ML_Project_HealthCare_CVD

March 25, 2023

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
```

Loading the data to panda data frame

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

0.0.1 PDA

[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
```

Lets check the shape of data to check number of rows and columns in the data

[4]:
```
df.shape
```

[4]: (303, 14)

From above command we understand that there are 303 rows and 14 columns in the dataset.

In the next step lets take basic understand on columns to understand the type of value present in the table & check if there is any null values

[5]:
```
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
```

0.0.2 Data Cleaning

[6]:
```
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
```

0.0.3 Its clear from above output that there is no null value in any column.

0.0.4 Lets Check for duplicate rows

[7]:
```python
print("Duplicate rows:", df.duplicated().sum())
```

```
Duplicate rows: 1
```

[8]:
```python
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
```

Since we noticed there is 1 duplicate value so first we need to remove the duplicate before proceding with the Statistical Summary

[9]:
```python
# Drop the duplicate rows and print the number of row after removing duplicate
df.drop_duplicates(inplace=True)
df.reset_index(drop=True, inplace=True)
print('No. of rows after removing duplicate: ',df.shape)
```

```
No. of rows after removing duplicate:  (302, 14)
```

0.0.5 Note: It would be good to perfirm statistical summary after removing duplicates as this can give more accurate results in the summary.

0.1 Preliminary Statistical Summary:

Measures of central tendencies. Summarizes the count, mean, standard deviation, min, and max for numeric variables

[10]:
```python
df.describe()
```

[10]:
```
             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
```

[11]:
```
type(df)
```

[11]:
```
pandas.core.frame.DataFrame
```

[12]:
```
df.columns
```

[12]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

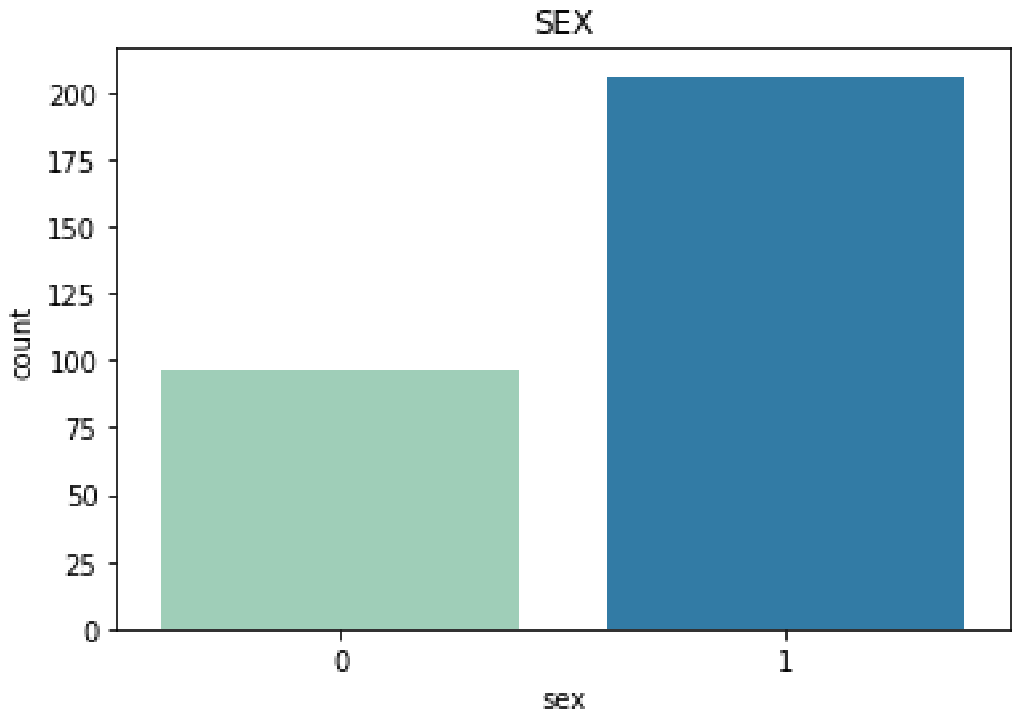
Identify the data variables which are categorical & describe & explore these variables using the appropriate tools, such as count plot identify the categorical variables

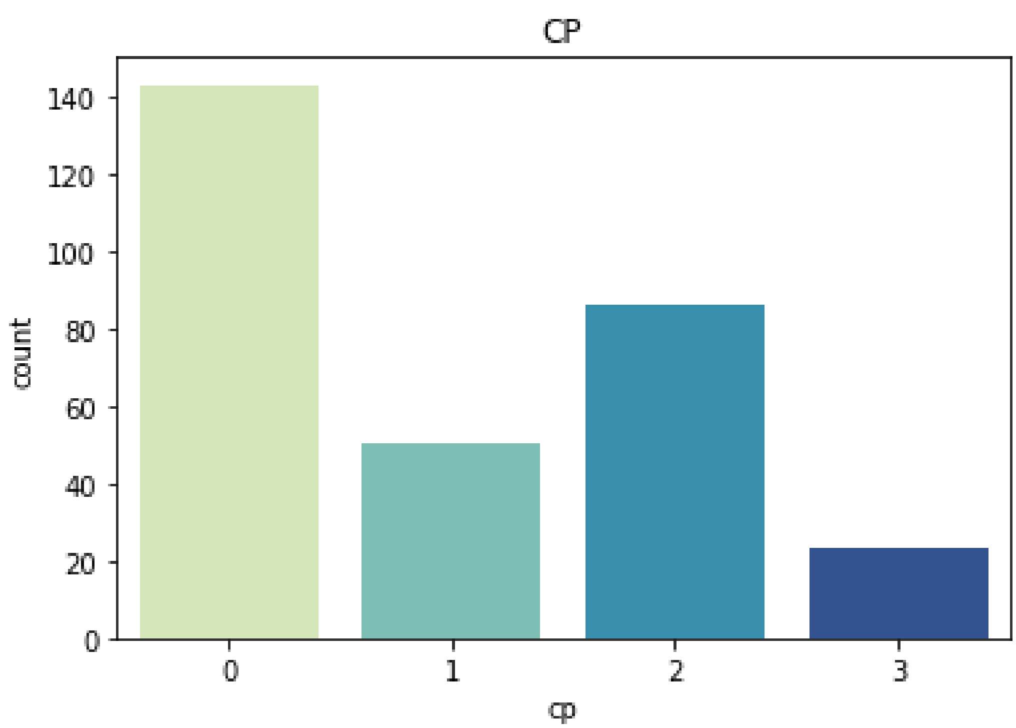
[13]:
```
cat_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal',␣
 ↪'target']
```

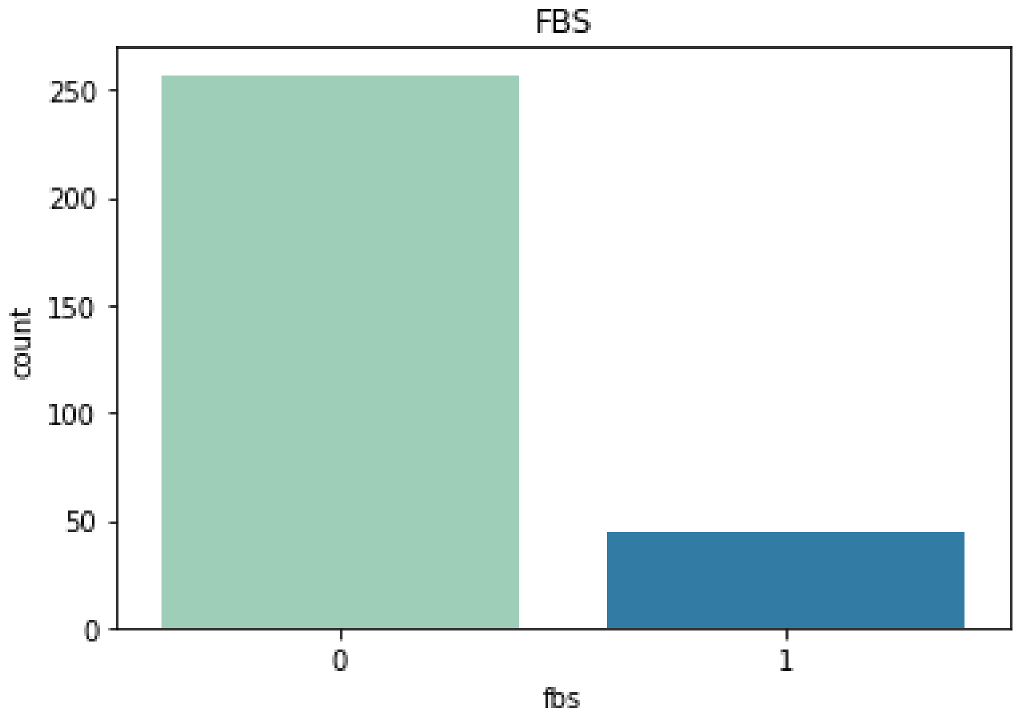
Create count plots for each categorical variable

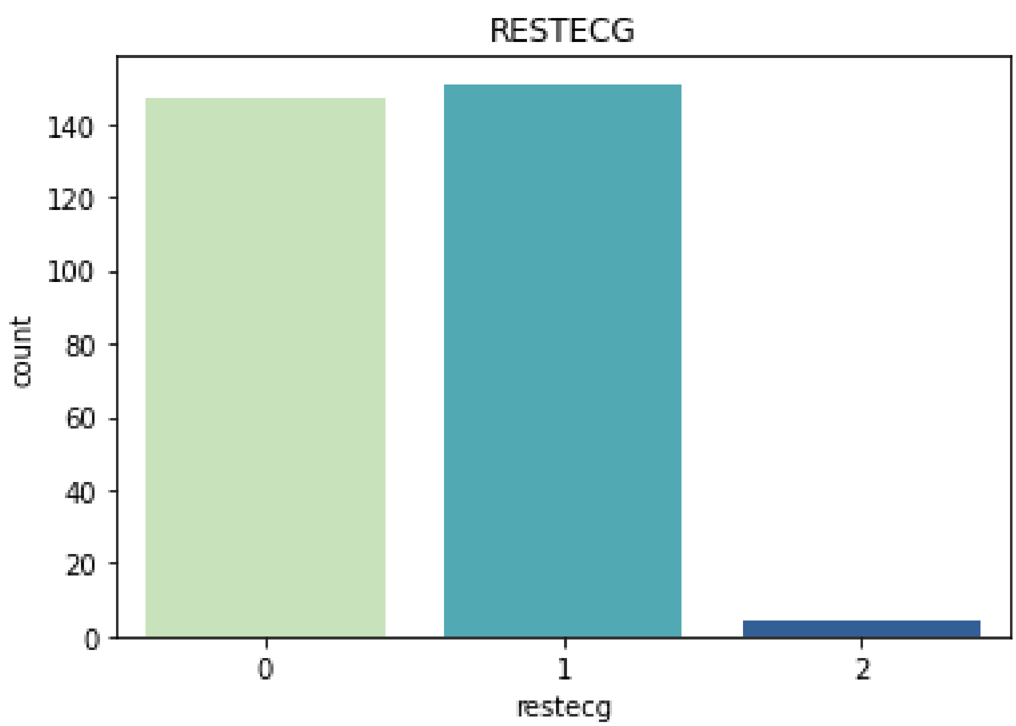
[14]:
```
for col in cat_cols:
    sns.countplot(x=col, data=df, palette='YlGnBu')
    plt.title(col.upper())
    plt.savefig(f"{col}_countplot.jpg")
```

