IITK DA: Data Analytics Capstone

Project on Healthcare

Description:

Cardiovascular diseases are the leading cause of death globally. To identify the causes and to develop a system to predict heart attack in an effective manner is necessary. The presented data has all information about all the relevant factors that might have an impact on heart health. The data needs to be explained in detail for any further analysis.

Preliminary analysis:

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

```
import pandas as pd
import numpy as np


df= pd.read_excel('data.xlsx')

df.head()
```

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

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
```

df.dtypes

```
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
```

df.shape

(303, 14)

print('The number of rows in the dataframe = ', df.shape[0])
print('The number of columns in the dataframe = ', df.shape[1])

The number of rows in the dataframe = 303
The number of columns in the dataframe = 14

[23]:

df['target'].value_counts()

target
1 165

0 138
Name: count, dtype: int64

[24]:

df.duplicated()

0 False
1 False
2 False
3 False
4 False
...
298 False
299 False
300 False
301 False
302 False
Length: 303, dtype: bool

df.duplicated().sum()
1
df.drop_duplicates(inplace=True)
df.reset_index(drop=True)

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
297	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
298	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
299	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
300	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
301	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

302 rows × 14 columns

df.isnull().sum()

```
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
```

```
# no missing values were found.
# if found then they could be dropped
# using 'new_df = df.dropna()'
# print 'new_df'
```

Prepare an informative report about the data explaining distribution of the disease and the related factors:

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

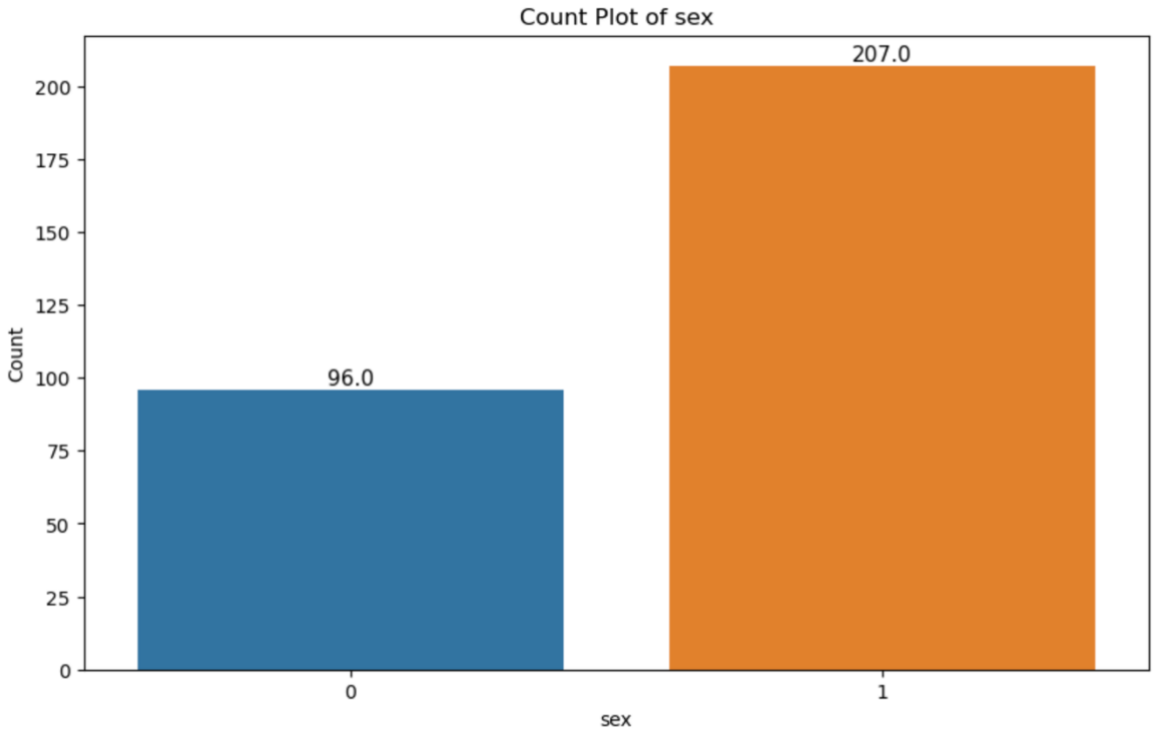
```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	0.526490	149.569536	0.327815	1.043046	1.397351	0.718543	2.314570	0.543046
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686	0.526027	22.903527	0.470196	1.161452	0.616274	1.006748	0.613026	0.498970
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.250000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	152.500000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

