Heart Disease Predictions Using Supervised Learning

In [4]:
```
# Import necessary Libraries

# For data analysis
import pandas as pd
import numpy as np

# For data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Data pre-processing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

#Classifier Libraries
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Ipip install xgboost
from xgboost import XGBClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Evaluation metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_
from sklearn.metrics import confusion_matrix

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

In [5]:
```
# Load the dataset
df = pd.read_csv(r"C:\Users\ADMIN\Desktop\New folder (2)\10Alytics Data Science\Machine Learn
df.head()
```

Out[5]:

	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

Features in the dataset and meaning:

- age – age in years
- sex – (1=male, 0=female)
- cp – chest pain type (1: typical angina, 2: atypical angina, 3: non-angina pain, 4: asymptomatic)
- treslbps – resting blood pressure (in mm Hg on admission to the hospital)
- chol – serum cholesterol in mg/dl,
- fbs – (fasting blood sugar>120mg/dl) (1=true, 0=false)

- restecg – resting electrocardiographic results
- thalach – maximum heart rate achieved
- exang – exercise induced by angina (1=yes, 0=no)
- oldpeak – ST depression induced by exercise relative to rest
- slope – the slope of the peak exercise ST segment
- ca – number of major vessels (0-3) colored by flourosopy
- thal – 3 = normal, 6 = fixed detect, 7 = reversable detect
- target – have disease or not (1=yes, 0=no)

In [6]:
```
# For better understanding and flow of analysis, I will rename some of the columns

df.columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol', 'fast
df.head()
```

Out[6]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_ecg	max_heart_rate
0	63	1	3	145	233	1	0	
1	37	1	2	130	250	0	1	
2	41	0	1	130	204	0	0	
3	56	1	1	120	236	0	1	
4	57	0	0	120	354	0	1	

In [7]:
```
# Data verification - Data type, number of features and rows, missing data, e.t.c

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   chest_pain_type          303 non-null    int64  
 3   resting_blood_pressure   303 non-null    int64  
 4   cholesterol              303 non-null    int64  
 5   fasting_blood_sugar      303 non-null    int64  
 6   rest_ecg                 303 non-null    int64  
 7   max_heart_rate_achieved  303 non-null    int64  
 8   exercise_induced_angina  303 non-null    int64  
 9   st_depression            303 non-null    float64
 10  st_slope                 303 non-null    int64  
 11  num_major_vessels        303 non-null    int64  
 12  thalassemia              303 non-null    int64  
 13  target                   303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [8]:
```
# Statistical Analysis of the data

df.describe()
```

Out[8]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.52
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.52
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.00
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.00

In [9]:

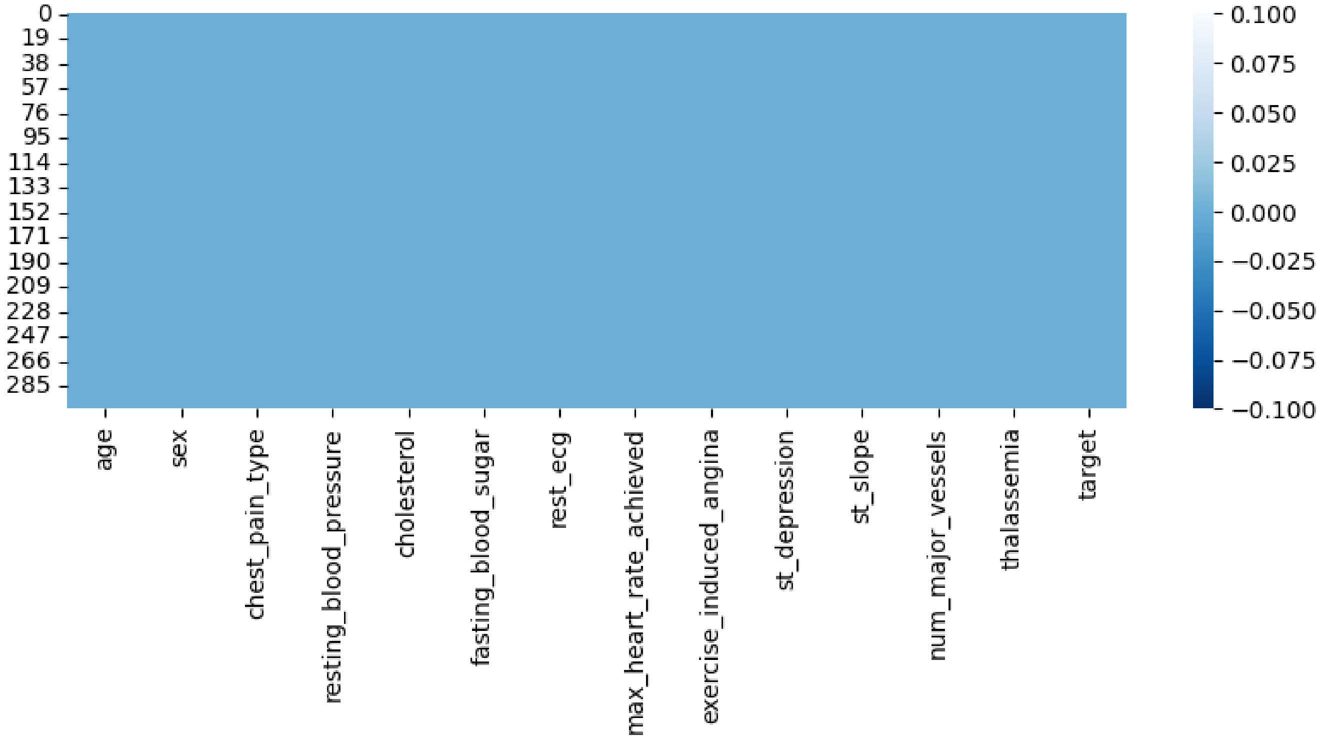
```
# Check for missing values

print(df.isnull().sum())
```

```
age                        0
sex                        0
chest_pain_type            0
resting_blood_pressure     0
cholesterol                0
fasting_blood_sugar        0
rest_ecg                   0
max_heart_rate_achieved    0
exercise_induced_angina    0
st_depression              0
st_slope                   0
num_major_vessels          0
thalassemia                0
target                     0
dtype: int64
```

In [10]:

```
# Visualization the missing data
plt.figure(figsize = (10,3))
sns.heatmap(df.isnull(), cbar=True, cmap="Blues_r")
```