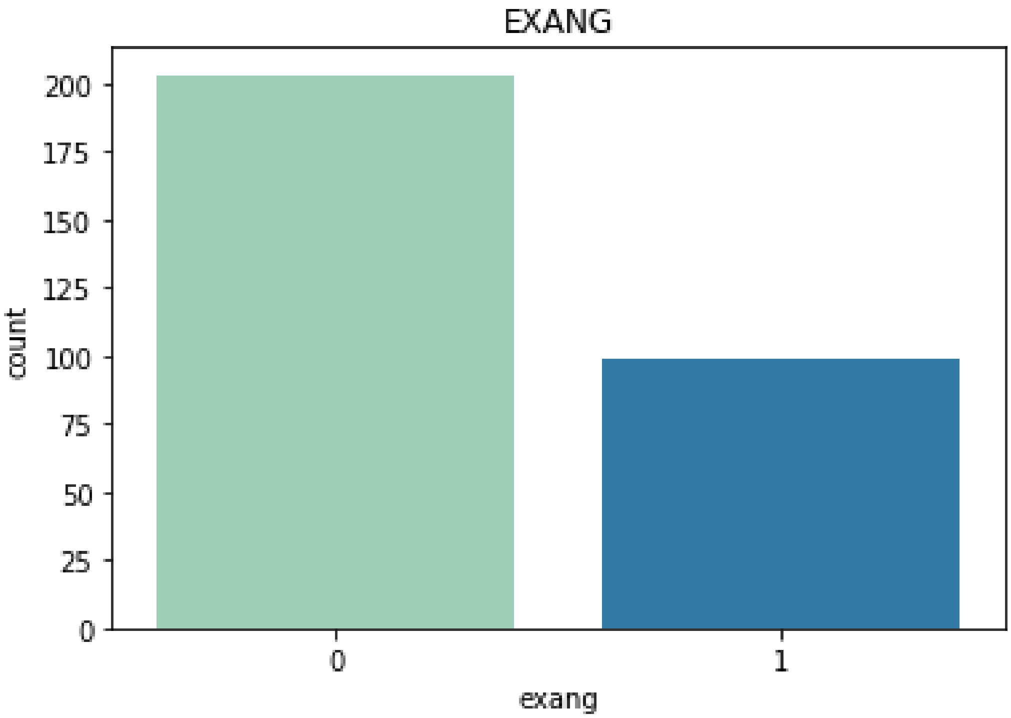
```
plt.show()
```

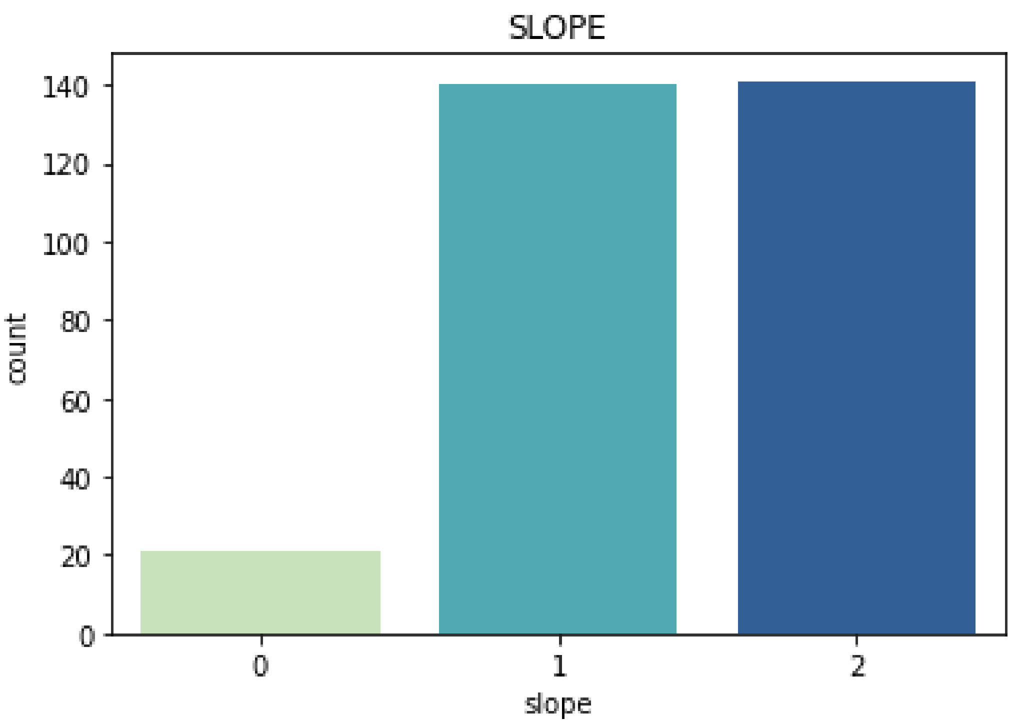

SEX
count
200
175
150
125
100
75
50
25
0
0
1
sex

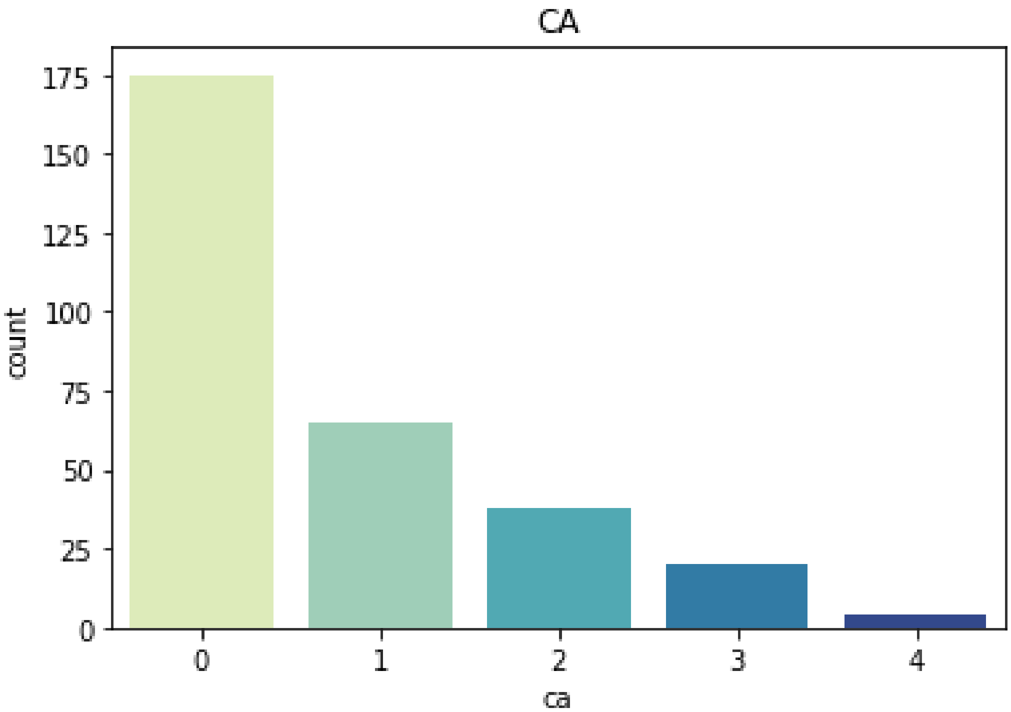

CP
count
140
120
100
80
60
40
20
0
0
1
2
3
cp

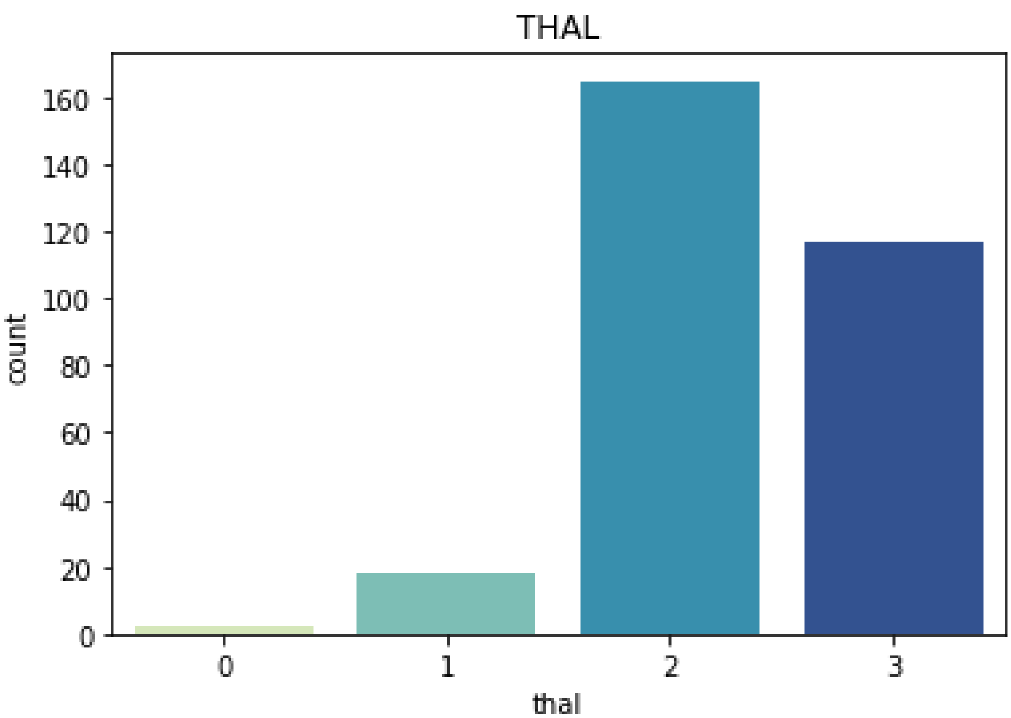

FBS
count
250
200
150
100
50
0
0
1
fbs

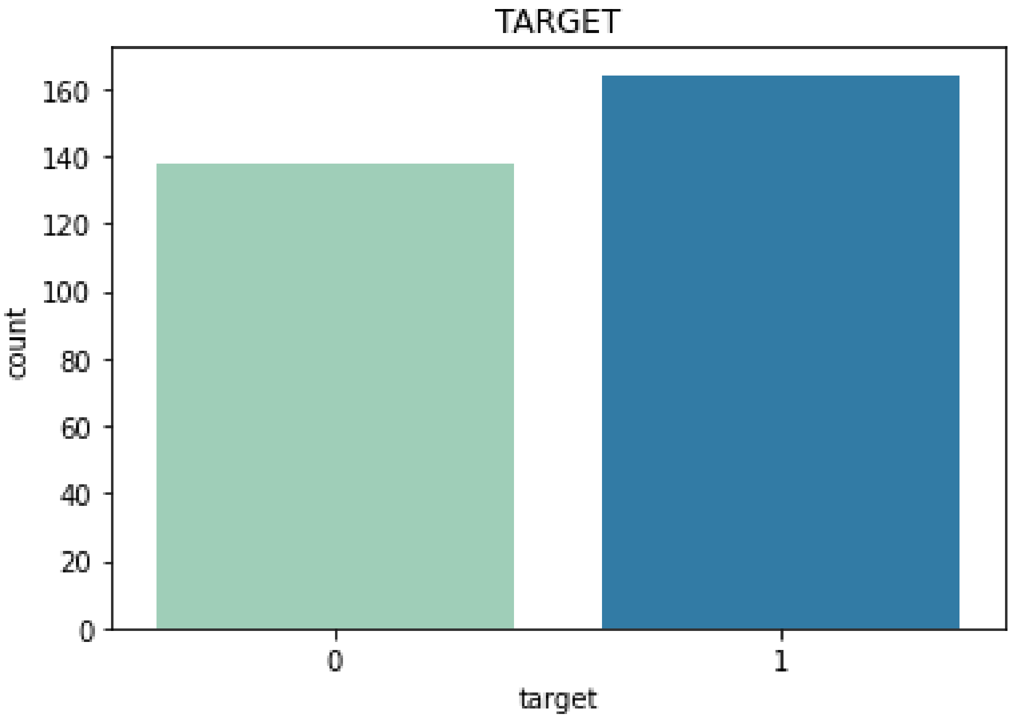

RESTECG
count
140
120
100
80
60
40
20
0
0
1
2
restecg


EXANG
200
175
150
125
100
75
50
25
0
count
0
1
exang


SLOPE
140
120
100
80
60
40
20
0
count
0
1
2
slope


CA
count
175
150
125
100
75
50
25
0
0
1
2
3
4
ca


THAL
count
160
140
120
100
80
60
40
20
0
0
1
2
3
thal

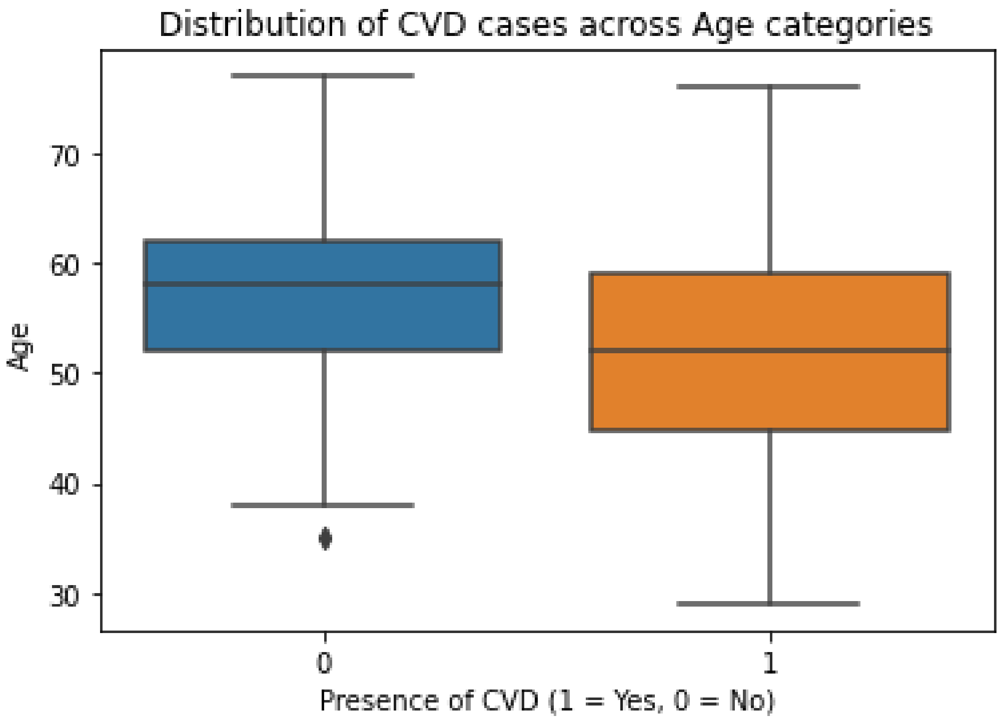


Study the occurrence of CVD across the Age category Calculate the prevalence of CVD for each age category

[15]:
```
age_cvd_counts = df.groupby('age')['target'].sum()
age_counts = df['age'].value_counts()
age_cvd_rates = age_cvd_counts / age_counts
```

Creating a box plot to visualize above calculation

[16]:
```
sns.boxplot(x='target', y='age', data=df)
plt.title("Distribution of CVD cases across Age categories")
plt.xlabel("Presence of CVD (1 = Yes, 0 = No)")
plt.ylabel("Age")
plt.savefig("Distribution of CVD cases across Age categories.jpg")
plt.show()
```



Abover code will generate a box plot that shows the distribution of CVD cases across different age categories. We can use this plot to visually analyze the prevalence of CVD amaong different age groups & identify any potential patterns or trends.

Study the composition of all patients with respect to the Sex category Below code will demonstrates how to calculate the proportion of male & female patients & create a pie chart to visualize the composition of all patients with respect to the sex category.

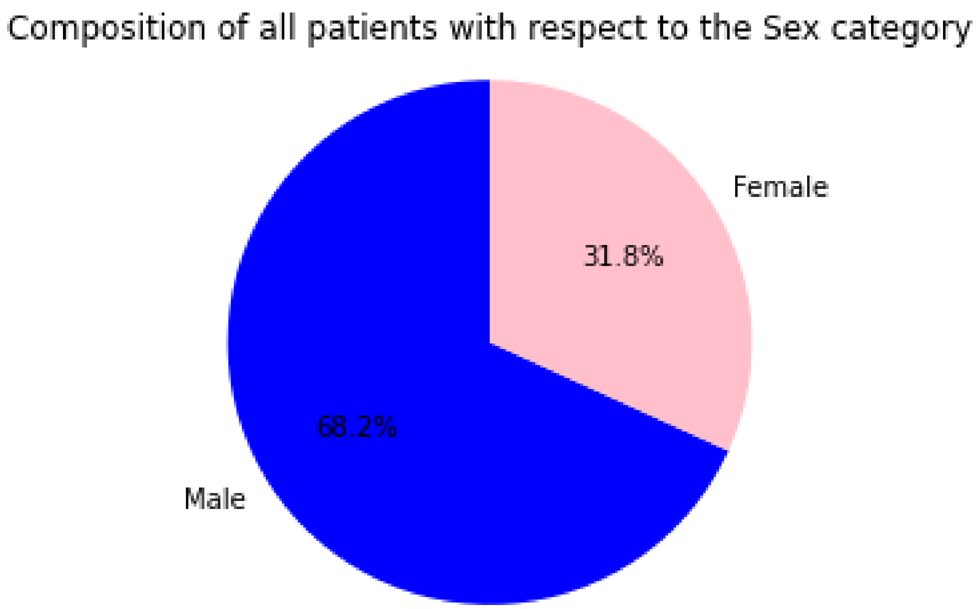
Calculate the proportion of male & female patients in the dataset

[17]:
```
sex_counts = df['sex'].value_counts()
total_patients = sex_counts.sum()
proportion_male = sex_counts[1] / total_patients
proportion_female = sex_counts[0] / total_patients
```

create a pie chart to visualize the composition of all patients with respect to the sex

[18]:
```
labels = ['Male', 'Female']
sizes = [proportion_male, proportion_female]
colors = ['blue', 'pink']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
plt.title("Composition of all patients with respect to the Sex category")
plt.axis('equal')
plt.savefig("Composition of all patients with respect to the Sex category.jpg")
```

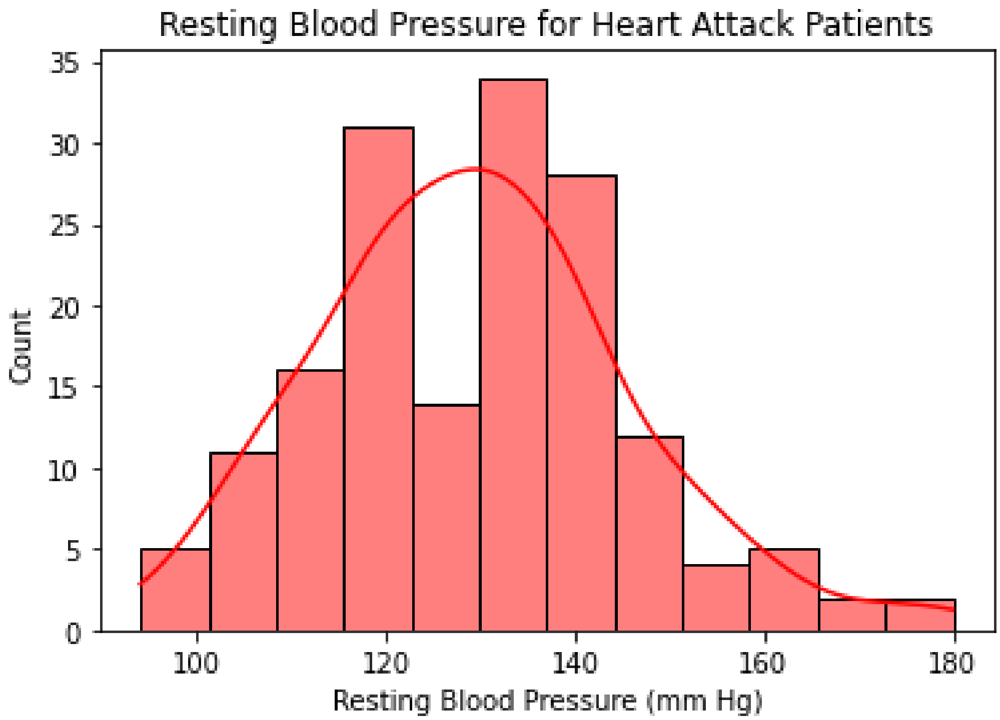
```
plt.show()
```



Study if one can detect heart attacks based on anomalies in the resting blood pressure (trestbps) of a patient Below code can help us understand if this analysis is possible or not based on anomalies in the resting blood pressure (trestbps) of a patient

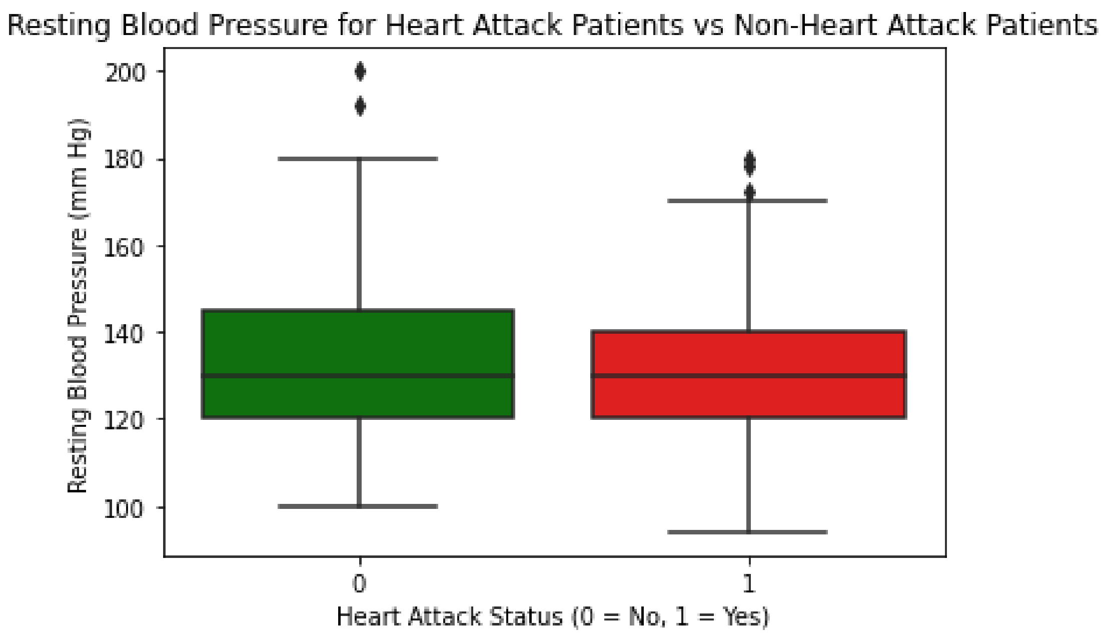
Lets create a histogram first of resting blood pressure for heart attack patients