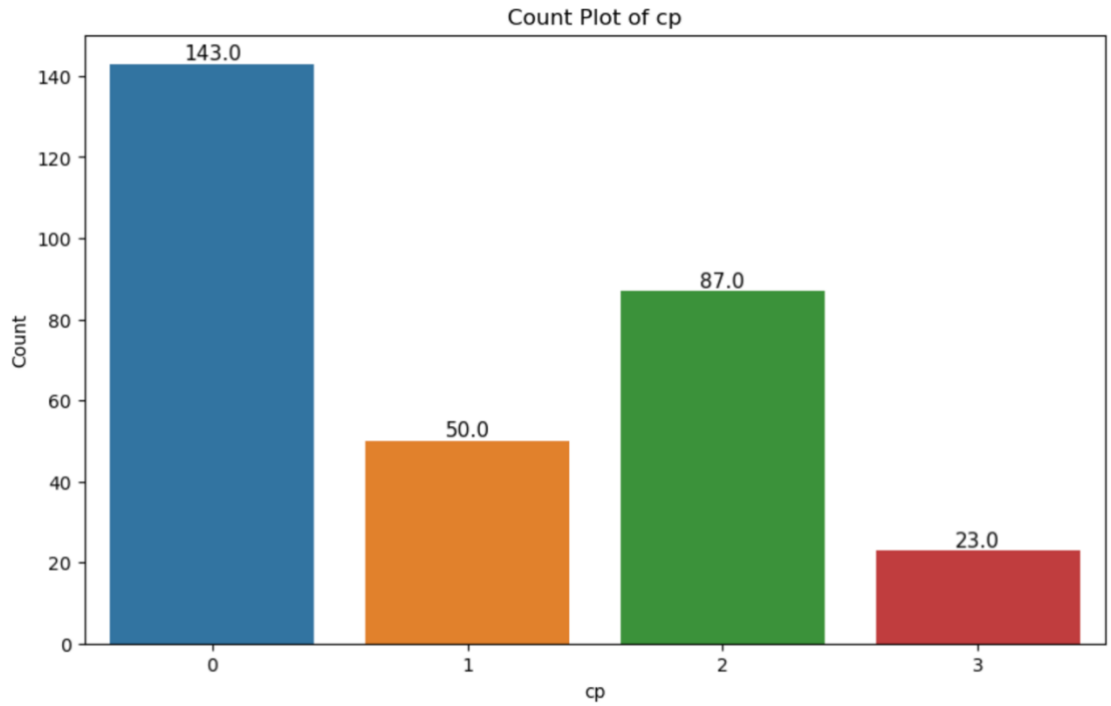
```
# Identify the data variables which might be categorical in nature.
# Describe and explore these variables using appropriate tools e.g. count plot
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

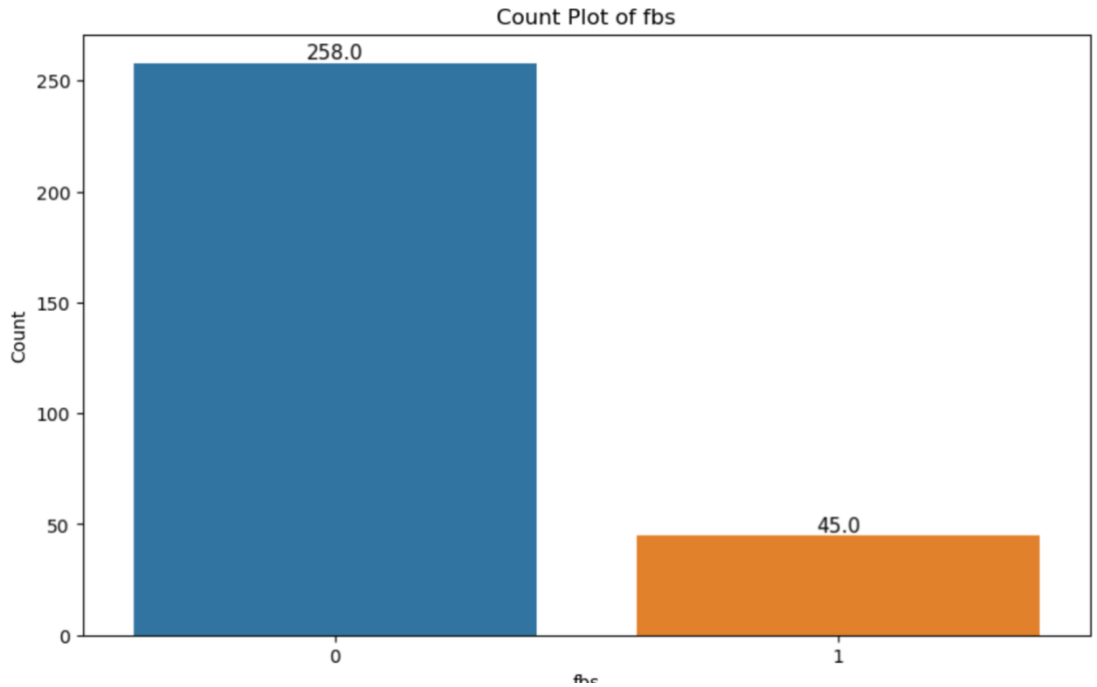
```
# Define the function to create count plot for gender(sex)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('sex')
```



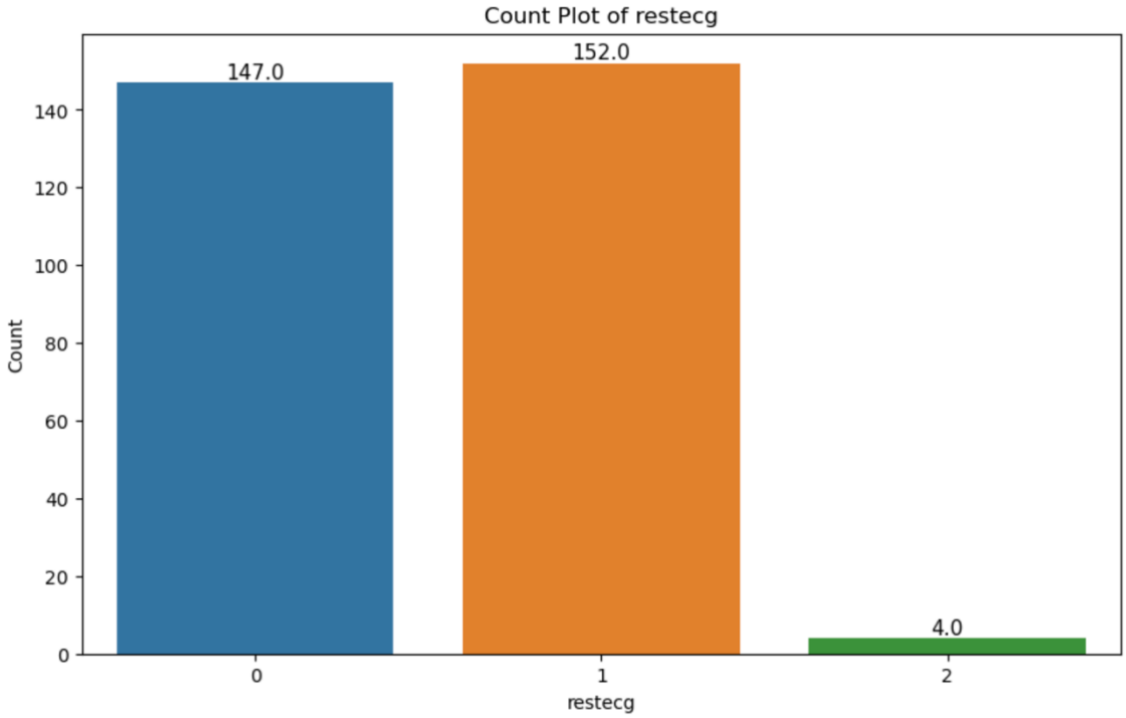
```
# Define the function to create count plot for chest pain type(cp)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('cp')
```



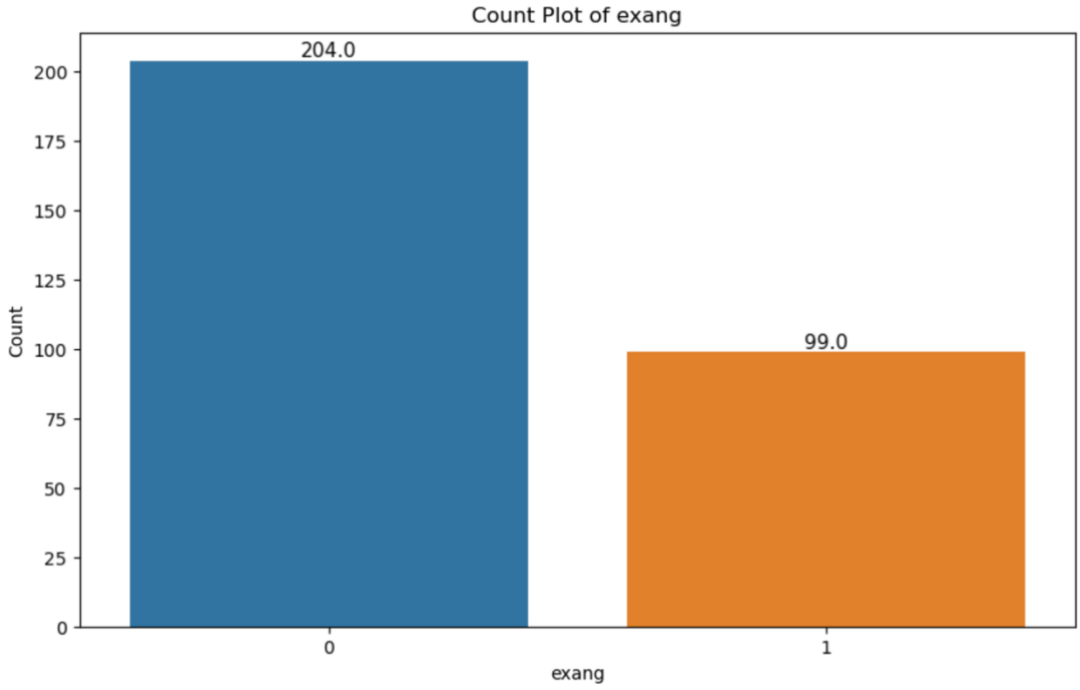
```
# Define the function to create count plot for fasting blood sugar(fbs)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('fbs')
```



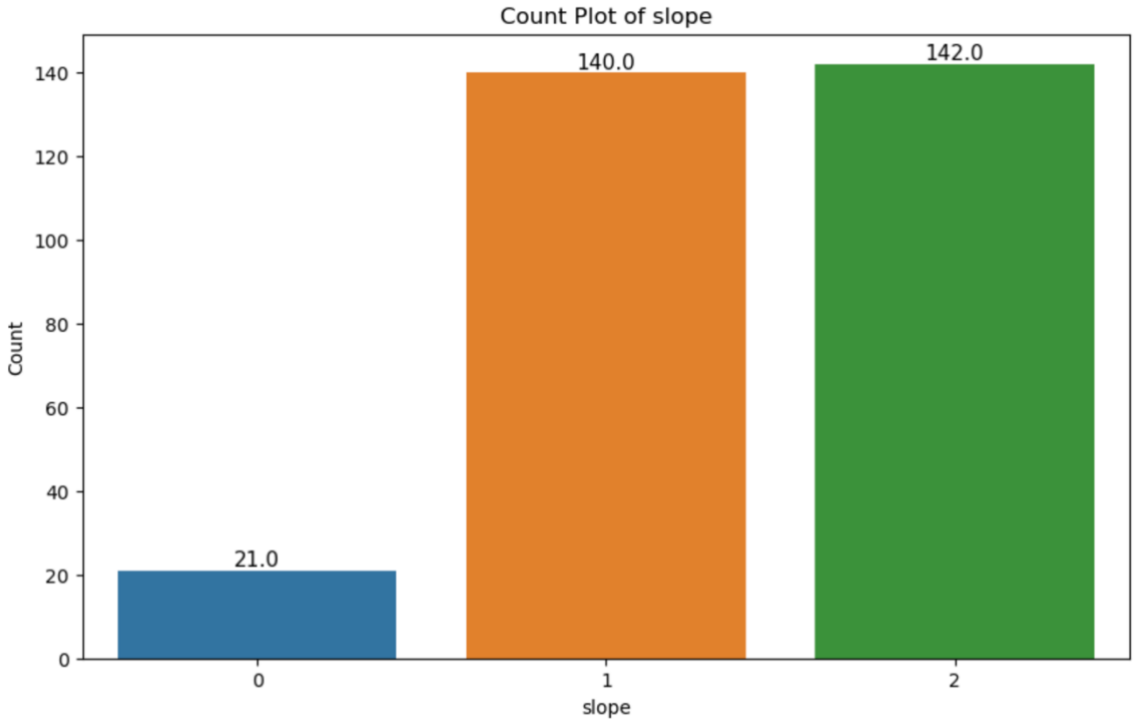
```
# Define the function to create count plot for resting electrocardiographic results(restecg)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('restecg')
```



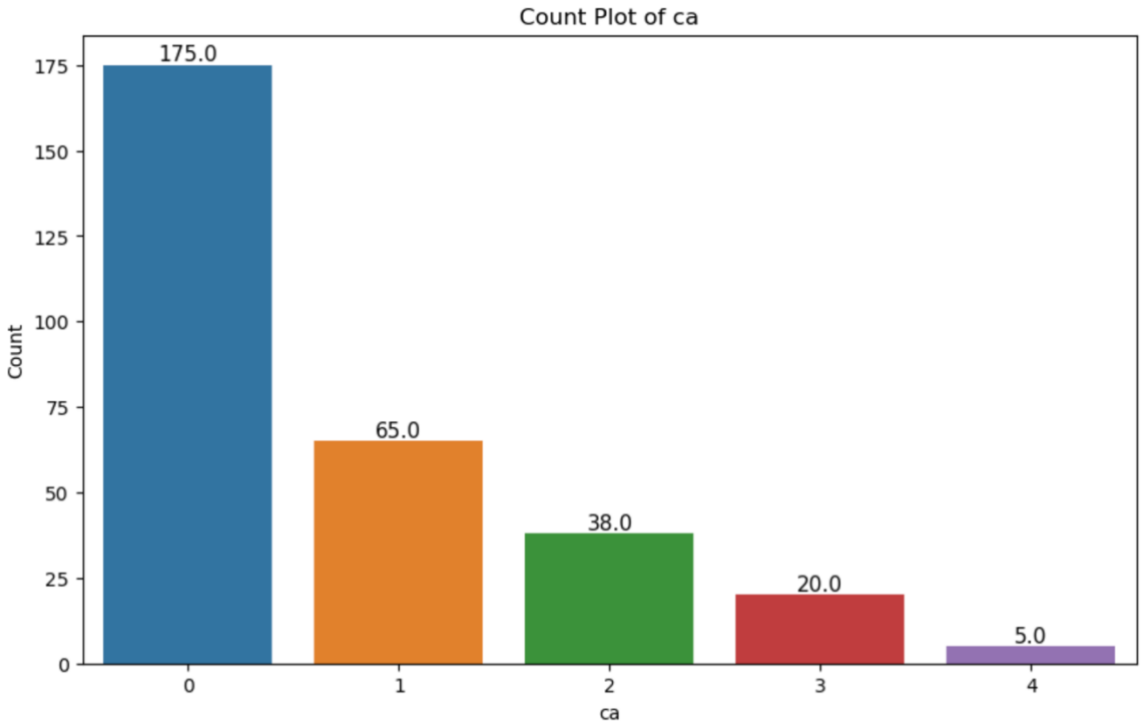
```
# Define the function to create count plot for exercise induced angina(exang)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('exang')
```



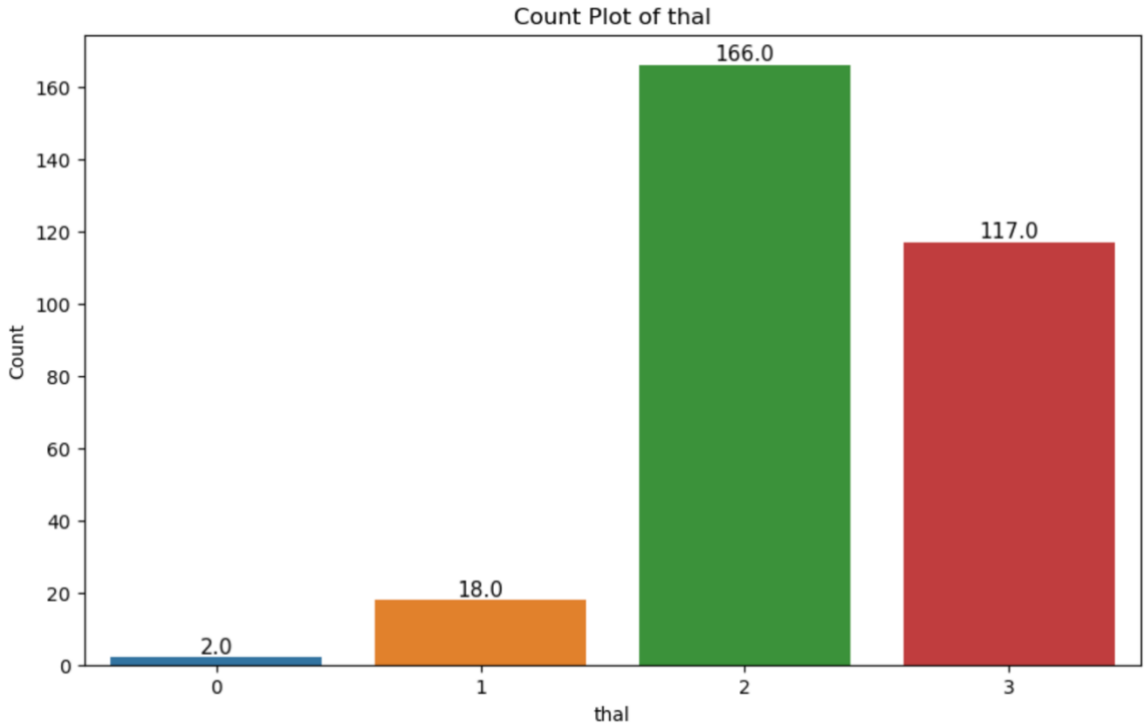
```
# Define the function to create count plot for the slope of the peak exercise ST
segment(slope)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('slope')
```



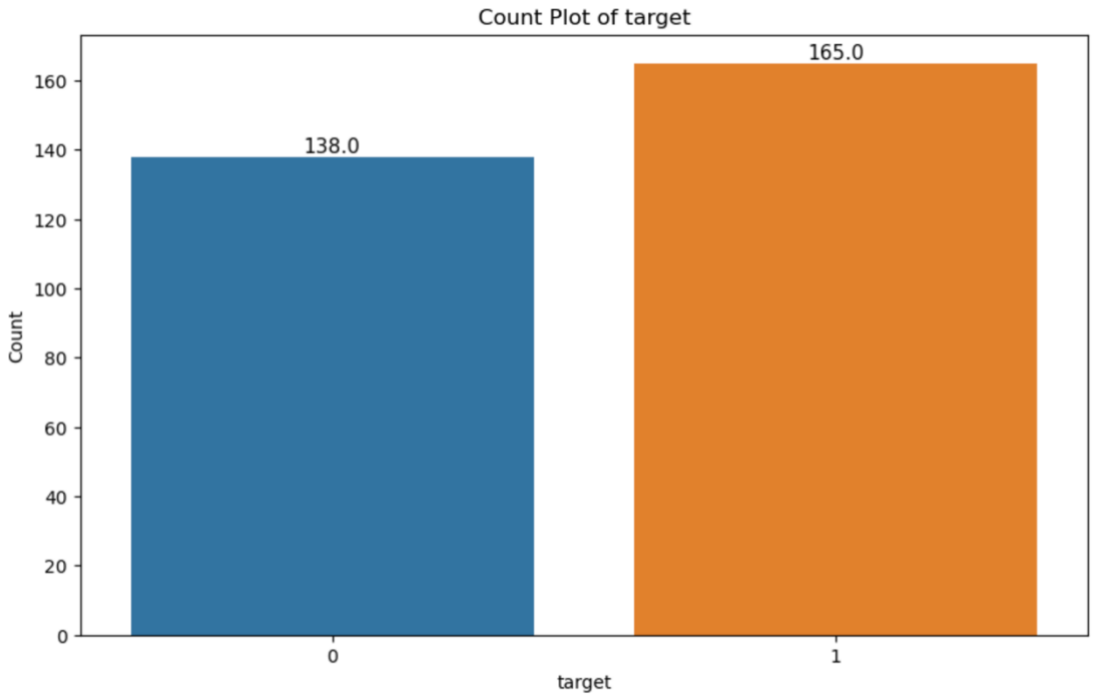
```
# Define the function to create count plot for number of major vessels (0-3) colored by flourosopy(ca)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('ca')
```



```
# Define the function to create count plot for thalassemia(thal)
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('thal')
```

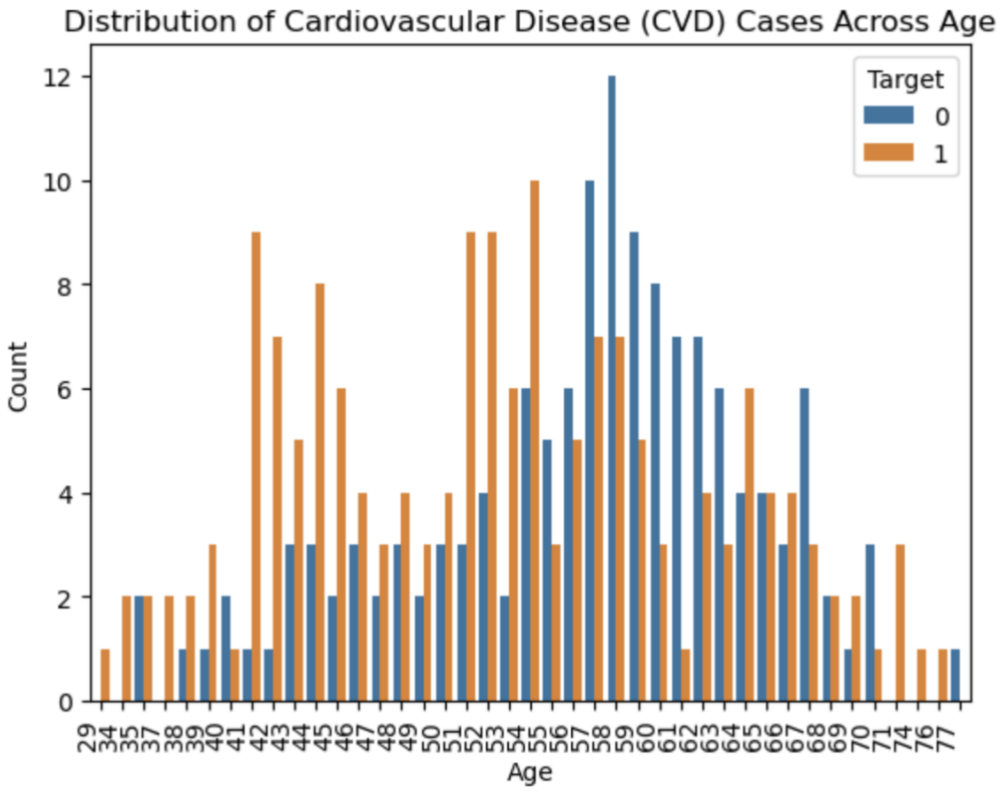


```
# Define the function to create count plot for target
def plot_cat(column):
    plt.figure(figsize=(10, 6))
    ax= sns.countplot(data=df, x=column)
    plt.title(f'Count Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=0)

    # Add count labels on top of each bar
    for p in ax.patches:
        ax.text(p.get_x() + p.get_width()/2., p.get_height(),
                f'{p.get_height()}',
                ha='center', va='bottom', fontsize=11, color='black')

    plt.show()

# Call the function with the categorical variable as argument
plot_cat('target')
```

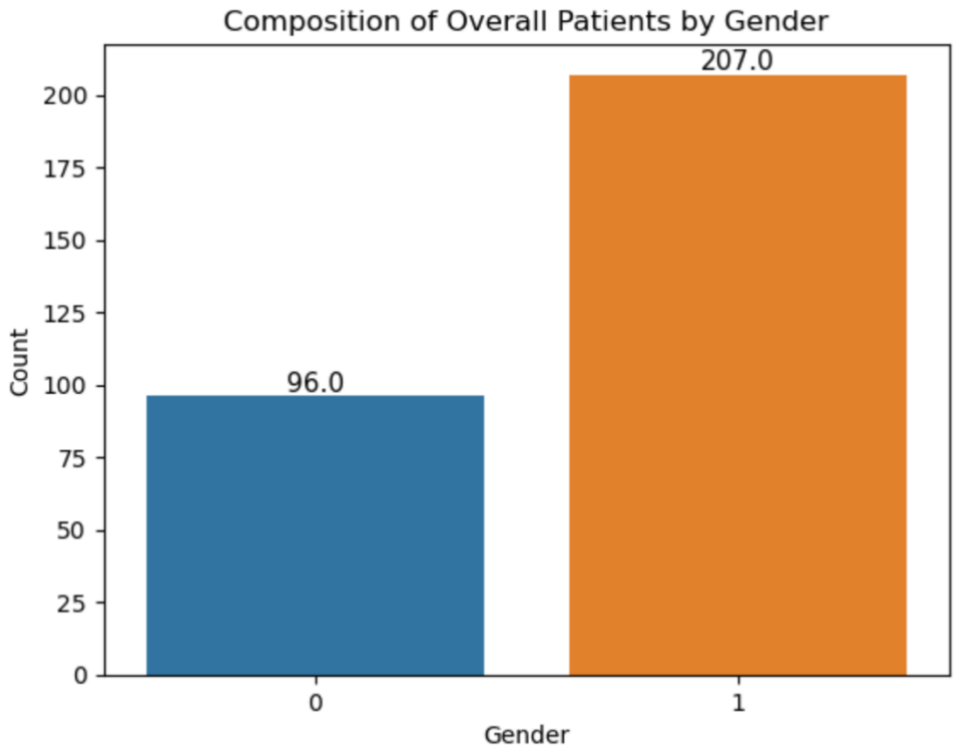


Study the occurrence of CVD across Age.

```
sns.countplot(data=df, x='age', hue='target')

# Customize the plot
plt.title('Distribution of Cardiovascular Disease (CVD) Cases Across Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend(title='Target')
plt.xticks(rotation=90, ha='right')

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



```
# Study the composition of overall patients w.r.t. Gender.

sns.countplot(data=df, x='sex')

# Customize the plot
plt.title('Composition of Overall Patients by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')

# Add count labels to the bars
for p in plt.gca().patches:
    plt.gca().annotate(f'{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')

# Show the plot
plt.show()

# where '0' in gender represents female and '1' as male.
```



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

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

X = df[['trestbps']]  # Feature matrix
y = df['target']  # Target variable

# Split data 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)

# Choose a classification model (Logistic Regression as an example)
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
print(classification_ report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.62	0.34	0.44	29
1	0.58	0.81	0.68	32
accuracy			0.59	61
macro avg	0.60	0.58	0.56	61
weighted avg	0.60	0.59	0.57	61

From the classification report, we can draw several conclusions about the performance of the model:

Accuracy: The overall accuracy of the model is 0.59, which means that it correctly predicts the target variable (presence or absence of a heart attack) for approximately 59% of the samples in the test set.

