit(X_train, y_train)
y_pred_rf = pipe.predict(X_test)
acc_rf = accuracy_score(y_test, y_pred_rf)
cm_rf = confusion_matrix(y_test, y_pred_rf)
print('Random Forest Accuracy: {:.3f}'.format(acc_rf*100))
print('Confusion Matrix:\n', cm_rf)
del pipe
```

```
Random Forest Accuracy: 88.333
Confusion Matrix:
 [[25  3]
 [ 4 28]]
```

[53]:
```
# Feature Selection with Logistic Regression
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.330081
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 target   No. Observations:                  296
Model:                          Logit   Df Residuals:                      283
```

```
Method:                           MLE   Df Model:                           12
Date:                Sun, 14 May 2023   Pseudo R-squ.:                  0.5215
Time:                        23:42:04   Log-Likelihood:                -97.704
converged:                       True   LL-Null:                       -204.20
Covariance Type:            nonrobust   LLR p-value:                 6.738e-39
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
age            0.0288      0.020      1.411      0.158      -0.011       0.069
sex           -1.6888      0.470     -3.595      0.000      -2.610      -0.768
cp             0.7783      0.190      4.095      0.000       0.406       1.151
trestbps      -0.0215      0.010     -2.074      0.038      -0.042      -0.001
chol          -0.0035      0.004     -0.877      0.380      -0.011       0.004
fbs            0.5447      0.584      0.932      0.351      -0.601       1.690
restecg        0.4380      0.358      1.222      0.222      -0.264       1.140
thalach        0.0333      0.009      3.818      0.000       0.016       0.050
exang         -0.8055      0.422     -1.910      0.056      -1.632       0.021
oldpeak       -0.3589      0.219     -1.637      0.102      -0.789       0.071
slope          0.8041      0.363      2.213      0.027       0.092       1.516
ca            -1.3353      0.274     -4.879      0.000      -1.872      -0.799
thal          -1.0149      0.313     -3.241      0.001      -1.629      -0.401
==============================================================================
```

[54]:
```
# Select the significant features based on p-values
sig_features = ['age', 'sex', 'cp', 'thalach', 'exang', 'oldpeak', 'slope',␣
 ↪'ca', 'thal']
X_sig = dataset[sig_features]

# Split the dataset into training and testing sets
X_train_sig, X_test_sig, y_train, y_test = train_test_split(X_sig, y,␣
 ↪test_size=0.2, random_state=42)
```

[55]:
```
# Logistic Regression Model with Significant Features
pipe = make_pipeline(StandardScaler(), LogisticRegression(solver='liblinear'))
pipe.fit(X_train_sig, y_train)
y_pred_lr_sig = pipe.predict(X_test_sig)
acc_lr_sig = accuracy_score(y_test, y_pred_lr_sig)
cm_lr_sig = confusion_matrix(y_test, y_pred_lr_sig)
print('Logistic Regression Accuracy with Significant Features: {:.3f}'.
 ↪format(acc_lr_sig*100))
print('Confusion Matrix:\n', cm_lr_sig)
del pipe
```

```
Logistic Regression Accuracy with Significant Features: 88.333
Confusion Matrix:
 [[25  3]
 [ 4 28]]
```

[56]:
```
# Random Forest Model with Significant Features
pipe = make_pipeline(StandardScaler(), RandomForestClassifier(n_estimators=500,␣
 ↪random_state=124))
pipe.fit(X_train_sig, y_train)
y_pred_rf_sig = pipe.predict(X_test_sig)
acc_rf_sig = accuracy_score(y_test, y_pred_rf_sig)
cm_rf_sig = confusion_matrix(y_test, y_pred_rf_sig)
print('Random Forest Accuracy with Significant Features: {:.3f}'.
 ↪format(acc_rf_sig*100))
print('Confusion Matrix:\n', cm_rf_sig)
```

```
Random Forest Accuracy with Significant Features: 85.000
Confusion Matrix:
 [[23  5]
 [ 4 28]]
```

[ ]: