MyProject

In [73]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from datetime import datetime

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import
GridSearchCV,train_test_split,cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc
import warnings
warnings.filterwarnings('ignore')
print(os.listdir("F:\Taran\Database"))
```

```
['Cleveland_data.csv']
```

In [54]:

```
data = pd.read_csv('F:\Taran\Database\Cleveland_data.csv')
# Now, our data is loaded. We're writing the following snippet to see the
loaded data.
# The purpose here is to see the top five of the loaded data.
print('Data First 5 Rows Show\n')
data.head()
```

```
Data First 5 Rows Show
```

In [13]:

```
print('Data Last 5 Rows Show\n')
data.tail()
```

```
Data Last 5 Rows Show
```

In [14]:

```
print('Data Show Describe\n')
data.describe()
```

```
Data Show Describe
```

In [55]:

```
print('Data Show Info\n')
data.info()
```

```
Data Show 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   restceg   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
```

In [68]:

```
data.sample(frac=0.01)
```

Out[68]:

	Age	Sex	cp	trestbps	chol	fbs	restceg	thalach	exang	oldpeak	slope	ca	thal	target
222	39	0	3	94	199	0	0	179	0	0.0	1	0	3	0
118	63	1	4	130	330	1	2	132	1	1.8	1	3	7	3
277	39	0	3	138	220	0	0	152	0	0.0	2	0	3	0

In [81]:

```
# sample; random rows in dataset
data.sample(5)
data = data.rename(columns={'age': 'Age', 'sex': 'Sex', 'cp': 'Cp',
'trestbps': 'Trestbps', 'chol': 'Chol',
                            'fbs': 'Fbs', 'restecg': 'Restecg', 'thalach':
'Thalach', 'exang': 'Exang',
                            'oldpeak': 'Oldpeak', 'slope': 'Slope', 'ca':
'Ca', 'thal': 'Thal', 'target': 'Target'})
# New show columns
pd.set_option('display.max_columns',None)
data.head(0)
```

Out[81]:

Age	Sex	Cp	Trestbps	Chol	Fbs	restceg	Thalach	Exang	Oldpeak	Slope	Ca	Thal	Target

In [89]:

```
# And, how many rows and columns are there for all data?
print('Data Shape Show\n')
pd.set_option('display.max_rows',None)
data
```

```
Data Shape Show
```

In [90]:

```
# Now,I will check null on all data and If data has null, I will sum of null
data's.
# In this way, how many missing data is in the data.
print('Data Sum of Null Values \n')
data.isnull().sum()
```

```
Data Sum of Null Values
```

In [91]:

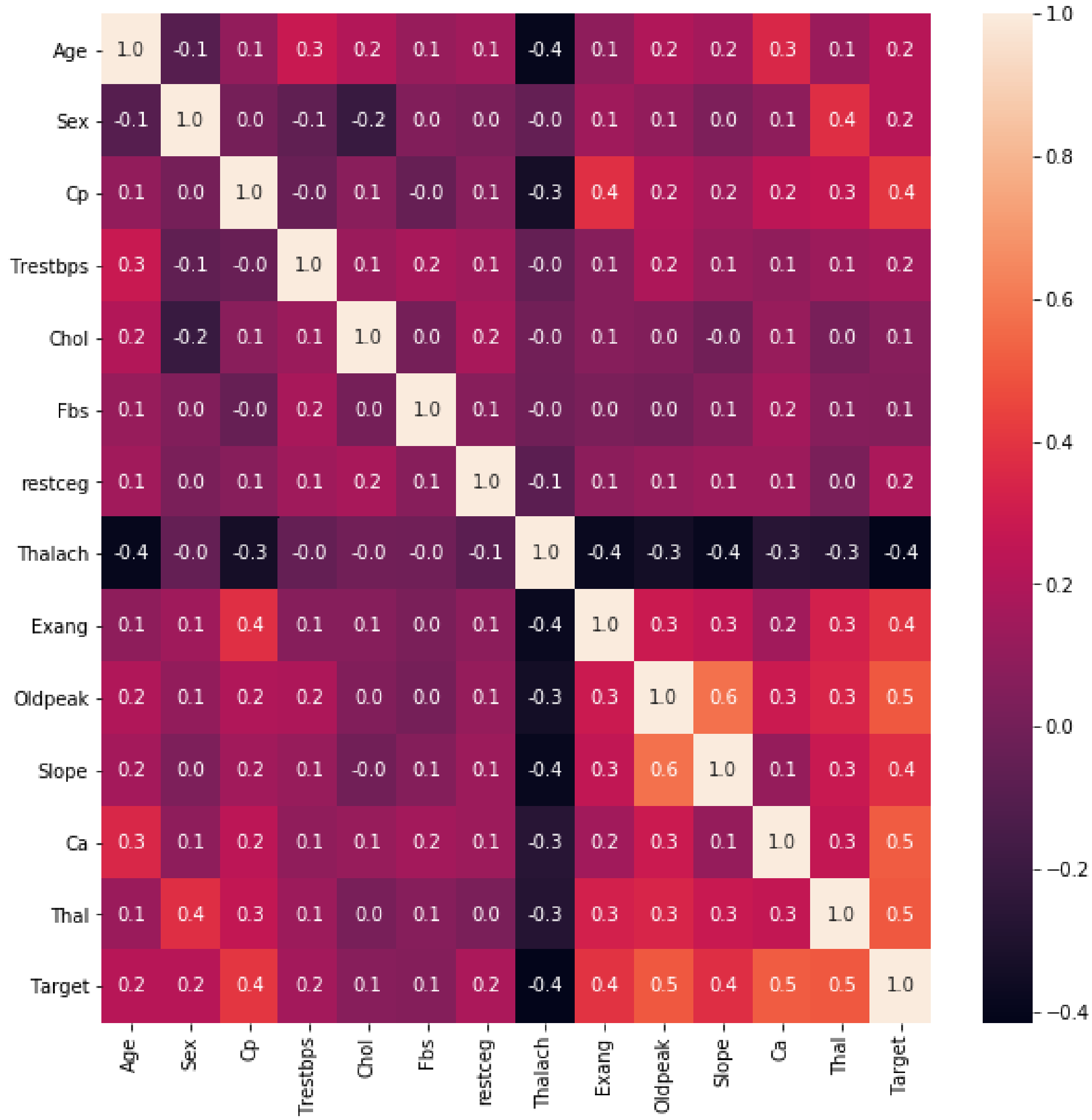
```
# all rows control for null values
data.isnull().values.any()
```

Out[91]:

```
False
```

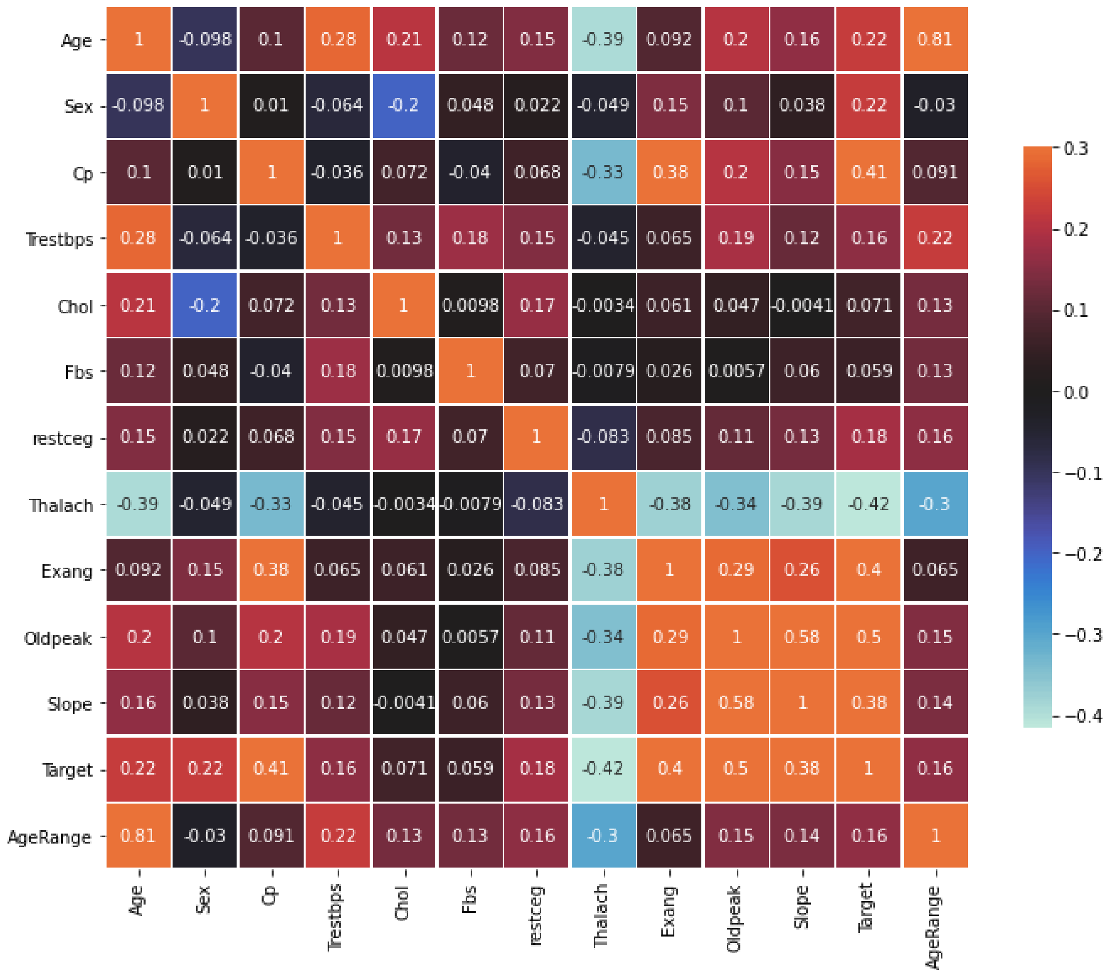
In [92]:

```
plt.figure(figsize=(10, 10))
sns.heatmap(data.corr(), annot=True, fmt='.1f')
plt.show()
```



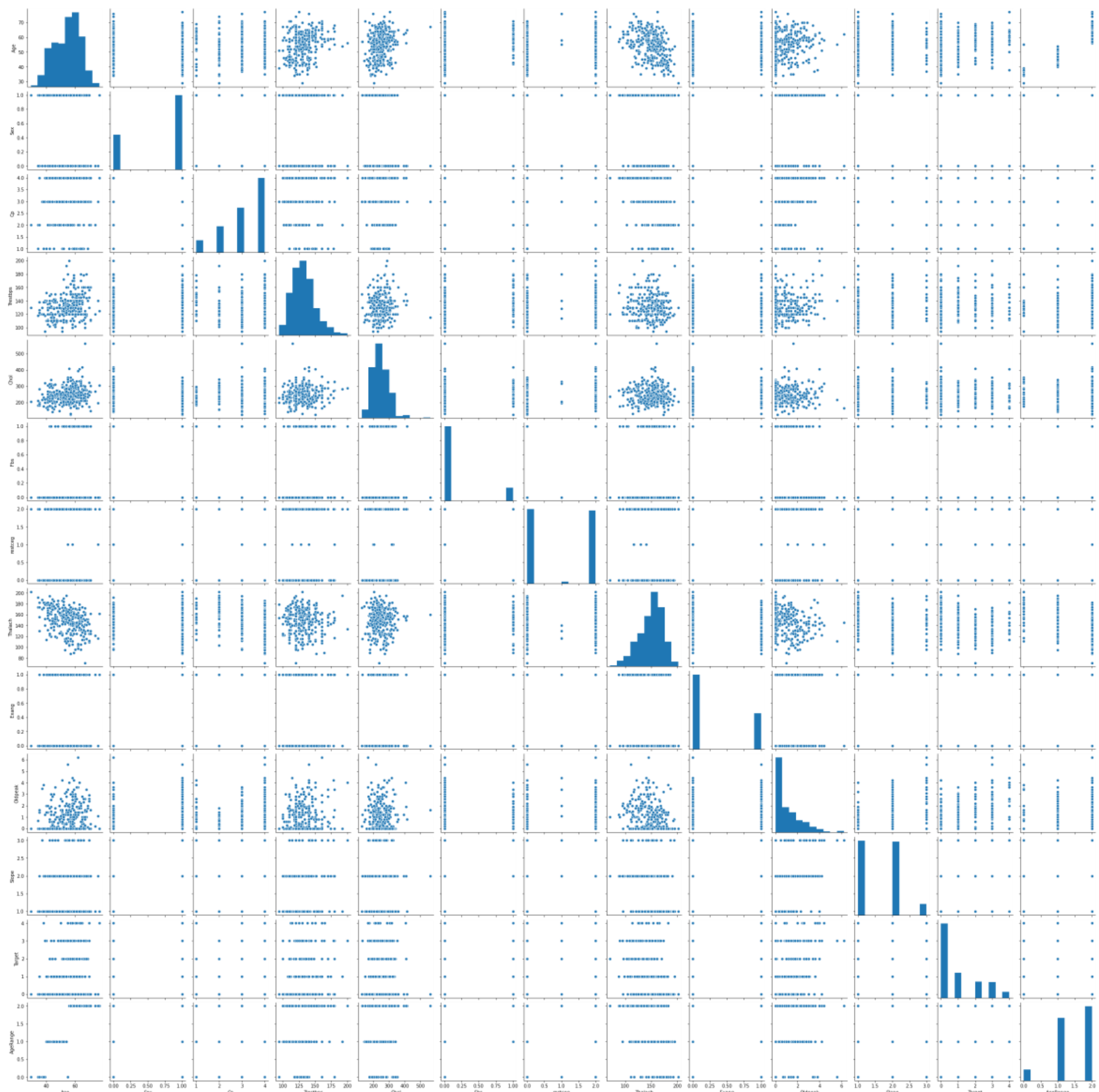
In [27]:

```
plt.figure(figsize=(10, 10))
sns.heatmap(data.corr(), vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
plt.tight_layout()
plt.show()
```



In [29]:

```
sns.pairplot(data)
plt.show()
```



In [33]:

```
data.Age.value_counts()[:10]
# data age show value counts for age least 10
```

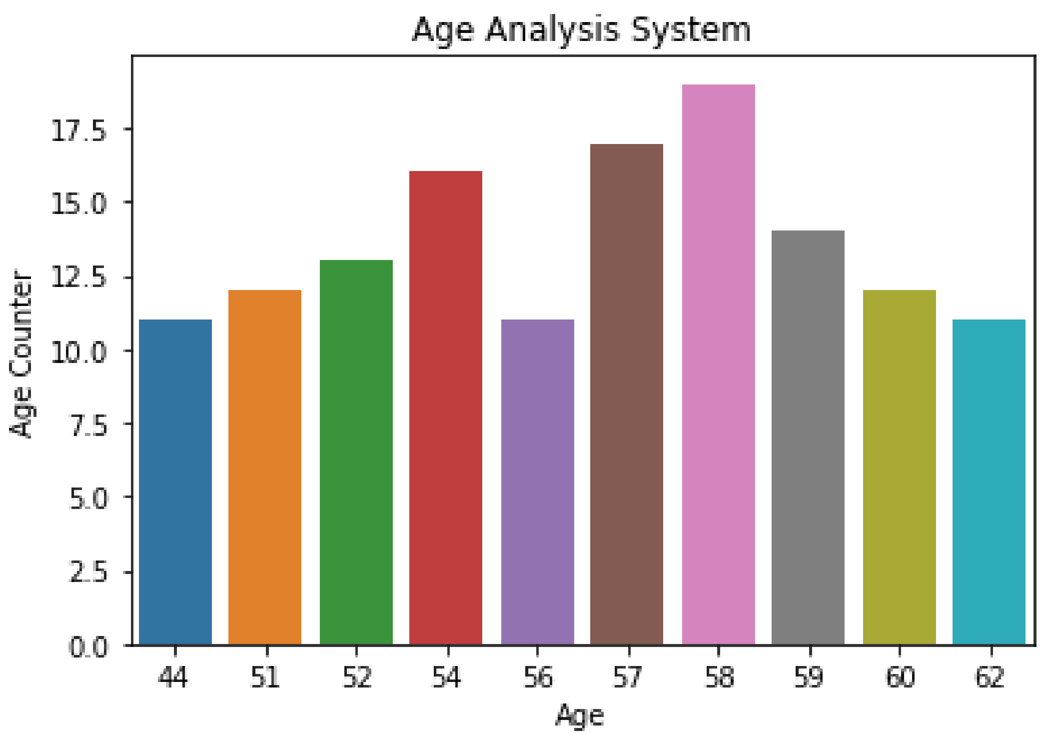
Out[33]:

```
58    19
57    17
54    16
59    14
52    13
51    12
60    12
62    11
44    11
```

```
56    11
Name: Age, dtype: int64
```

In [34]:

```
sns.barplot(x=data.Age.value_counts()[:10].index,
y=data.Age.value_counts()[:10].values)
plt.xlabel('Age')
plt.ylabel('Age Counter')
plt.title('Age Analysis System')
plt.show()
```



In [35]:

```
# firstly find min and max ages
minAge = min(data.Age)
maxAge = max(data.Age)
meanAge = data.Age.mean()
print('Min Age :', minAge)
print('Max Age :', maxAge)
print('Mean Age :', meanAge)
```