Out[10]: <Axes: >



Observation

- There is no missing value in the dataset

Exploratory Data Analysis

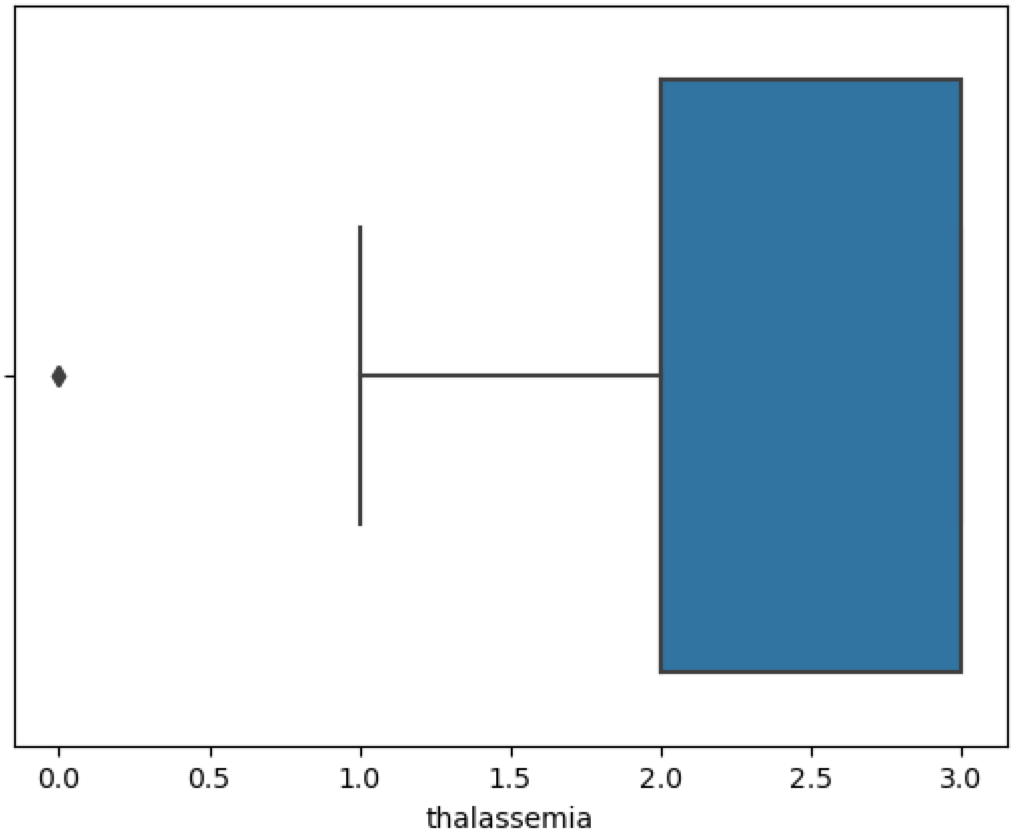
Univariate Analysis

In [11]:
```
df.columns
```

Out[11]:
```
Index(['age', 'sex', 'chest_pain_type', 'resting_blood_pressure',
       'cholesterol', 'fasting_blood_sugar', 'rest_ecg',
       'max_heart_rate_achieved', 'exercise_induced_angina', 'st_depression',
       'st_slope', 'num_major_vessels', 'thalassemia', 'target'],
      dtype='object')
```

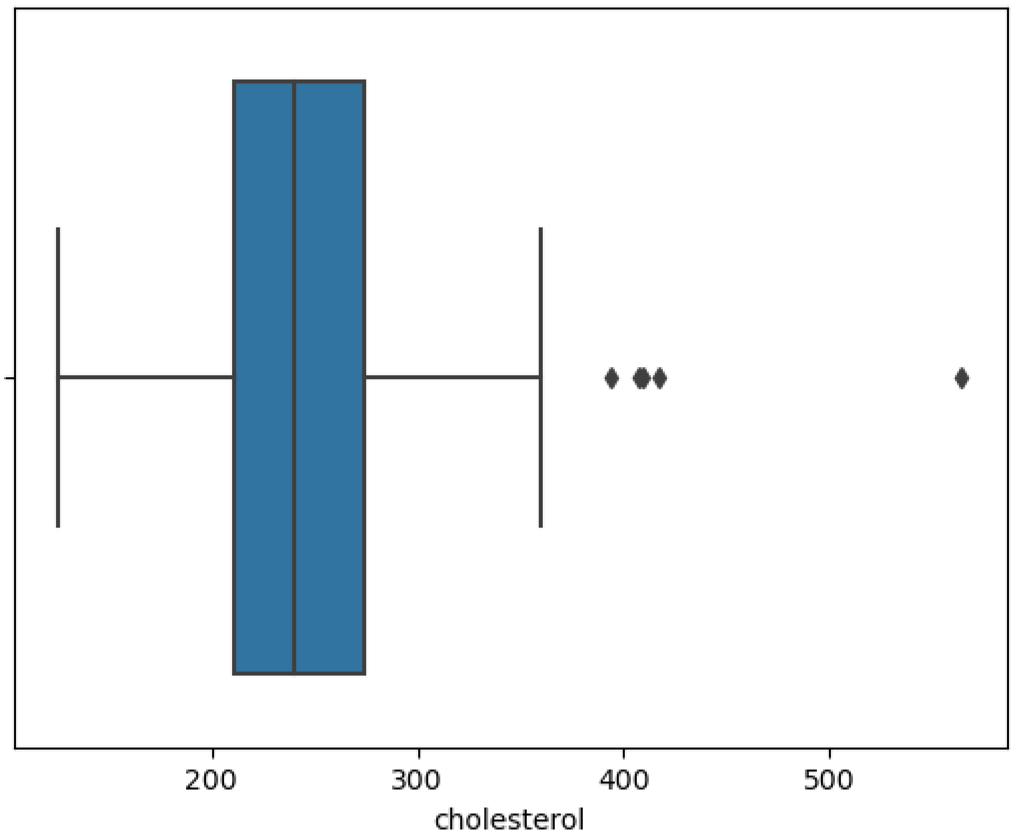
In [12]:
```
# Check for outliers
sns.boxplot (x=df["thalassemia"])
```

Out[12]:
```
<Axes: xlabel='thalassemia'>
```



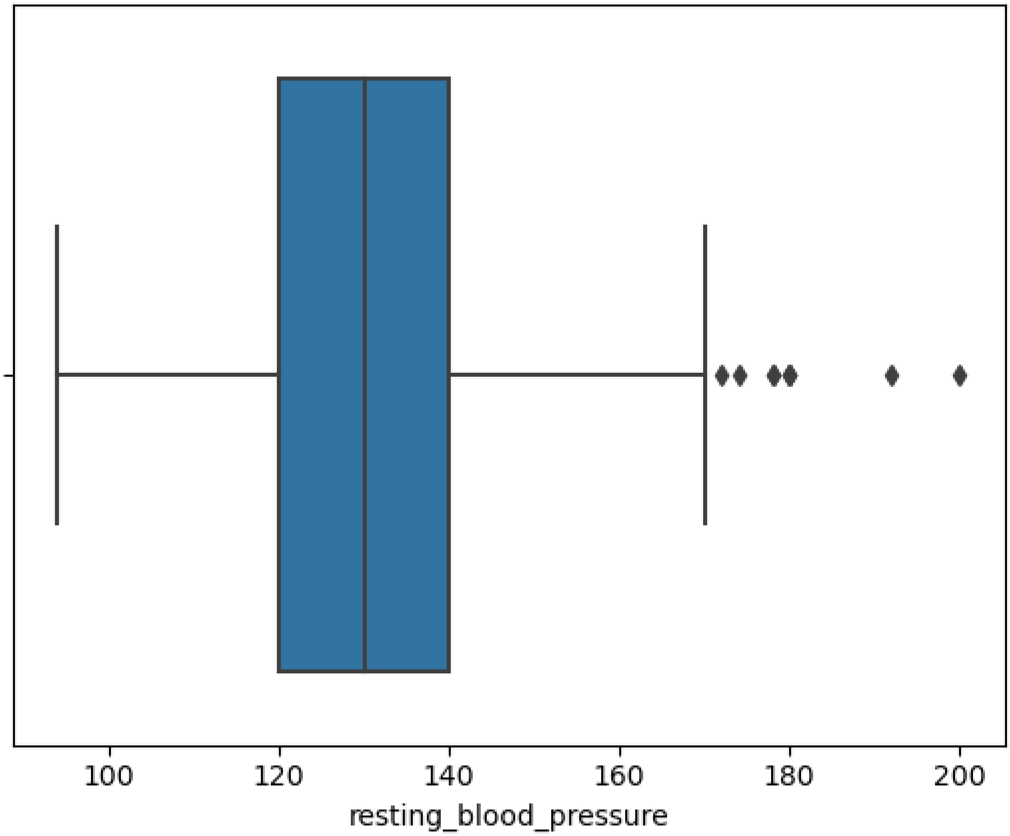
In [13]:
```
# Check for outliers
sns.boxplot (x=df["cholesterol"])
```

Out[13]: `<Axes: xlabel='cholesterol'>`



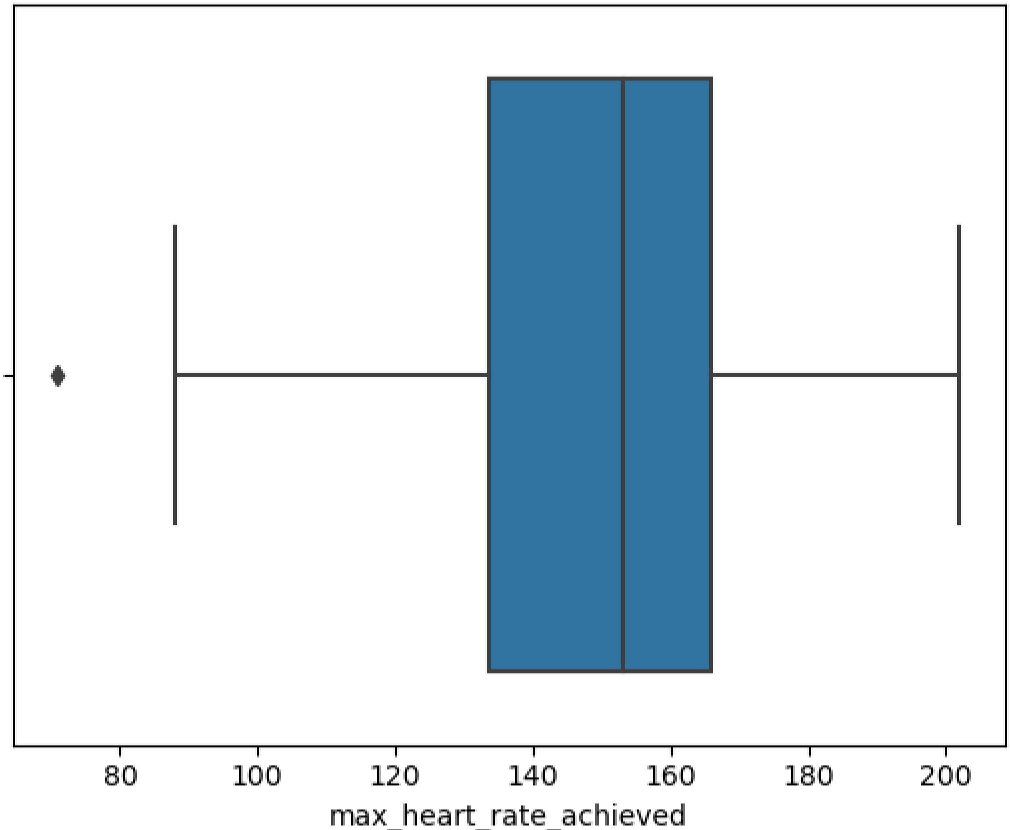
In [14]:
```
#check for outliers
sns.boxplot (x=df["resting_blood_pressure"])
```

Out[14]: `<Axes: xlabel='resting_blood_pressure'>`



In [15]:
```
# check for outliers
sns.boxplot (x=df["max_heart_rate_achieved"])
```

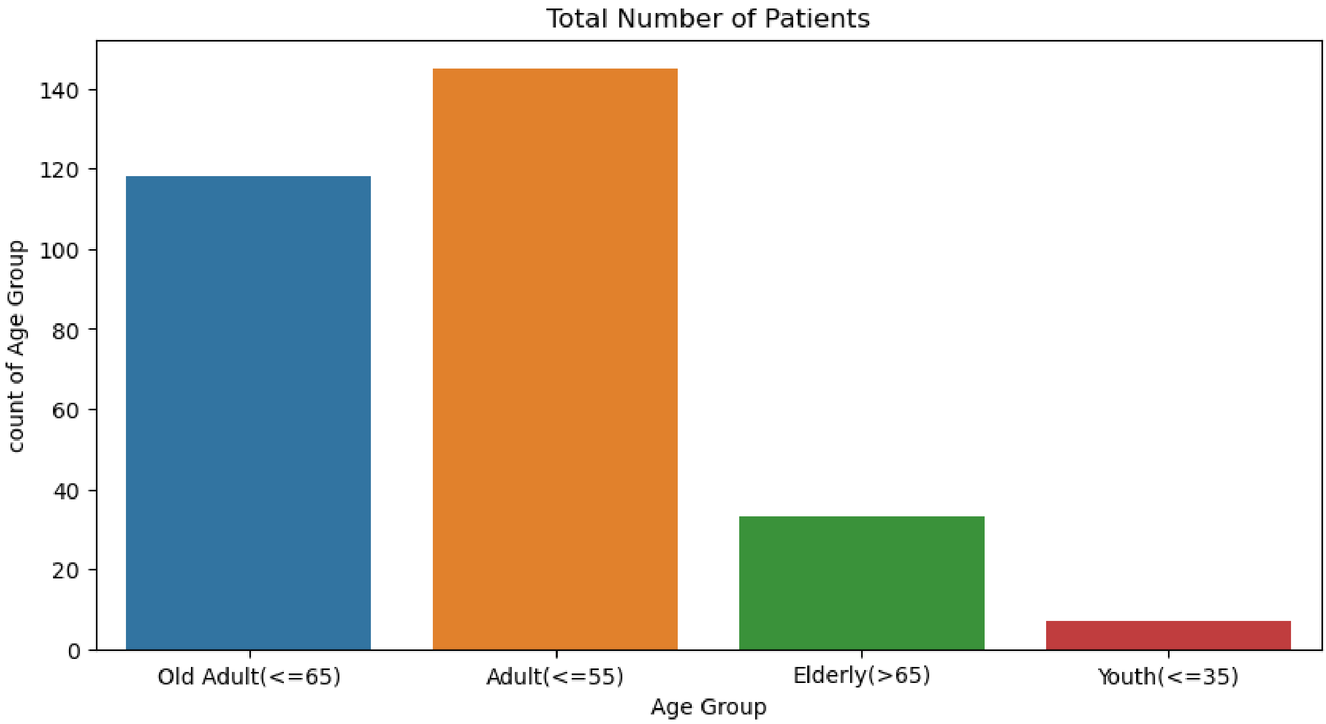
Out[15]: <Axes: xlabel='max_heart_rate_achieved'>



In [16]:
```
# Data visualization
# Age bracket

def age_bracket(age):
    if age <= 35:
        return "Youth(<=35)"
```

```
    elif age <= 55:
        return "Adult(<=55)"
    elif age <= 65:
        return "Old Adult(<=65)"
    else:
        return "Elderly(>65)"

df['age_bracket'] = df['age'].apply(age_bracket)

# Investigating the age group of patients

plt.figure(figsize = (10, 5))
sns.countplot(x='age_bracket', data=df)
plt.xlabel('Age Group')
plt.ylabel('count of Age Group')
plt.title('Total Number of Patients')
```

Out[16]:
```
Text(0.5, 1.0, 'Total Number of Patients')
```

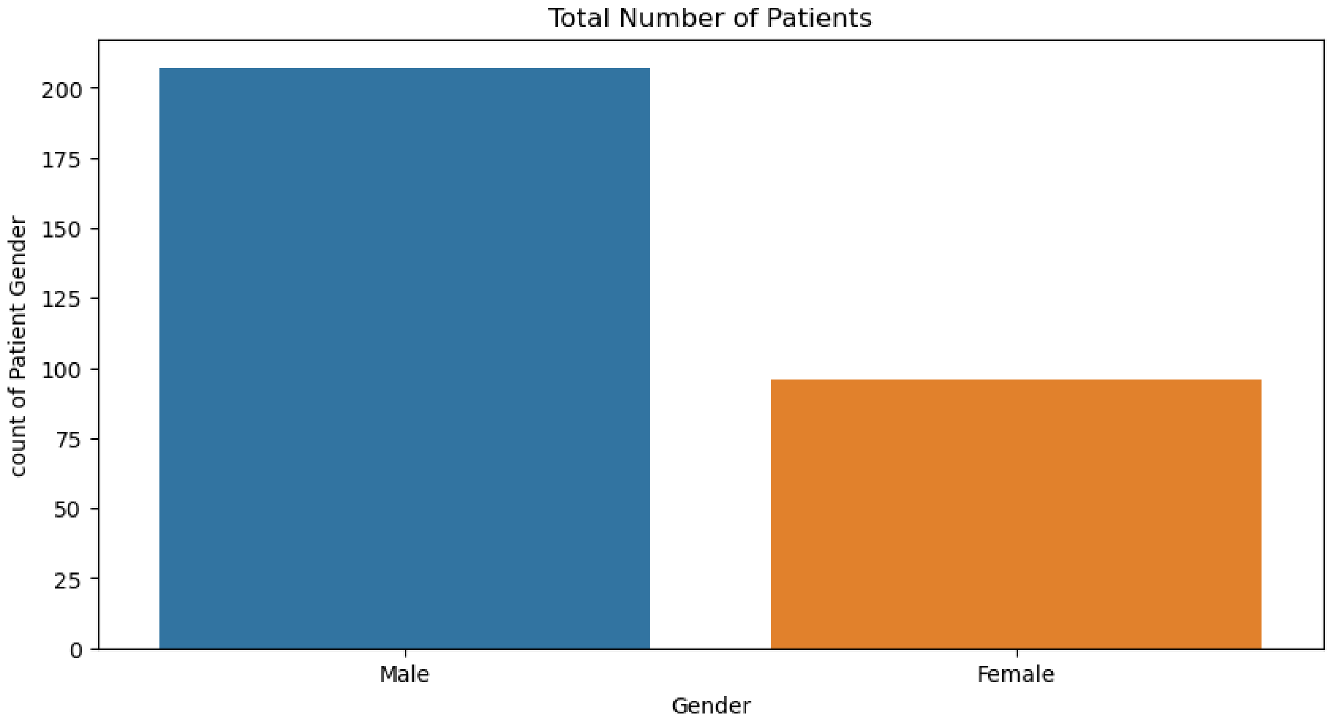


Observation

Based on the chart above, majority of patients age is less than or equal to 55 years.

In [17]:
```
# Data visualization
# Sex

def gender(sex):
    if sex == 1:
        return "Male"
    else:
        return "Female"

df['gender'] = df['sex'].apply(gender)

# Investigating the age group of patients

plt.figure(figsize = (10, 5))
sns.countplot(x='gender', data=df)
plt.xlabel('Gender')
plt.ylabel('count of Patient Gender')
plt.title('Total Number of Patients')
```

Out[17]: Text(0.5, 1.0, 'Total Number of Patients')

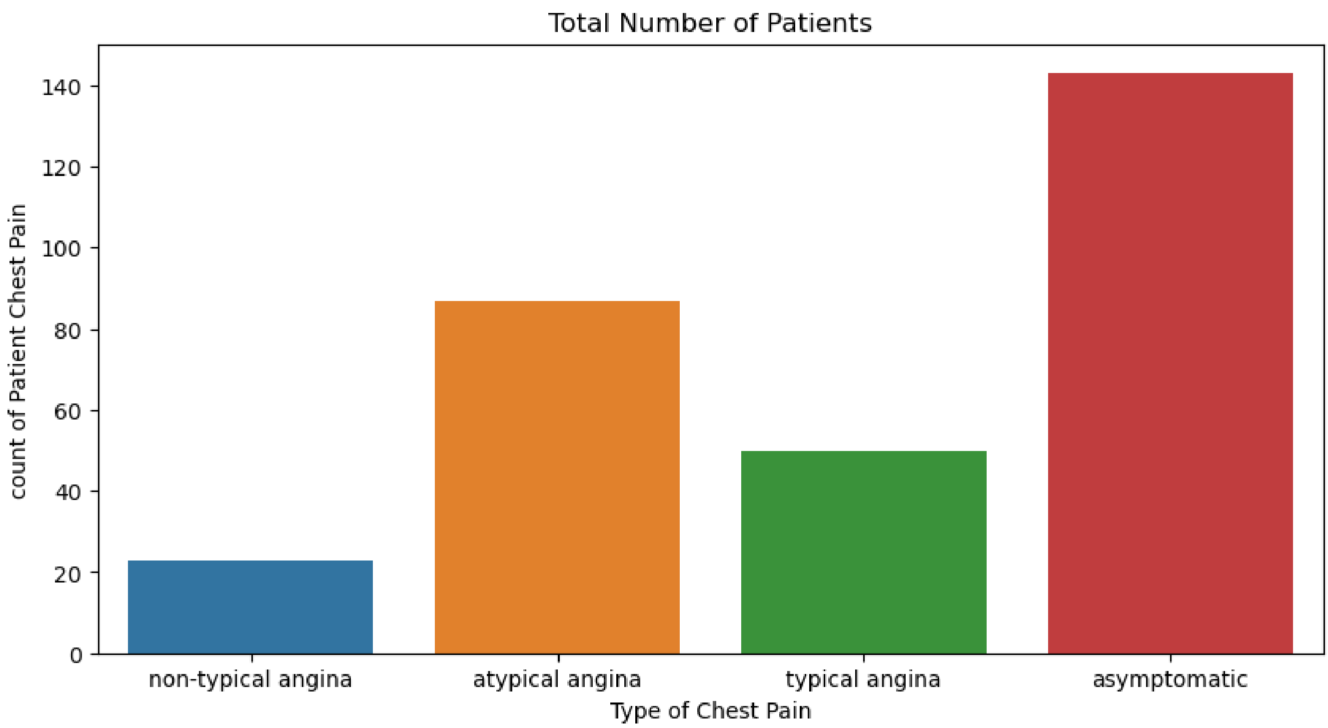


Observation

Based on the gender, the number of male patients is more than double of the female patients.

In [18]:
```
# Data visualization
# Chest pain type (1: typical angina, 2: atypical angina, 3: non-angina pain, 4: asymptomatic

def chest_pain(cp):
    if cp == 1:
        return "typical angina"
    elif cp == 2:
        return "atypical angina"
    elif cp == 3:
        return "non-typical angina"
    else:
        return "asymptomatic"

df['cp_cat'] = df['chest_pain_type'].apply(chest_pain)

# Investigating the age group of patients

plt.figure(figsize = (10, 5))
sns.countplot(x='cp_cat', data=df)
plt.xlabel('Type of Chest Pain')
plt.ylabel('count of Patient Chest Pain')
plt.title('Total Number of Patients')
```

Out[18]: Text(0.5, 1.0, 'Total Number of Patients')



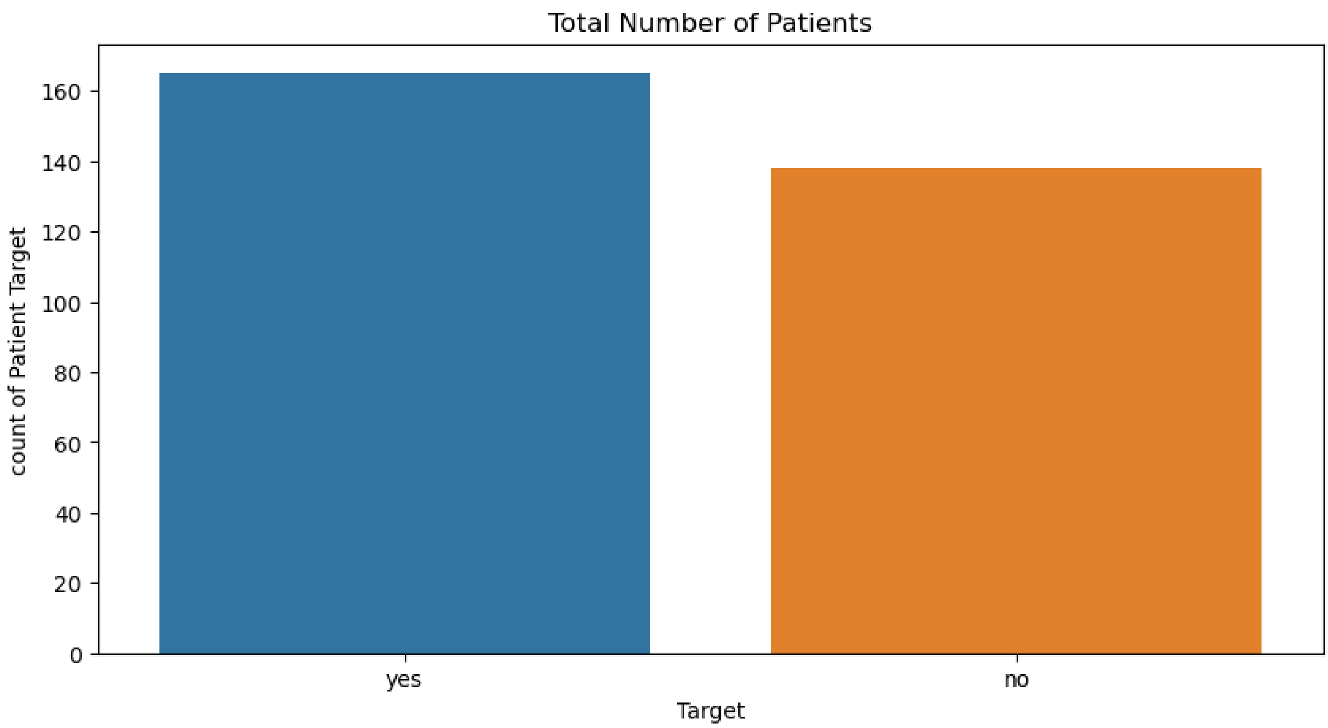
In [19]:

```
# Data visualization
# target - have disease or not (1=yes, 0=no)

def label(tg):
    if tg == 1:
        return "yes"
    else:
        return "no"

df['label'] = df['target'].apply(label)

# total patients in each category

print(df["label"].value_counts())

# Investigating the target of patients

plt.figure(figsize = (10, 5))
sns.countplot(x='label', data=df)
plt.xlabel('Target')
plt.ylabel('count of Patient Target')
plt.title('Total Number of Patients')
```

```
yes    165
no     138
Name: label, dtype: int64
```

Out[19]:

```
Text(0.5, 1.0, 'Total Number of Patients')
```

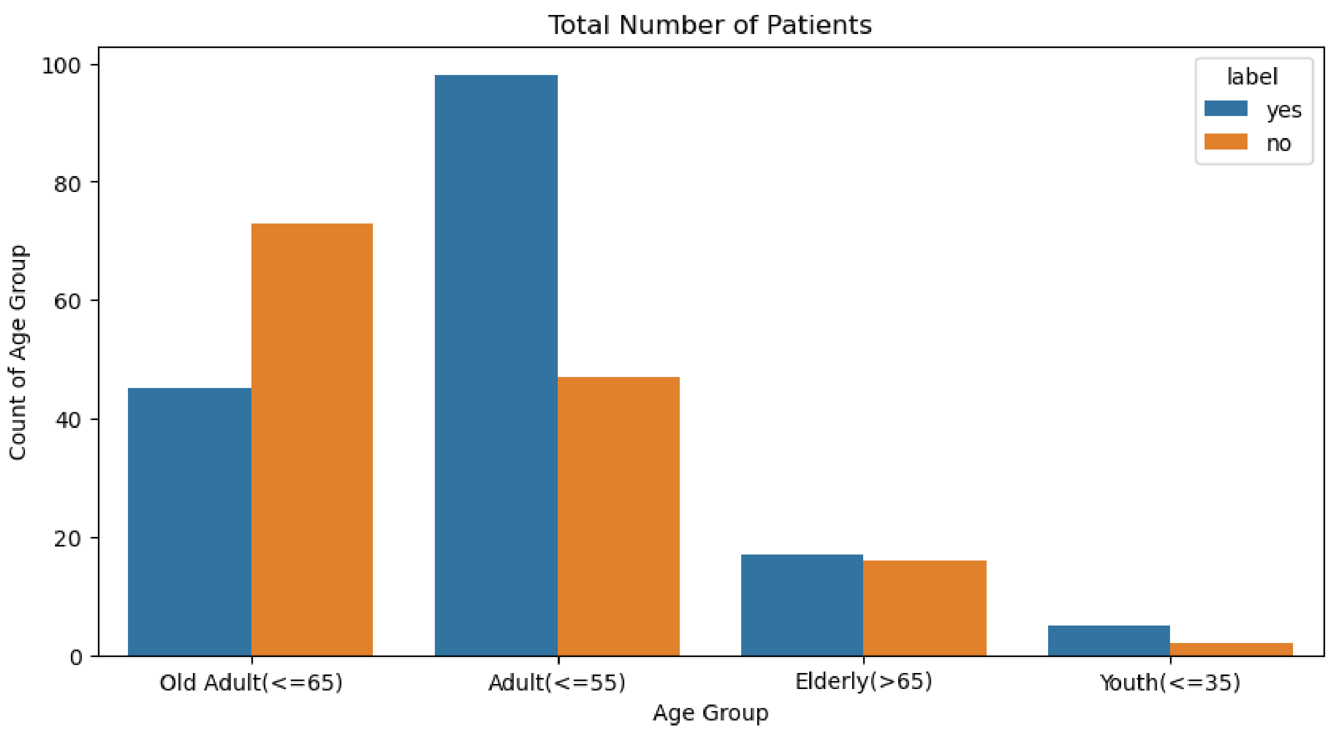


Exploratory Data Analysis

BIVARIATE ANALYSIS

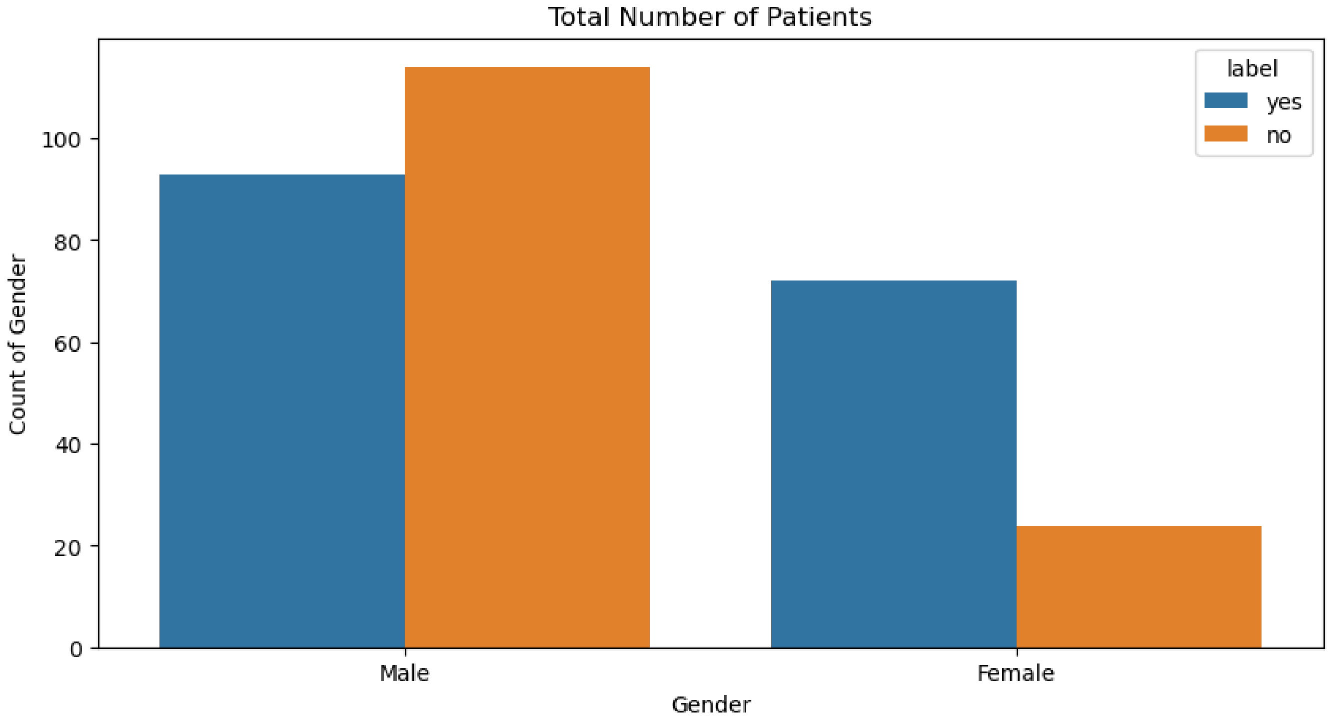
In [20]:
```
# Investigating the age group of patients by the target feature

plt.figure(figsize = (10, 5))
sns.countplot(x='age_bracket', data=df, hue='label')
plt.xlabel('Age Group')
plt.ylabel('Count of Age Group')
plt.title('Total Number of Patients')
```