Precision: For class 0 (absence of a heart attack), the precision is 0.62. This means that out of all the instances predicted as class 0, 62% were actually class 0. For class 1 (presence of a heart attack), the precision is 0.58. This means that out of all the instances predicted as

class 1, 58% were actually class 1. Precision is a measure of the correctness of positive predictions.

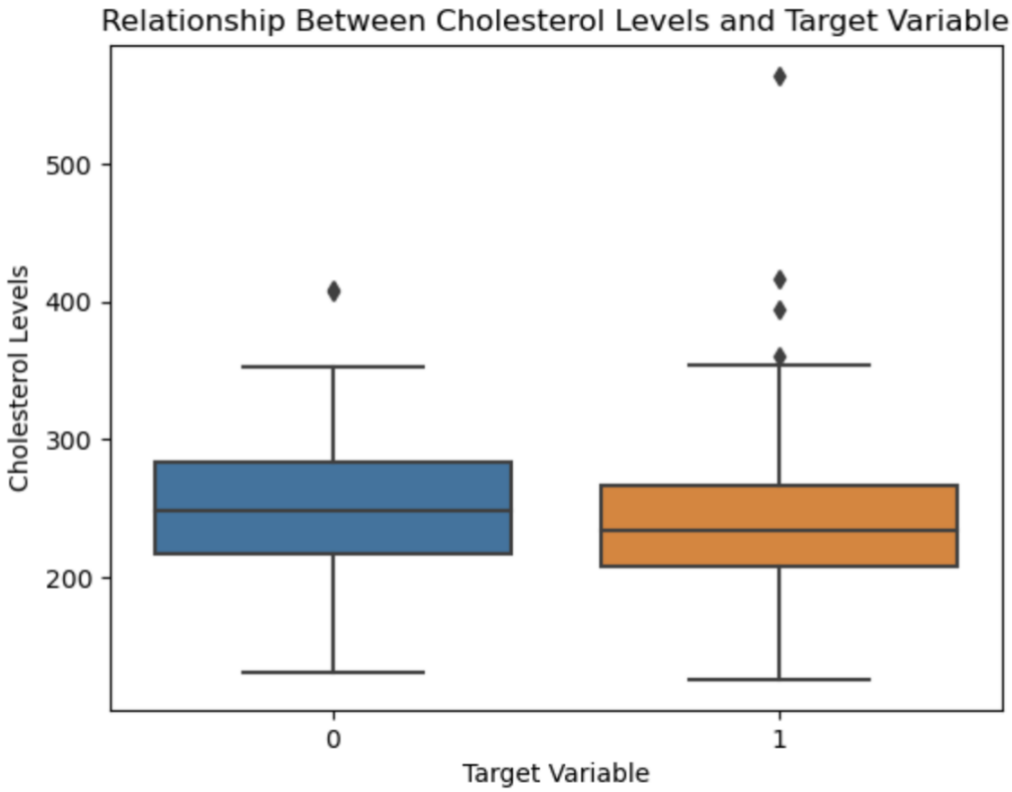
Recall: For class 0, the recall is 0.34. This means that out of all the instances that are actually class 0, only 34% were correctly predicted as class 0. For class 1, the recall is 0.81. This means that out of all the instances that are actually class 1, 81% were correctly predicted as class 1. Recall is a measure of the completeness of positive predictions.

F1-score: The F1-score is the harmonic mean of precision and recall. It balances both precision and recall. For class 0, the F1-score is 0.44. For class 1, the F1-score is 0.68.

Support: Support indicates the number of actual occurrences of each class in the test set.

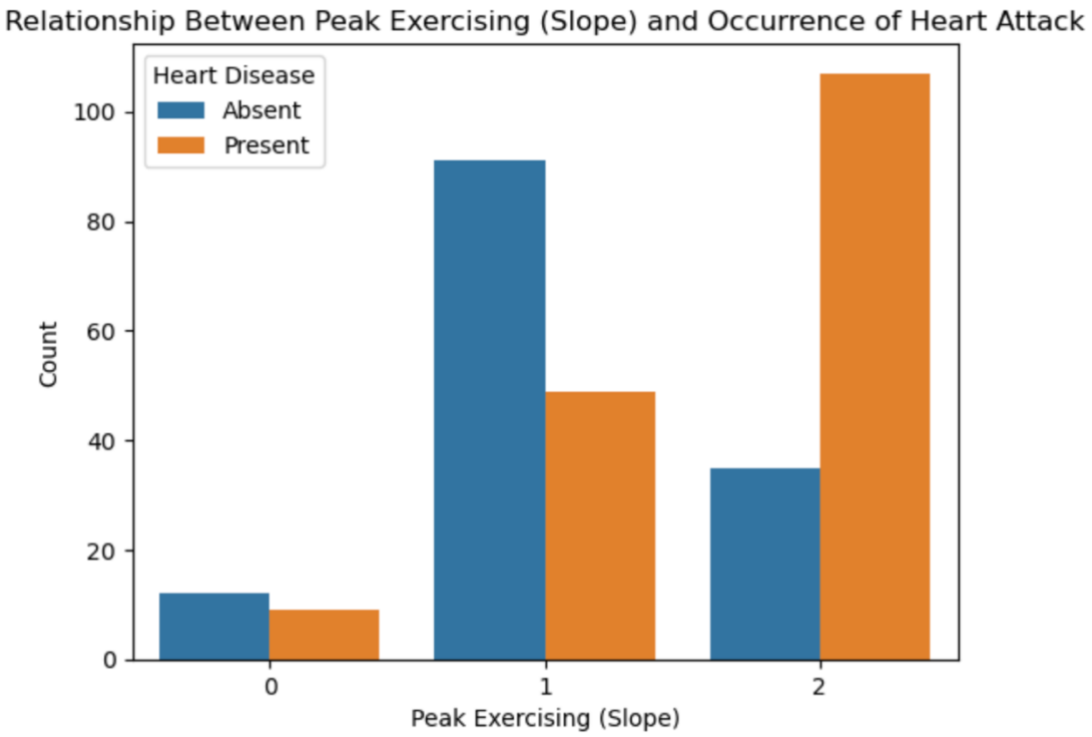
In summary, the model performs relatively better in predicting the presence of a heart attack (class 1) compared to the absence of a heart attack (class 0), as indicated by higher precision, recall, and F1-score for class 1.

```
# Describe the relationship between Cholesterol levels and our target variable.

sns.boxplot(data=df, x='target', y='chol')

# Customize the plot
plt.title('Relationship Between Cholesterol Levels and Target Variable')
plt.xlabel('Target Variable')
plt.ylabel('Cholesterol Levels')

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

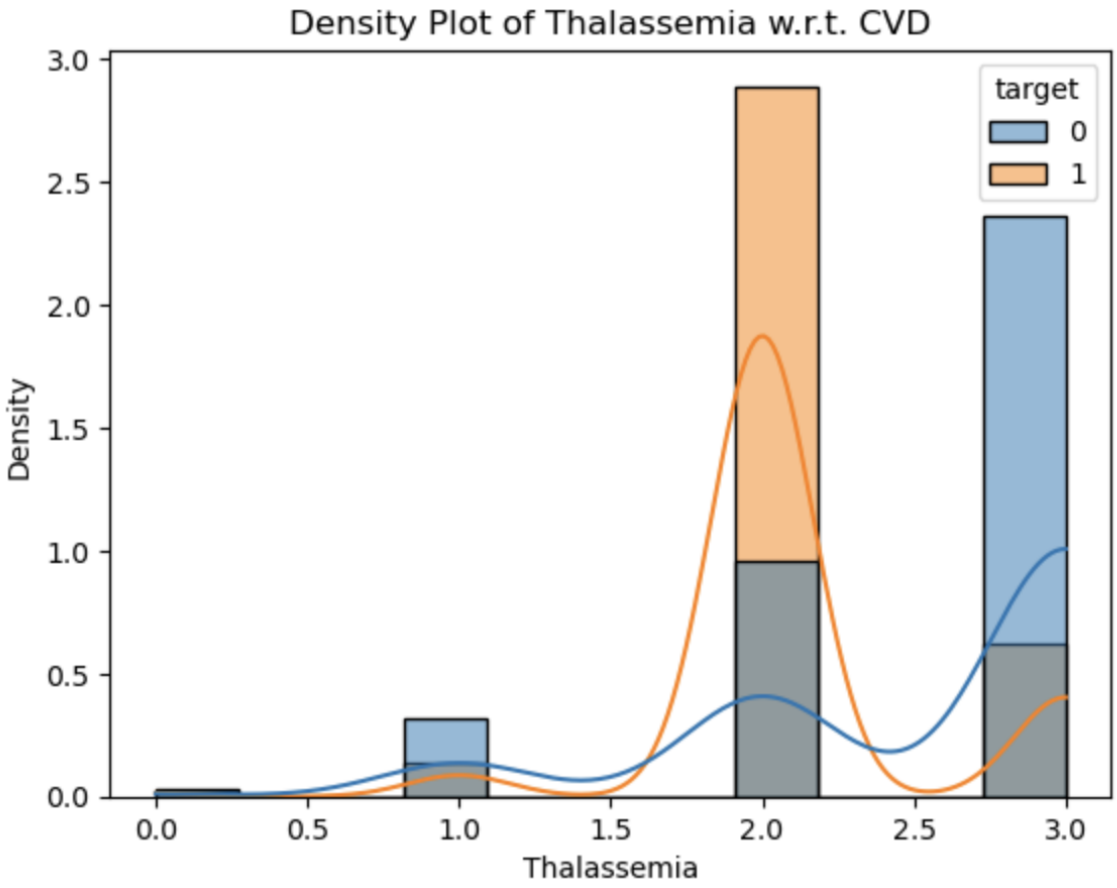


What can be concluded about the relationship between peak exercising and occurrence of heart attack.