```
Min Age : 29
Max Age : 77
Mean Age : 54.43894389438944
```

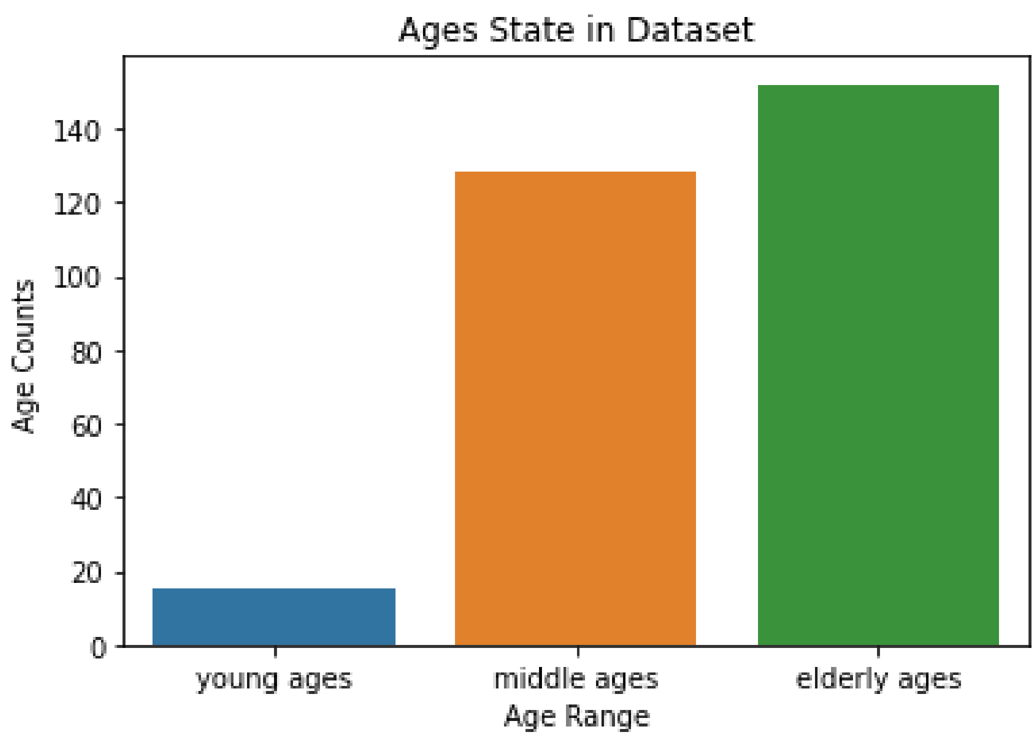
In [36]:

```
young_ages = data[(data.Age >= 29) & (data.Age < 40)]
middle_ages = data[(data.Age >= 40) & (data.Age < 55)]
elderly_ages = data[(data.Age > 55)]
print('Young Ages :', len(young_ages))
print('Middle Ages :', len(middle_ages))
print('Elderly Ages :', len(elderly_ages))
```

```
Young Ages : 15
Middle Ages : 128
Elderly Ages : 152
```

In [37]:

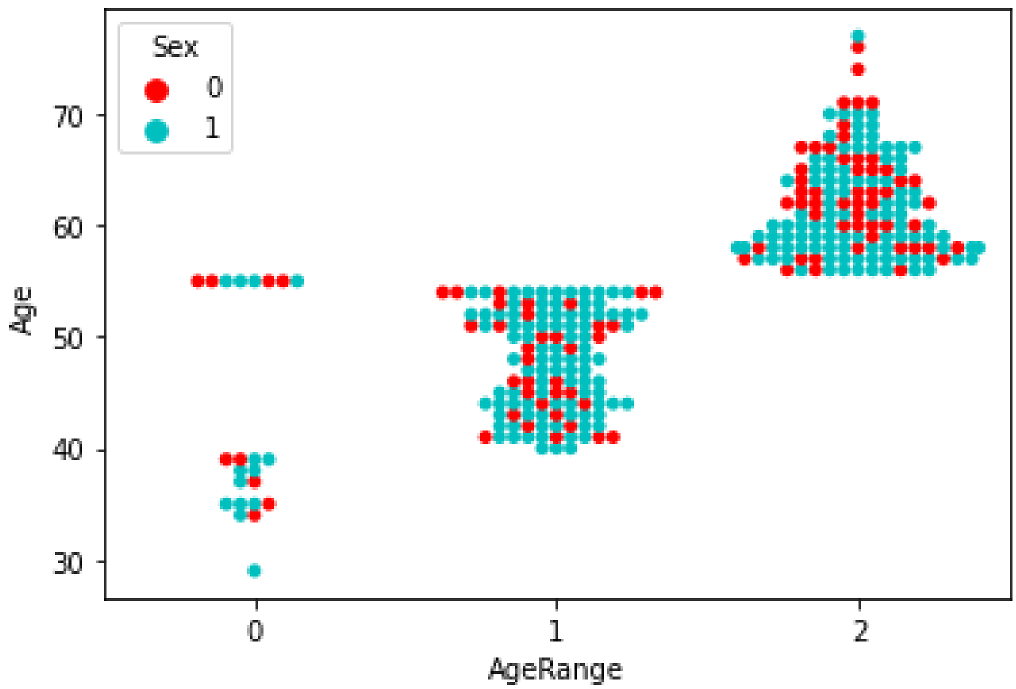
```
sns.barplot(x=['young ages', 'middle ages', 'elderly ages'],
y=[len(young_ages), len(middle_ages), len(elderly_ages)])
plt.xlabel('Age Range')
plt.ylabel('Age Counts')
plt.title('Ages State in Dataset')
plt.show()
```



In [39]:

```
data['AgeRange'] = 0
youngAge_index = data[(data.Age >= 29) & (data.Age < 40)].index
middleAge_index = data[(data.Age >= 40) & (data.Age < 55)].index
elderlyAge_index = data[(data.Age > 55)].index
for index in elderlyAge_index:
    data.loc[index, 'AgeRange'] = 2
```

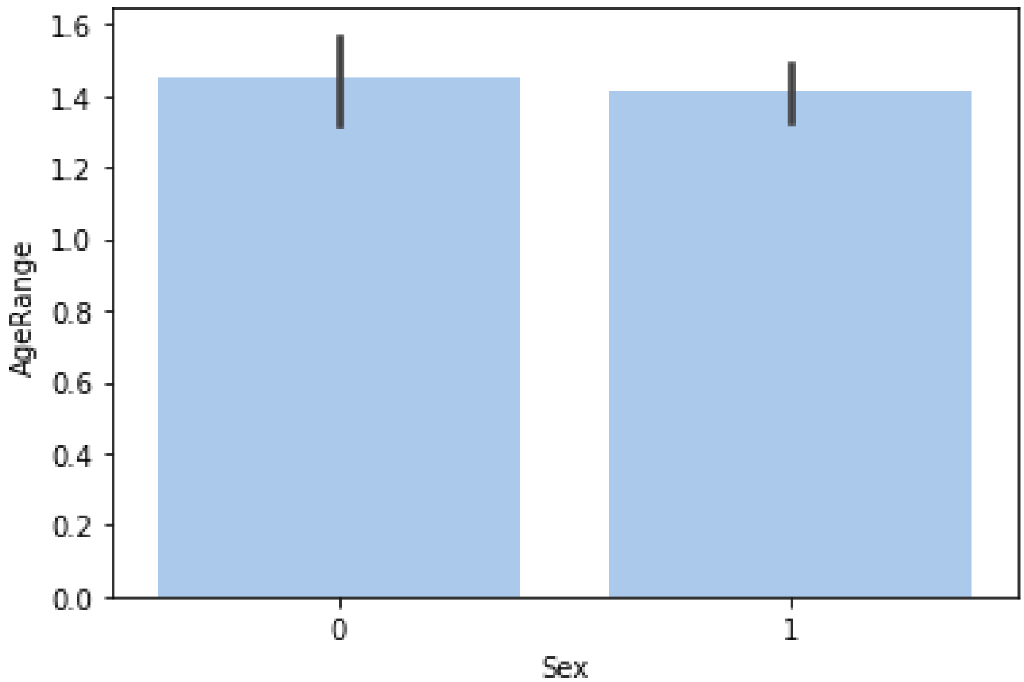
```
for index in middleAge_index:
    data.loc[index, 'AgeRange'] = 1

for index in youngAge_index:
    data.loc[index, 'AgeRange'] = 0

# Draw a categorical scatter-plot to show each observation
sns.swarmplot(x="AgeRange", y="Age", hue='Sex',
              palette=["r", "c", "y"], data=data)
plt.show()
```



In [94]:

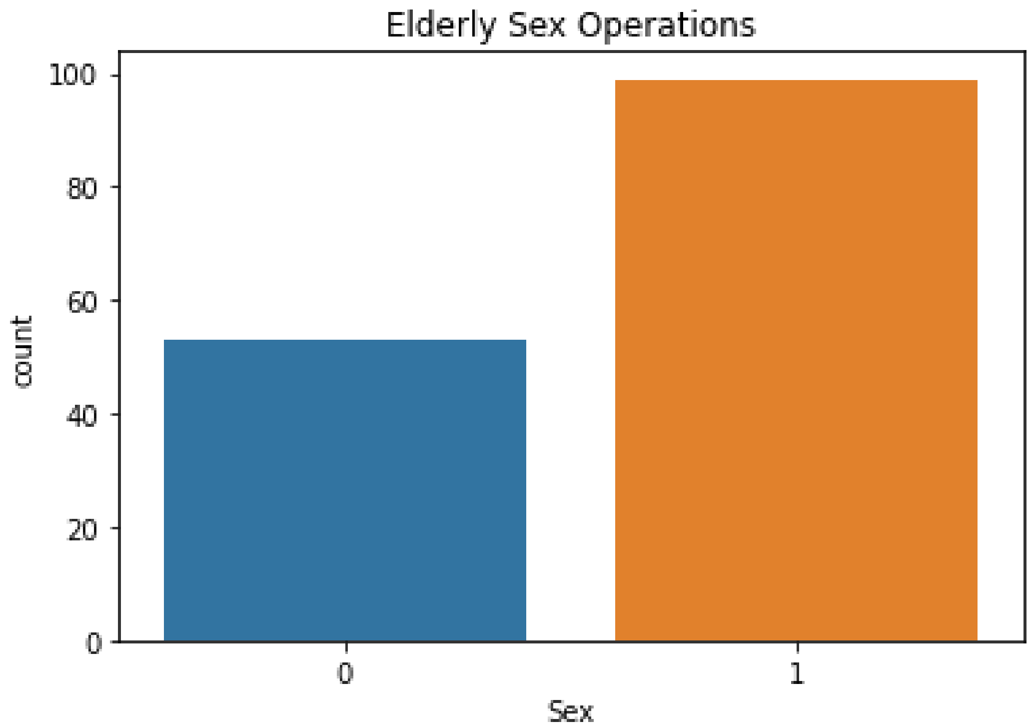
```
data['AgeRange'] = 0
youngAge_index = data[(data.Age >= 29) & (data.Age < 40)].index
middleAge_index = data[(data.Age >= 40) & (data.Age < 55)].index
elderlyAge_index = data[(data.Age > 55)].index
for index in elderlyAge_index:
    data.loc[index, 'AgeRange'] = 2

for index in middleAge_index:
    data.loc[index, 'AgeRange'] = 1

for index in youngAge_index:
    data.loc[index, 'AgeRange'] = 0

# Plot the total crashes
```

```
sns.set_color_codes("pastel")
sns.barplot(y="AgeRange", x="Sex", data=data, label="Total", color="b")
plt.show()
```



In [41]:

```
sns.countplot(elderly_ages.Sex)
plt.title("Elderly Sex Operations")
plt.show()
```



In [42]:

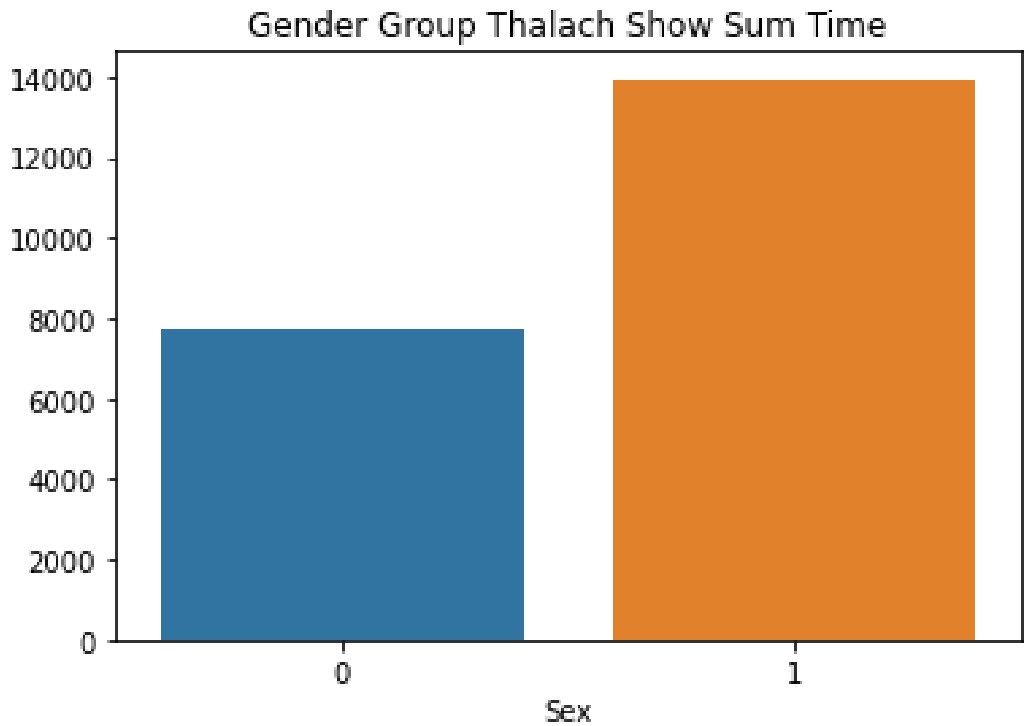
```
elderly_ages.groupby(elderly_ages['Sex'])['Thalach'].agg('sum')
```

Out[42]:

```
Sex
0     7739
1    13948
Name: Thalach, dtype: int64
```

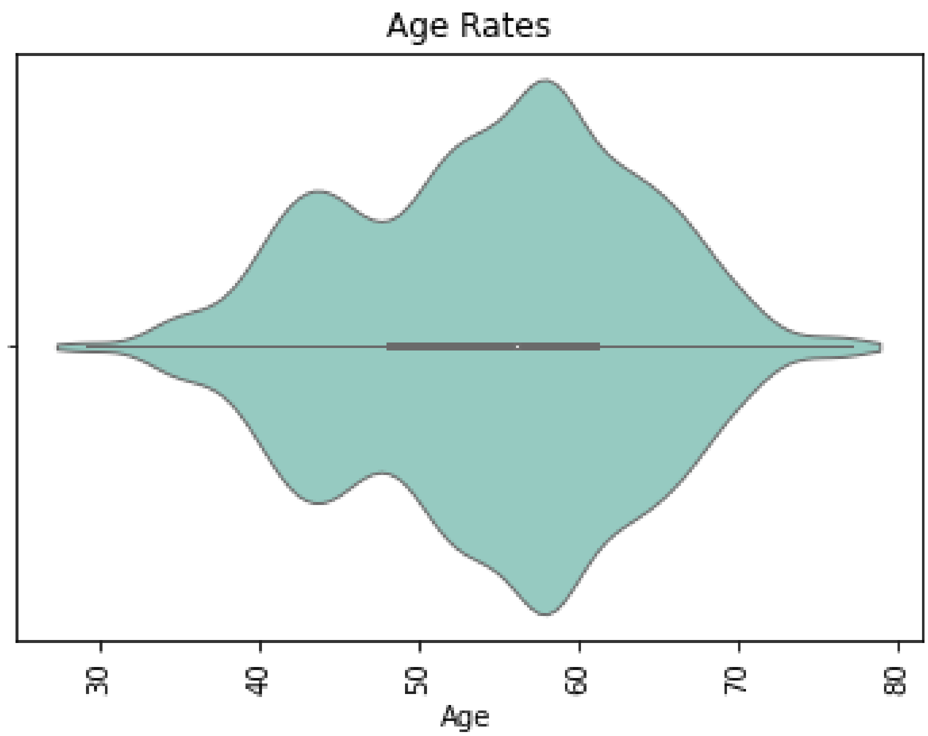
In [43]:

```
sns.barplot(x=elderly_ages.groupby(elderly_ages['Sex'])['Thalach'].agg('sum')
.index,

y=elderly_ages.groupby(elderly_ages['Sex'])['Thalach'].agg('sum').values)
plt.title("Gender Group Thalach Show Sum Time")
plt.show()
```



In [44]:

```
sns.violinplot(data.Age, palette="Set3", bw=.2, cut=1, linewidth=1)
plt.xticks(rotation=90)
plt.title("Age Rates")
plt.show()
```



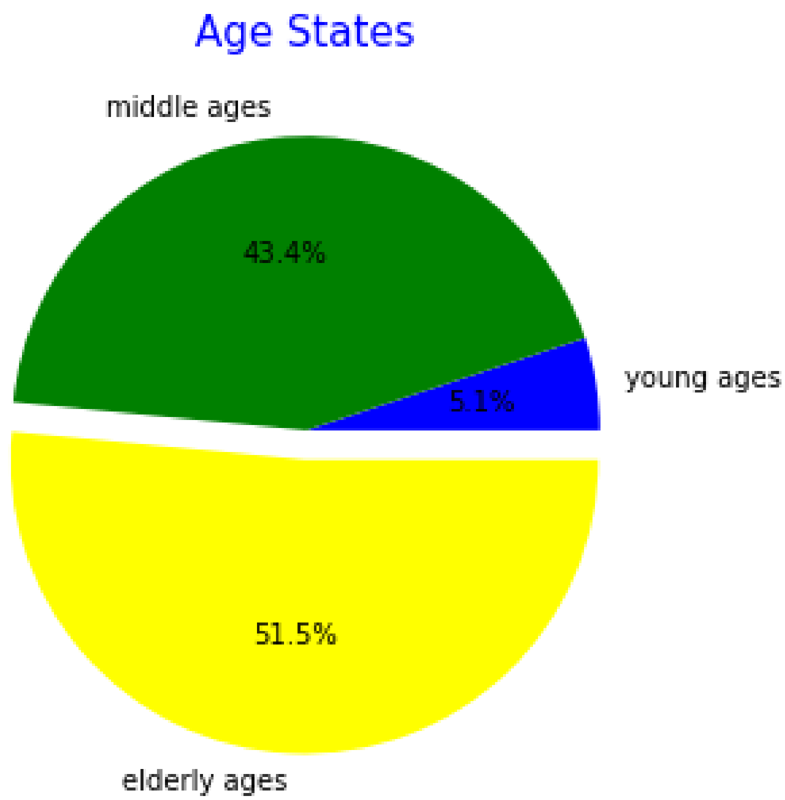
In [45]:

```
plt.figure(figsize=(15, 7))
sns.violinplot(x=data.Age, y=data.Target)
plt.xticks(rotation=90)
plt.legend()
plt.title("Age & Target System")
plt.show()
```

```
No handles with labels found to put in legend.
```

In [46]:

```
colors = ['blue', 'green', 'yellow']
explode = [0, 0, 0.1]
plt.figure(figsize=(5, 5))
# plt.pie([target_0_agerang_0,target_1_agerang_0], explode=explode,
labels=['Target 0 Age Range 0',
# 'Target 1 Age Range 0'], colors=colors, autopct='%1.1f%%')
plt.pie([len(young_ages), len(middle_ages), len(elderly_ages)],
labels=['young ages', 'middle ages', 'elderly ages'],
        explode=explode, colors=colors, autopct='%1.1f%%')
plt.title('Age States', color='blue', fontsize=15)
plt.show()
```



In [47]:

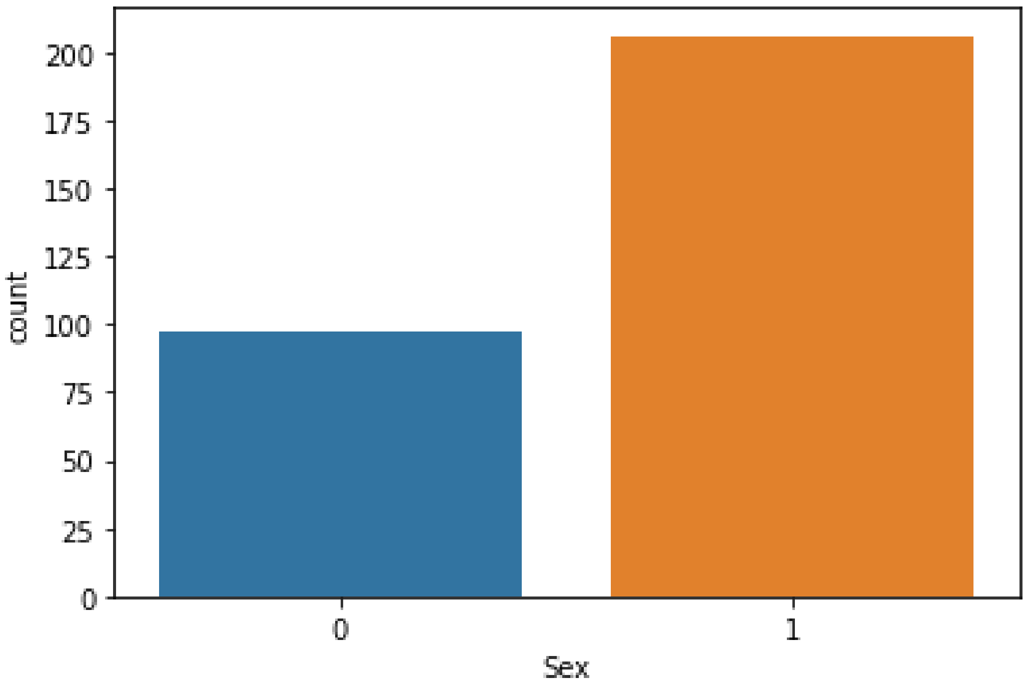
```
data.Sex.value_counts()
```

Out[47]:

```
1    206
0     97
Name: Sex, dtype: int64
```

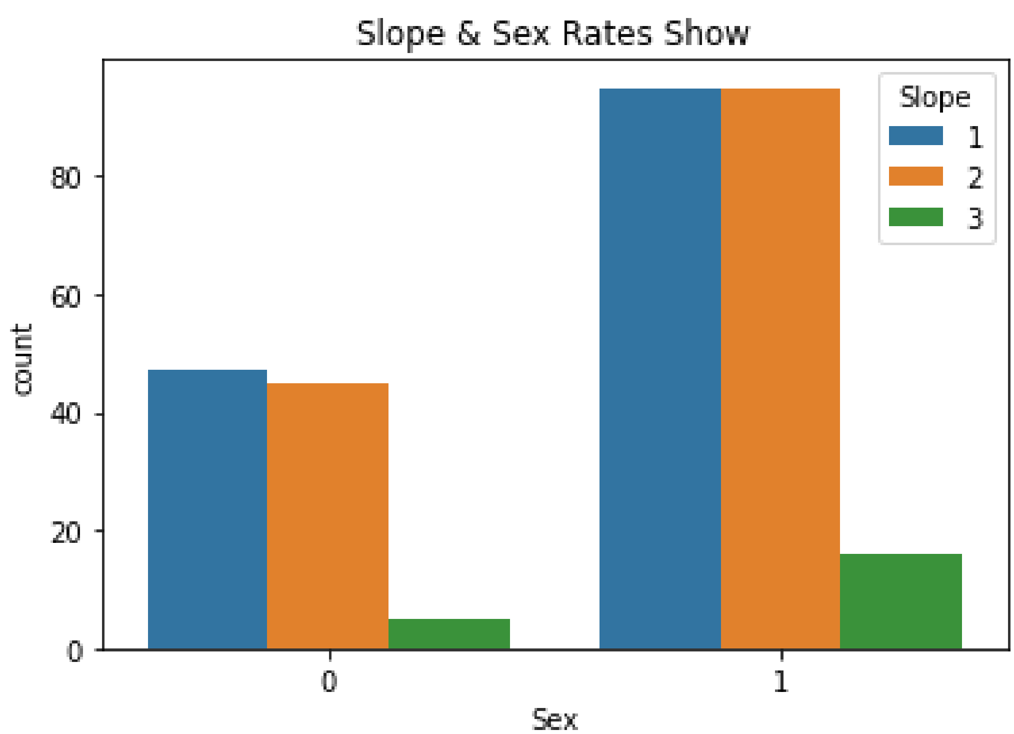
In [48]:

```
# Sex (1 = male; 0 = female)
sns.countplot(data.Sex)
plt.show()
```



In [49]:

```
sns.countplot(data.Sex, hue=data.Slope)
plt.title('Slope & Sex Rates Show')
plt.show()
```



In [50]:

```
total_genders_count = len(data.Sex)
male_count = len(data[data['Sex'] == 1])
female_count = len(data[data['Sex'] == 0])
print('Total Genders :', total_genders_count)
print('Male Count    :', male_count)
print('Female Count  :', female_count)
```

```
Total Genders : 303
Male Count    : 206
Female Count  : 97
```

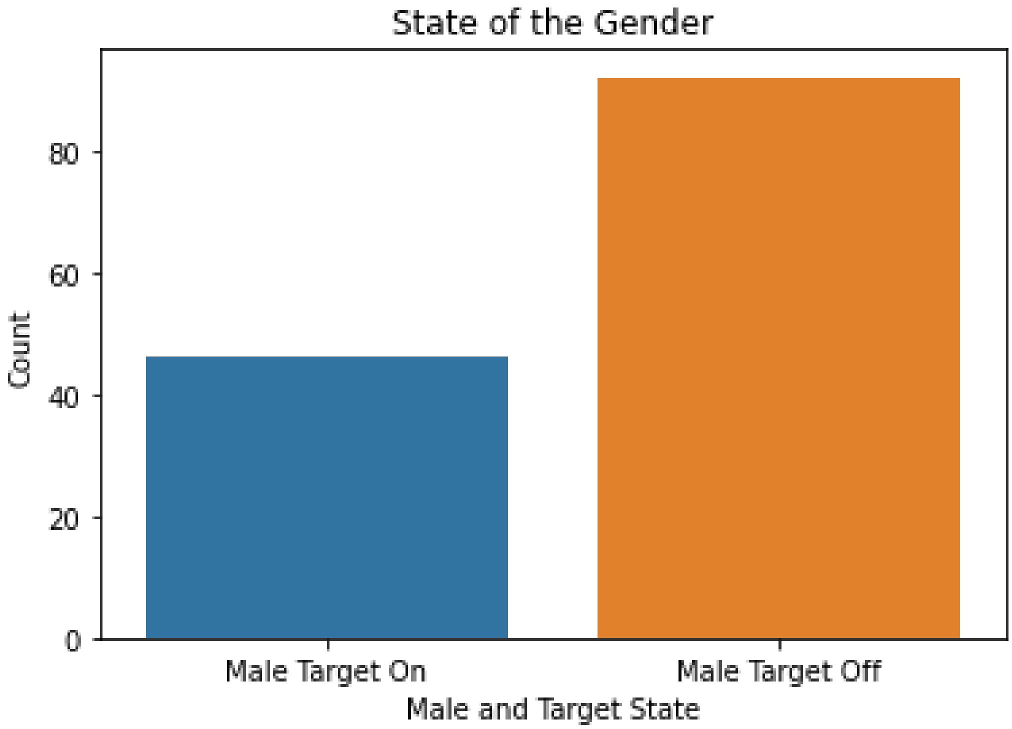
In [51]:

```
# Percentage ratios
print("Male State: {:.2f}%".format((male_count / total_genders_count * 100)))
print("Female State: {:.2f}%".format((female_count / total_genders_count *
100)))
```

```
Male State: 67.99%
Female State: 32.01%
```

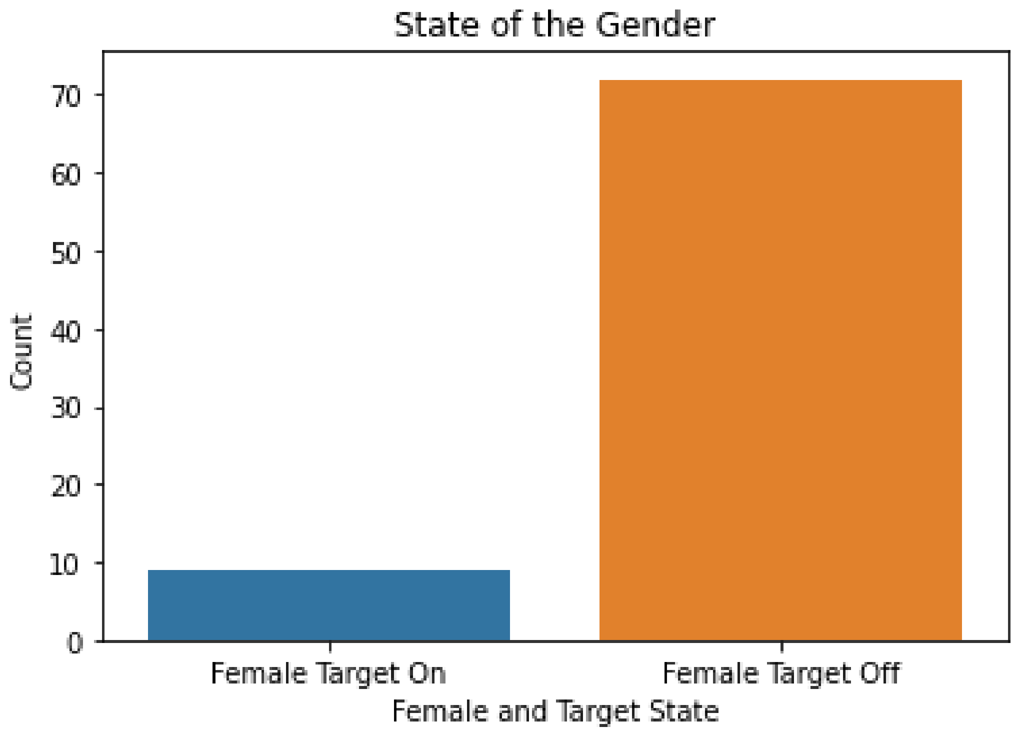
In [52]:

```
# Male State & target 1 & 0
male_andtarget_on = len(data[(data.Sex == 1) & (data['Target'] == 1)])
male_andtarget_off = len(data[(data.Sex == 1) & (data['Target'] == 0)])
sns.barplot(x=['Male Target On', 'Male Target Off'], y=[male_andtarget_on,
male_andtarget_off])
plt.xlabel('Male and Target State')
plt.ylabel('Count')
plt.title('State of the Gender')
plt.show()
```



In [53]:

```
# Female State & target 1 & 0
female_andtarget_on = len(data[(data.Sex == 0) & (data['Target'] == 1)])
female_andtarget_off = len(data[(data.Sex == 0) & (data['Target'] == 0)])
sns.barplot(x=['Female Target On', 'Female Target Off'],
y=[female_andtarget_on, female_andtarget_off])
plt.xlabel('Female and Target State')
plt.ylabel('Count')
plt.title('State of the Gender')
plt.show()
```



In [54]:

```
# Plot miles per gallon against horsepower with other semantics
sns.relplot(x="Trestbps", y="Age",
            sizes=(40, 400), alpha=.5, palette="muted",
            height=6, data=data)
```

Out[54]:

```
<seaborn.axisgrid.FacetGrid at 0x135f970>
```

In [55]:

```
data.head()
```

Out[55]:

	Age	Sex	Cp	Trestbps	Chol	Fbs	restceg	Thalach	Exang	Oldpeak	Slope	Ca	Thal	Target	AgeRange
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0	2
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1	2
3	37	1	3	130	25	0	0	187	0	3.5	3	0	3	0	0

4	41	0	2	130	0 20 4	0	2	172	0	1.4	1	0	3	0	1

In [56]:

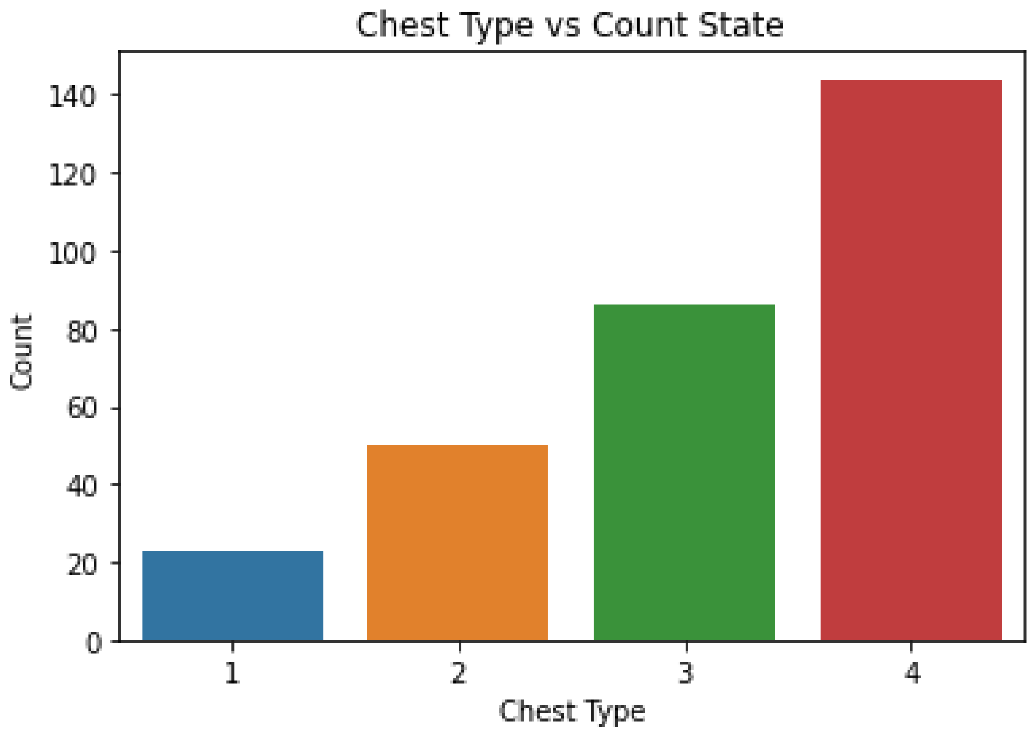
```
# As seen, there are 4 types of chest pain.
data.Cp.value_counts()
```

Out[56]:

```
4    144
3     86
2     50
1     23
Name: Cp, dtype: int64
```

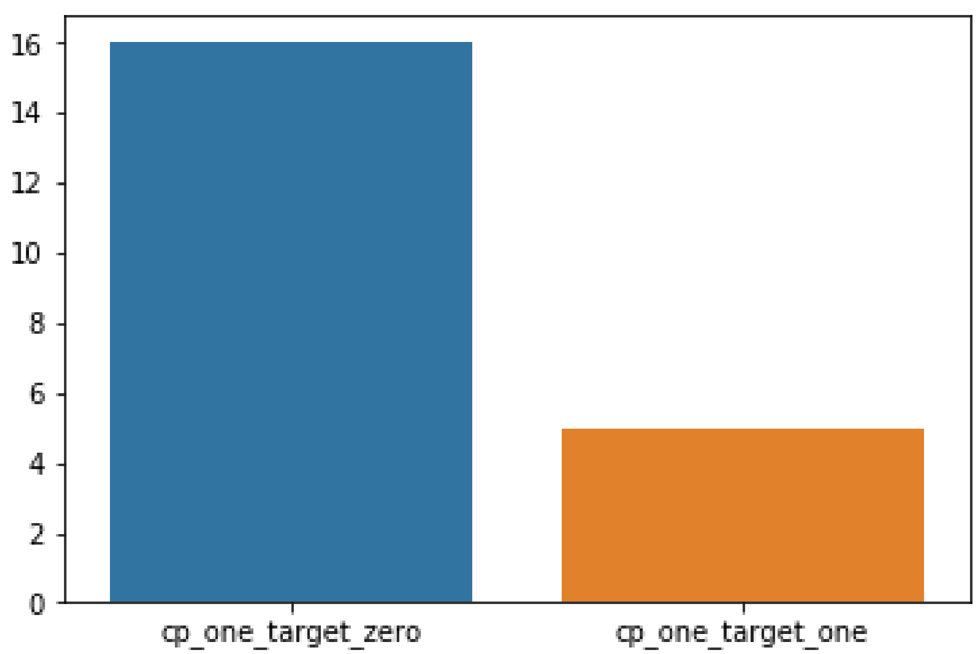
In [57]:

```
sns.countplot(data.Cp)
plt.xlabel('Chest Type')
plt.ylabel('Count')
plt.title('Chest Type vs Count State')
plt.show()

# 0 status at least
# 1 condition slightly distressed
# 2 condition medium problem
# 3 condition too bad
```



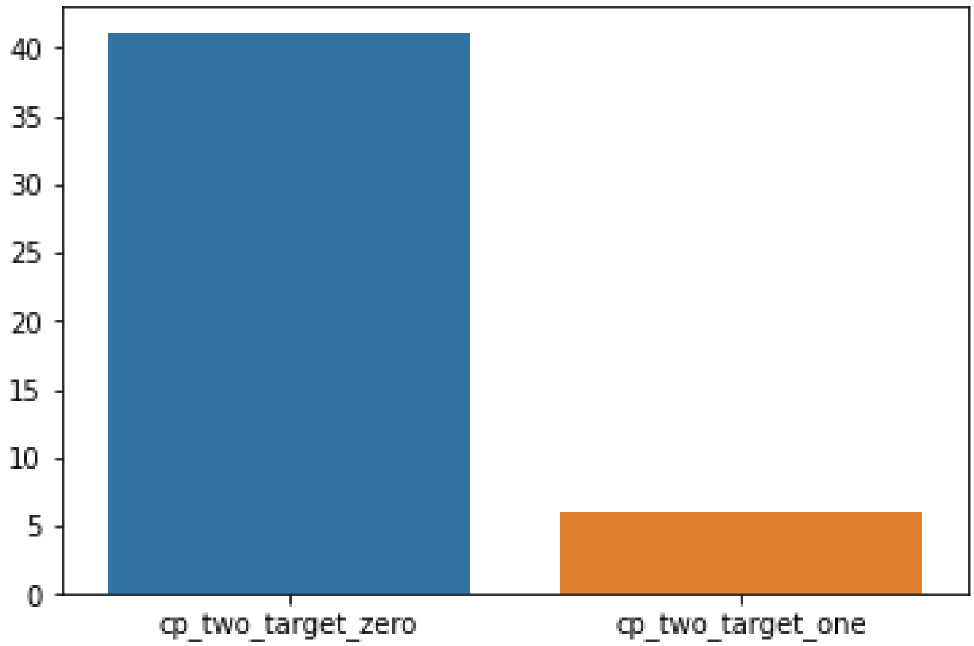
In [109]:

```
cp_one_target_zero = len(data[(data.Cp == 1) & (data.Target == 0)])
cp_one_target_one = len(data[(data.Cp == 1) & (data.Target == 1)])
sns.barplot(x=['cp_one_target_zero', 'cp_one_target_one'],
y=[cp_one_target_zero, cp_one_target_one])
plt.show()
```



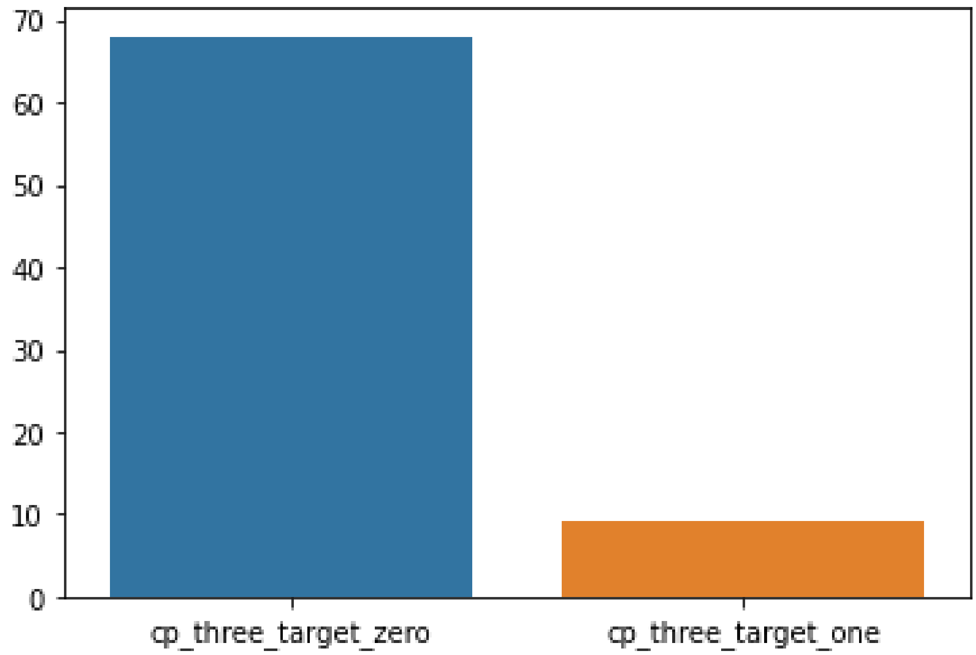
In [62]:

```
cp_two_target_zero = len(data[(data.Cp == 2) & (data.Target == 0)])
cp_two_target_one = len(data[(data.Cp == 2) & (data.Target == 1)])
sns.barplot(x=['cp_two_target_zero', 'cp_two_target_one'],
y=[cp_two_target_zero, cp_two_target_one])
plt.show()
```



In [63]:

```
cp_three_target_zero = len(data[(data.Cp == 3) & (data.Target == 0)])
cp_three_target_one = len(data[(data.Cp == 3) & (data.Target == 1)])
sns.barplot(x=['cp_three_target_zero', 'cp_three_target_one'],
y=[cp_three_target_zero, cp_three_target_one])
plt.show()
```



In [64]:

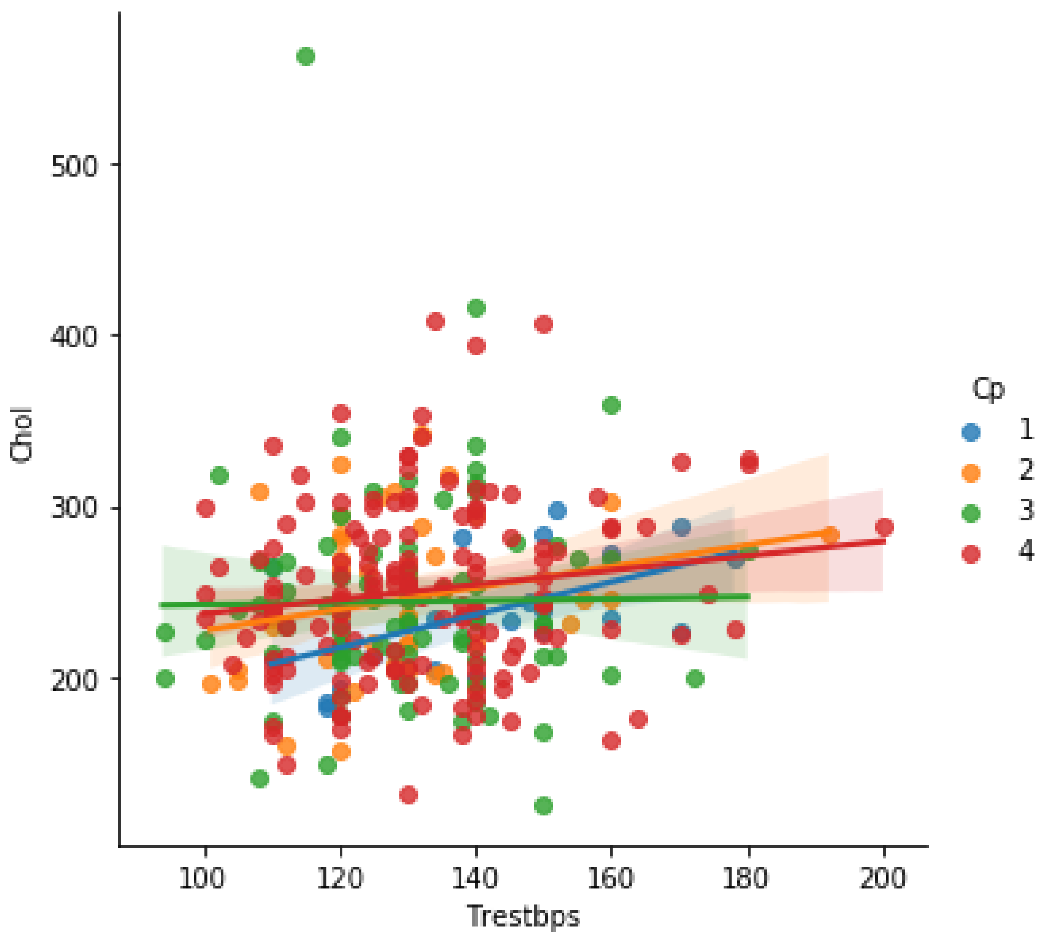
```
data.head(1)
```

Out[64]:

	Age	Sex	Cp	Trestbps	Chol	Fbs	restceg	Thalach	Exang	Oldpeak	Slope	Ca	Thal	Target	AgeRange
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0	2

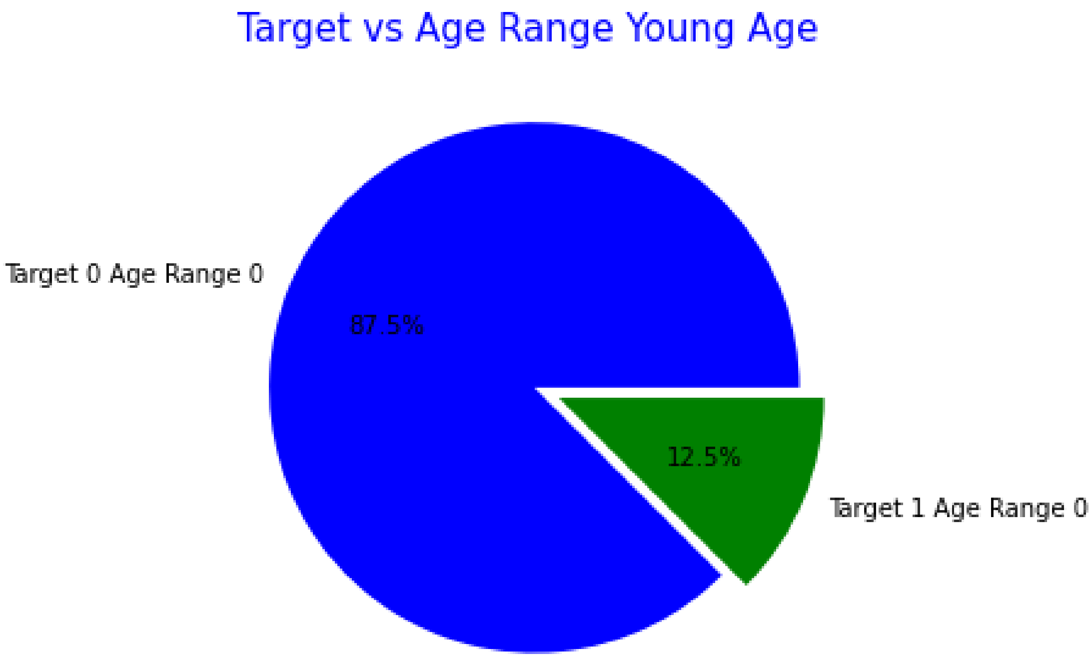
In [114]:

```
# Show the results of a linear regression within each dataset
sns.lmplot(x="Trestbps", y="Chol", data=data, hue="Cp")
plt.show()
```



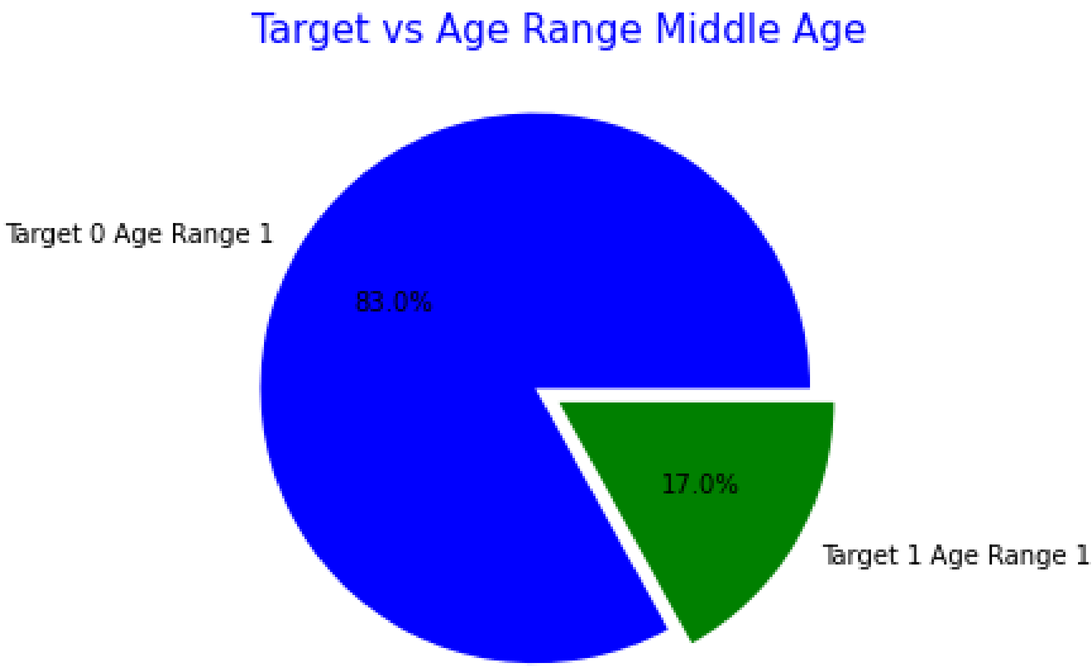
In [66]:

```
target_0_agerang_0 = len(data[(data.Target == 0) & (data.AgeRange == 0)])
target_1_agerang_0 = len(data[(data.Target == 1) & (data.AgeRange == 0)])
colors = ['blue', 'green']
explode = [0, 0.1]
plt.figure(figsize=(5, 5))
plt.pie([target_0_agerang_0, target_1_agerang_0], explode=explode,
labels=['Target 0 Age Range 0',

'Target 1 Age Range 0'],
        colors=colors, autopct='%1.1f%%')
plt.title('Target vs Age Range Young Age ', color='blue', fontsize=15)
plt.show()
```



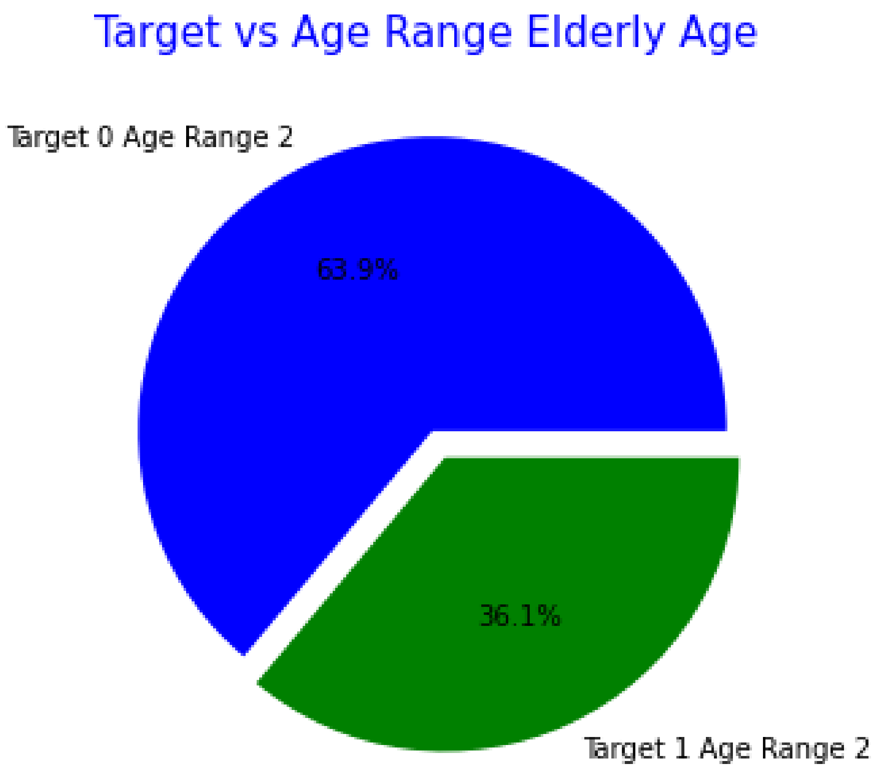
In [67]:

```
target_0_agerang_1 = len(data[(data.Target == 0) & (data.AgeRange == 1)])
target_1_agerang_1 = len(data[(data.Target == 1) & (data.AgeRange == 1)])
colors = ['blue', 'green']
explode = [0.1, 0]
plt.figure(figsize=(5, 5))
plt.pie([target_0_agerang_1, target_1_agerang_1], explode=explode,
labels=['Target 0 Age Range 1',

'Target 1 Age Range 1'],
        colors=colors, autopct='%1.1f%%')
plt.title('Target vs Age Range Middle Age', color='blue', fontsize=15)
plt.show()
```



In [68]:

```
target_0_agerang_2 = len(data[(data.Target == 0) & (data.AgeRange == 2)])
target_1_agerang_2 = len(data[(data.Target == 1) & (data.AgeRange == 2)])
colors = ['blue', 'green']
explode = [0, 0.1]
plt.figure(figsize=(5, 5))
plt.pie([target_0_agerang_2, target_1_agerang_2], explode=explode,
labels=['Target 0 Age Range 2',

'Target 1 Age Range 2'],
        colors=colors, autopct='%1.1f%%')
plt.title('Target vs Age Range Elderly Age ', color='blue', fontsize=15)
plt.show()
```



In [112]:

```
data.Thalach.value_counts()[:20]
# First show 20 rows
```

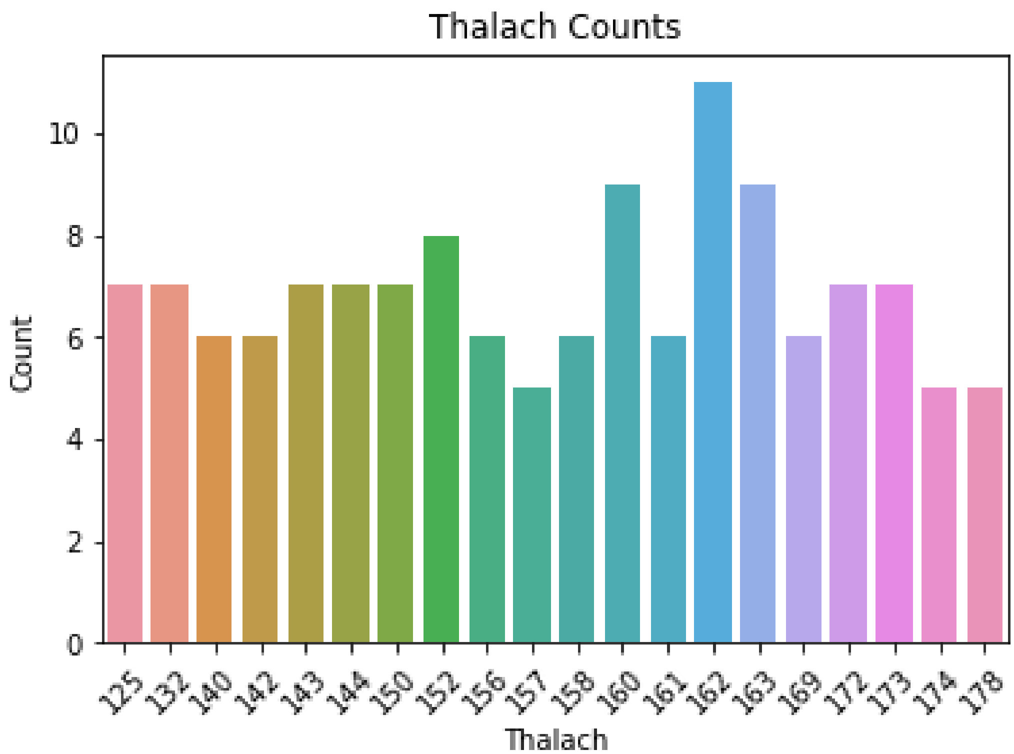
Out[112]:

```
162    11
160     9
163     9
152     8
173     7
125     7
132     7
150     7
143     7
144     7
172     7
156     6
161     6
169     6
158     6
142     6
140     6
174     5
157     5
```

```
178     5
Name: Thalach, dtype: int64
```

In [70]:

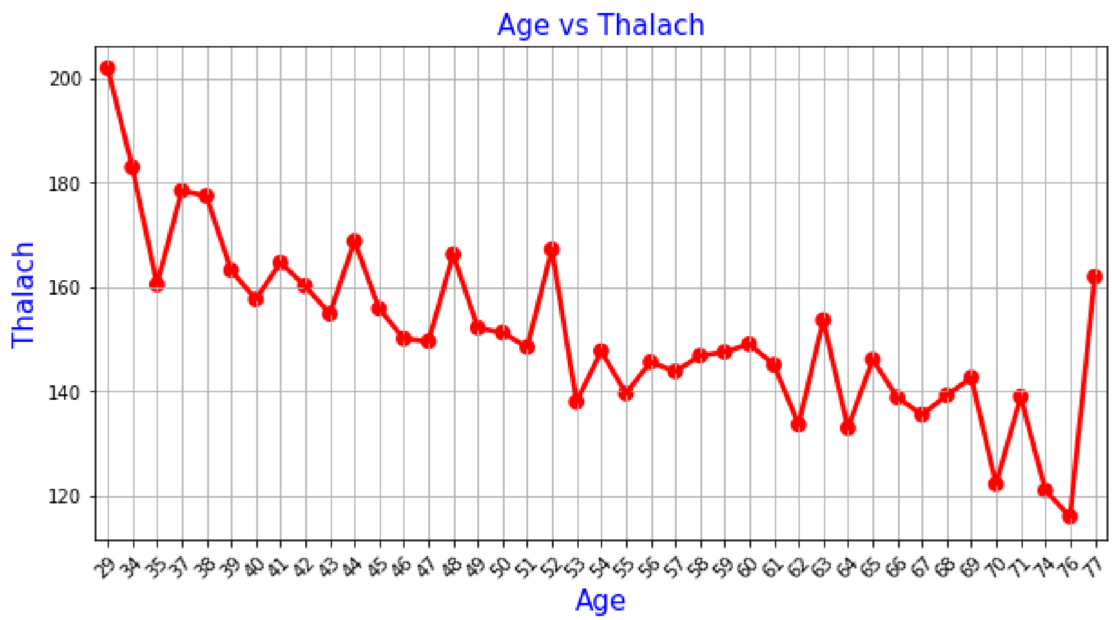
```
sns.barplot(x=data.Thalach.value_counts()[:20].index,
y=data.Thalach.value_counts()[:20].values)
plt.xlabel('Thalach')
plt.ylabel('Count')
plt.title('Thalach Counts')
plt.xticks(rotation=45)
plt.show()
```



In [71]:

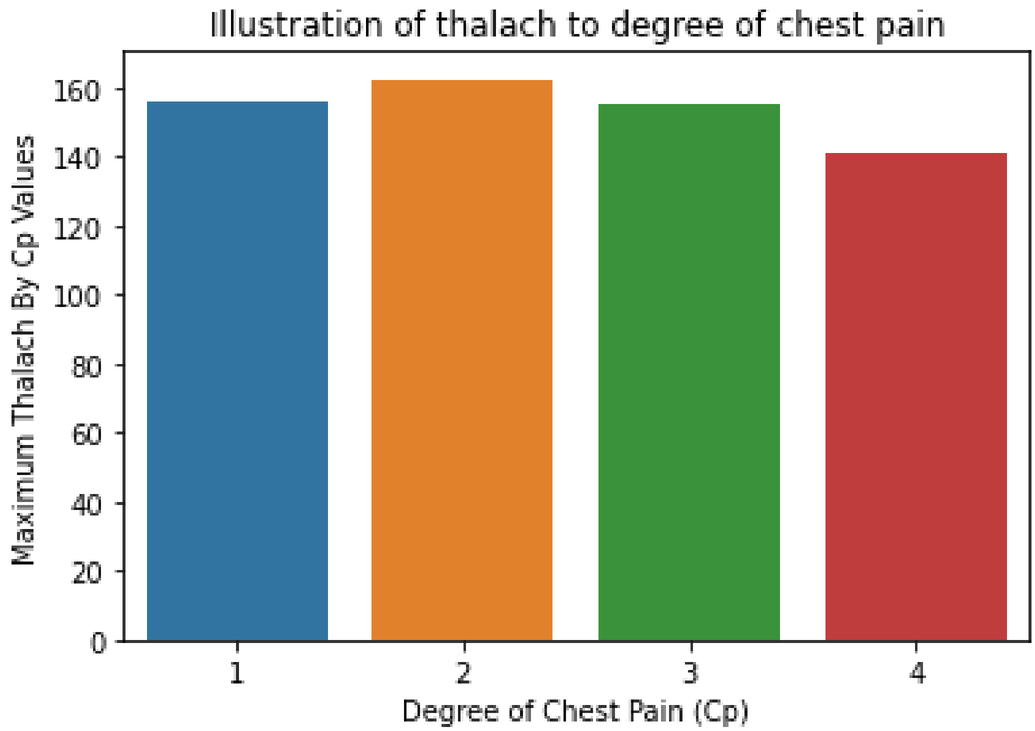
```
age_unique = sorted(data.Age.unique())
age_thalach_values = data.groupby('Age')['Thalach'].count().values
mean_thalach = []
for i, age in enumerate(age_unique):
    mean_thalach.append(sum(data[data['Age'] ==
age].Thalach)/age_thalach_values[i])
# data_sorted=data.sort_values(by='Age',ascending=True)
plt.figure(figsize=(10, 5))
sns.pointplot(x=age_unique, y=mean_thalach, color='red', alpha=0.8)
plt.xlabel('Age', fontsize=15, color='blue')
plt.xticks(rotation=45)
plt.ylabel('Thalach', fontsize=15, color='blue')
```

```
plt.title('Age vs Thalach', fontsize=15, color='blue')
plt.grid()
plt.show()
age_range_thalach = data.groupby('AgeRange')['Thalach'].mean()
sns.barplot(x=age_range_thalach.index, y=age_range_thalach.values)
plt.xlabel('Age Range Values')
plt.ylabel('Maximum Thalach By Age Range')
plt.title('illustration of the thalach to the age range')
plt.show()
```



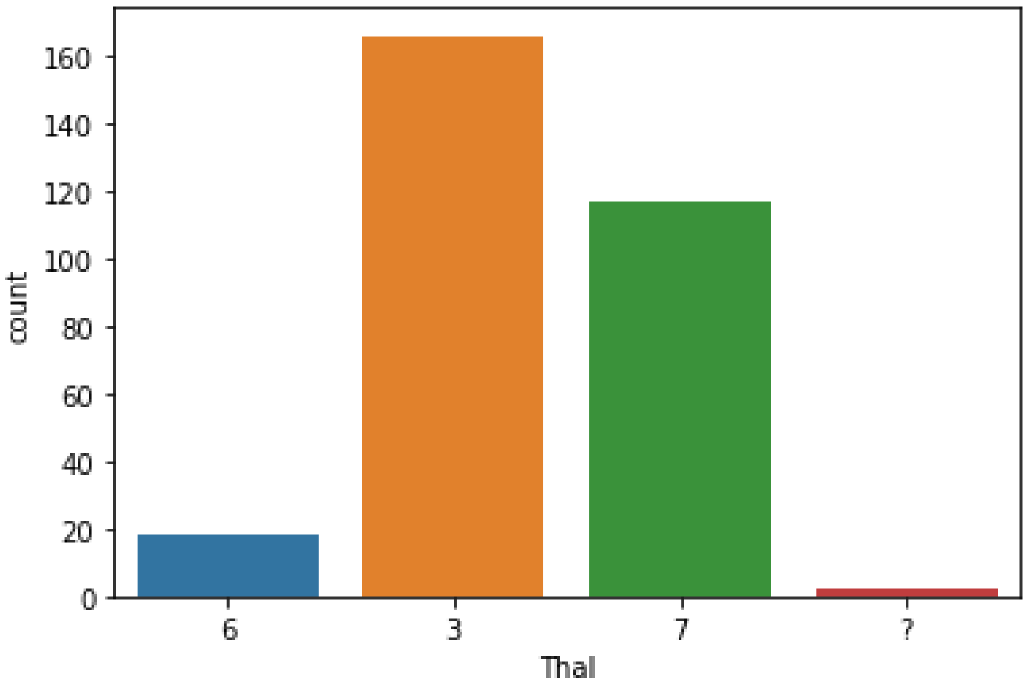
In [72]:

```
# As shown in this graph, this rate decreases as the heart rate is faster and
in old age areas.
cp_thalach = data.groupby('Cp')['Thalach'].mean()
sns.barplot(x=cp_thalach.index, y=cp_thalach.values)
plt.xlabel('Degree of Chest Pain (Cp)')
plt.ylabel('Maximum Thalach By Cp Values')
plt.title('Illustration of thalach to degree of chest pain')
plt.show()
```



In [73]:

```
# As seen in this graph, it is seen that the heart rate is less when the chest pain is low. But in cases where chest
# pain is 1, it is observed that the area is more. 2 and 3 were found to be of the same degree.
data.Thal.value_counts()
sns.countplot(data.Thal)
plt.show()
```



In [75]:

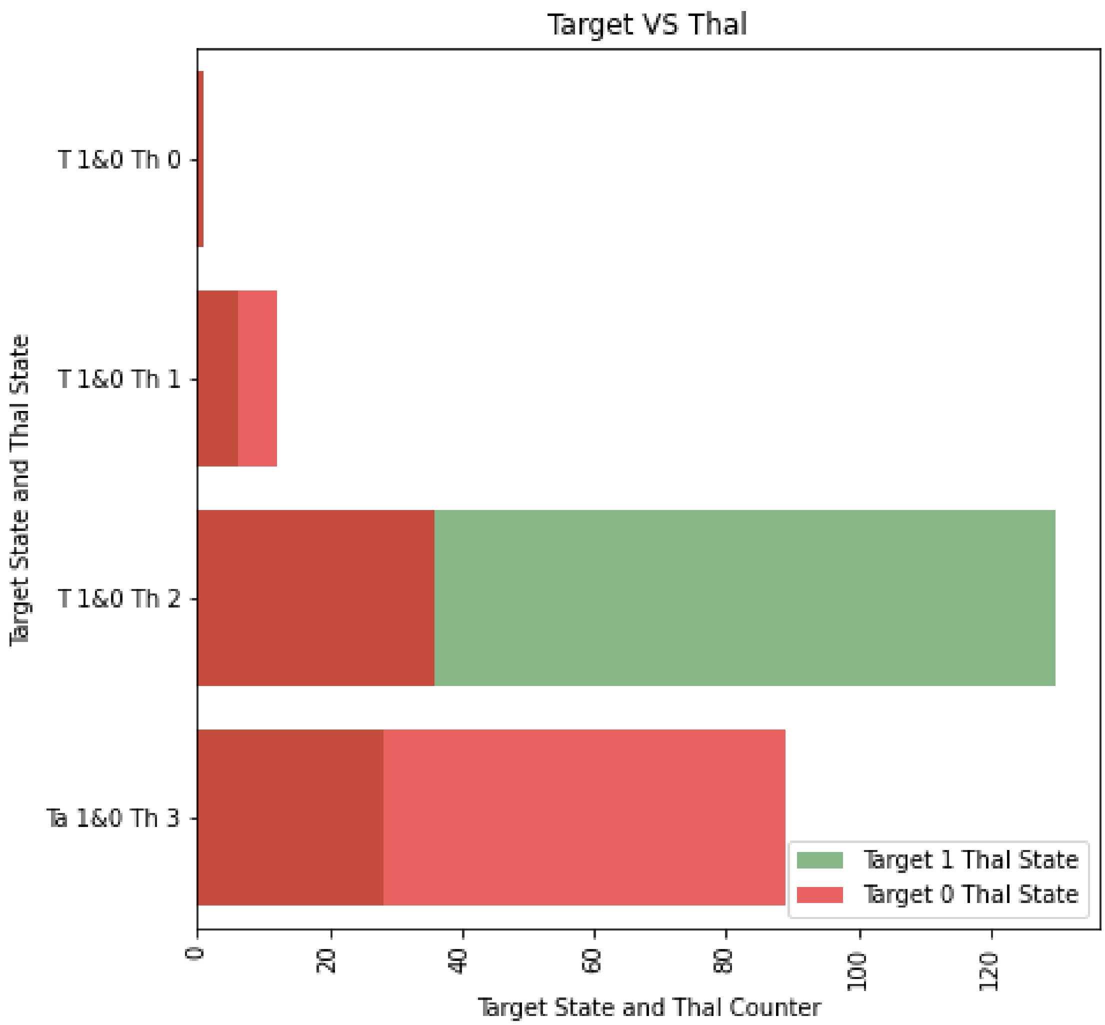
```
# Target 1
a = len(data[(data['Target'] == 1) & (data['Thal'] == 0)])
b = len(data[(data['Target'] == 1) & (data['Thal'] == 1)])
c = len(data[(data['Target'] == 1) & (data['Thal'] == 2)])
d = len(data[(data['Target'] == 1) & (data['Thal'] == 3)])
print('Target 1 Thal 0: ', a)
print('Target 1 Thal 1: ', b)
print('Target 1 Thal 2: ', c)
print('Target 1 Thal 3: ', d)