[19]:
```
heart_attack_patients = df[df['target'] == 1]
sns.histplot(data=heart_attack_patients, x='trestbps', kde=True, color='red')
plt.title("Resting Blood Pressure for Heart Attack Patients")
plt.xlabel("Resting Blood Pressure (mm Hg)")
plt.savefig("Resting Blood Pressure for Heart Attack Patients.jpg")
plt.show()
```



Create a boxplot of resting blood pressure for heart attack patients vs non-heart attack patients

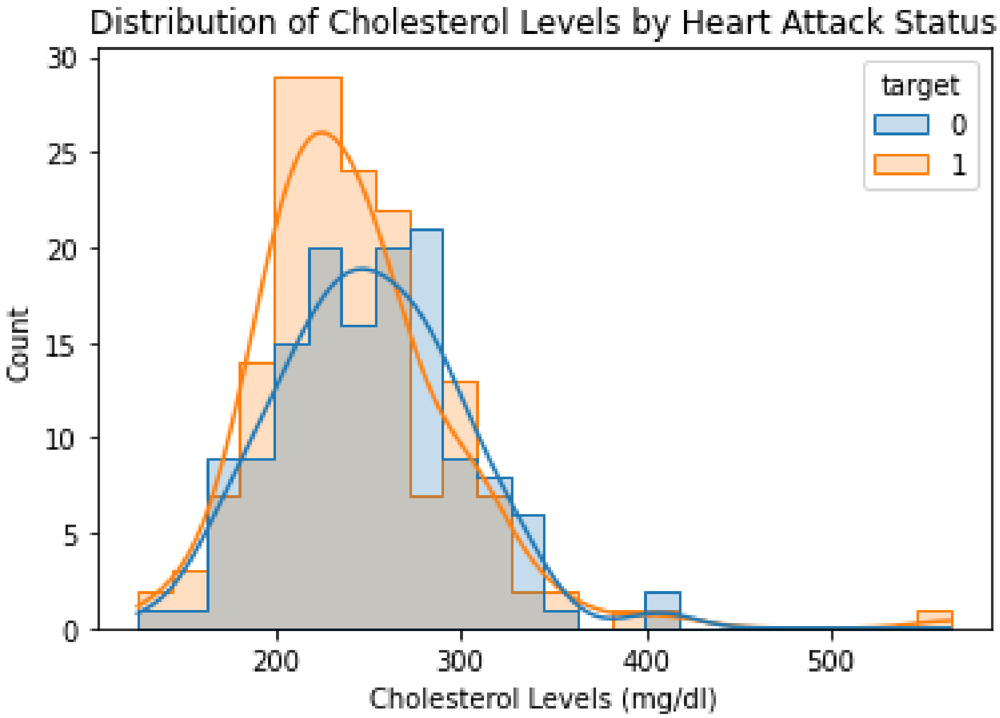
[20]:
```
sns.boxplot(data=df, x='target', y='trestbps', palette=["green", "red"])
plt.title("Resting Blood Pressure for Heart Attack Patients vs Non-Heart Attack␣
↪Patients")
plt.xlabel("Heart Attack Status (0 = No, 1 = Yes)")
plt.ylabel("Resting Blood Pressure (mm Hg)")
plt.savefig("Resting Blood Pressure for Heart Attack Patients vs Non-Heart␣
↪Attack Patients.jpg")
plt.show()
```



The histogram shows the distribution of resting blood pressure for heart attack patients, and the boxplot shows the median, quartiles, and outliers for resting blood pressure in heart attack patients vs non-heart attack patients. By analyzing these plots, we can determine if there are any anomalies in the resting blood pressure of heart attack patients compared to non-heart attack patients. If there are significant differences in the resting blood pressure between these groups, we can consider resting blood pressure as a potential predictor of heart attacks.

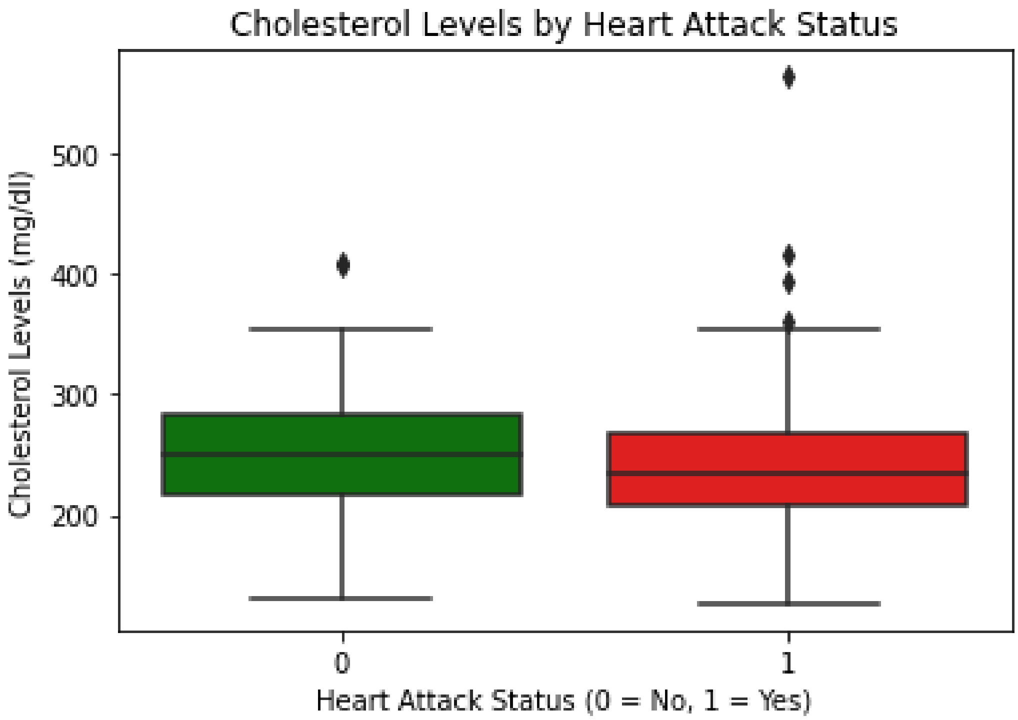
Describe the relationship between cholesterol levels and a target variable Plot the distribution of cholesterol levels in heart attack patients and non-heart attack patients

[21]:
```
sns.histplot(data=df, x='chol', hue='target', element='step', kde=True)
plt.title('Distribution of Cholesterol Levels by Heart Attack Status')
plt.xlabel('Cholesterol Levels (mg/dl)')
plt.ylabel('Count')
plt.savefig("Distribution of Cholesterol Levels by Heart Attack Status.jpg")
plt.show()
```



plot the relationship between cholesterol levels and heart attack occurrence using a boxplot

[22]:
```
sns.boxplot(data=df, x='target', y='chol', palette=["green", "red"])
plt.title('Cholesterol Levels by Heart Attack Status')
plt.xlabel('Heart Attack Status (0 = No, 1 = Yes)')
plt.ylabel('Cholesterol Levels (mg/dl)')
plt.savefig("Cholesterol Levels by Heart Attack Status.jpg")
plt.show()
```



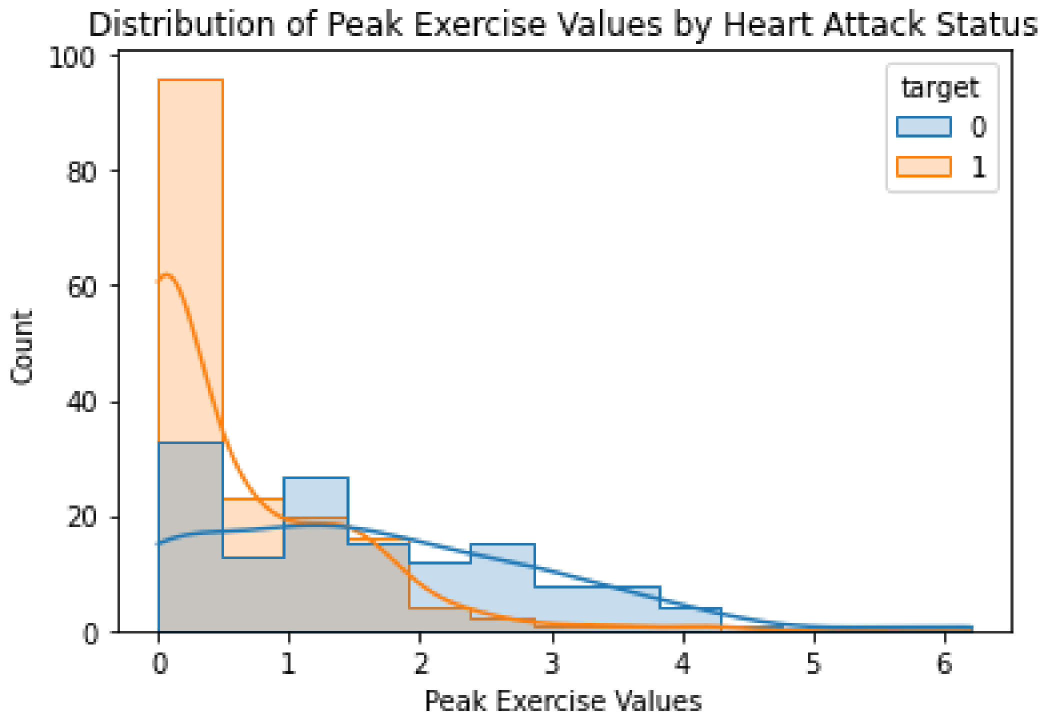
Calculate the correlation coefficient between cholesterol levels and the target variable

[23]:
```
correlation = df['chol'].corr(df['target'])
print(f'The Correlation Coefficient between cholestrol levels & the target␣
 ↪variable is: {correlation}')
```

```
The Correlation Coefficient between cholestrol levels & the target variable is:
-0.08143720051844144
```

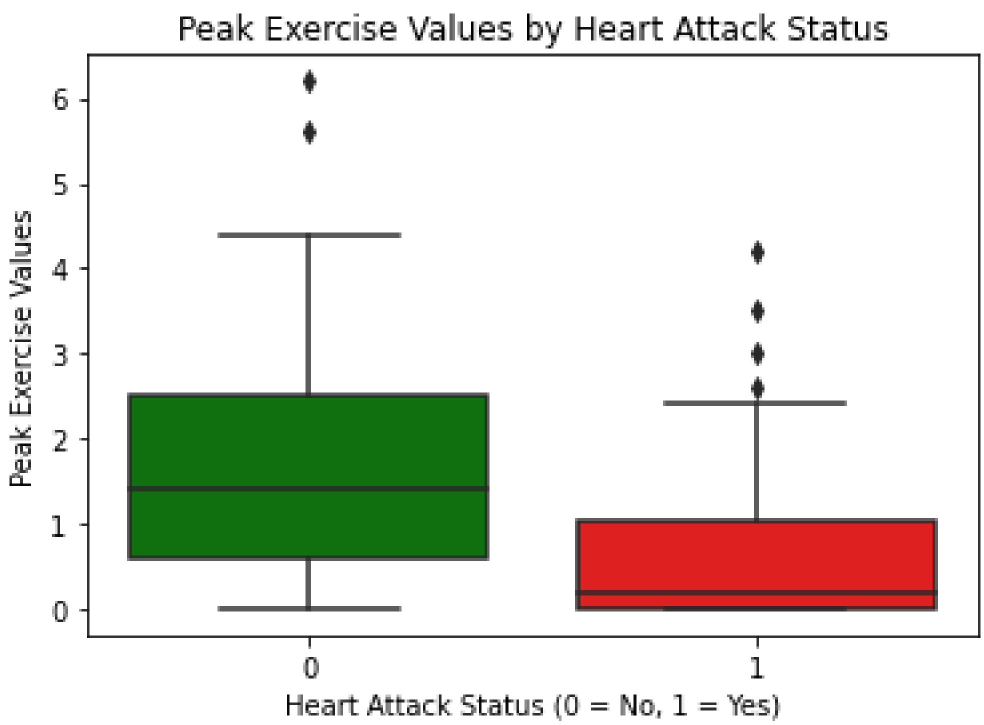
State what relationship exists between peak exercising & the occurrence of a heart attack Plot the distribution of peak exercise values in heart attack patients & non-heart attack patients

[24]:
```
sns.histplot(data=df, x='oldpeak', hue='target', element='step', kde=True)
plt.title('Distribution of Peak Exercise Values by Heart Attack Status')
plt.xlabel('Peak Exercise Values')
plt.ylabel('Count')
plt.savefig("Distribution of Peak Exercise Values by Heart Attack Status.jpg")
plt.show()
```



plot the relationship between peak exercise values and heart attack occurrence using a boxplot

[25]:
```
sns.boxplot(data=df, x='target', y='oldpeak', palette=["green", "red"])
plt.title('Peak Exercise Values by Heart Attack Status')
plt.xlabel('Heart Attack Status (0 = No, 1 = Yes)')
plt.ylabel('Peak Exercise Values')
plt.savefig("Peak Exercise Values by Heart Attack Status.jpg")
plt.show()
```



Calculate the correlation coefficient between peak exercise values and the target variable

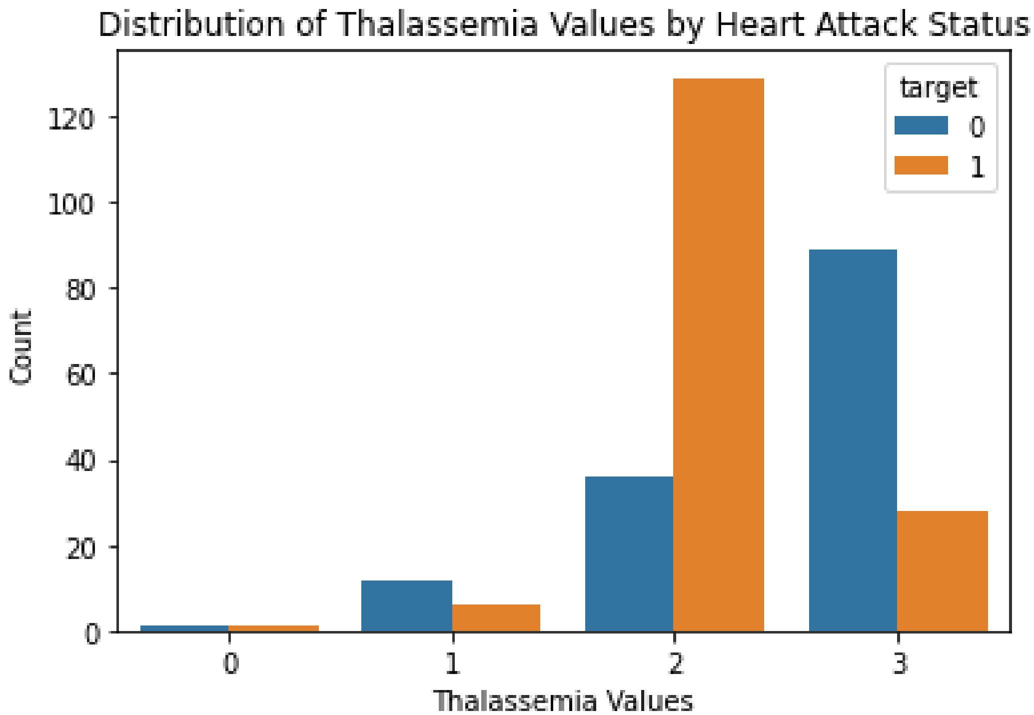
[26]:
```
correlation = df['oldpeak'].corr(df['target'])
print(f'The correlation coefficient between peak exercise values and the target␣
 ↪variable is: {correlation}')
```

```
The correlation coefficient between peak exercise values and the target variable
is: -0.429145832886738
```

By analyzing these plots and the correlation coefficient, we can determine if there is a relationship between peak exercising and the occurrence of heart attacks. The histogram shows the distribution of peak exercise values for heart attack patients vs non-heart attack patients, and the boxplot shows the median, quartiles, and outliers for peak exercise values by heart attack status.