Out[20]:
```
Text(0.5, 1.0, 'Total Number of Patients')
```



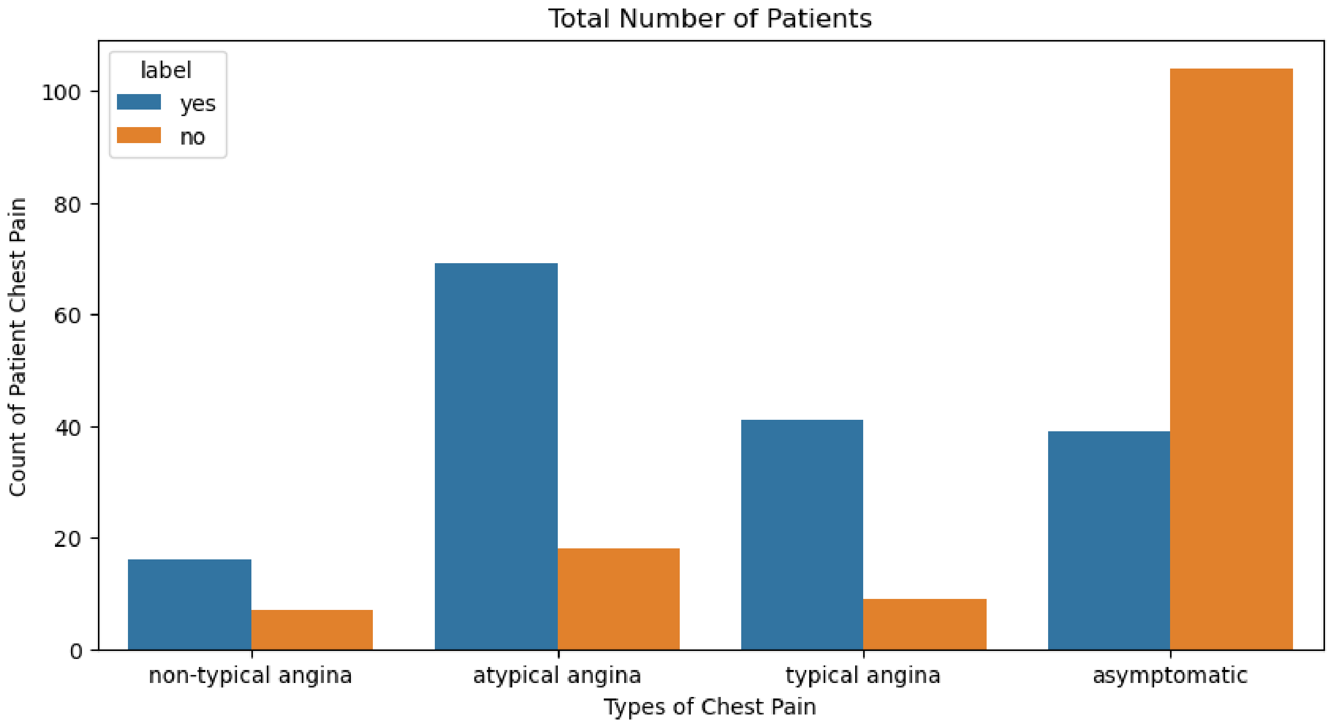
In [21]:
```
# Investigating the gender of patients by the target feature

plt.figure(figsize = (10, 5))
sns.countplot(x='gender', data=df, hue='label')
plt.xlabel('Gender')
plt.ylabel('Count of Gender')
plt.title('Total Number of Patients')
```

Out[21]:
```
Text(0.5, 1.0, 'Total Number of Patients')
```



In [22]:
```
# Investigating the chest pain type by the target featur

plt.figure(figsize = (10, 5))
sns.countplot(x='cp_cat', data=df, hue='label')
plt.xlabel('Types of Chest Pain')
plt.ylabel('Count of Patient Chest Pain')
plt.title('Total Number of Patients')
```

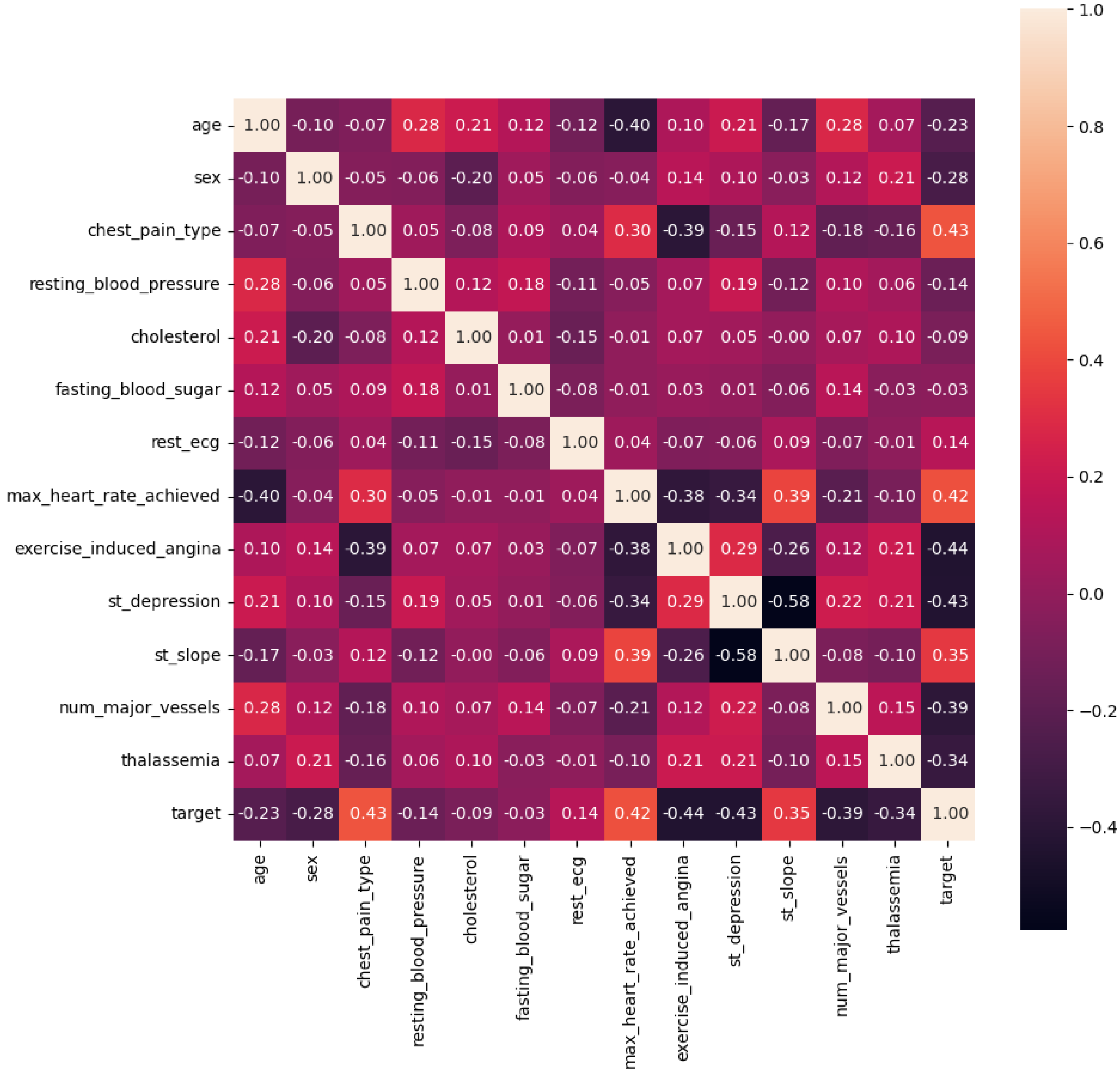
Out[22]:
```
Text(0.5, 1.0, 'Total Number of Patients')
```