# So,Apparently, there is a rate at Thal 2.Now, draw graph
print('*'*50)
# Target 0
e = len(data[(data['Target'] == 0) & (data['Thal'] == 0)])
f = len(data[(data['Target'] == 0) & (data['Thal'] == 1)])
g = len(data[(data['Target'] == 0) & (data['Thal'] == 2)])
h = len(data[(data['Target'] == 0) & (data['Thal'] == 3)])
print('Target 0 Thal 0: ', e)
print('Target 0 Thal 1: ', f)
print('Target 0 Thal 2: ', g)
print('Target 0 Thal 3: ', h)
```

```
Target 1 Thal 0:  0
Target 1 Thal 1:  0
Target 1 Thal 2:  0
Target 1 Thal 3:  0
**************************************************
```

```
Target 0 Thal 0:  0
Target 0 Thal 1:  0
Target 0 Thal 2:  0
Target 0 Thal 3:  0
```

In [76]:

```
f, ax = plt.subplots(figsize=(7, 7))
sns.barplot(y=['T 1&0 Th 0', 'T 1&0 Th 1', 'T 1&0 Th 2', 'Ta 1&0 Th 3'],
x=[1, 6, 130, 28], color='green', alpha=0.5,
            label='Target 1 Thal State')
sns.barplot(y=['T 1&0 Th 0', 'T 1&0 Th 1', 'T 1&0 Th 2', 'Ta 1&0 Th 3'],
x=[1, 12, 36, 89], color='red', alpha=0.7,
            label='Target 0 Thal State')
ax.legend(loc='lower right', frameon=True)
ax.set(xlabel='Target State and Thal Counter', ylabel='Target State and Thal
State', title='Target VS Thal')
plt.xticks(rotation=90)
plt.show()

# So, there has been a very nice graphic display. This is the situation that
best describes the situation.
```



In [78]:

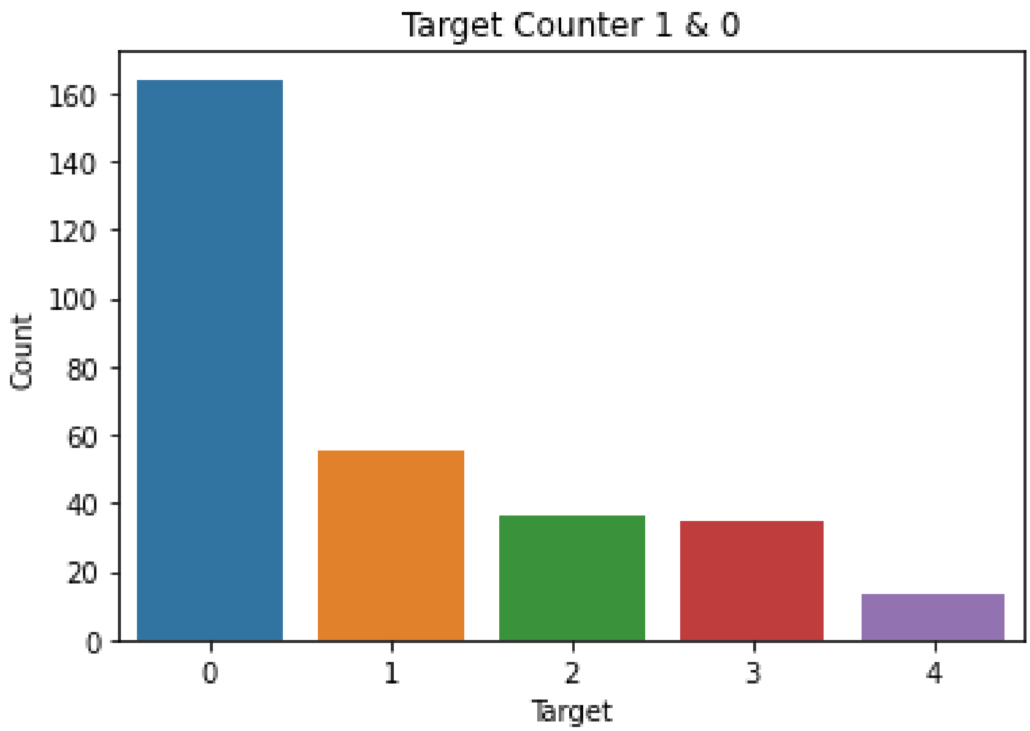
```
data.Target.unique()
# Only two values are shown.
# A value of 1 is the value of patient 0.
```

Out[78]:

```
array([0, 2, 1, 3, 4], dtype=int64)
```

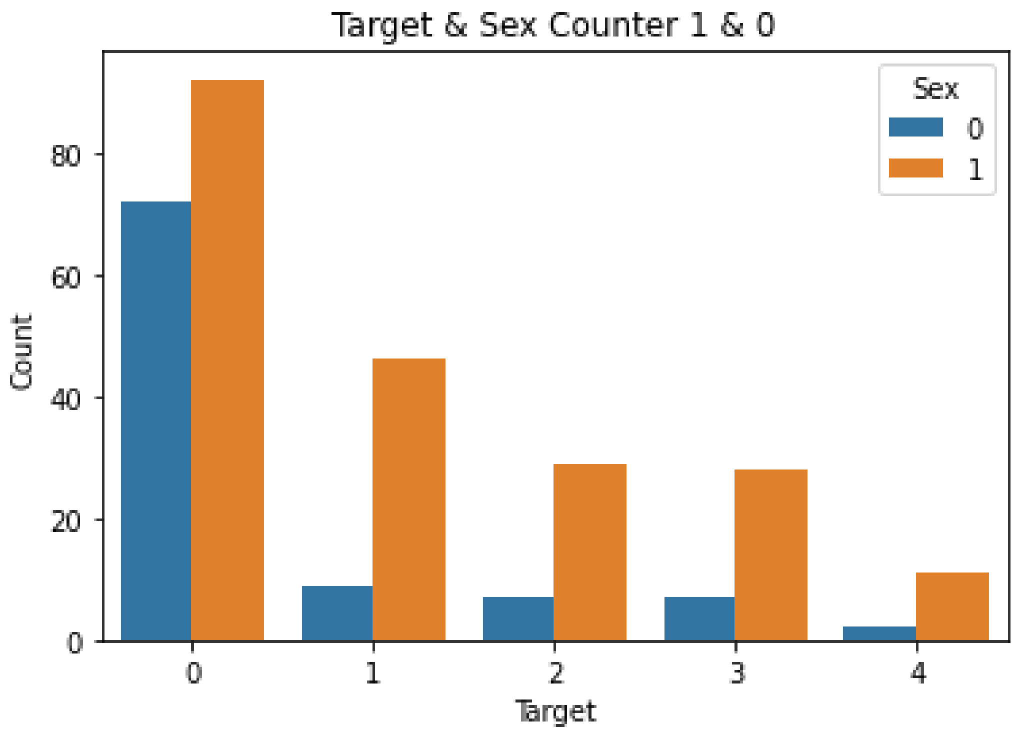
In [79]:

```
sns.countplot(data.Target)
plt.xlabel('Target')
plt.ylabel('Count')
plt.title('Target Counter 1 & 0')
plt.show()
```



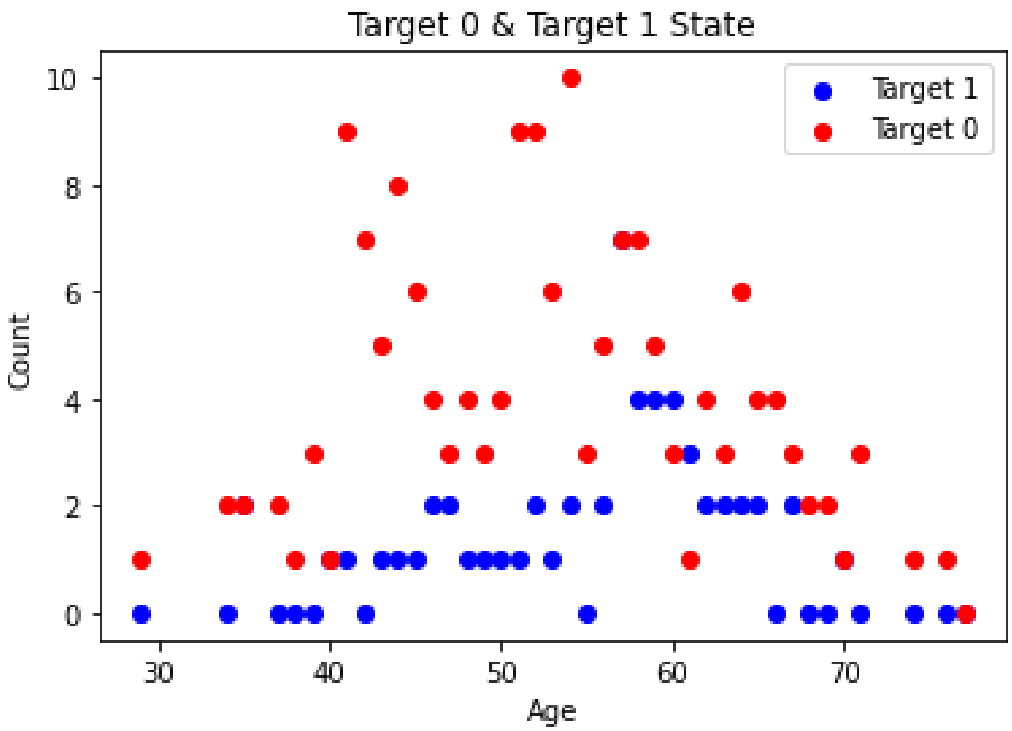
In [80]:

```
sns.countplot(data.Target, hue=data.Sex)
plt.xlabel('Target')
plt.ylabel('Count')
plt.title('Target & Sex Counter 1 & 0')
plt.show()
```



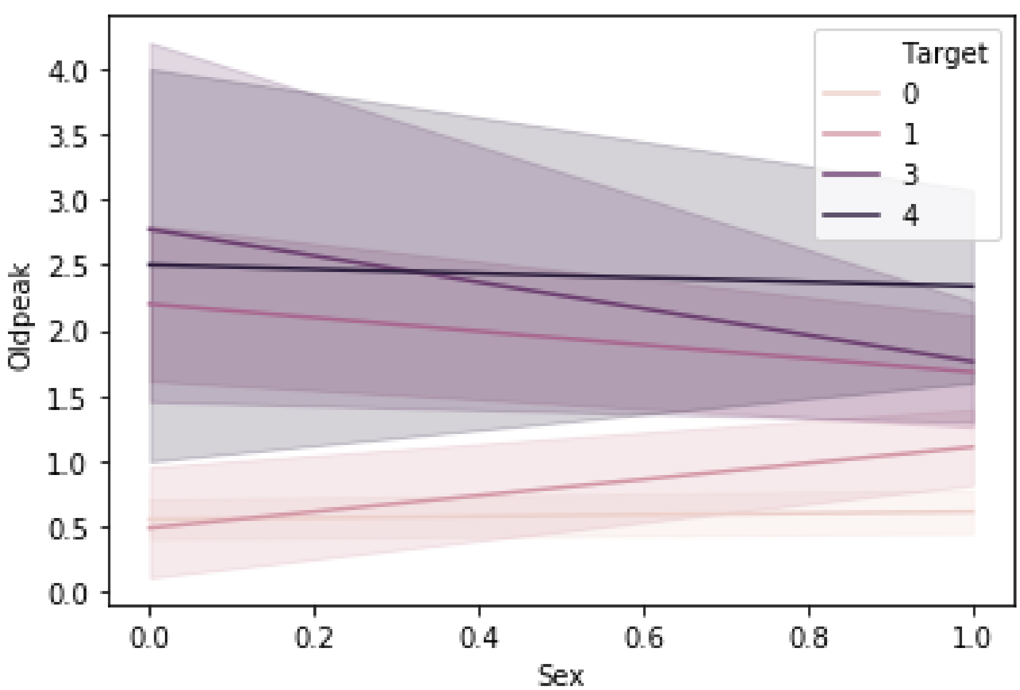
In [81]:

```
# Determine the age ranges of patients with and without sickness and make
analyzes about them
age_counter_target_1 = []
age_counter_target_0 = []
for age in data.Age.unique():
    age_counter_target_1.append(len(data[(data['Age'] == age) & (data.Target
== 1)]))
    age_counter_target_0.append(len(data[(data['Age'] == age) & (data.Target
== 0)]))

# Now, draw show on graph
# Target 1 & 0 show graph on scatter
plt.scatter(x=data.Age.unique(), y=age_counter_target_1, color='blue',
label='Target 1')
plt.scatter(x=data.Age.unique(), y=age_counter_target_0, color='red',
label='Target 0')
plt.legend(loc='upper right', frameon=True)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Target 0 & Target 1 State')
plt.show()
```



In [123]:

```
sns.lineplot(x="Sex", y="Oldpeak", hue="Target", data=data)
plt.show()
```



In [83]:

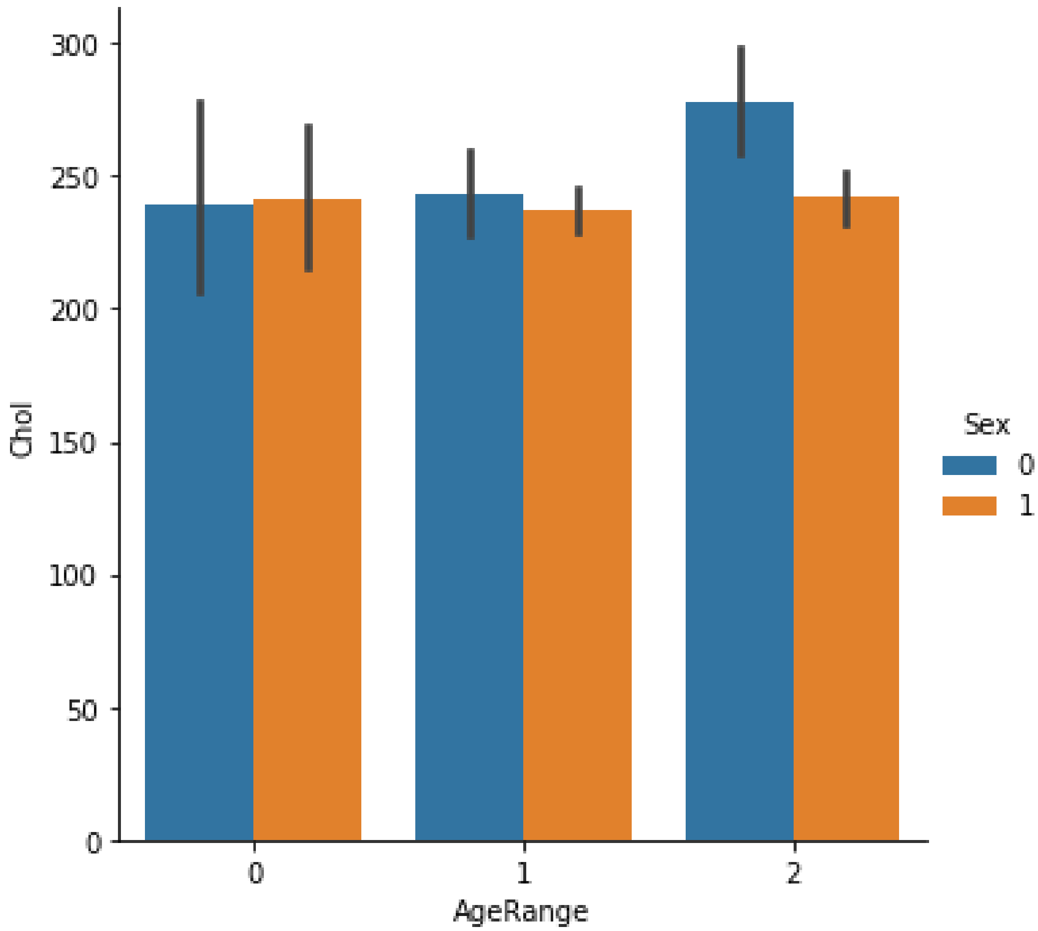
```
data.head()
```

Out[83]:

	Age	Sex	Cp	Trestbps	Chol	Fbs	restceg	Thalach	Exang	Oldpeak	Slope	Ca	Thal	Target	AgeRange
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0	2
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1	2
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0	1

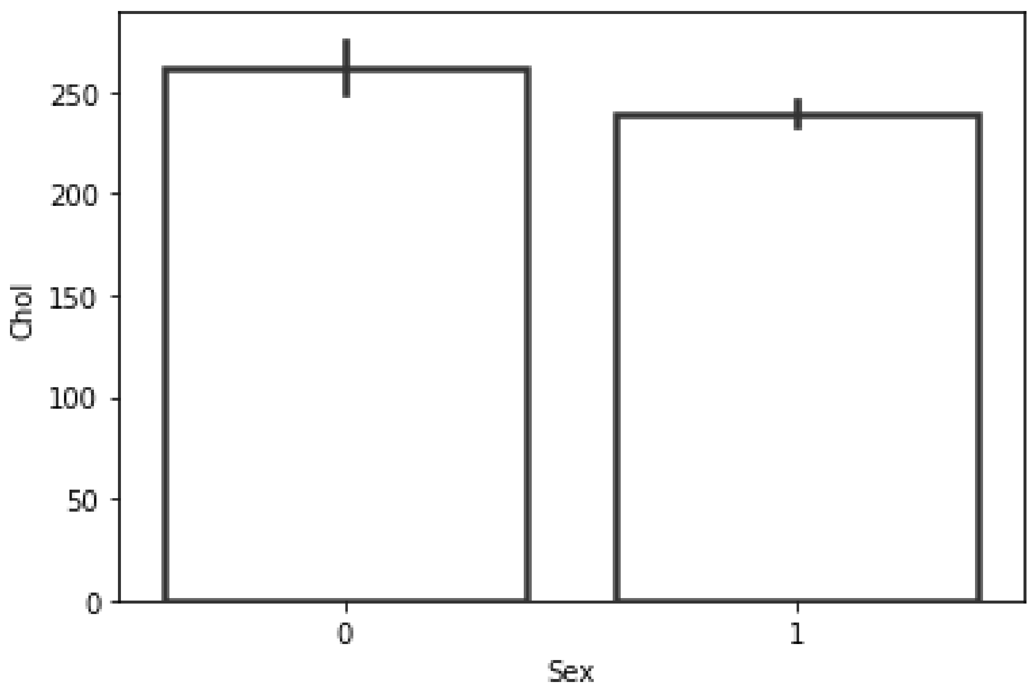
In [124]:

```
g = sns.catplot(x="AgeRange", y="Chol", hue="Sex", data=data, kind="bar")
plt.show()
```



In [125]:

```
ax = sns.barplot("Sex", "Chol", data=data, linewidth=2.5, facecolor=(1, 1, 1,
0), errcolor=".2", edgecolor=".2")
plt.show()
```



In [86]:

```
male_young_t_1 = data[(data['Sex'] == 1) & (data['AgeRange'] == 0) &
(data['Target'] == 1)]
male_middle_t_1 = data[(data['Sex'] == 1) & (data['AgeRange'] == 1) &
(data['Target'] == 1)]
male_elderly_t_1 = data[(data['Sex'] == 1) & (data['AgeRange'] == 2) &
(data['Target'] == 1)]
print(len(male_young_t_1))
print(len(male_middle_t_1))
print(len(male_elderly_t_1))
```

```
2
18
26
```

In [128]:

```
male_young_t_1 = data[(data['Sex'] == 1) & (data['AgeRange'] == 0) &
(data['Target'] == 1)]
male_middle_t_1 = data[(data['Sex'] == 1) & (data['AgeRange'] == 1) &
(data['Target'] == 1)]
male_elderly_t_1 = data[(data['Sex'] == 1) & (data['AgeRange'] == 2) &
(data['Target'] == 1)]

f, ax1 = plt.subplots(figsize=(20, 10))
sns.pointplot(x=np.arange(len(male_young_t_1)), y=male_young_t_1.Trestbps,
color='lime', alpha=0.8, label='Young')
sns.pointplot(x=np.arange(len(male_middle_t_1)), y=male_middle_t_1.Trestbps,
```

```
color='black', alpha=0.8, label='Middle')
sns.pointplot(x=np.arange(len(male_elderly_t_1)),
y=male_elderly_t_1.Trestbps, color='red', alpha=0.8, label='Elderly')
plt.xlabel('Range', fontsize=15, color='blue')
plt.xticks(rotation=90)
plt.legend(loc='upper right', frameon=True)
plt.ylabel('Trestbps', fontsize=15, color='blue')
plt.title('Age Range Values vs Trestbps', fontsize=20, color='blue')
plt.grid()
plt.show()
```

```
No handles with labels found to put in legend.
```

In [127]:

```
data.head()
```

Out[127]:

	Age	Sex	Cp	Trestbps	Chol	Fbs	restceg	Thalach	Exang	Oldpeak	Slope	Ca	Thal	Target	AgeRange
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0	2
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1	2
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0	1

In [89]:

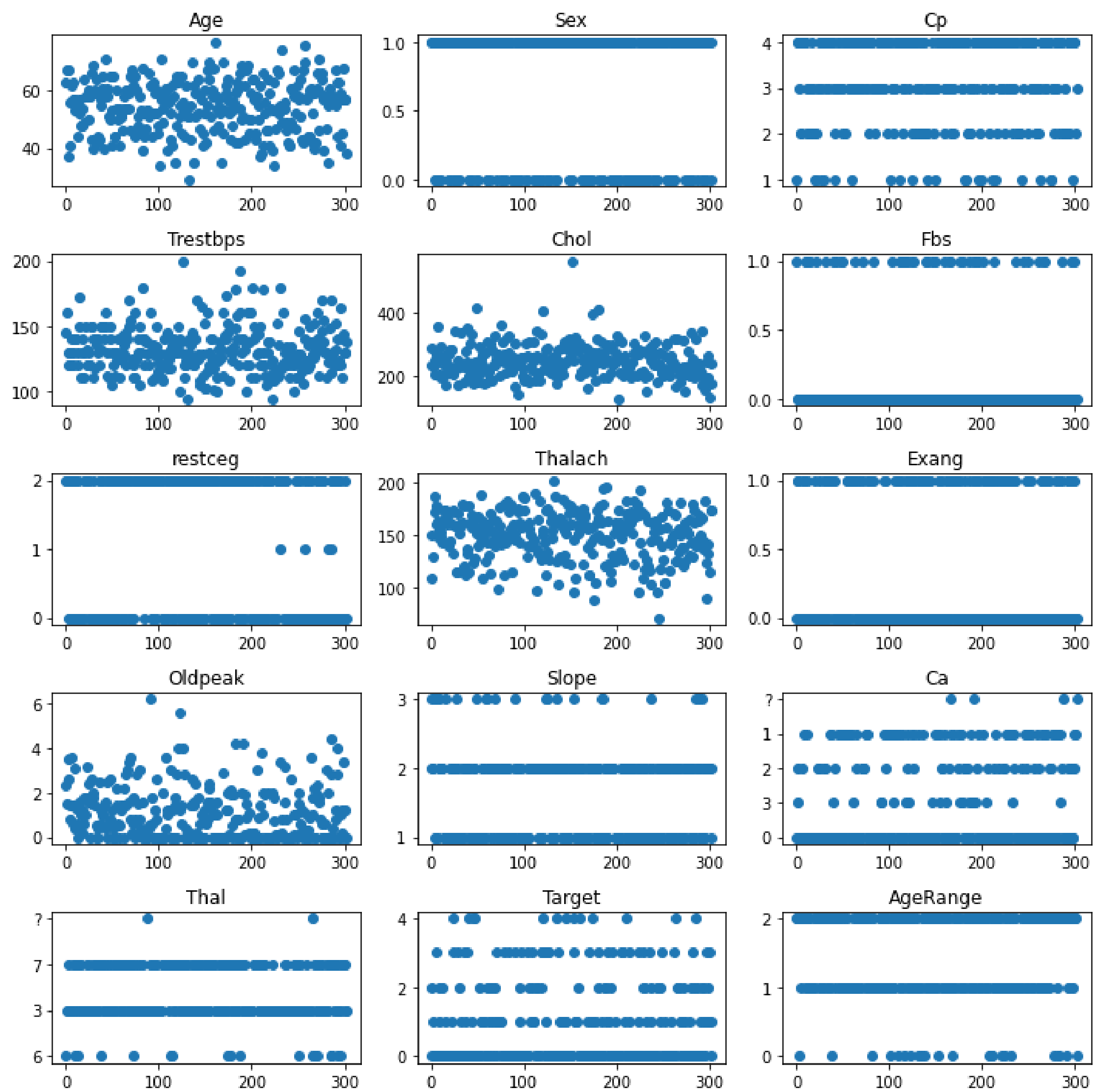
```
data_filter_mean = data[(data['Target'] == 1) & (data['Age'] >
50)].groupby('Sex')[['Trestbps', 'Chol', 'Thalach']].\
    mean()
data_filter_mean.unstack()
```

Out[89]:

```
          Sex
Trestbps  0      139.888889
          1      136.062500
Chol      0      264.222222
          1      254.093750
Thalach   0      152.777778
          1      142.375000
dtype: float64
```

In [90]:

```
for i, col in enumerate(data.columns.values):
    plt.subplot(5, 3, i+1)
    plt.scatter([i for i in range(303)], data[col].values.tolist())
    plt.title(col)
    fig, ax = plt.gcf(), plt.gca()
    fig.set_size_inches(10, 10)
    plt.tight_layout()
plt.show()
```



In [91]:

```
# Let's see how the correlation values between them
data.corr()
```

Out[91]:

	Age	Sex	Cp	Trestbps	Chol	Fbs	restceg	Thalach	Exang	Oldpeak	Slope	Target	Age Range
Age	1.000000	-0.097542	0.104139	0.284946	0.208950	0.118530	0.148868	-0.393806	0.091661	0.203805	0.161770	0.222853	0.806614
Sex	-0.097542	1.000000	0.010084	-0.064456	-0.199915	0.047862	0.021647	-0.048663	0.146201	0.102173	0.037533	0.224469	-0.030375
Cp	0.104139	0.010084	1.000000	-0.036077	0.072319	-0.039975	0.067505	-0.334422	0.384060	0.202277	0.152050	0.407075	0.090596
Trestbps	0.284946	-0.064456	-0.036077	1.000000	0.130120	0.175340	0.146560	-0.045351	0.064762	0.189171	0.117382	0.157754	0.222292
Chol	0.208950	-0.199915	0.072319	0.130120	1.000000	0.009841	0.171043	-0.003432	0.061310	0.046564	-0.004062	0.070909	0.132921
Fbs	0.118530	0.047862	-0.039975	0.175340	0.009841	1.000000	0.069564	-0.007854	0.025665	0.005747	0.059894	0.059186	0.130347
restceg	0.148868	0.021647	0.067505	0.146560	0.171043	0.069564	1.000000	-0.083389	0.084867	0.114133	0.133946	0.183696	0.159797
Thalach	-0.393806	-0.048663	-0.334422	-0.045351	-0.003432	-0.007854	-0.083389	1.000000	-0.378103	-0.343085	-0.385601	-0.415040	-0.299427
Exang	0.091661	0.146201	0.384060	0.064762	0.061310	0.025665	0.084867	-0.378103	1.000000	0.288223	0.257748	0.397057	0.065406
Oldpeak	0.203805	0.102173	0.202277	0.189171	0.046564	0.005747	0.114133	-0.343085	0.288223	1.000000	0.577537	0.504092	0.146949

Slope	0.161770	0.037533	0.152050	0.117382	-0.004062	0.059894	0.133946	-0.385601	0.257748	0.577537	1.000000	0.377957	0.140733
Target	0.222853	0.224469	0.407075	0.157754	0.070909	0.059186	0.183696	-0.415040	0.397057	0.504092	0.377957	1.000000	0.162808
Age Range	0.806614	-0.030375	0.090596	0.222292	0.132921	0.130347	0.159797	-0.299427	0.065406	0.146949	0.140733	0.162808	1.000000

In [92]:

```
dataX = data.drop('Target', axis=1)
dataY = data['Target']
X_train, X_test, y_train, y_test = train_test_split(dataX, dataY,
test_size=0.2, random_state=42)
print('X_train', X_train.shape)
print('X_test', X_test.shape)
print('y_train', y_train.shape)
print('y_test', y_test.shape)
```

```
X_train (242, 14)
X_test (61, 14)
y_train (242,)
y_test (61,)
```

In [131]:

```
# Normalization as the first process
# Normalize
dataX = data.drop('Target', axis=1)
dataY = data['Target']
X_train, X_test, y_train, y_test = train_test_split(dataX, dataY,
test_size=0.2, random_state=42)
X_train = (X_train-np.min(X_train))/(np.max(X_train)-np.min(X_train)).values
X_test = (X_test-np.min(X_test))/(np.max(X_test)-np.min(X_test)).values
```

In [132]:

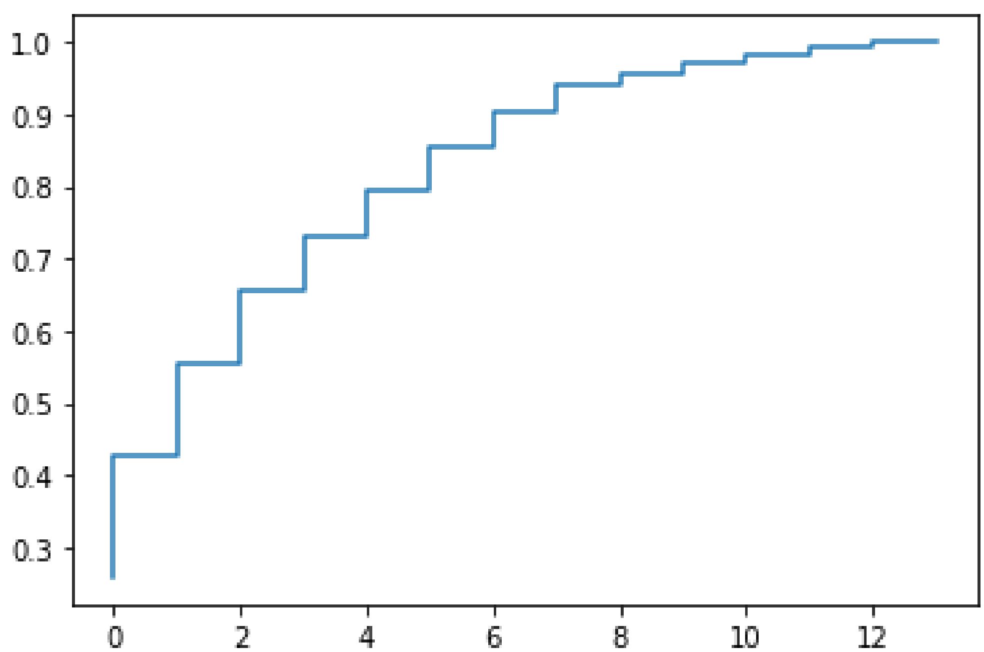
```
from sklearn.decomposition import PCA
pca = PCA().fit(X_train)
print(pca.explained_variance_ratio_)
print()
print(X_train.columns.values.tolist())
print(pca.components_)
```

```
[0.25839852 0.16992593 0.12696313 0.10128235 0.0738321  0.06476136
 0.05818582 0.04940414 0.03705724 0.01712078 0.01484224 0.01199703
 0.01097685 0.00525251]

['Age', 'Sex', 'Cp', 'Trestbps', 'Chol', 'Fbs', 'restceg', 'Thalach',
'Exang', 'Oldpeak', 'Slope', 'Ca', 'Thal', 'AgeRange']
[[ 7.87028281e-02  3.89468273e-01  2.13730356e-01  5.08781009e-02
   1.13427160e-02  5.29550199e-02  2.11857425e-01 -1.16739153e-01
   5.15409305e-01  1.44039038e-01  1.96736511e-01  1.83863126e-01
   5.97018978e-01  1.30652336e-01]
 [ 1.18141641e-01 -4.66405350e-01  1.14474934e-01  6.43998672e-02
   8.28803036e-02 -1.37526042e-02  7.90486164e-01 -6.02346267e-02
   5.16648682e-02  4.14826441e-02  1.07917183e-01  1.15959437e-01
  -2.19092602e-01  1.95493066e-01]
 [ 4.01028996e-02 -5.85459291e-01  3.14184812e-01 -2.85120506e-02
   2.21564089e-02 -6.94465392e-02 -4.86322623e-01 -9.48594471e-02
   5.37736269e-01  3.92103203e-02  7.01040963e-02  4.32243402e-02
  -8.70364639e-02  1.50461887e-02]
 [ 2.38243315e-01 -3.03887007e-01 -9.94028596e-02  1.26678598e-01
   2.97014595e-02  4.26432472e-01 -2.32577924e-01 -4.16491179e-02
  -4.38097794e-01  6.47094148e-02  1.25176596e-01  2.49046308e-01
   3.83174489e-01  4.07157865e-01]
 [ 1.33205217e-01  2.86685577e-01 -1.38333535e-01  3.76232378e-02
  -1.65542237e-03  6.12384486e-01 -4.85570442e-02 -4.79887325e-03
   3.42183707e-01 -7.43701090e-02 -1.53829890e-01  1.39197290e-01
  -5.21484449e-01  2.53991351e-01]
 [-2.77421957e-01 -2.47437667e-01 -2.01810857e-01  3.88637354e-02
  -3.66251472e-02  5.74003537e-01  1.38782990e-01  1.00744436e-01
   1.39857097e-01  1.02374703e-02  1.88065864e-01 -3.29212271e-01
   2.12912495e-01 -5.00595638e-01]
 [ 1.08037028e-01  7.14781464e-02 -4.63050499e-01  6.52980070e-02
  -6.25990952e-02 -2.06203830e-01 -9.60467842e-02 -1.22394348e-01
   8.67214177e-02  2.28653925e-01  7.02161563e-01 -2.56963993e-01
  -1.95339092e-01  1.92024261e-01]
 [-1.07978083e-01  1.78048016e-01  5.52237340e-01 -9.30042019e-02
  -7.72899874e-02  1.87131206e-01 -5.26071587e-02 -8.12516609e-02
  -2.97461160e-01  2.08263023e-01  4.70689231e-01  2.87823229e-01
  -2.80137843e-01 -2.79174132e-01]
 [-1.13147124e-01 -8.47503187e-02 -4.75916581e-01  5.63699532e-03
   8.88568900e-02 -1.40345703e-01 -1.73862132e-02  1.12720013e-01
   1.12679445e-01  1.41223119e-01 -2.11876413e-03  7.58373121e-01
  -1.42542836e-02 -3.23895211e-01]
 [ 3.77593835e-02  7.45371717e-02  1.27955818e-01  7.27410835e-01
   5.34916264e-01 -3.84939781e-02 -6.87208198e-02  2.56421743e-01
  -1.66767296e-02  2.53008378e-01 -4.12976929e-02 -1.02600861e-01
  -7.67298434e-02 -8.37622154e-02]
 [-2.85393791e-02 -4.53696587e-02  2.81438113e-02  4.50825015e-01
  -8.06577147e-01 -4.20290174e-02  2.18283008e-02  7.18567133e-02
   1.58272198e-02  3.13518914e-01 -1.87953912e-01  2.29368580e-02
  -2.58256889e-02  7.38152869e-03]
```

```
 [ 1.83438589e-01  1.85457193e-04 -9.59261938e-02 -4.22234956e-02
   1.37840880e-01  5.44247248e-02  8.36914980e-03 -7.54398927e-01
  -6.48388367e-02  4.51104247e-01 -3.06674534e-01 -9.94743875e-02
  -1.53938951e-02 -2.28768741e-01]
 [ 1.59039382e-01  2.39436902e-02 -1.45016330e-02  4.50832356e-01
  -9.72560734e-02 -4.89665884e-02 -1.07362023e-02 -4.28949255e-01
  -1.31780332e-02 -6.95835906e-01  1.55201009e-01  1.12994785e-01
   2.37973368e-04 -2.31273354e-01]
 [ 8.54082362e-01  8.73110005e-03  1.73976165e-02 -1.52085608e-01
  -8.56509332e-02  1.14485041e-03  1.86615924e-02  3.21744640e-01
   1.83729790e-02  2.41121563e-02  1.67320465e-02 -5.81398371e-02
   1.61788473e-02 -3.61922916e-01]]
```

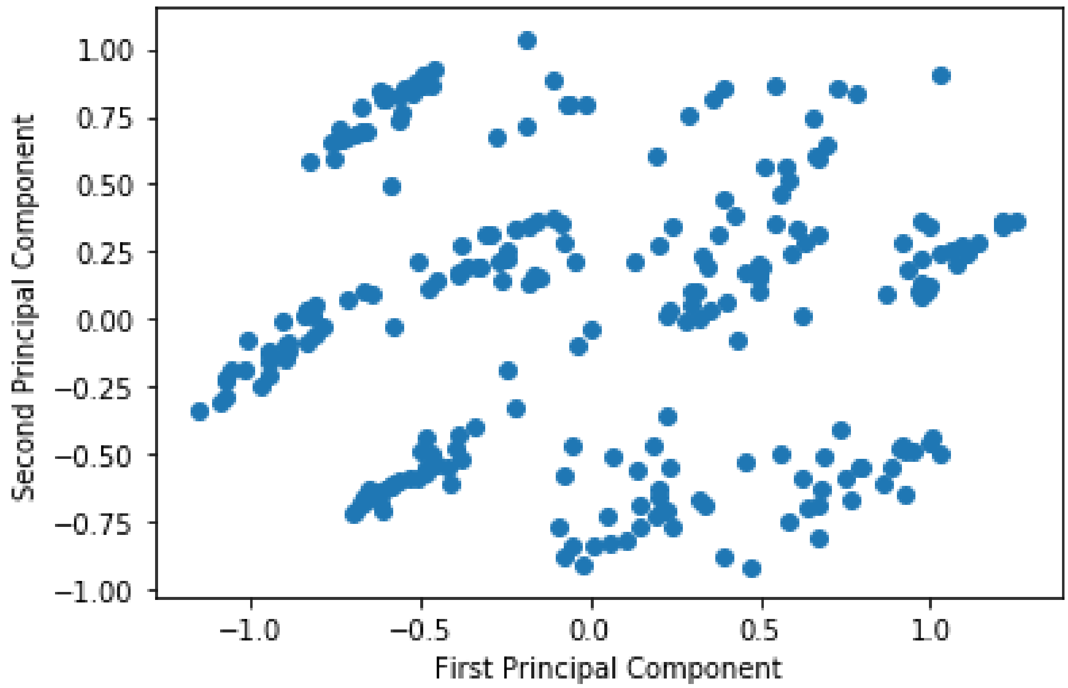
In [133]:

```
cumulative = np.cumsum(pca.explained_variance_ratio_)
plt.step([i for i in range(len(cumulative))], cumulative)
plt.show()
```



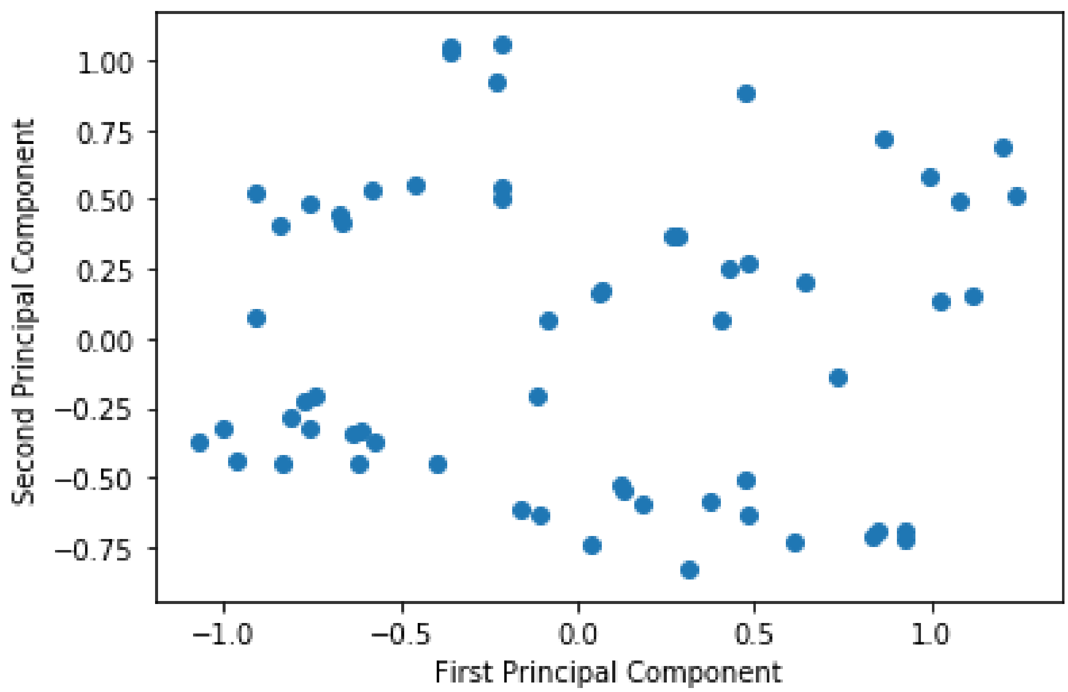
In [134]:

```
pca = PCA(n_components=8)
pca.fit(X_train)
reduced_data_train = pca.transform(X_train)
# inverse_data = pca.inverse_transform(reduced_data)
plt.scatter(reduced_data_train[:, 0], reduced_data_train[:, 1],
label='reduced')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```



In [135]:

```
pca = PCA(n_components=8)
pca.fit(X_test)
reduced_data_test = pca.transform(X_test)
# inverse_data = pca.inverse_transform(reduced_data)
plt.scatter(reduced_data_test[:, 0], reduced_data_test[:, 1],
label='reduced')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```



In [136]:

```
reduced_data_train = pd.DataFrame(reduced_data_train,
                                  columns=['Dim1', 'Dim2', 'Dim3', 'Dim4',
'Dim5', 'Dim6', 'Dim7', 'Dim8'])
reduced_data_test = pd.DataFrame(reduced_data_test,
                                 columns=['Dim1', 'Dim2', 'Dim3', 'Dim4',
'Dim5', 'Dim6', 'Dim7', 'Dim8'])
X_train = reduced_data_train
X_test = reduced_data_test
```

In [137]:

```
def plot_roc_(false_positive_rate, true_positive_rate, roc_auc):
    plt.figure(figsize=(5, 5))
    plt.title('Receiver Operating Characteristic')
    plt.plot(false_positive_rate, true_positive_rate, color='red', label='AUC
= %0.2f' % roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.axis('tight')
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()


def plot_feature_importances(gbm):
    n_features = X_train.shape[1]
    plt.barh(range(n_features), gbm.feature_importances_, align='center')
```

```
    plt.yticks(np.arange(n_features), X_train.columns)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)


combine_features_list = [
    ('Dim1', 'Dim2', 'Dim3'),
    ('Dim4', 'Dim5', 'Dim5', 'Dim6'),
    ('Dim7', 'Dim8', 'Dim1'),
    ('Dim4', 'Dim8', 'Dim5')
]
```

In [168]:

```
parameters = [
    {
        'penalty': ['l1', 'l2'],'C': [0.1, 0.4, 0.5],'random_state': [0]
    },
]

for features in combine_features_list:
    print(features)
    print("*" * 50)

    X_train_set = X_train.loc[:, features]
    X_test_set = X_test.loc[:, features]

    gslog = GridSearchCV(LogisticRegression(), parameters,
scoring='accuracy')
    gslog.fit(X_train_set, y_train)
    print('Best parameters set:')
    print(gslog.best_params_)
    print()
    predictions = [
        (gslog.predict(X_train_set), y_train, 'Train'),
        (gslog.predict(X_test_set), y_test, 'Test'),
    ]

    for pred in predictions:
        print(pred[2] + ' Classification Report:')
        print("*" * 50)
        print(classification_report(pred[1], pred[0]))
        print("*" * 50)
        print(pred[2] + ' Confusion Matrix:')
        print(confusion_matrix(pred[1], pred[0]))
        print("*" * 50)

    print("*" * 50)
    basari = cross_val_score(estimator=LogisticRegression(), X=X_train,
```

```
y=y_train, cv=12)
    print(basari.mean())
    print(basari.std())
    print("*" * 50)

('Dim1', 'Dim2', 'Dim3')
**************************************************
Best parameters set:
{'C': 0.4, 'penalty': 'l2', 'random_state': 0}

Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.72      0.96      0.83       135
           1       0.25      0.09      0.14        43
           2       0.27      0.11      0.16        27
           3       0.43      0.54      0.48        28
           4       0.00      0.00      0.00         9

    accuracy                           0.63       242
   macro avg       0.33      0.34      0.32       242
weighted avg       0.53      0.63      0.56       242

**************************************************
Train Confusion Matrix:
[[130   2   1   2   0]
 [ 30   4   2   7   0]
 [  9   6   3   9   0]
 [  7   2   4  15   0]
 [  4   2   1   2   0]]
**************************************************
Test Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.62      1.00      0.76        29
           1       0.00      0.00      0.00        12
           2       0.00      0.00      0.00         9
           3       0.12      0.14      0.13         7
           4       0.00      0.00      0.00         4

    accuracy                           0.49        61
   macro avg       0.15      0.23      0.18        61
weighted avg       0.31      0.49      0.38        61

**************************************************
Test Confusion Matrix:
[[29  0  0  0  0]
```

```
 [ 7  0  1  4  0]
 [ 6  2  0  1  0]
 [ 4  2  0  1  0]
 [ 1  1  0  2  0]]
**************************************************
**************************************************
0.6248015873015873
0.07996277911843525
**************************************************
('Dim4', 'Dim5', 'Dim5', 'Dim6')
**************************************************
Best parameters set:
{'C': 0.1, 'penalty': 'l2', 'random_state': 0}

Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.56      1.00      0.72       135
           1       0.00      0.00      0.00        43
           2       0.00      0.00      0.00        27
           3       0.00      0.00      0.00        28
           4       0.00      0.00      0.00         9

    accuracy                           0.56       242
   macro avg       0.11      0.20      0.14       242
weighted avg       0.31      0.56      0.40       242

**************************************************
Train Confusion Matrix:
[[135   0   0   0   0]
 [ 43   0   0   0   0]
 [ 27   0   0   0   0]
 [ 28   0   0   0   0]
 [  9   0   0   0   0]]
**************************************************
Test Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.48      1.00      0.64        29
           1       0.00      0.00      0.00        12
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00         7
           4       0.00      0.00      0.00         4

    accuracy                           0.48        61
   macro avg       0.10      0.20      0.13        61
weighted avg       0.23      0.48      0.31        61
```

```
**************************************************
Test Confusion Matrix:
[[29  0  0  0  0]
 [12  0  0  0  0]
 [ 9  0  0  0  0]
 [ 7  0  0  0  0]
 [ 4  0  0  0  0]]
**************************************************
**************************************************
0.6248015873015873
0.07996277911843525
**************************************************
('Dim7', 'Dim8', 'Dim1')
**************************************************
Best parameters set:
{'C': 0.5, 'penalty': 'l2', 'random_state': 0}

Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.72      0.95      0.82       135
           1       0.27      0.14      0.18        43
           2       0.00      0.00      0.00        27
           3       0.40      0.61      0.48        28
           4       0.00      0.00      0.00         9

    accuracy                           0.62       242
   macro avg       0.28      0.34      0.30       242
weighted avg       0.50      0.62      0.55       242

**************************************************
Train Confusion Matrix:
[[128   6   0   1   0]
 [ 30   6   0   7   0]
 [  8   5   0  14   0]
 [  7   4   0  17   0]
 [  4   1   0   4   0]]
**************************************************
Test Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.62      1.00      0.76        29
           1       0.25      0.08      0.12        12
           2       0.00      0.00      0.00         9
           3       0.20      0.29      0.24         7
           4       0.00      0.00      0.00         4
```

```
    accuracy                           0.52        61
   macro avg       0.21      0.27      0.22        61
weighted avg       0.37      0.52      0.41        61

**************************************************
Test Confusion Matrix:
[[29  0  0  0  0]
 [ 7  1  0  4  0]
 [ 6  1  0  2  0]
 [ 4  1  0  2  0]
 [ 1  1  0  2  0]]
**************************************************
**************************************************
0.6248015873015873
0.07996277911843525
**************************************************
('Dim4', 'Dim8', 'Dim5')
**************************************************
Best parameters set:
{'C': 0.1, 'penalty': 'l2', 'random_state': 0}

Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.56      1.00      0.72       135
           1       0.00      0.00      0.00        43
           2       0.00      0.00      0.00        27
           3       0.00      0.00      0.00        28
           4       0.00      0.00      0.00         9

    accuracy                           0.56       242
   macro avg       0.11      0.20      0.14       242
weighted avg       0.31      0.56      0.40       242

**************************************************
Train Confusion Matrix:
[[135   0   0   0   0]
 [ 43   0   0   0   0]
 [ 27   0   0   0   0]
 [ 28   0   0   0   0]
 [  9   0   0   0   0]]
**************************************************
Test Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.48      1.00      0.64        29
```

```
           1       0.00      0.00      0.00        12
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00         7
           4       0.00      0.00      0.00         4

    accuracy                           0.48        61
   macro avg       0.10      0.20      0.13        61
weighted avg       0.23      0.48      0.31        61

**************************************************
Test Confusion Matrix:
[[29  0  0  0  0]
 [12  0  0  0  0]
 [ 9  0  0  0  0]
 [ 7  0  0  0  0]
 [ 4  0  0  0  0]]
**************************************************
**************************************************
0.6248015873015873
0.07996277911843525
**************************************************
```

In [185]:

```
from sklearn.linear_model import LogisticRegression

lr=LogisticRegression(C=0.1,penalty='l1',random_state=0)
lr.fit(X_train,y_train)

y_pred=lr.predict(X_test)



y_proba=lr.predict_proba(X_test)

false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_test,y_proba[:,1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plot_roc_(false_positive_rate,true_positive_rate,roc_auc)



from sklearn.metrics import r2_score,accuracy_score

#print('Hata Oranı :',r2_score(y_test,y_pred))
print('Accurancy Oranı :',accuracy_score(y_test, y_pred))
print("Logistic TRAIN score with ",format(lr.score(X_train, y_train)))
print("Logistic TEST score with ",format(lr.score(X_test, y_test)))
print()

cm=confusion_matrix(y_test,y_pred)
```

```
print(cm)
sns.heatmap(cm,annot=True)
plt.show()
```

In [158]:

```
print('CoEf:\n')
print(lr.coef_)
print('Intercept_\n')
print(lr.intercept_)
print('Proba:\n')
print(lr.predict_log_proba)
```

```
CoEf:

[[-3.34097905 -0.80751848 -0.47890134 -0.6846189   0.38891206  1.78352102
  -0.02035305 -2.55218308]
 [-0.8192547  -0.40994335 -0.42572259 -0.72779789 -0.12709345 -1.25538147
   0.59724349 -1.9629514 ]
 [ 1.18128871 -0.09661853  0.28048053  1.22872231  1.36730348 -0.67506767
   0.00882897  0.55233691]
 [ 1.81701203  0.58620694  0.59761475  0.50628221 -0.02962596  0.98297915
  -0.97634016  2.67671325]
 [ 1.16193301  0.72787341  0.02652866 -0.32258773 -1.59949612 -0.83605102
   0.39062075  1.28608432]]
Intercept_

[ 2.06614236  1.01667714 -0.3662306  -0.98611531 -1.73047359]
Proba:

<bound method LogisticRegression.predict_log_proba of
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)>
```