Check if thalassemia is a major cause of CVD Plot the distribution of thalassemia values in heart attack patients and non-heart attack patients

[27]:
```
sns.countplot(data=df, x='thal', hue='target')
plt.title('Distribution of Thalassemia Values by Heart Attack Status')
plt.xlabel('Thalassemia Values')
plt.ylabel('Count')
plt.savefig("Distribution of Thalassemia Values by Heart Attack Status.jpg")
plt.show()
```



Calculate the proportion of heart attack patients by thalassemia value

[28]:
```
thal_counts = df.groupby('thal')['target'].value_counts(normalize=True).
 ↪reset_index(name='proportion')
thal_counts = thal_counts[thal_counts['target'] == 1].
 ↪sort_values(by='proportion', ascending=False)
print(thal_counts)
```

```
   thal  target  proportion
4     2       1    0.781818
1     0       1    0.500000
3     1       1    0.333333
7     3       1    0.239316
```

By analyzing these plots and the proportion of heart attack patients by thalassemia value, we can determine if thalassemia is a major cause of CVD. The countplot shows the distribution of thalassemia values for heart attack patients vs non-heart attack patients. If there are significant differences in thalassemia values between these groups, we can consider thalassemia as a potential predictor of heart attacks.

0.2 List how the other factors determine the occurrence of CVD

0.2.1 1. Correlation analysis:

Calculate correlation coefficients between variables

[29]:
```
corr_matrix = df.corr()
```

Show correlations with target variable

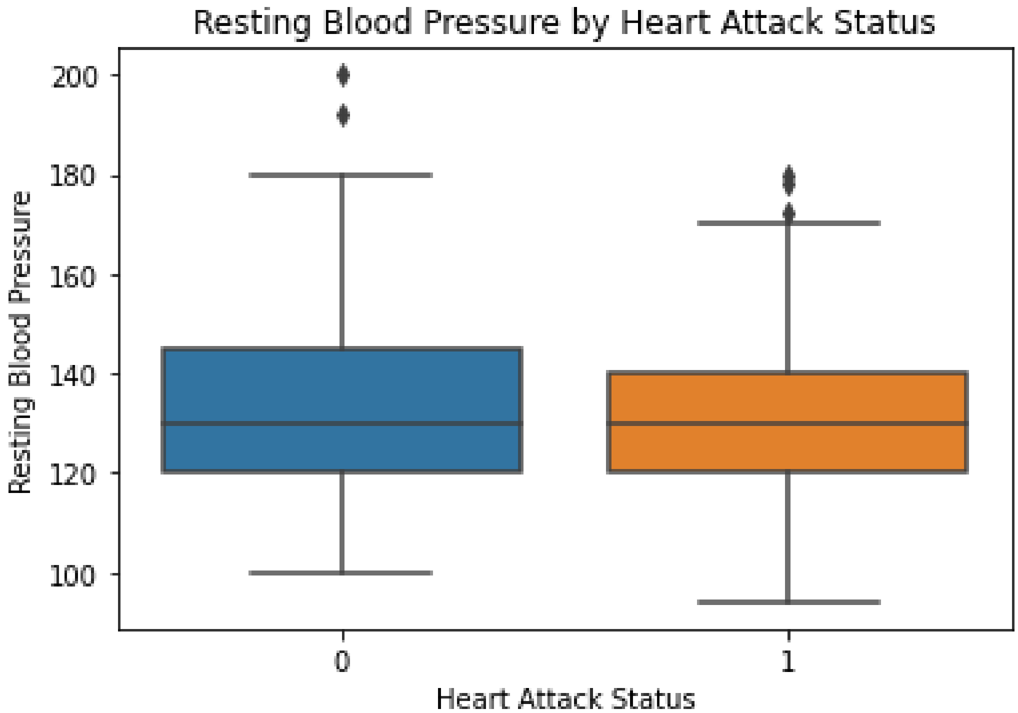
[30]:
```
print(corr_matrix['target'].sort_values(ascending=False))
```

```
target      1.000000
cp          0.432080
thalach     0.419955
slope       0.343940
restecg     0.134874
fbs        -0.026826
chol       -0.081437
trestbps   -0.146269
age        -0.221476
sex        -0.283609
thal       -0.343101
ca         -0.408992
oldpeak    -0.429146
exang      -0.435601
Name: target, dtype: float64
```

0.2.2 2. Data visualization:

Create boxplot of resting blood pressure (trestbps) by heart attack statu

[31]:
```
sns.boxplot(data=df, x='target', y='trestbps')
plt.title('Resting Blood Pressure by Heart Attack Status')
plt.xlabel('Heart Attack Status')
plt.ylabel('Resting Blood Pressure')
plt.savefig("Resting Blood Pressure by Heart Attack Status.jpg")
plt.show()
```



0.2.3 3. Hypothesis testing:

Conduct t-test of cholesterol level by heart attack status

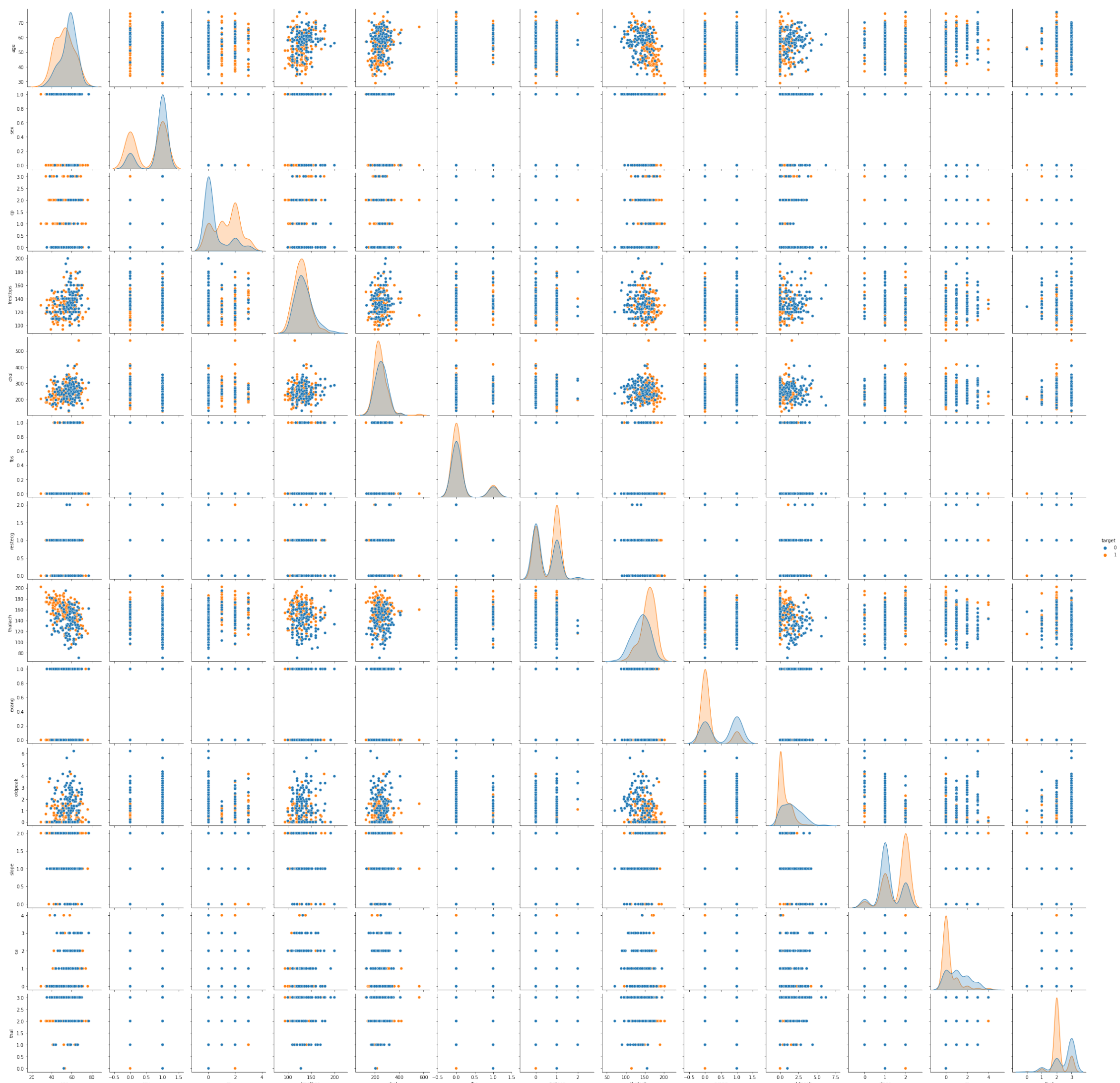
[32]:
```
heart_attack_chol = df[df['target'] == 1]['chol']
no_heart_attack_chol = df[df['target'] == 0]['chol']
t_stat, p_val = stats.ttest_ind(heart_attack_chol, no_heart_attack_chol)
print('t-statistic:', t_stat)
print('p-value:', p_val)
```

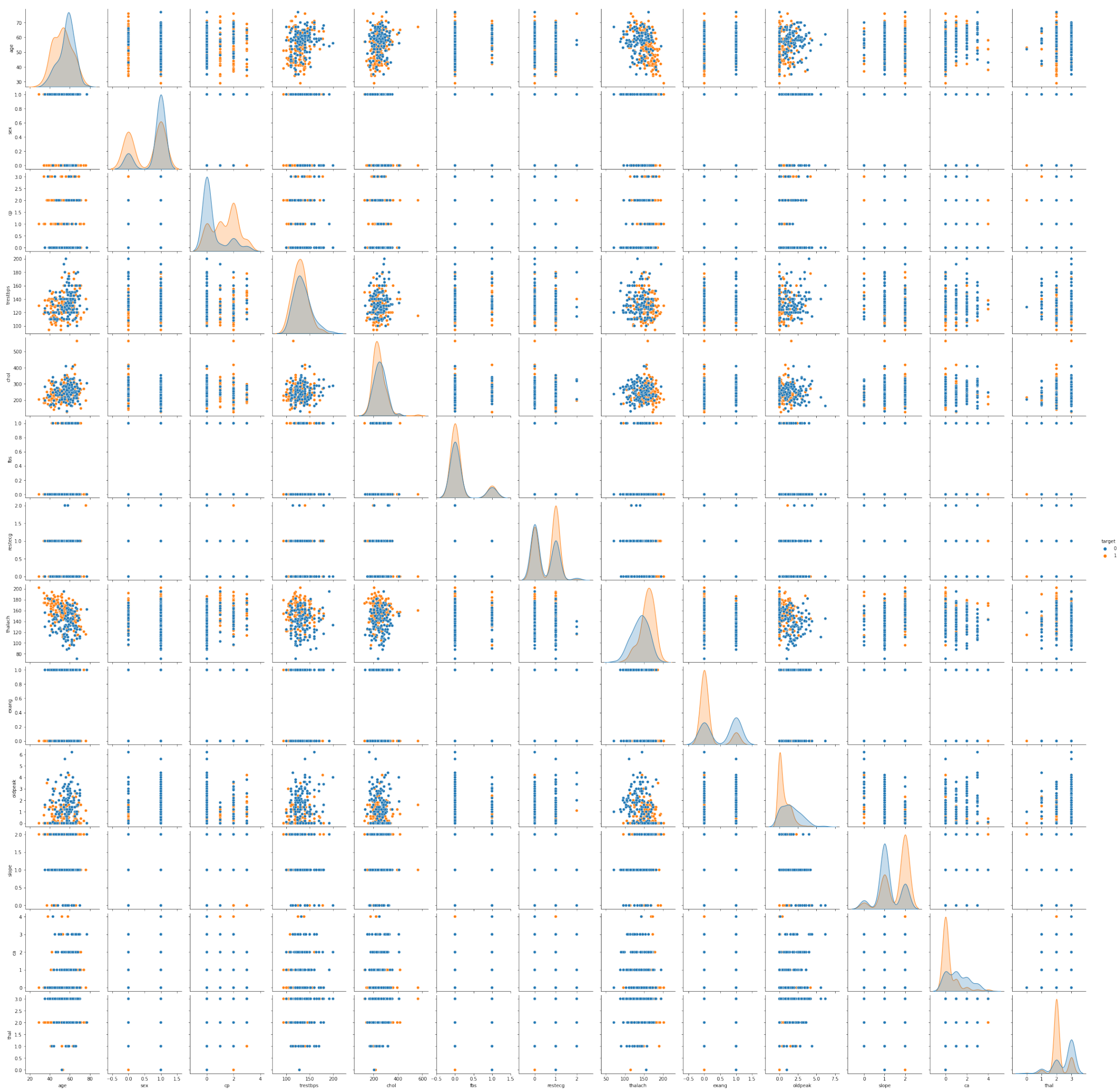
```
t-statistic: -1.4152344258787561
p-value: 0.15803697464249714
```

By using these techniques, we can identify how different factors in the heart attack dataset determine the occurrence of CVD. We can use the resulting insights to develop predictive models or identify risk factors that can help prevent the onset of CVD.

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

[33]:
```
sns.pairplot(data=df, hue='target')
pairplot_image=sns.pairplot(data=df, hue='target')
pairplot_image.savefig("pairplt.png")
```


age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target
0
1



The resulting pair plot will give us a visual overview of the relationships between all the variables in the dataset, and can help us identify potential predictors of heart attack.

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 Split the data into training and test sets

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('target', axis=1),␣
↪df['target'], test_size=0.2, random_state=42)
```

Fit logistic regression model

[35]:
```
logreg = LogisticRegression(max_iter=770)
logreg.fit(X_train, y_train)
```

[35]:
```
LogisticRegression(max_iter=770)
```

Evaluate logistic regression model on test set

[36]:
```
y_pred = logreg.predict(X_test)
acc_logreg = accuracy_score(y_test, y_pred)
cm_logreg = confusion_matrix(y_test, y_pred)
```

[37]:
```
print("Logistic Regression Accuracy:", acc_logreg)
print("Logistic Regression Confusion Matrix:\n", cm_logreg)
```

```
Logistic Regression Accuracy: 0.819672131147541
Logistic Regression Confusion Matrix:
 [[24  5]
 [ 6 26]]
```

Fit random forest model

[38]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

[38]:
```
RandomForestClassifier(random_state=42)
```

Evaluate random forest model on test set

[39]:
```
y_pred = rf.predict(X_test)
acc_rf = accuracy_score(y_test, y_pred)
cm_rf = confusion_matrix(y_test, y_pred)
```

[40]:
```
print("Random Forest Accuracy:", acc_rf)
print("Random Forest Confusion Matrix:\n", cm_rf)
```

```
Random Forest Accuracy: 0.8688524590163934
Random Forest Confusion Matrix:
 [[26  3]
 [ 5 27]]
```

Perform correlation analysis and logistic regression for feature selection

[41]:
```
X_train = sm.add_constant(X_train)
logit_model=sm.Logit(y_train,X_train)
result=logit_model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.347733
         Iterations 7
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                 target   No. Observations:                  241
Model:                          Logit   Df Residuals:                      227
Method:                           MLE   Df Model:                           13
Date:                Sat, 25 Mar 2023   Pseudo R-squ.:                  0.4950
Time:                        17:45:06   Log-Likelihood:                -83.804
converged:                       True   LL-Null:                       -165.95
Covariance Type:            nonrobust   LLR p-value:                 2.662e-28
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          3.8752      2.924      1.325      0.185      -1.855       9.606
age            0.0028      0.026      0.110      0.913      -0.048       0.053
sex           -1.8149      0.523     -3.467      0.001      -2.841      -0.789
cp             0.6955      0.202      3.450      0.001       0.300       1.091
trestbps      -0.0279      0.012     -2.393      0.017      -0.051      -0.005
chol          -0.0045      0.004     -1.068      0.285      -0.013       0.004
fbs            0.5405      0.637      0.849      0.396      -0.708       1.789
restecg        0.5445      0.395      1.380      0.168      -0.229       1.318
thalach        0.0251      0.012      2.098      0.036       0.002       0.049
exang         -1.1128      0.465     -2.391      0.017      -2.025      -0.200
oldpeak       -0.4721      0.253     -1.867      0.062      -0.968       0.023
slope          0.8707      0.391      2.227      0.026       0.104       1.637
ca            -0.8744      0.244     -3.587      0.000      -1.352      -0.397
thal          -1.1090      0.346     -3.206      0.001      -1.787      -0.431
==============================================================================
```

This gives a summary of our analysis and the best algorithm to have a predictive analysis is Random Forest with a Accuracy of 0.8688524590163934