```
sns.countplot(data=df, x='slope', hue='target')

# Customize the plot
plt.title('Relationship Between Peak Exercising (Slope) and Occurrence of Heart Attack')
plt.xlabel('Peak Exercising (Slope)')
plt.ylabel('Count')
plt.legend(title='Heart Disease', labels=['Absent', 'Present'])

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




```
# Is thalassemia a major cause of CVD?

sns.histplot(data=df, x='thal', hue='target', kde=True, stat='density', common_norm=False)

# Customize the plot
plt.title('Density Plot of Thalassemia w.r.t. CVD')
plt.xlabel('Thalassemia')
plt.ylabel('Density')

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

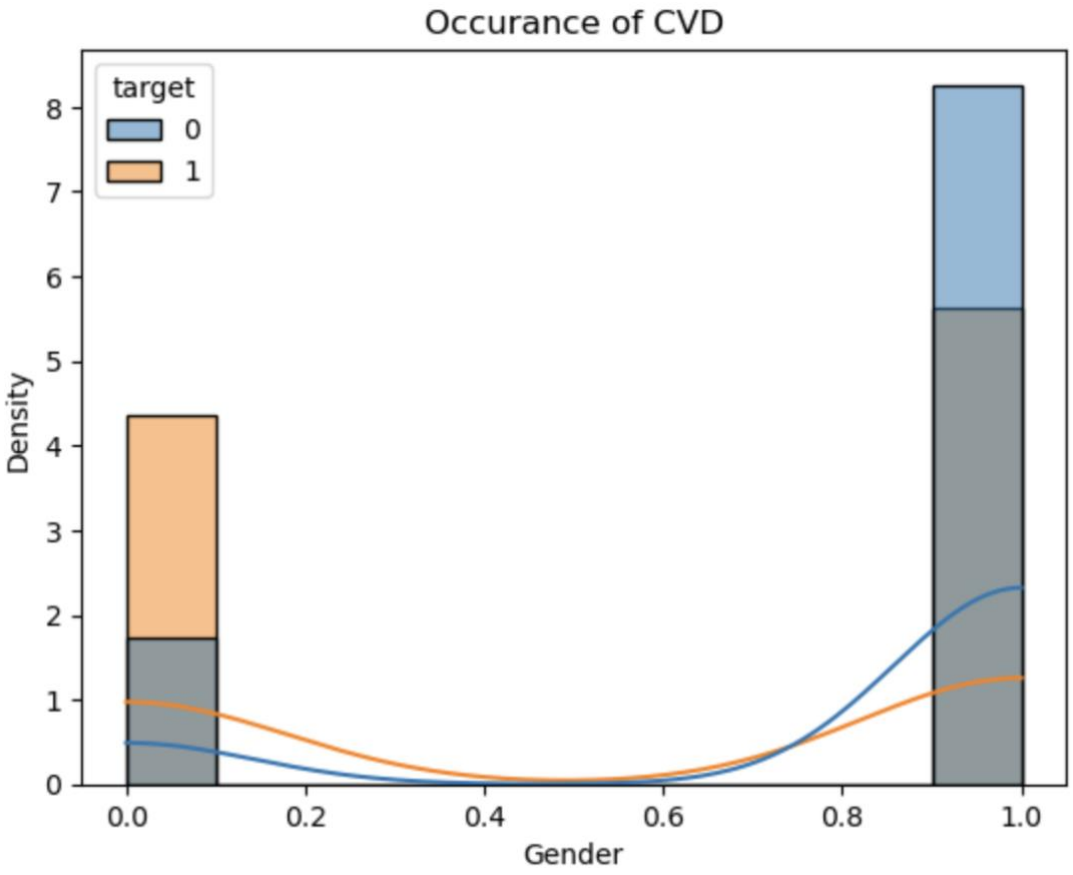


How are the other factors determining the occurrence of CVD?

1- Occurrence of CVD based on gender:
sns.histplot(data=df, x='sex', hue='target', kde=True, stat='density', common_norm=False)

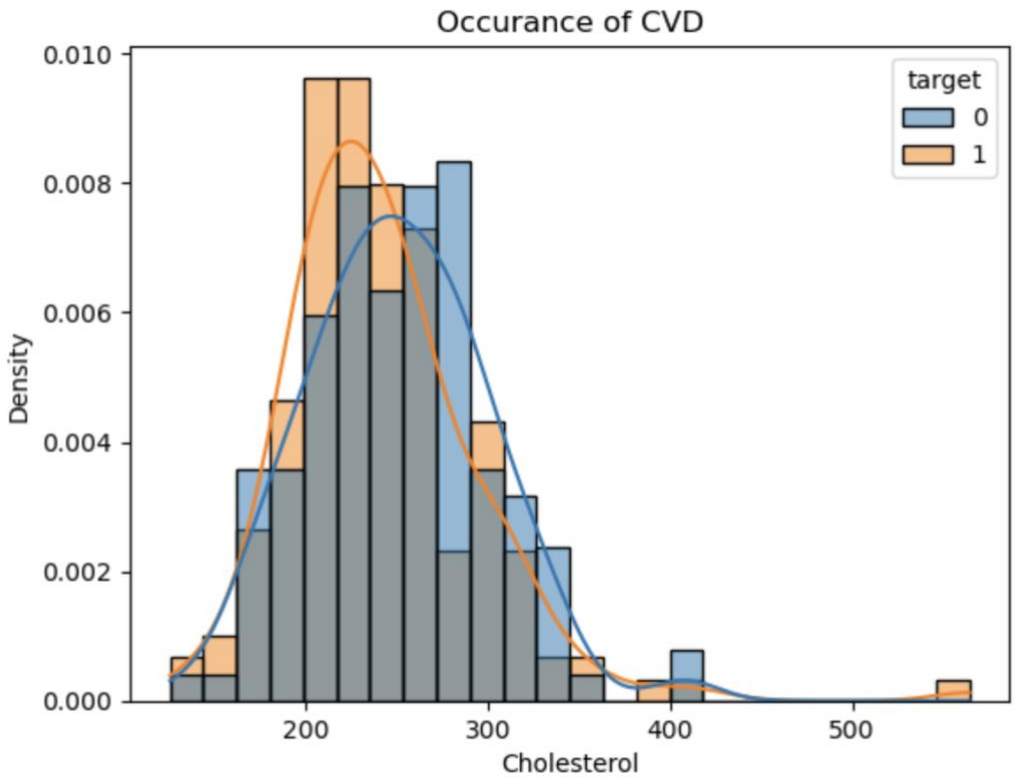
# Customize the plot
plt.title('Occurance of CVD')
plt.xlabel('Gender')
plt.ylabel('Density')

# Show the plot
plt.show()



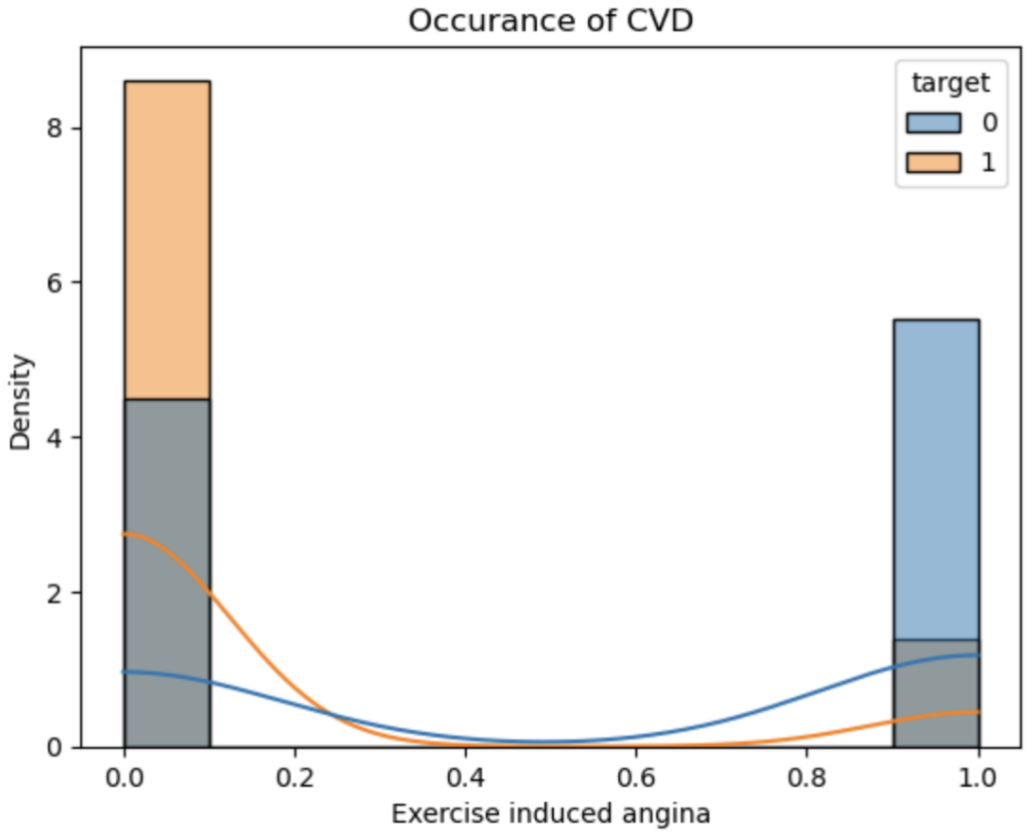
2- Occurrence of CVD based on cholesterol levels:

```
sns.histplot(data=df, x='chol', hue='target', kde=True, stat='density', common_norm=False)

# Customize the plot
plt.title('Occurance of CVD')
plt.xlabel('Cholesterol')
plt.ylabel('Density')

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



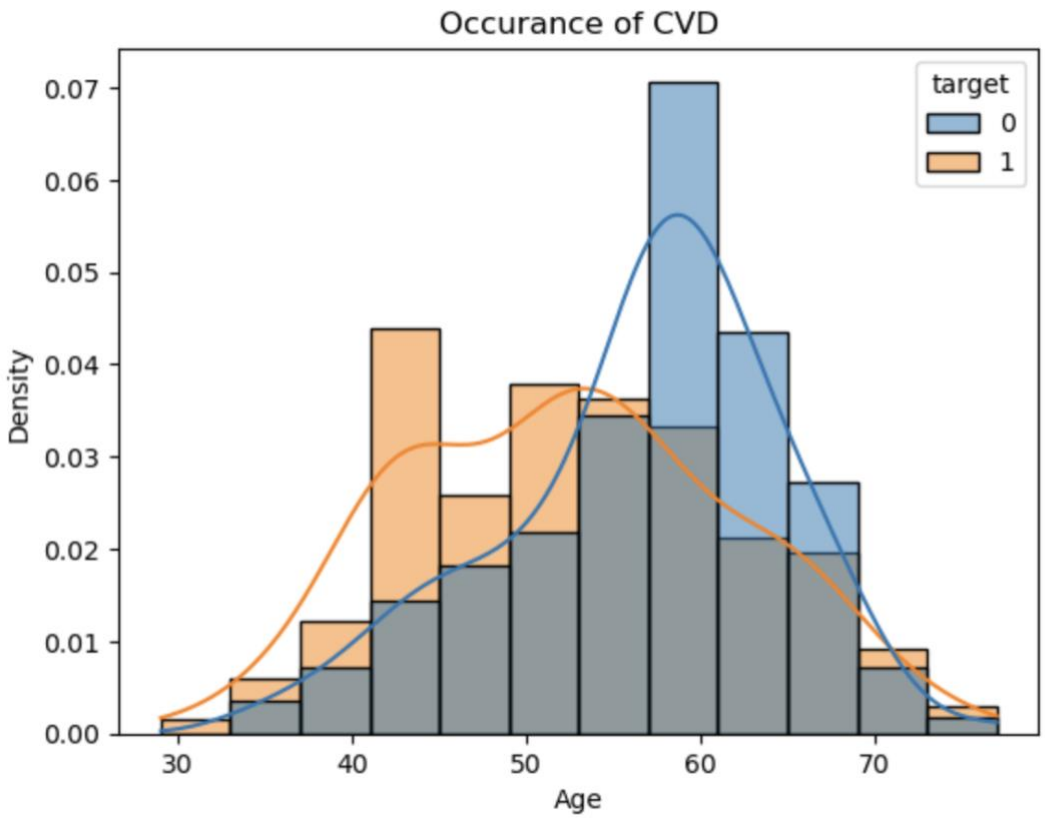
3- Occurrence of CVD based on exercise induced angina:

```
sns.histplot(data=df, x='exang', hue='target', kde=True, stat='density', common_norm=False)

# Customize the plot
plt.title('Occurance of CVD')
plt.xlabel('Exercise induced angina')
plt.ylabel('Density')

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



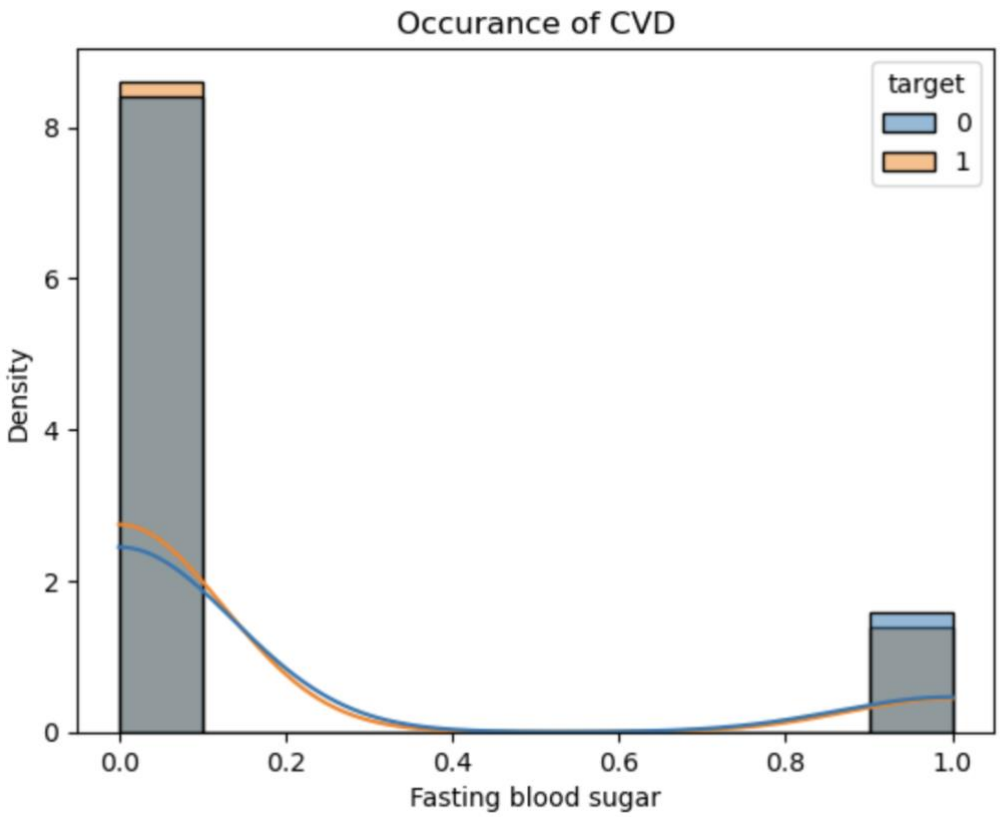
4- Occurrence of CVD based on age:

```
sns.histplot(data=df, x='age', hue='target', kde=True, stat='density', common_norm=False)

# Customize the plot
plt.title('Occurance of CVD')
plt.xlabel('Age')
plt.ylabel('Density')

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



5- Occurrence of CVD based on fasting blood sugar:

```
sns.histplot(data=df, x='fbs', hue='target', kde=True, stat='density', common_norm=False)

# Customize the plot
plt.title('Occurance of CVD')
plt.xlabel('Fasting blood sugar')
plt.ylabel('Density')

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



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

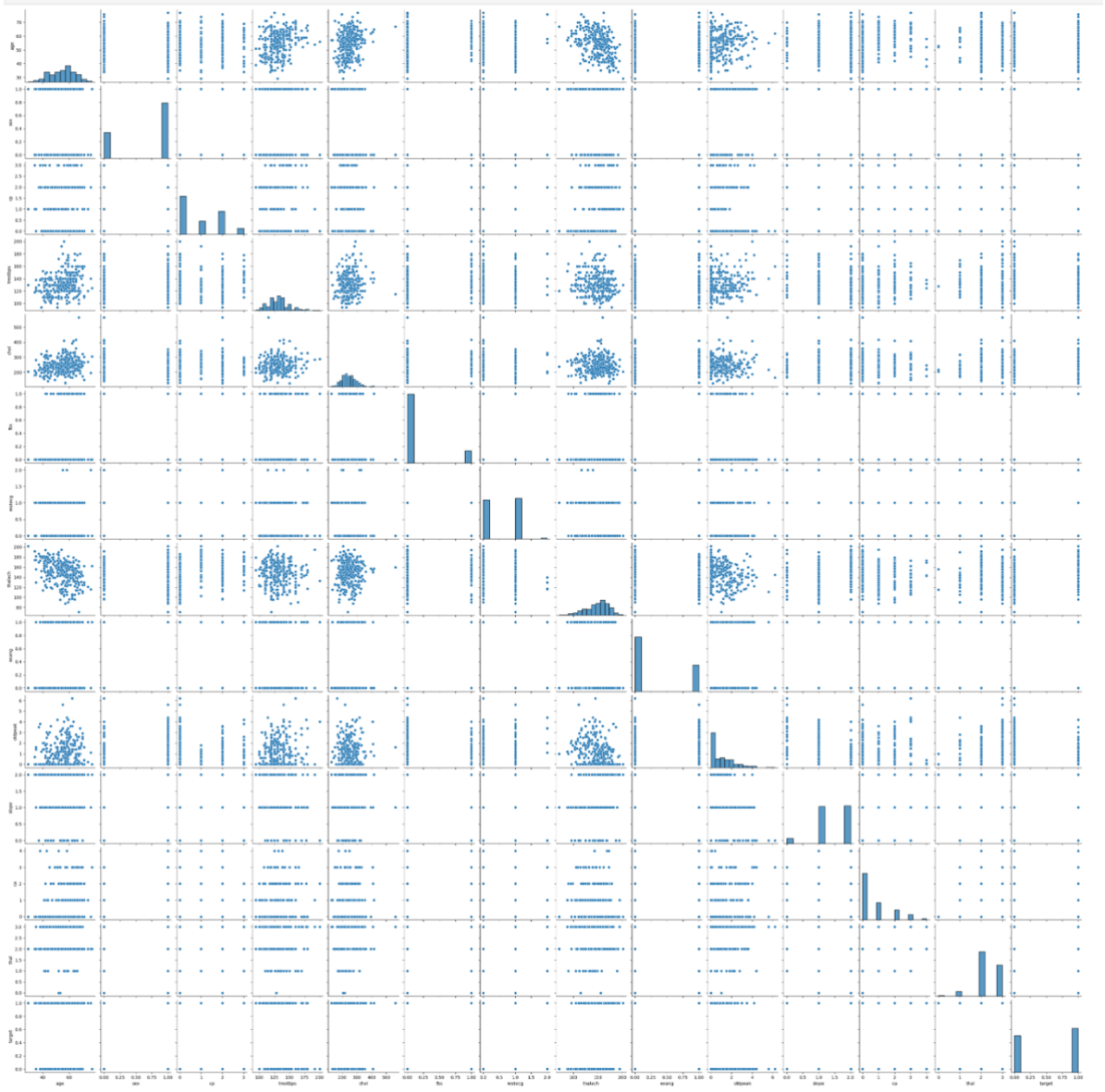
numeric_data = df.select_dtypes(include=['number'])

# Create a pair plot
sns.pairplot(numeric_data)

# Show the plot
plt.show()

import warnings

warnings.filterwarnings("ignore")



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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the logistic regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nROC-AUC Score:", roc_auc_score(y_test, y_pred))
```

```
Confusion Matrix:
[[10 19]
 [ 6 26]]

Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.34      0.44        29
           1       0.58      0.81      0.68        32

    accuracy                           0.59        61
   macro avg       0.60      0.58      0.56        61
weighted avg       0.60      0.59      0.57        61


Accuracy Score: 0.5901639344262295

ROC-AUC Score: 0.5786637931034483
```

From the evaluation metrics of the baseline logistic regression model, we can draw several conclusions:

Confusion Matrix: The confusion matrix shows that out of 29 instances of class 0 (no cardiovascular disease), the model correctly predicted 10 instances and incorrectly predicted 19 instances as class 1 (presence of cardiovascular disease). Out of 32 instances of class 1, the model correctly predicted 26 instances and incorrectly predicted 6 instances as class 0.

Classification Report: The classification report provides a detailed overview of the model's performance for each class (0 and 1) based on metrics such as precision, recall, and F1-score. Precision measures the proportion of true positive predictions among all positive predictions. Recall measures the proportion of true positive predictions among all actual positive instances. F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. The report indicates that the model performs better in predicting class 1 (presence of cardiovascular disease) compared to class 0 (no cardiovascular disease), as reflected in higher precision, recall, and F1-score for class 1.

Accuracy Score: The accuracy score of the model is 0.59, indicating that it correctly predicts the target variable (presence or absence of cardiovascular disease) for approximately 59% of the samples in the test set. However, accuracy alone may not provide a complete picture of the model's performance, especially if the classes are imbalanced.

ROC-AUC Score: The ROC-AUC score is 0.579, which measures the area under the receiver operating characteristic (ROC) curve. The ROC curve is a graphical representation of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different threshold values. A higher ROC-AUC score indicates better discrimination capability of the model.