Exploratory Data Analysis

Multivariate Analysis

In [23]:
```
# Correlation between heart disease and other variables in the dataset
plt.figure(figsize = (10, 10))
hm = sns.heatmap(df.corr(), cbar=True, annot=True, square=True, fmt=' .2f', annot_kws={'size'
```



Observation

Based on the heatmap presented above. There is negative and postive relationship.

Feature Engineering/Data Pre-Processing

In [24]:
```
# Create a copy of the data (Exlude target/label alongside other columns that was created)
df1 = df[['age', 'sex', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol', 'fasting_
        'max_heart_rate_achieved', 'exercise_induced_angina', 'st_depression','st_slope', 'num

label = df[['target']]
```

In [25]:
```
df1.head()
```

Out[25]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_ecg	max_heart_rate
0	63	1	3	145	233	1	0	
1	37	1	2	130	250	0	1	
2	41	0	1	130	204	0	0	
3	56	1	1	120	236	0	1	
4	57	0	0	120	354	0	1	

In [26]:

```
df1.dtypes
```

Out[26]:

```
age                          int64
sex                          int64
chest_pain_type              int64
resting_blood_pressure       int64
cholesterol                  int64
fasting_blood_sugar          int64
rest_ecg                     int64
max_heart_rate_achieved      int64
exercise_induced_angina      int64
st_depression              float64
st_slope                     int64
num_major_vessels            int64
thalassemia                  int64
dtype: object
```

In [27]:

```
# Dealing with outliers - 'resting_blood_pressure', cholesterol, thalassmia

# Normalize the data

scaler = MinMaxScaler()

df1["Scaled_RBP"] = scaler.fit_transform(df1['resting_blood_pressure'].values.reshape(-1, 1))
df1["Scaled_chol"] = scaler.fit_transform(df1['cholesterol'].values.reshape(-1, 1))
df1["Scaled_thal"] = scaler.fit_transform(df1['thalassemia'].values.reshape(-1, 1))
df1["Scaled_max_heart_rate"] = scaler.fit_transform(df1['max_heart_rate_achieved'].values.res

df1.drop(['resting_blood_pressure', 'cholesterol', 'thalassemia', 'max_heart_rate_achieved'],

df1.head()
```

Out[27]:

	age	sex	chest_pain_type	fasting_blood_sugar	rest_ecg	exercise_induced_angina	st_depression	st_slope	n
0	63	1	3	1	0	0	2.3	0	
1	37	1	2	0	1	0	3.5	0	
2	41	0	1	0	0	0	1.4	2	
3	56	1	1	0	1	0	0.8	2	
4	57	0	0	0	1	1	0.6	2	

In [ ]:

Machine Learning

In [28]:

```
# Split the dataset into training and testing sets - x = questions while y = answers
```

```
X_train, X_test, y_train, y_test = train_test_split(df1, label, test_size=0.2, random_state=4
```

In [50]:
```
X_test.head(3)
```

Out[50]:

	age	sex	chest_pain_type	fasting_blood_sugar	rest_ecg	exercise_induced_angina	st_depression	st_slope
179	57	1	0	0	0	1	0.6	1
228	59	1	3	0	0	0	0.2	1
111	57	1	2	1	1	0	0.2	2

In [49]:
```
y_test.head(3)
```

Out[49]:

	target
179	0
228	0
111	1

In [48]:
```
X_train.head(3)
```

Out[48]:

	age	sex	chest_pain_type	fasting_blood_sugar	rest_ecg	exercise_induced_angina	st_depression	st_slope
132	42	1	1	0	1	0	0.0	2
202	58	1	0	0	0	1	0.8	2
196	46	1	2	0	1	0	3.6	1

In [47]:
```
y_train.head(3)
```

Out[47]:

	target
132	1
202	0
196	0

In [31]:
```
# Model Building
# Logistic Regression

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

ly_pred = logreg.predict(X_test)

print("Logistic Regression")
print("Accuracy:", accuracy_score(y_test, ly_pred))
print("Precision:", precision_score(y_test, ly_pred))
print("Recall:", recall_score(y_test, ly_pred))
print("F1-score:", f1_score(y_test, ly_pred))
print("AUC-ROC:", roc_auc_score(y_test, ly_pred))
```

```
Logistic Regression
Accuracy: 0.8688524590163934
Precision: 0.875
Recall: 0.875
F1-score: 0.875
AUC-ROC: 0.8685344827586206
```

In [32]:

```
ly_pred
```

Out[32]:

```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [33]:

```
y_test
```

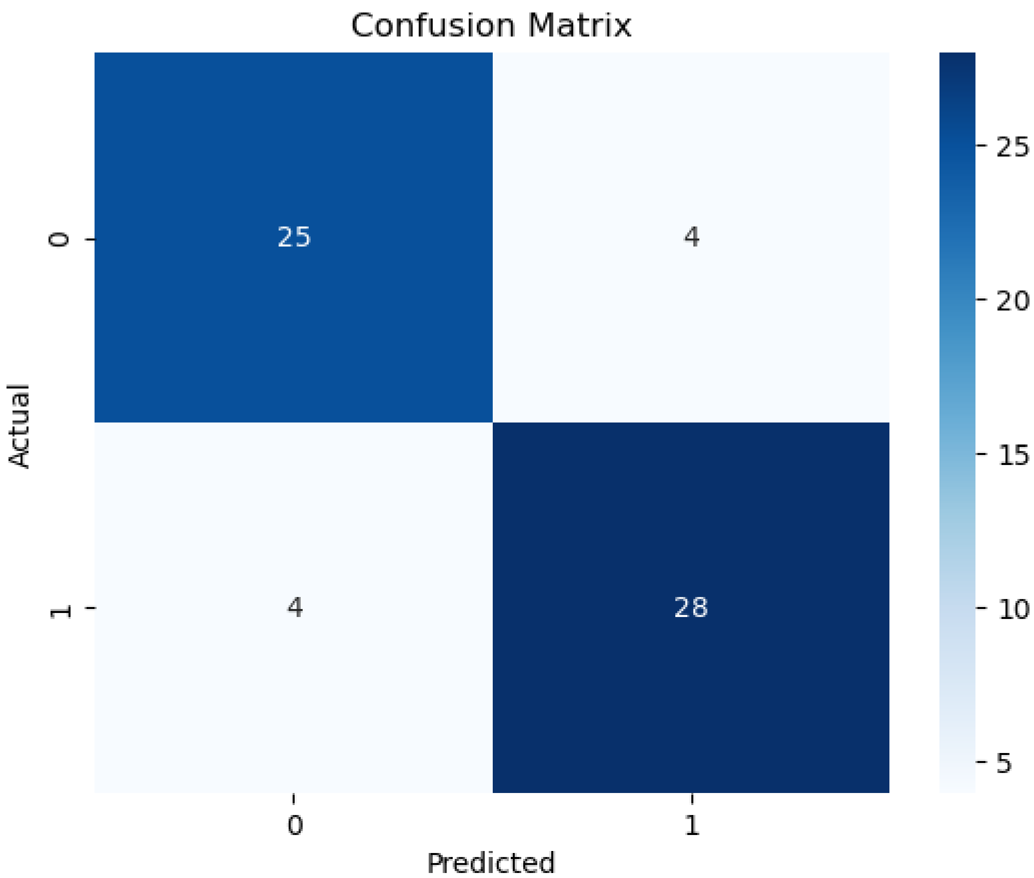
Out[33]:

	target
179	0
228	0
111	1
246	0
60	1
...	...
249	0
104	1
300	0
193	0
184	0

61 rows × 1 columns

In [34]:

```
# Create a confusion matrix

lcm = confusion_matrix(y_test, ly_pred)

# Visualize the confusion matrix

sns.heatmap(lcm, annot=True, cmap="Blues", fmt="g")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



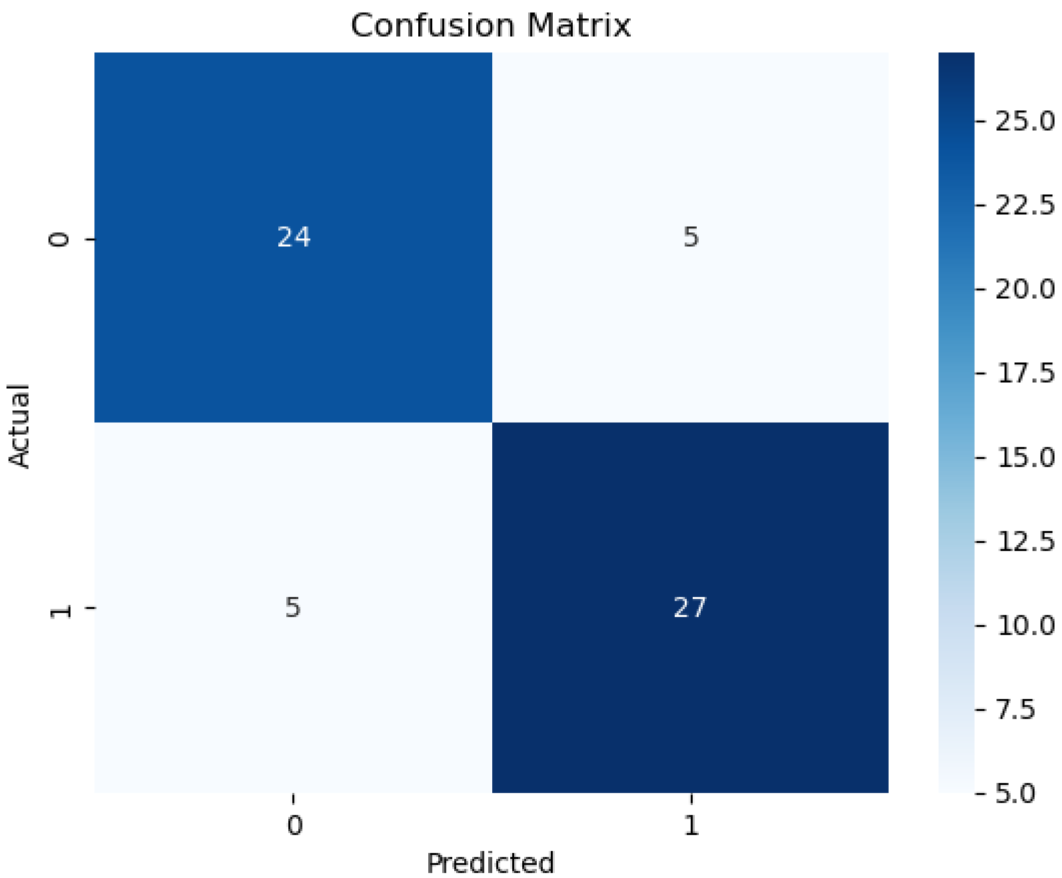
In [ ]:

In [35]:
```
# Model Building

# Random Forest Classifier

rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
rfy_pred = rfc.predict(X_test)
print("Logistic Regression")
print("Accuracy:", accuracy_score(y_test, rfy_pred))
print("Precision:", precision_score(y_test, rfy_pred))
print("Recall:", recall_score(y_test, rfy_pred))
print("F1-score:", f1_score(y_test, rfy_pred))
print("AUC-ROC:", roc_auc_score(y_test, rfy_pred))
```

```
Logistic Regression
Accuracy: 0.8360655737704918
Precision: 0.84375
Recall: 0.84375
F1-score: 0.84375
AUC-ROC: 0.8356681034482758
```

In [ ]:

In [36]:
```
# Create a confusion matrix

rcm = confusion_matrix(y_test, rfy_pred)

# Visualize the confusion matrix

sns.heatmap(rcm, annot=True, cmap="Blues", fmt="g")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [ ]:

In [51]:
```
# 8 Machine Learning Algorithms will be applied to the dataset

classifiers = [[XGBClassifier(), 'XGB Classifier'],
               [RandomForestClassifier(), 'Random forest'],
               [KNeighborsClassifier(), 'K-Nearest Neighbors'],
               [SGDClassifier(), 'SGD Classifier'],
               [SVC(), 'SVC'],
               [GaussianNB(), "Naive Bayes"],
               [DecisionTreeClassifier(random_state = 42), "Decision tree"],
               [LogisticRegression(), 'Logistics Regression']
              ]
```

In [38]:
```
acc_list = {}
precision_list = {}
recall_list = {}
roc_list = {}

for classifier in classifiers:
    model = classifier[0]
    model.fit(X_train, y_train)
    model_name = classifier[1]

    pred = model.predict(X_test)

    a_score = accuracy_score(y_test, pred)
    p_score = precision_score(y_test, pred)
    r_score = recall_score(y_test, pred)
    roc_score = roc_auc_score(y_test, pred)

    acc_list[model_name] = ([str(round(a_score*100, 2)) + '%'])
    precision_list[model_name] = ([str(round(p_score*100, 2)) + '%'])
    recall_list[model_name] = ([str(round(r_score*100, 2)) + '%'])
    roc_list[model_name] = ([str(round(roc_score*100, 2)) + '%'])
```

```
    if model_name != classifiers[-1][1]:
        print('')
```

In [39]:
```
print("Accuracy Score")
s1 = pd.DataFrame(acc_list)
s1.head()
```

```
Accuracy Score
```

Out[39]:

	XGB Classifier	Random forest	K-Nearest Neighbors	SGD Classifier	SVC	Naive Bayes	Decision tree	Logistics Regression
0	81.97%	85.25%	75.41%	73.77%	65.57%	86.89%	85.25%	86.89%

In [40]:
```
print("Precision Score")
s2 = pd.DataFrame(precision_list)
s2.head()
```

```
Precision Score
```

Out[40]:

	XGB Classifier	Random forest	K-Nearest Neighbors	SGD Classifier	SVC	Naive Bayes	Decision tree	Logistics Regression
0	86.21%	84.85%	79.31%	67.39%	65.71%	90.0%	92.59%	87.5%

In [41]:
```
print("Recall Score")
s3 = pd.DataFrame(recall_list)
s3.head()
```

```
Recall Score
```

Out[41]:

	XGB Classifier	Random forest	K-Nearest Neighbors	SGD Classifier	SVC	Naive Bayes	Decision tree	Logistics Regression
0	78.12%	87.5%	71.88%	96.88%	71.88%	84.38%	78.12%	87.5%

In [42]:
```
print("ROC Score")
s4 = pd.DataFrame(roc_list)
s4.head()
```

```
ROC Score
```

Out[42]:

	XGB Classifier	Random forest	K-Nearest Neighbors	SGD Classifier	SVC	Naive Bayes	Decision tree	Logistics Regression
0	82.17%	85.13%	75.59%	72.58%	65.25%	87.02%	85.61%	86.85%

In Conclusion

From the analysis above, Logistic Regression performed better than Random Forest Classifier with accurancy of 86.89% and precision of 87.5%.

In [ ]: