PROJECT ON CARDIOVASCULAR DISEASE PREDICTION

(using different machine learning models)

Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns

import pandas_profiling #Pandas Profiling is a Python library that allows you to generate a very \
                        #detailed report on our pandas dataframe without much input from the user.

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

from sklearn.metrics import  confusion_matrix , ConfusionMatrixDisplay
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn import svm

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

# Importing required package for visualization
import matplotlib.pyplot as plt
```

Loading the data set

In [2]:
```
heart_data = pd.read_csv('C:/Users/HP1/OneDrive/Desktop/heart data.csv')
```

Information of the Data Set

In [3]:
```
heart_data.shape
```

Out[3]:
```
(1025, 14)
```

In [4]:
```
hd = heart_data.drop_duplicates()
```

In [5]:
```
hd.shape
```

Out[5]:
```
(302, 14)
```

In [6]:
```
hd.head(6)
```

Out[6]:

	Age	Gender	C.P	Trestbps	Chol	F.B.S	Restecg	Thalach	Exang	Old peak	Slope	C.A	Thal	Target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1

In [7]:
```
hd.tail()
```

Out[7]:

	Age	Gender	C.P	Trestbps	Chol	F.B.S	Restecg	Thalach	Exang	Old peak	Slope	C.A	Thal	Target
723	68	0	2	120	211	0	0	115	0	1.5	1	0	2	1
733	44	0	2	108	141	0	1	175	0	0.6	1	0	2	1
739	52	1	0	128	255	0	1	161	1	0.0	2	1	3	0
843	59	1	3	160	273	0	0	125	0	0.0	2	0	2	0
878	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

Meaning of the column headers : -

1.**Age**: The person's age in years

2.**Sex**: The person's sex (*1 = male, 0 = female*)

3.**C.P**: The chest pain experienced (*Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic*

4.**Trestbps**: The person's resting blood pressure (*mm Hg*)

5.**Chol**: The person's cholesterol measurement in mg/dl

6.**F.B.S**: The person's fasting blood sugar (*> 120 mg/dl, 1 = true; 0 = false*)

7.**Rest_ecg**: Resting electrocardiographic measurement (*0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria*)

8.**Thalach**: The person's maximum heart rate achieved

9.**Exang**: Exercise induced angina (*1 = yes; 0 = no*)

10.**Oldpeak**: ST depression induced by exercise relative to rest (*'ST' relates to positions on the ECG plot.*)

11.**Slope**: the slope of the peak exercise ST segment (*Value 1: upsloping, Value 2: flat, Value 3: downsloping*)

12.**C.A**: The number of major vessels (*0-3*)

13. **Thal**: A blood disorder called thalassemia (*3 = normal; 6 = fixed defect; 7 = reversable defect*)

14. **Target**: Heart disease (*0 = no, 1 = yes*)

In [8]:
```
hd.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Age	302.0	54.420530	9.047970	29.0	48.00	55.5	61.00	77.0
Gender	302.0	0.682119	0.466426	0.0	0.00	1.0	1.00	1.0
C.P	302.0	0.963576	1.032044	0.0	0.00	1.0	2.00	3.0
Trestbps	302.0	131.602649	17.563394	94.0	120.00	130.0	140.00	200.0
Chol	302.0	246.500000	51.753489	126.0	211.00	240.5	274.75	564.0
F.B.S	302.0	0.149007	0.356686	0.0	0.00	0.0	0.00	1.0
Restecg	302.0	0.526490	0.526027	0.0	0.00	1.0	1.00	2.0
Thalach	302.0	149.569536	22.903527	71.0	133.25	152.5	166.00	202.0
Exang	302.0	0.327815	0.470196	0.0	0.00	0.0	1.00	1.0
Old peak	302.0	1.043046	1.161452	0.0	0.00	0.8	1.60	6.2
Slope	302.0	1.397351	0.616274	0.0	1.00	1.0	2.00	2.0
C.A	302.0	0.718543	1.006748	0.0	0.00	0.0	1.00	4.0
Thal	302.0	2.314570	0.613026	0.0	2.00	2.0	3.00	3.0
Target	302.0	0.543046	0.498970	0.0	0.00	1.0	1.00	1.0

In [9]:
```
hd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 302 entries, 0 to 878
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Age       302 non-null    int64  
 1   Gender    302 non-null    int64  
 2   C.P       302 non-null    int64  
 3   Trestbps  302 non-null    int64  
 4   Chol      302 non-null    int64  
 5   F.B.S     302 non-null    int64  
 6   Restecg   302 non-null    int64  
 7   Thalach   302 non-null    int64  
 8   Exang     302 non-null    int64  
 9   Old peak  302 non-null    float64
 10  Slope     302 non-null    int64  
 11  C.A       302 non-null    int64  
 12  Thal      302 non-null    int64  
 13  Target    302 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 35.4 KB
```

In [10]:
```
hd.isnull().sum()
```

Out[10]:
```
Age         0
Gender      0
C.P         0
Trestbps    0
Chol        0
F.B.S       0
Restecg     0
Thalach     0
Exang       0
Old peak    0
Slope       0
C.A         0
Thal        0
Target      0
dtype: int64
```

In [11]:
```
hd
```

Out[11]:

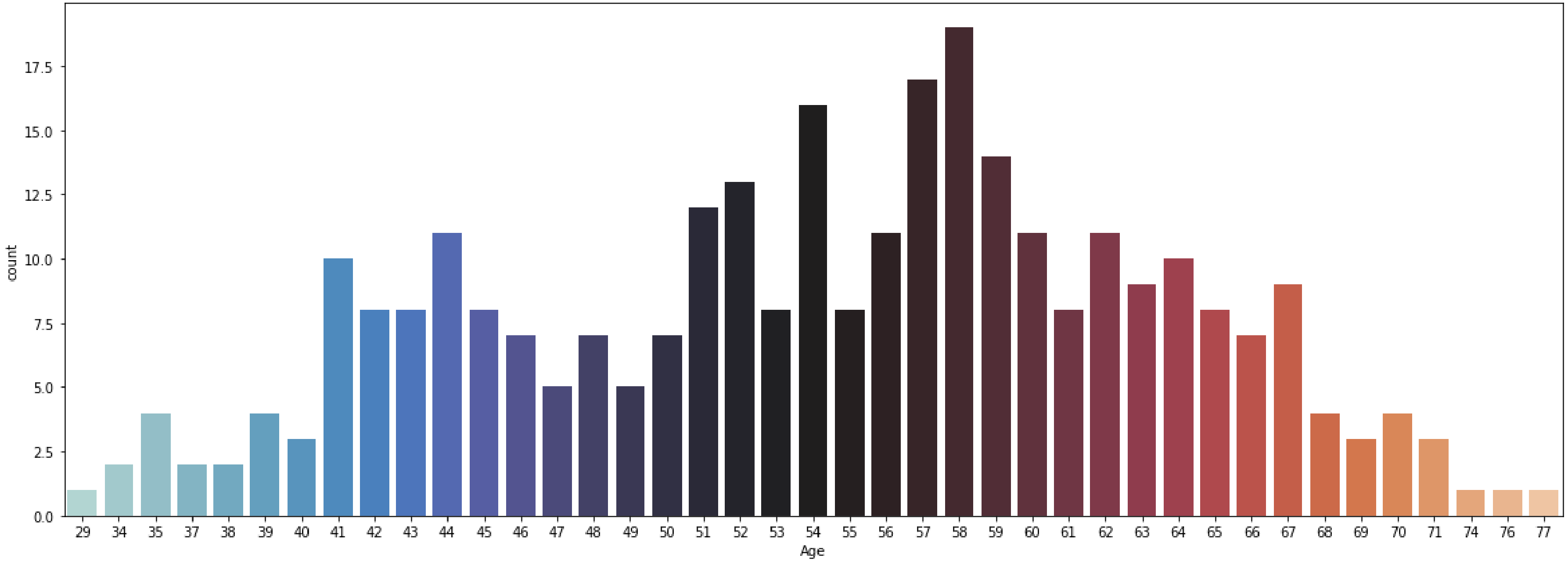
	Age	Gender	C.P	Trestbps	Chol	F.B.S	Restecg	Thalach	Exang	Old peak	Slope	C.A	Thal	Target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
723	68	0	2	120	211	0	0	115	0	1.5	1	0	2	1
733	44	0	2	108	141	0	1	175	0	0.6	1	0	2	1
739	52	1	0	128	255	0	1	161	1	0.0	2	1	3	0
843	59	1	3	160	273	0	0	125	0	0.0	2	0	2	0
878	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

302 rows × 14 columns

Countplot for the following :

1.Person's Age

In [12]:
```
plt.figure(figsize = (20,7))
sns.countplot(x='Age', data=hd, palette='icefire')
plt.show()
```

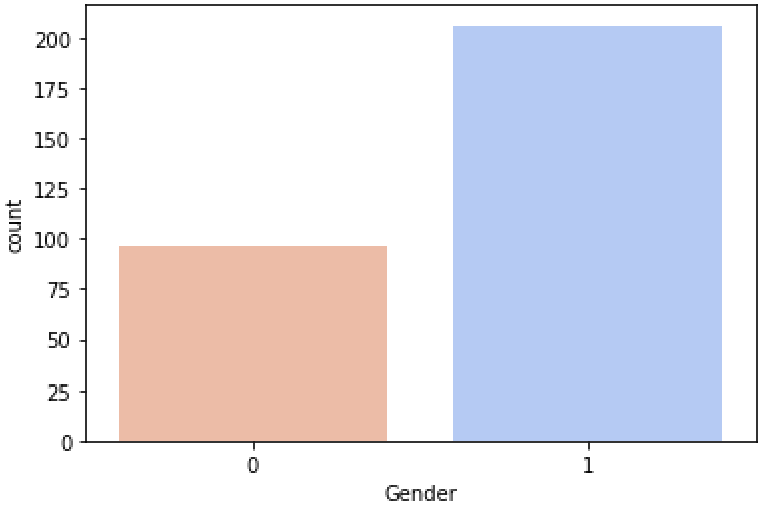


2. Gender of the person

In [13]:
```
hd['Gender'].value_counts()
```

Out[13]:
```
1    206
0     96
Name: Gender, dtype: int64
```

In [14]:
```
sns.countplot(x='Gender', data=hd, palette='coolwarm_r')
plt.show()
```



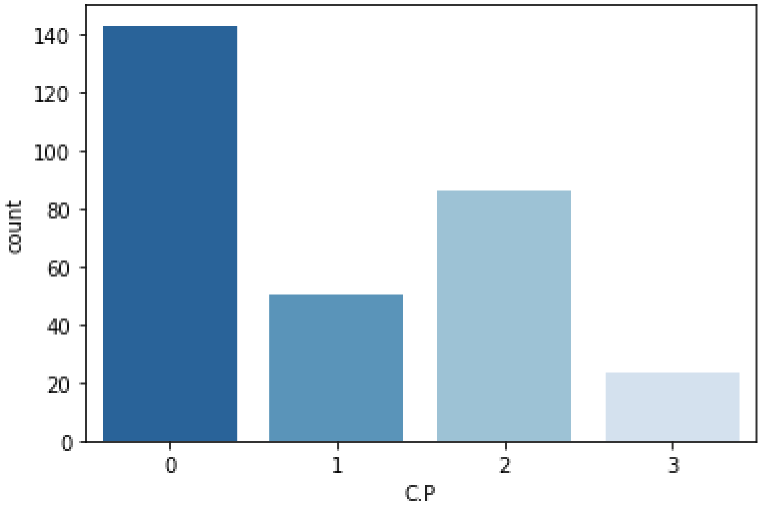
0 => female , 1 = male

3. Chest Pain experienced

In [15]:
```
hd['C.P'].value_counts()
```

Out[15]:
```
0    143
2     86
1     50
3     23
Name: C.P, dtype: int64
```

In [16]:
```
sns.countplot(x='C.P', data=hd, palette='Blues_r')
plt.show()
```



C.P Values :

0 = typical angina,

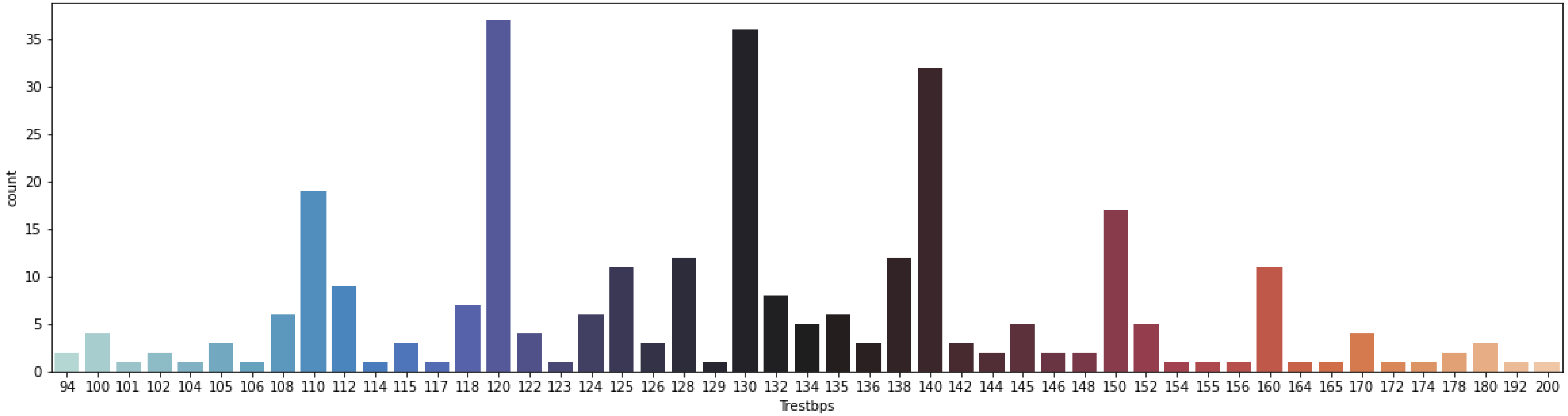
1 = atypical angina,

2 = non-anginal pain,

3 = asymptotic

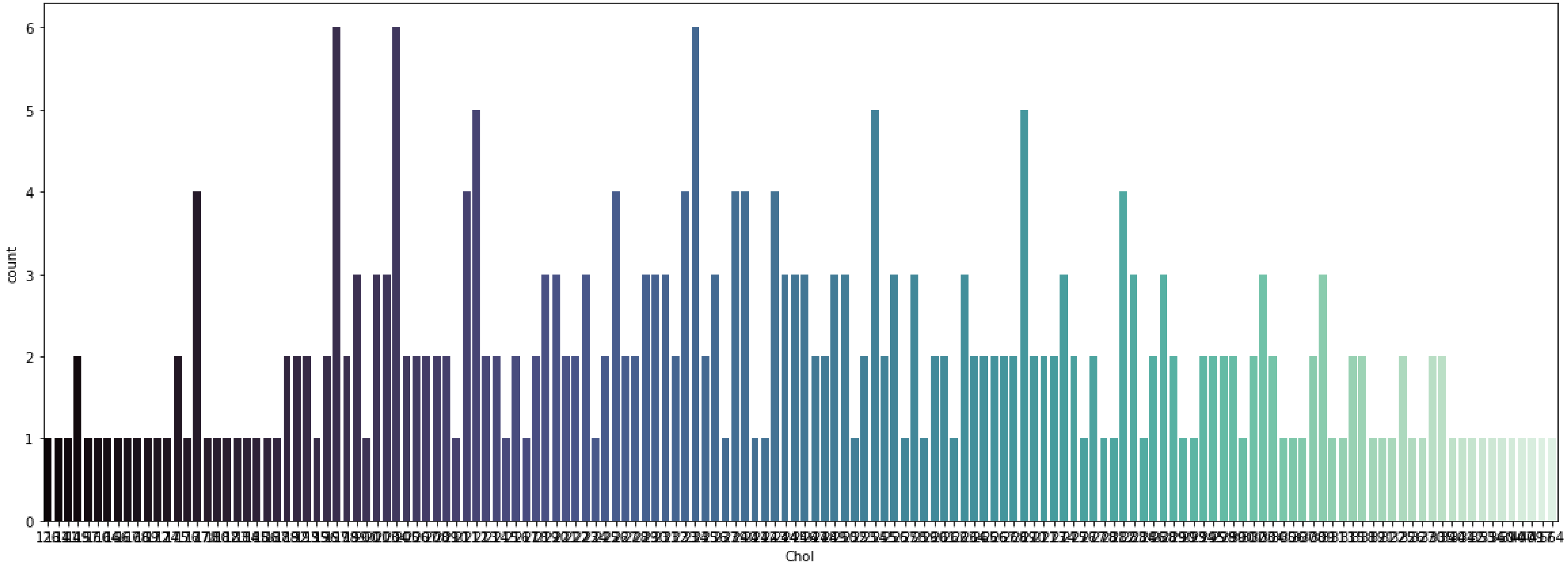
4.Person's Resting Blood Pressure

In [17]:
```
plt.figure(figsize = (20,5))
sns.countplot(x='Trestbps', data=hd, palette='icefire')
plt.show()
```



5. Cholestrol level of the person

In [18]:
```
plt.figure(figsize = (20,7))
sns.countplot(x='Chol', data=hd, palette='mako')
plt.show()
```

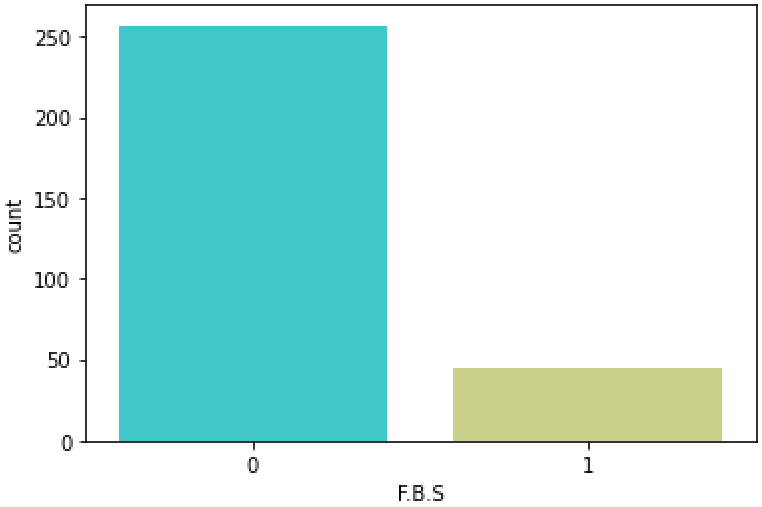


6. Fasting Blood Sugar

In [19]:
```
hd['F.B.S'].value_counts()
```

Out[19]:
```
0    257
1     45
Name: F.B.S, dtype: int64
```

In [20]:
```
sns.countplot(x='F.B.S', data=hd, palette='rainbow')
plt.show()
```



F.B.S

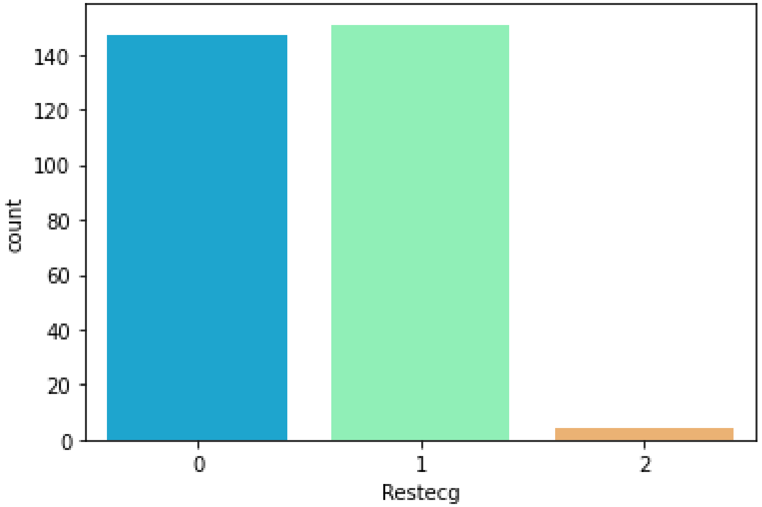
0 = false **1** = true

7.Resting electrocardiographic measurement

In [21]:
```
hd['Restecg'].value_counts()
```

Out[21]:
```
1    151
0    147
2      4
Name: Restecg, dtype: int64
```

In [22]:
```
sns.countplot(x='Restecg', data=hd, palette='rainbow')
plt.show()
```



Restecg (Resting electrocardiographic measurement)

0 = normal,

1 = having abnormality,

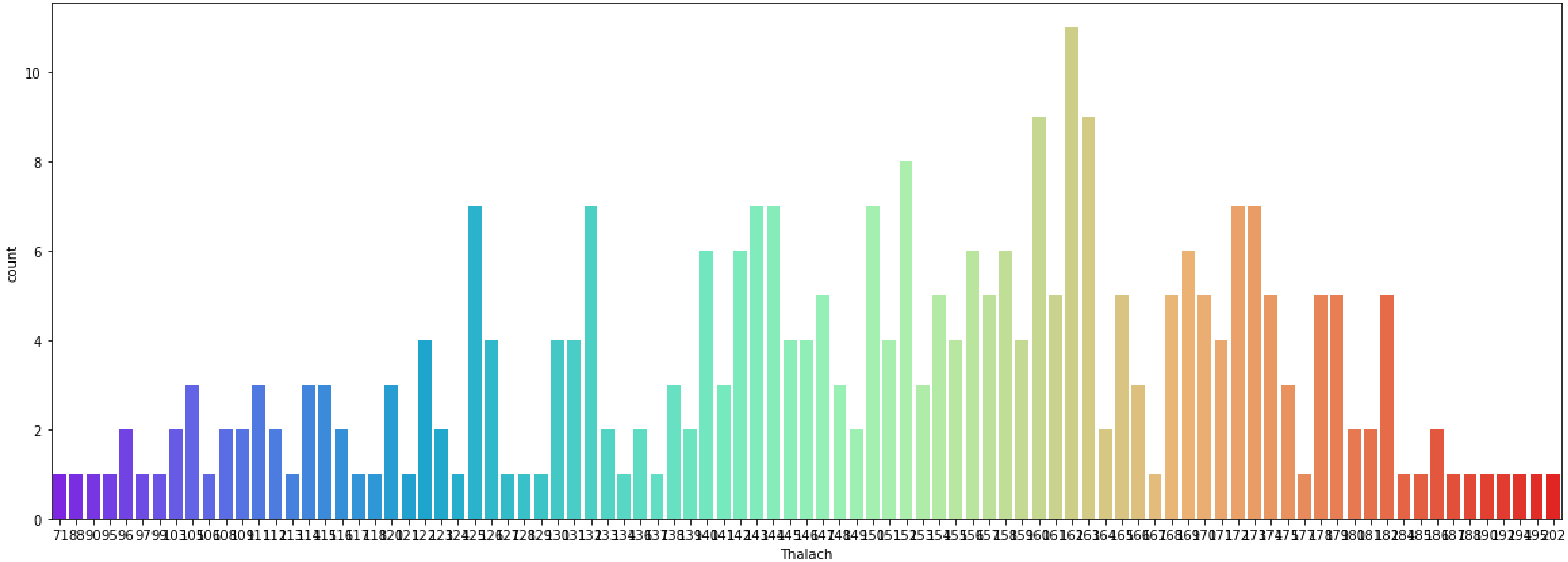
2 = showing probability

8.Person's maximum heart rate achieved

In [23]:
```
hd['Thalach'].value_counts()
```

Out[23]:
```
162    11
163     9
160     9
152     8
144     7
       ..
167     1
134     1
177     1
95      1
113     1
Name: Thalach, Length: 91, dtype: int64
```

In [24]:
```
plt.figure(figsize = (20,7))
sns.countplot(x='Thalach', data=hd, palette='rainbow')
plt.show()
```

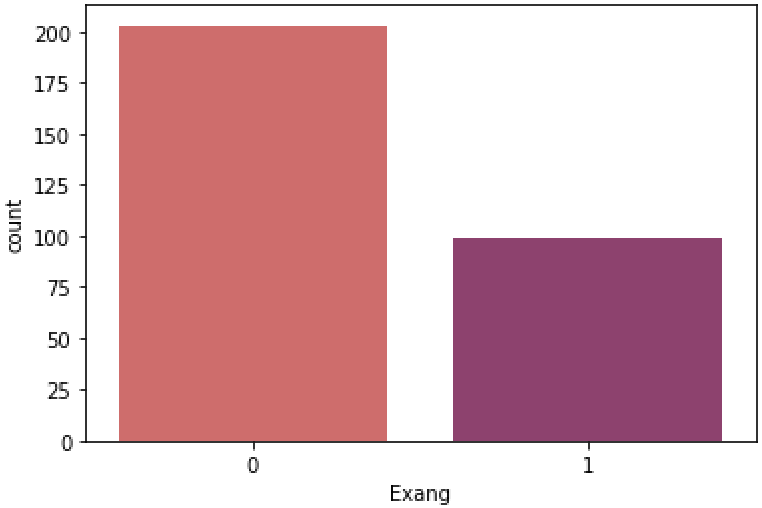


9.Exercise induced angina

In [25]:
```
hd['Exang'].value_counts()
```

Out[25]:
```
0    203
1     99
Name: Exang, dtype: int64
```

In [26]:
```
sns.countplot(x='Exang', data=hd, palette='flare')
plt.show()
```



0 = no , 1 = yes

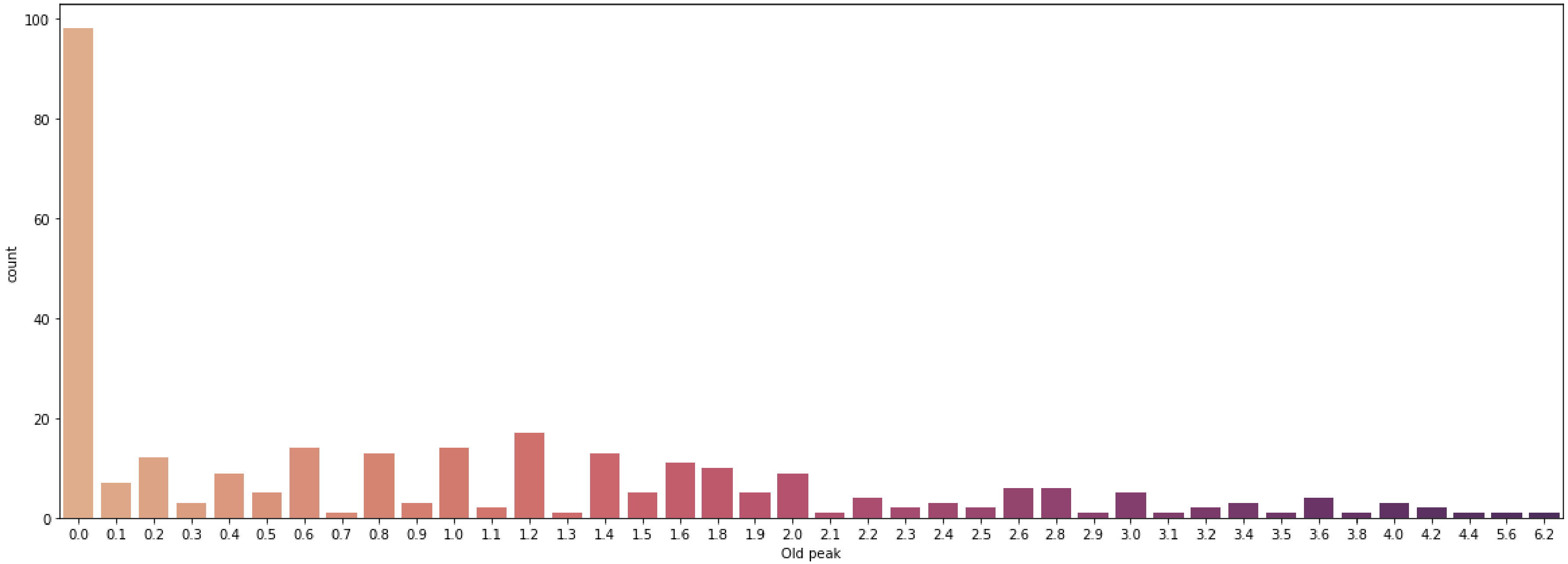
10. Old peak

Depression induced by exercise relative to rest

In [27]:
```
hd['Old peak'].value_counts
```

Out[27]:
```
<bound method IndexOpsMixin.value_counts of 0      1.0
1      3.1
2      2.6
3      0.0
4      1.9
      ...
723    1.5
733    0.6
739    0.0
843    0.0
878    1.4
Name: Old peak, Length: 302, dtype: float64>
```

In [28]:
```
plt.figure(figsize = (20,7))
sns.countplot(x='Old peak', data=hd, palette='flare')
plt.show()
```

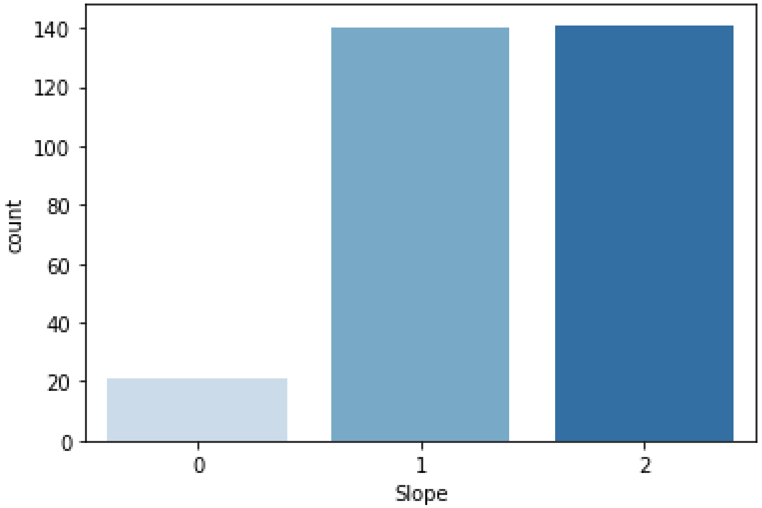


11. Slope

In [29]:
```
hd['Slope'].value_counts
```

Out[29]:
```
<bound method IndexOpsMixin.value_counts of 0      2
1      0
2      0
3      2
4      1
      ..
723    1
733    1
739    2
843    2
878    1
Name: Slope, Length: 302, dtype: int64>
```

In [30]:
```
sns.countplot(x='Slope', data=hd, palette='Blues')
plt.show()
```



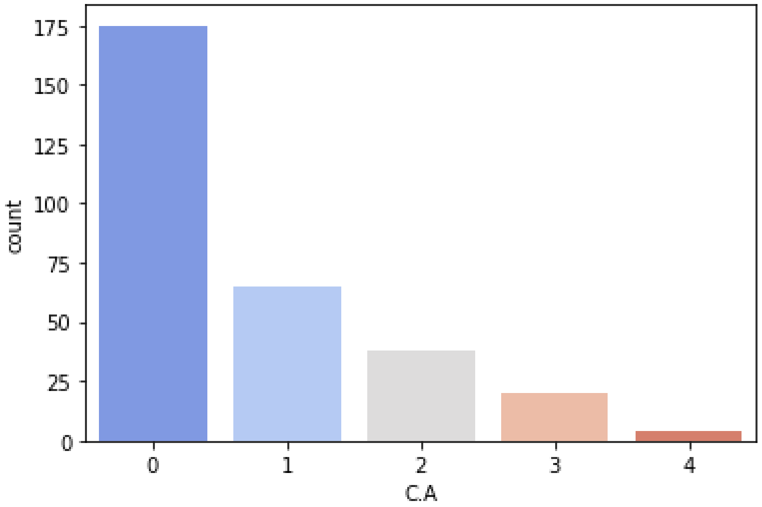
Values :
1: upsloping,
2: flat,
3: downsloping

12.Number of major vessels (CA)

In [31]:
```
hd['C.A'].value_counts
```

Out[31]:
```
<bound method IndexOpsMixin.value_counts of 0      2
1      0
2      0
3      1
4      3
      ..
723    0
733    0
739    1
843    0
878    1
Name: C.A, Length: 302, dtype: int64>
```

In [32]:
```
sns.countplot(x='C.A', data=hd, palette='coolwarm')
plt.show()
```

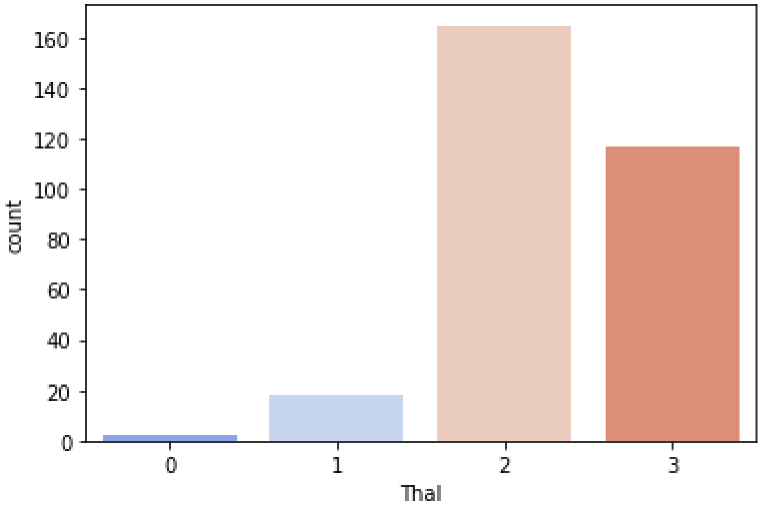


13.Thalassemia

In [33]:
```
hd['Thal'].value_counts
```

Out[33]:
```
<bound method IndexOpsMixin.value_counts of 0      3
1      3
2      3
3      3
4      2
      ..
723    2
733    2
739    3
843    2
878    3
Name: Thal, Length: 302, dtype: int64>
```

In [34]:
```
sns.countplot(x='Thal', data=hd, palette='coolwarm')
plt.show()
```



0 = normal

1 = fixed defect

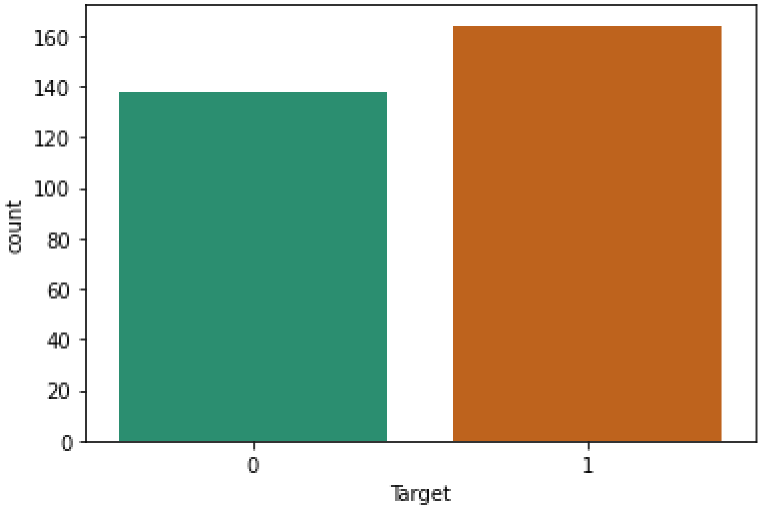
2 = reversable defect

14.Target

In [35]:
```
hd['Target'].value_counts()
```

Out[35]:
```
1    164
0    138
Name: Target, dtype: int64
```

In [36]:
```
sns.countplot(x='Target', data=hd, palette='Dark2')
plt.show()
```



0 => Healthy heart ,
1 => Defective Heart*

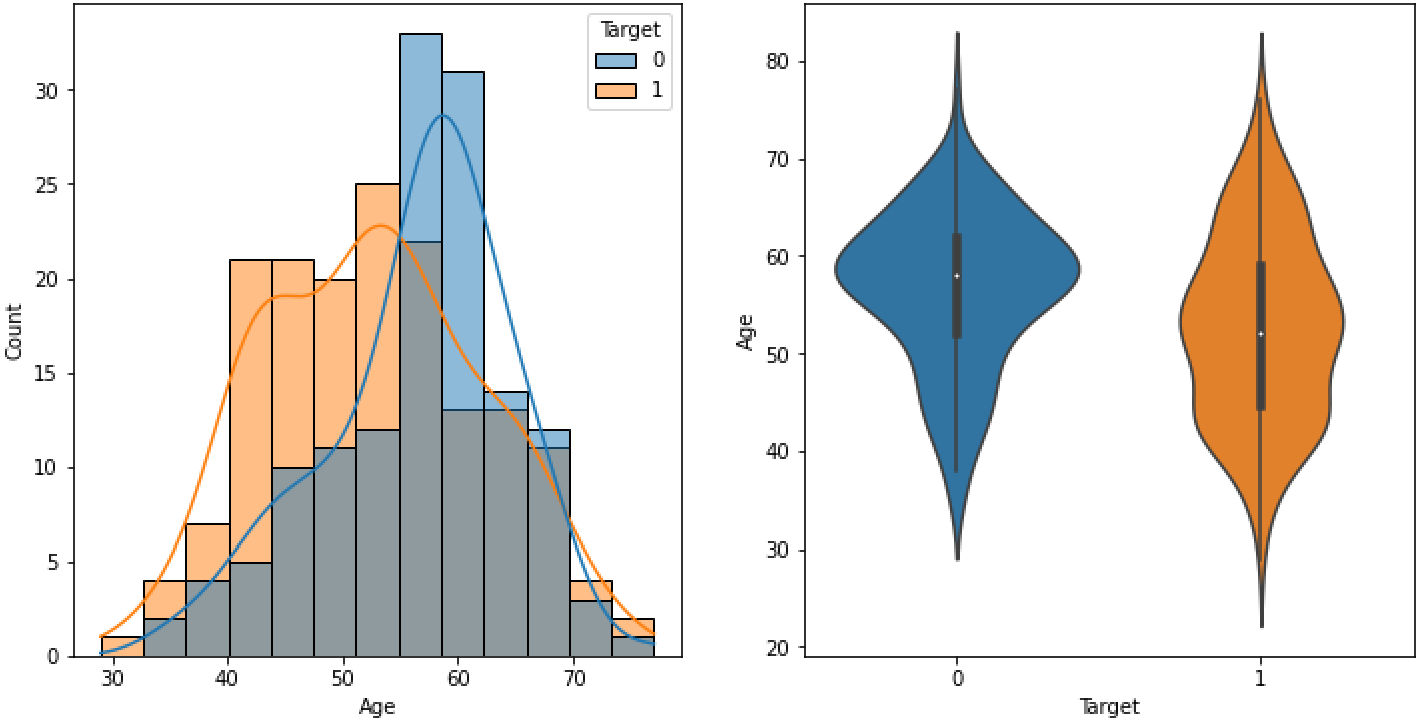
In [37]:
```
hd.groupby('Target').mean()
```

Out[37]:

Target	Age	Gender	C.P	Trestbps	Chol	F.B.S	Restecg	Thalach	Exang	Old peak	Slope	C.A	Thal
0	56.601449	0.826087	0.478261	134.398551	251.086957	0.159420	0.449275	139.101449	0.550725	1.585507	1.166667	1.166667	2.543478
1	52.585366	0.560976	1.371951	129.250000	242.640244	0.140244	0.591463	158.378049	0.140244	0.586585	1.591463	0.341463	2.121951

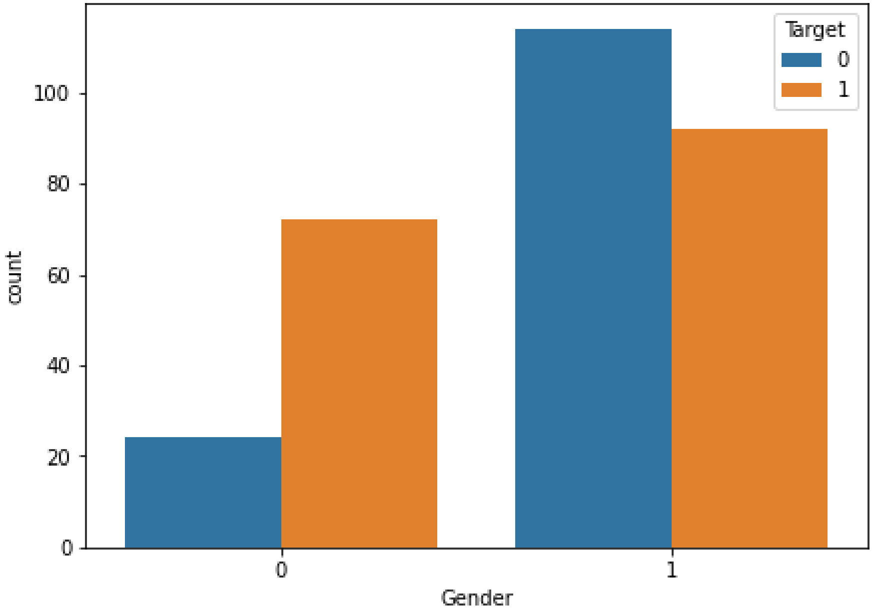
In [38]:
```
fig, ax = plt.subplots(1,2, figsize=(12,6))
sns.histplot(x='Age', data=hd, kde=True, hue='Target', ax=ax[0])
sns.violinplot(x='Target', data=hd, y='Age', ax=ax[1])
```

Out[38]:
```
<AxesSubplot:xlabel='Target', ylabel='Age'>
```



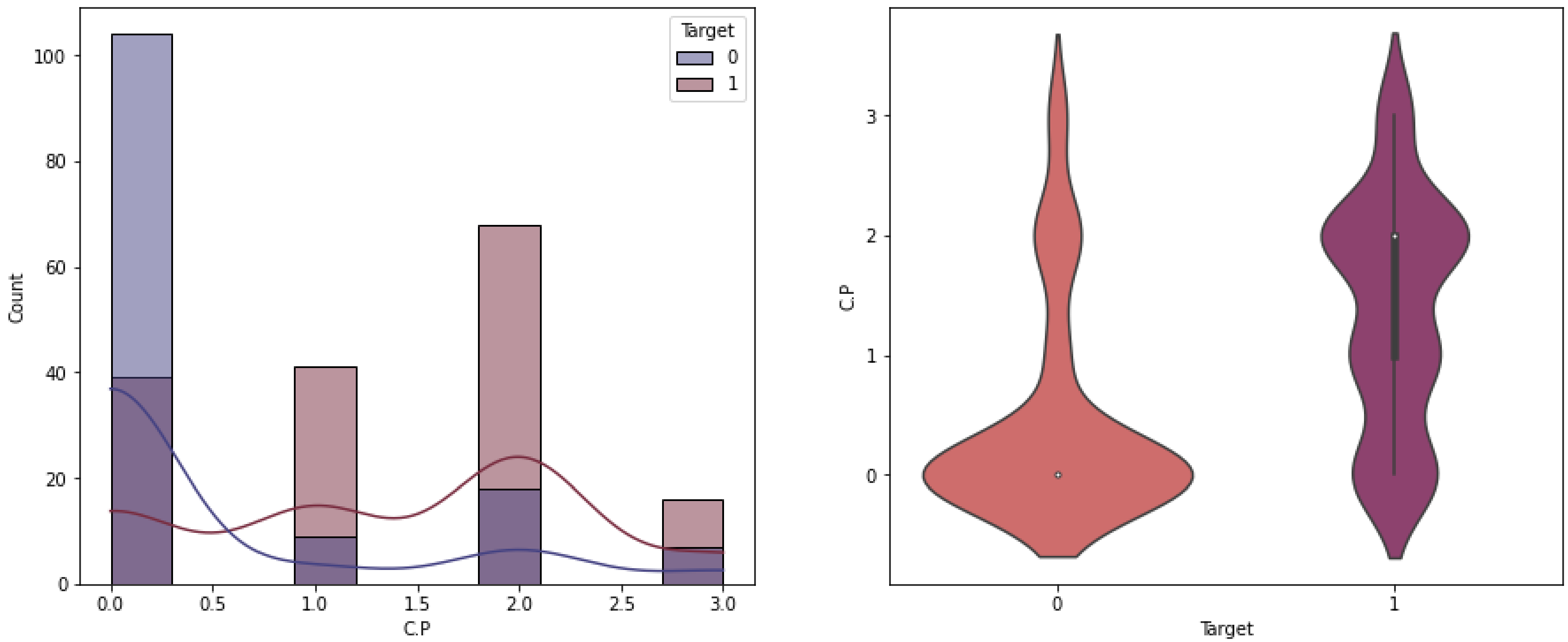
In [39]:
```
fig, ax= plt.subplots(1,1, figsize=(7,5))
sns.countplot(x='Gender', data=hd, hue='Target')
```

Out[39]: <AxesSubplot:xlabel='Gender', ylabel='count'>



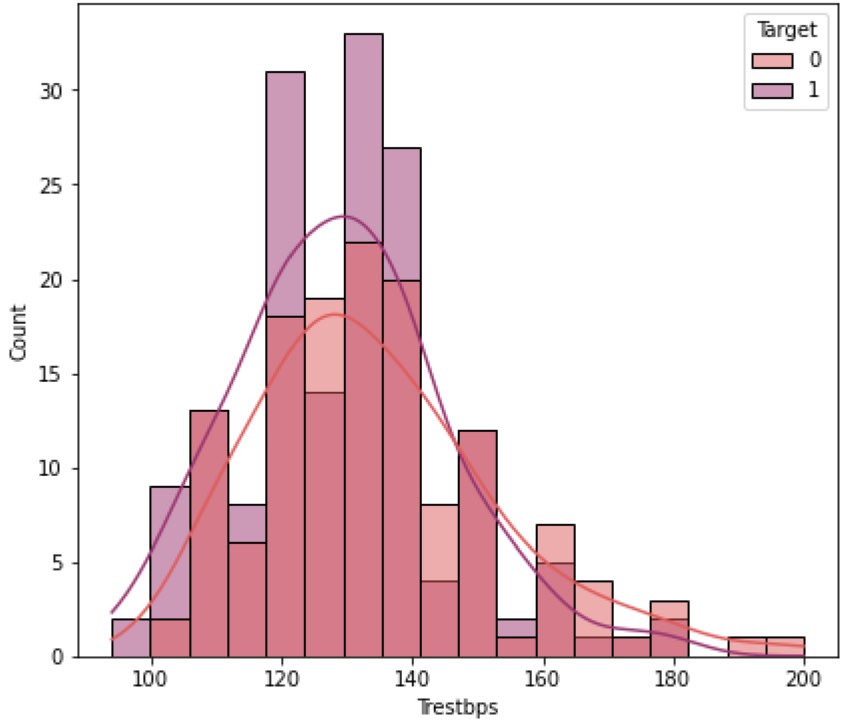
In [40]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='C.P', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'icefire')
sns.violinplot(x='Target', data=hd, y = 'C.P', ax=ax[1], palette = 'flare')
```

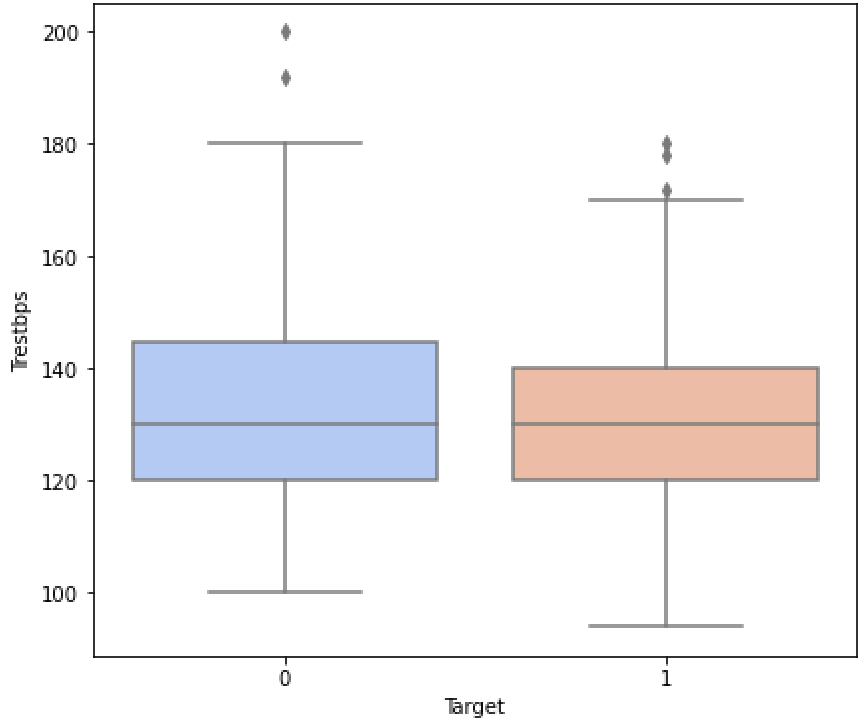
Out[40]:
```
<AxesSubplot:xlabel='Target', ylabel='C.P'>
```



In [41]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Trestbps', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'flare')
sns.boxplot(x='Target', data=hd, y = 'Trestbps', ax=ax[1], palette = 'coolwarm')
```

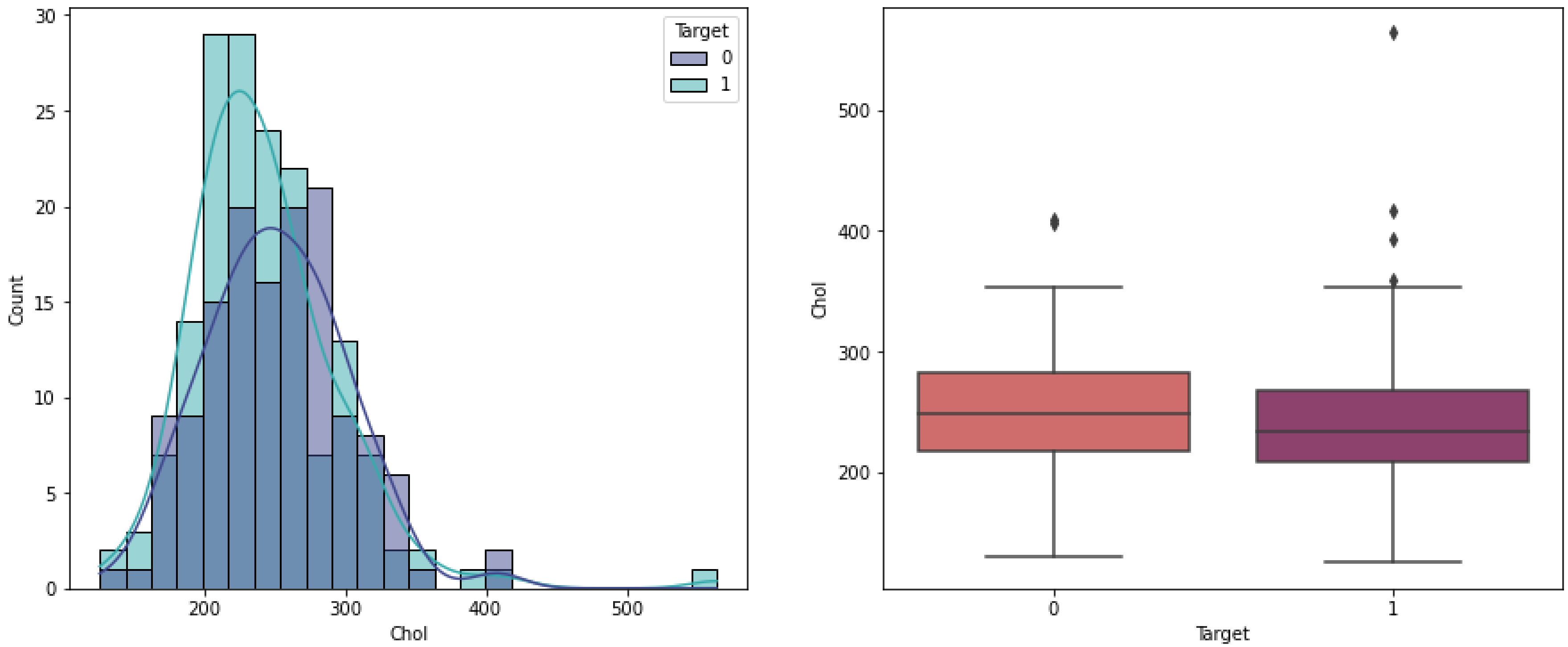
Out[41]: <AxesSubplot:xlabel='Target', ylabel='Trestbps'>





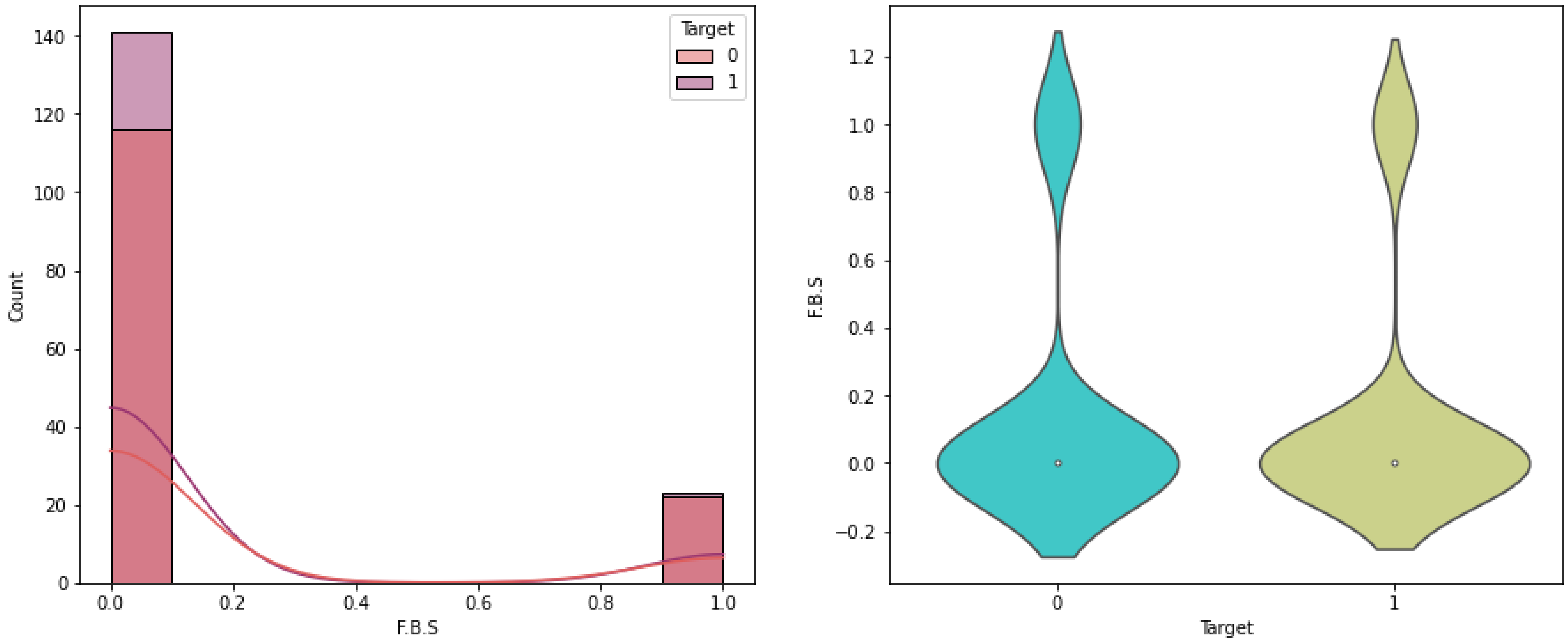
In [42]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Chol', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'mako')
sns.boxplot(x='Target', data=hd, y = 'Chol', ax=ax[1], palette = 'flare')
```

Out[42]: <AxesSubplot:xlabel='Target', ylabel='Chol'>



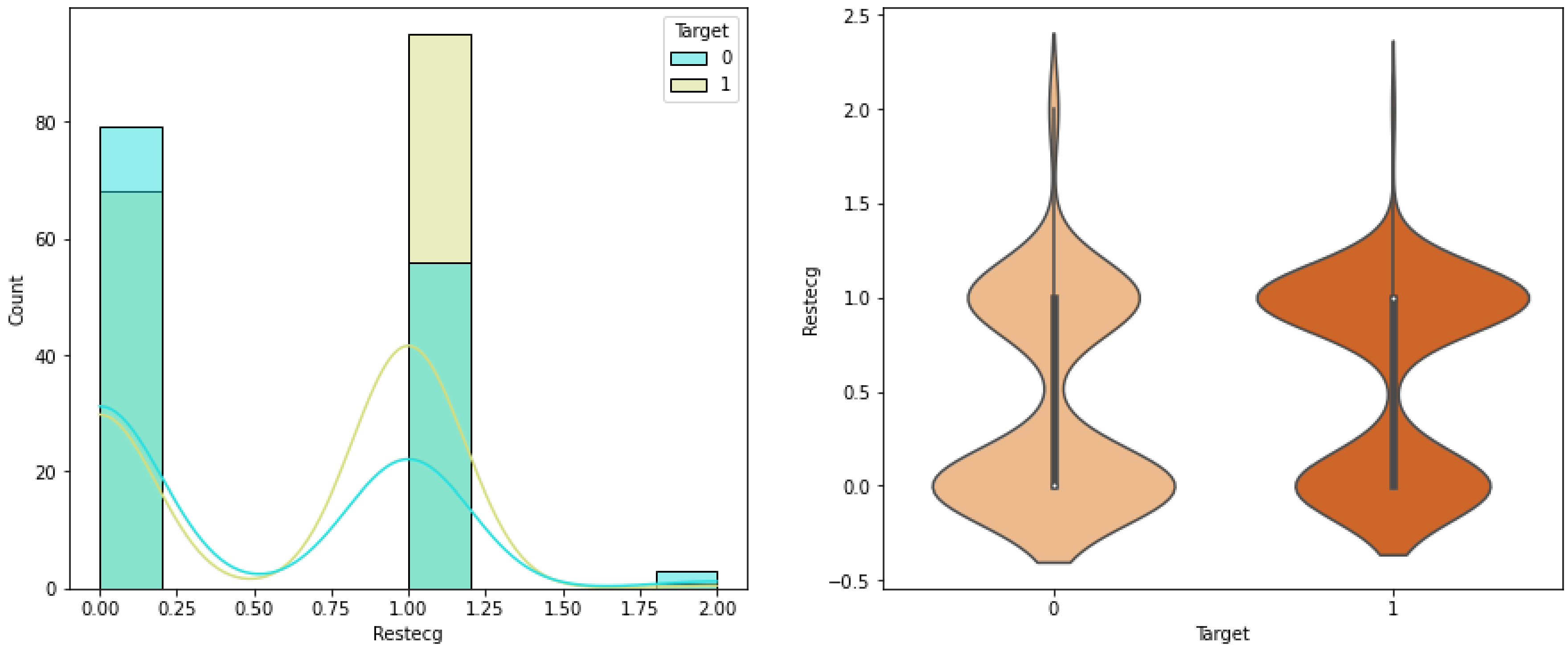
In [43]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='F.B.S', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'flare')
sns.violinplot(x='Target', data=hd, y = 'F.B.S', ax=ax[1], palette = 'rainbow')
```

Out[43]: <AxesSubplot:xlabel='Target', ylabel='F.B.S'>



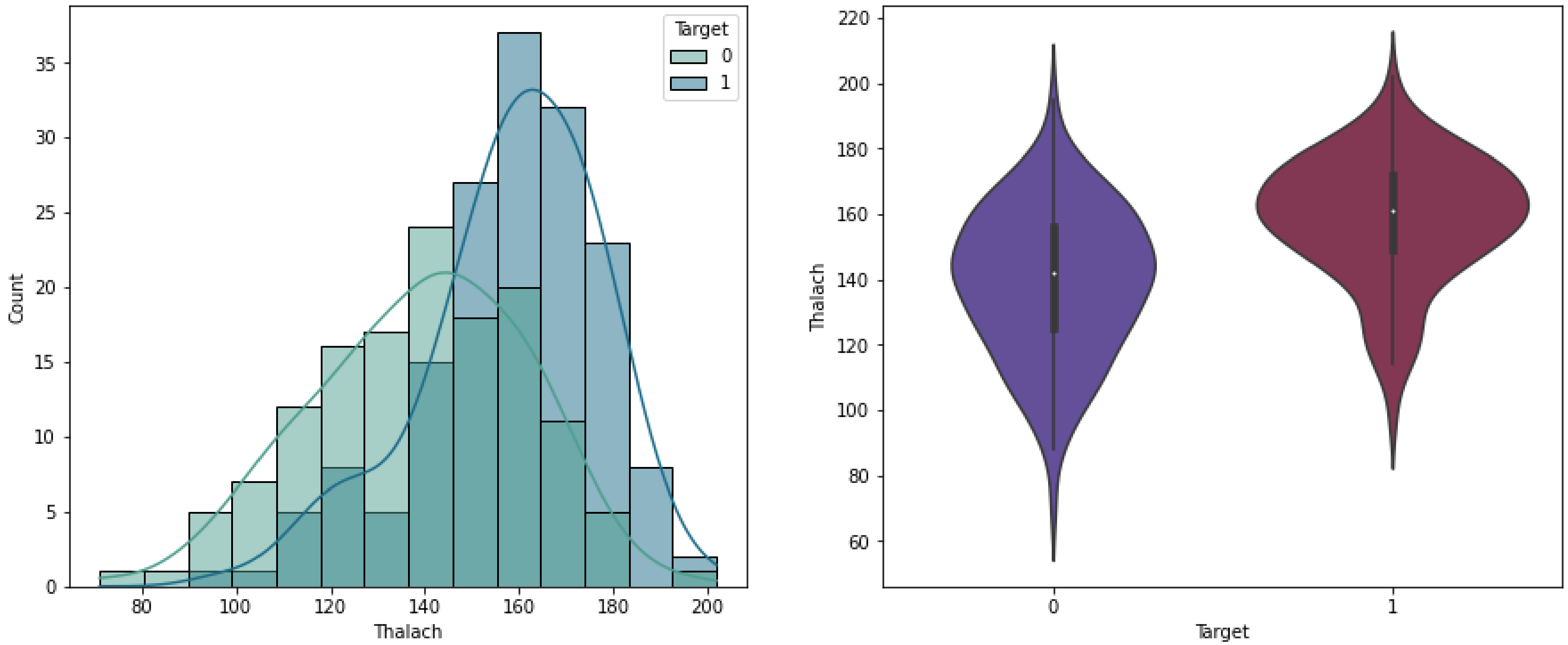
In [44]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Restecg', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'rainbow')
sns.violinplot(x='Target', data=hd, y = 'Restecg', ax=ax[1], palette = 'Oranges')
```

Out[44]: <AxesSubplot:xlabel='Target', ylabel='Restecg'>



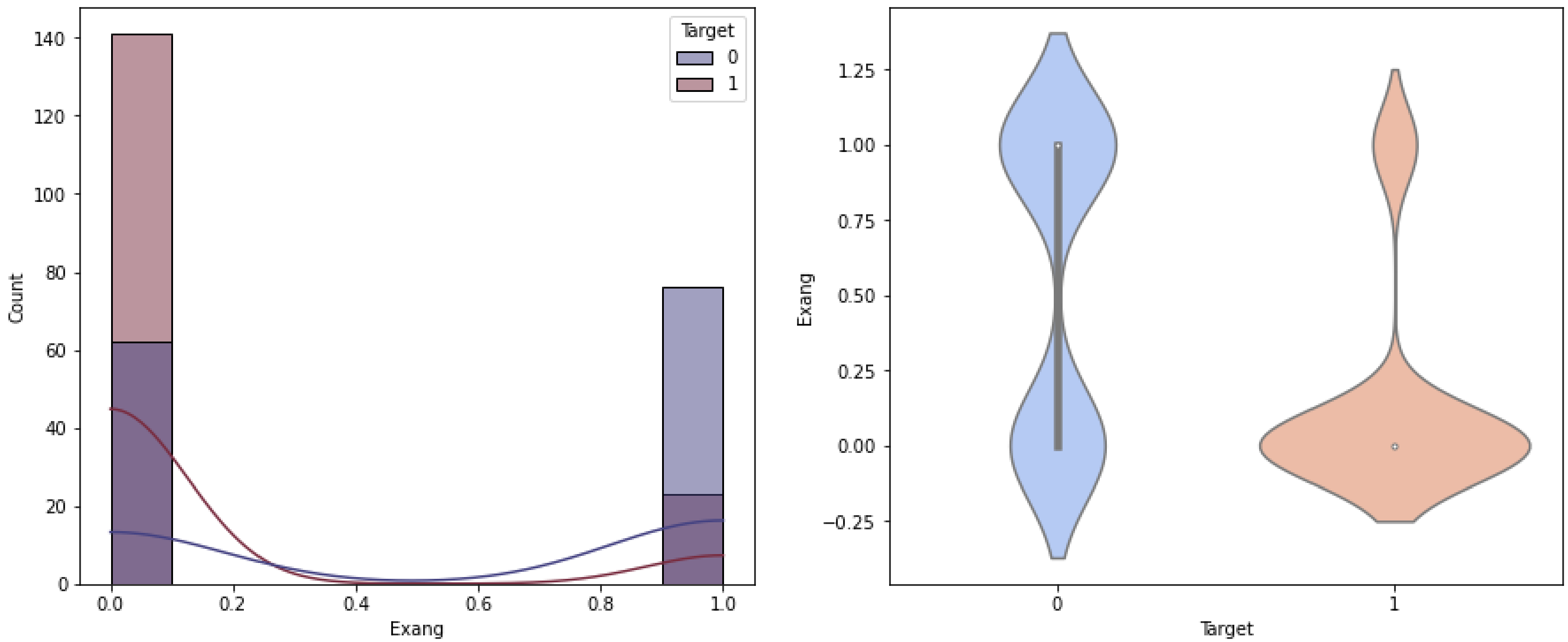
In [45]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Thalach', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'crest')
sns.violinplot(x='Target', data=hd, y = 'Thalach', ax=ax[1], palette = 'twilight')
```

Out[45]: <AxesSubplot:xlabel='Target', ylabel='Thalach'>



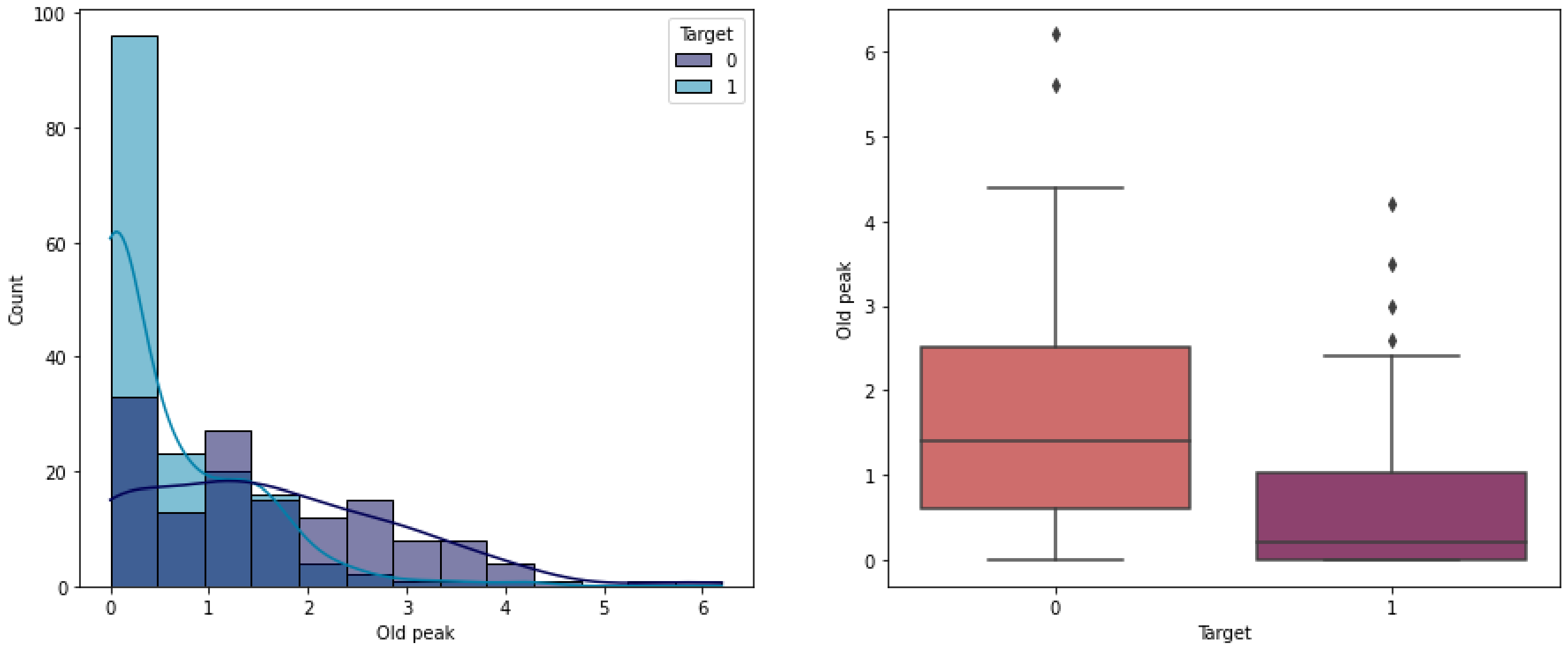
In [46]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Exang', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'icefire')
sns.violinplot(x='Target', data=hd, y = 'Exang', ax=ax[1], palette = 'coolwarm')
```

Out[46]: <AxesSubplot:xlabel='Target', ylabel='Exang'>



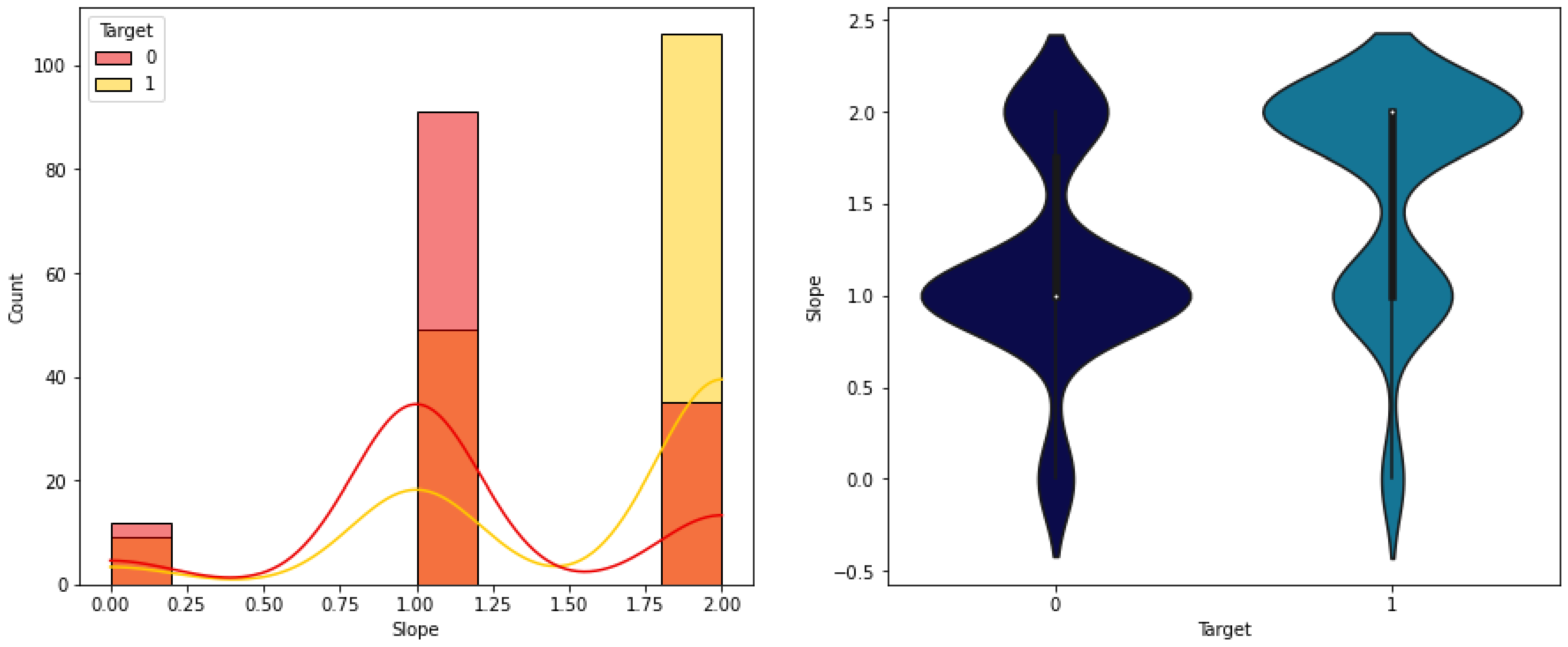
In [47]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Old peak', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'ocean')
sns.boxplot(x='Target', data=hd, y = 'Old peak', ax=ax[1], palette = 'flare')
```

Out[47]: <AxesSubplot:xlabel='Target', ylabel='Old peak'>



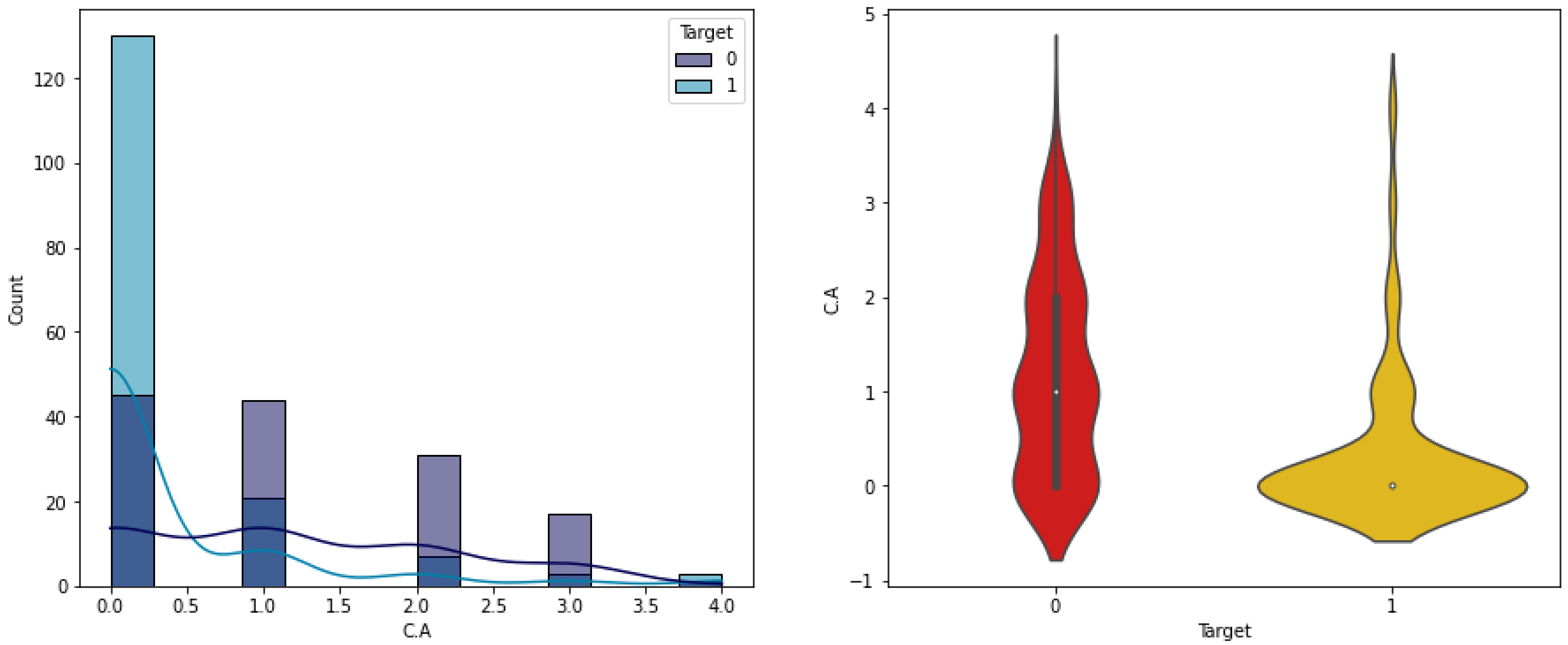
In [48]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Slope', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'hot')
sns.violinplot(x='Target', data=hd, y = 'Slope', ax=ax[1], palette = 'ocean')
```

Out[48]:
```
<AxesSubplot:xlabel='Target', ylabel='Slope'>
```



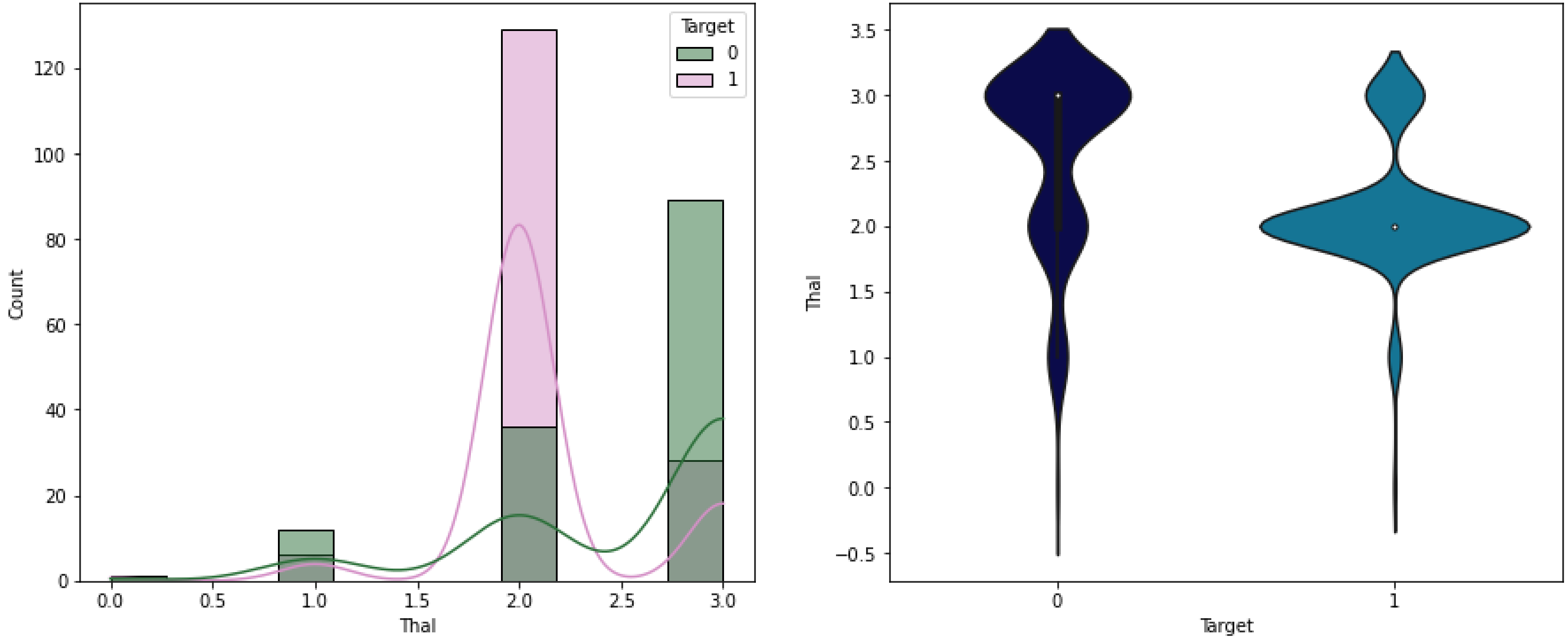
In [49]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='C.A', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'ocean')
sns.violinplot(x='Target', data=hd, y = 'C.A', ax=ax[1], palette = 'hot')
```

Out[49]: <AxesSubplot:xlabel='Target', ylabel='C.A'>



In [50]:
```
fig, ax = plt.subplots(1,2, figsize=(15,6))
sns.histplot(x='Thal', data=hd, hue='Target', kde=True, ax=ax[0], palette= 'cubehelix')
sns.violinplot(x='Target', data=hd, y = 'Thal', ax=ax[1], palette = 'ocean')
```

Out[50]: <AxesSubplot:xlabel='Target', ylabel='Thal'>



Overview of the Data Set

In [51]:
```
prof = pandas_profiling.ProfileReport(hd)
prof
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	14
Number of observations	302
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	43.5 KiB
Average record size in memory	147.5 B

Variable types

Numeric	5
Categorical	9

Alerts

`C.P` is highly overall correlated with `Exang` and 1 other fields (Exang, Target)	**High correlation**
`Thal` is highly overall correlated with `Gender` and 1 other fields (Gender, Target)	**High correlation**
`Target` is highly overall correlated with `C.P` and 3 other fields (C.P, Thalach, Exang, Thal)	**High correlation**
`Old peak` is highly overall correlated with `Restecg` and 1 other fields (Restecg, Slope)	**High correlation**

Out[51]:

Slope is highly overall correlated with Old peak	High correlation
Age is highly overall correlated with Thalach	High correlation

Splitting the Features and the Target values

In [52]:
```
X = hd.drop(columns='Target',axis=1)
Y = hd['Target']
```

In [53]:
```
print(X)
```

```
     Age  Gender  C.P  Trestbps  Chol  F.B.S  Restecg  Thalach  Exang  \
0     52       1    0       125   212      0        1      168      0   
1     53       1    0       140   203      1        0      155      1   
2     70       1    0       145   174      0        1      125      1   
3     61       1    0       148   203      0        1      161      0   
4     62       0    0       138   294      1        1      106      0   
..   ...     ...  ...       ...   ...    ...      ...      ...    ...   
723   68       0    2       120   211      0        0      115      0   
733   44       0    2       108   141      0        1      175      0   
739   52       1    0       128   255      0        1      161      1   
843   59       1    3       160   273      0        0      125      0   
878   54       1    0       120   188      0        1      113      0   

     Old peak  Slope  C.A  Thal  
0         1.0      2    2     3  
1         3.1      0    0     3  
2         2.6      0    0     3  
3         0.0      2    1     3  
4         1.9      1    3     2  
..        ...    ...  ...   ...  
723       1.5      1    0     2  
733       0.6      1    0     2  
739       0.0      2    1     3  
843       0.0      2    0     2  
878       1.4      1    1     3  

[302 rows x 13 columns]
```

In [54]:
```
print(Y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
723    1
733    1
739    0
843    0
878    0
Name: Target, Length: 302, dtype: int64
```

Splitting the data into training data and test data

In [55]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.2, stratify = Y, random_state = 2)
```

In [56]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(302, 13) (241, 13) (61, 13)
```

Model Training

1.Logistic Regression

This type of statistical model is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring.

In [57]:
```
model = LogisticRegression()
```

In [58]:
```
#training the logistic regression model with training data
model.fit(X_train, Y_train)
```

```
C:\Users\HP1\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/m
odules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[58]:
```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

Checking the accuracy Score

In [59]:
```
#accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [60]:
```
print('Accuracy on training data : ', training_data_accuracy)
```

```
Accuracy on training data :  0.8506224066390041
```

The accuracy score for the training data is 85%

In [61]:
```
#accuracy on test data
X_test_prediction = model.predict(X_test)
LR_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [62]:
```
print('Accuracy on test data : ', LR_accuracy)
```

```
Accuracy on test data :  0.9016393442622951
```

The accuracy score for the test data is 90%

Building a Predictive system

In [63]:
```
input_data = (71,0,0,112,149,0,1,125,0,1.6,1,0,2)

#Changing the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

#reshapping the numpy array as we are predicting for only one instance
input_data_reshape = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshape)
print(prediction)

if(prediction[0]==0):
    print('The person is healthy')
else:
    print('The person has a heart disease')
```

```
[1]
The person has a heart disease

C:\Users\HP1\anaconda3\lib\site-packages\sklearn\base.py:409: UserWarning: X does not have valid feature names, but
LogisticRegression was fitted with feature names
  warnings.warn(
```

2. Decision Tree

Creating the object for the model

In [64]:
```
dtc = DecisionTreeClassifier(random_state=0)
```

Fitting the Data to the Object

In [67]:
```
parameters = {'max_features': ['log2','sqrt','auto'],'criterion':['entropy'],
              'max_depth':[5,10,25,35,50],'min_samples_split':[10,20,30,50,100],
              'min_samples_leaf':[2,3,5]}

grid_obj = GridSearchCV(dtc,parameters)
grid_obj = grid_obj.fit(X_train, Y_train)

dtc = grid_obj.best_estimator_
dtc.fit(X_train, Y_train)
```

```
C:\Users\HP1\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269: FutureWarning: `max_features='auto'` has b
een deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqr
t'`.
  warnings.warn(
C:\Users\HP1\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269: FutureWarning: `max_features='auto'` has b
een deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqr
t'`.
  warnings.warn(
C:\Users\HP1\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269: FutureWarning: `max_features='auto'` has b
een deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqr
t'`.
  warnings.warn(
C:\Users\HP1\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269: FutureWarning: `max_features='auto'` has b
een deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqr
t'`.
  warnings.warn(
C:\Users\HP1\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269: FutureWarning: `max_features='auto'` has b
een deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqr
t'`.
```

In [68]:
```
dtc.tree_.max_depth
```

Out[68]: 5

Prediction on Test Data

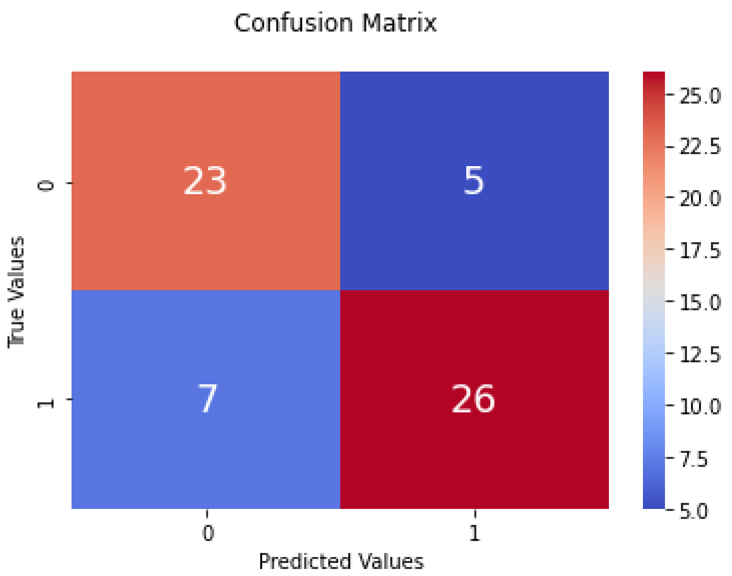
In [69]:
```
Y_pred = dtc.predict(X_test)
```

Confusion Matrix

A confusion matrix is a table that is used to define(visualize and summarize) the performance of a classification algorithm.

In [70]:
```
cm = confusion_matrix(Y_test, Y_pred)  # comparing the actual values with the predicted values
sns.heatmap(cm, annot=True, cmap= 'coolwarm', annot_kws={'size':19})  #  representing the confusion matrix

plt.title('Confusion Matrix \n')
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.show()
```



Checking classification Report

Checking the Accuracy of the Model

In [71]:
```
X_test_pred = dtc.predict(X_test)
DT_accuracy = accuracy_score(X_test_pred,Y_test)
```

In [72]:
```
print(DT_accuracy)
```

```
0.8032786885245902
```

The accuracy of the decision tree model is 80%

3. Support Vector Machine (SVM)

Creating the object for the model

In [73]:
```
svm = svm.SVC(kernel='linear')
```

Fitting the data into the object

In [74]:
```
svm.fit(X_train,Y_train)
```

Out[74]:
```
SVC
SVC(kernel='linear')
```

Model Evaluation

Checking the Accuracy score

In [75]:
```
X_train_pred = svm.predict(X_train)
training_data_accuracy = accuracy_score(X_train_pred,Y_train)
```

In [76]:
```
print(training_data_accuracy)
```

```
0.8464730290456431
```

The accuracy score for the training data is 85%

In [77]:
```
X_test_pred = svm.predict(X_test)
SVM_accuracy = accuracy_score(X_test_pred,Y_test)
```

In [78]:
```
print("Accuracy of training data: ", SVM_accuracy)
```

```
Accuracy of training data:  0.8524590163934426
```

The accuracy score for the training data is 85%

4. K - Nearest Neighbors

Creating the object for the model

In [79]:
```
knn = KNeighborsClassifier()
```

Fitting the data into the object

In [80]:
```
knn.fit(X_train,Y_train)
```

Out[80]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Model Evaluation

In [81]:
```
y_pred = knn.predict(X_test)
```

Checking the accuracy score

In [82]:
```
KNN_accuracy = accuracy_score(Y_test,y_pred)
```

In [83]:
```
print("Accuracy of training data: ", KNN_accuracy)
```

```
Accuracy of training data:  0.639344262295082
```

The accuracy score for the test data is 64%

5. Gaussian Naive Bayes

Creating the object for the model

In [84]:
```
model = GaussianNB()
```

Fitting the data into the object

In [85]:
```
model.fit(X_train,Y_train)
```

Out[85]:
```
▾ GaussianNB
GaussianNB()
```

Model Evaluation

In [86]:
```
Y_Pred = model.predict(X_test)
```

Checking the accuracy score

In [87]:
```
GNB_accuracy = accuracy_score(Y_test,Y_Pred)
```

In [88]:
```
print("Accuracy of training data: ", GNB_accuracy)
```

```
Accuracy of training data:  0.8032786885245902
```

The Accuracy score for the test data is 80%

In [89]:

```
input_data = (71,0,0,112,149,0,1,125,0,1.6,1,0,2)

#Changing the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

#reshapping the numpy array as we are predicting for only one instance
input_data_reshape = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshape)
print(prediction)

if(prediction[0]==0):
    print('The person is healthy')
else:
    print('The person has a heart disease')
```

```
[1]
The person has a heart disease

C:\Users\HP1\anaconda3\lib\site-packages\sklearn\base.py:409: UserWarning: X does not have valid feature names, but GaussianNB was fitted with feature names
  warnings.warn(
```

Comparison and Evaluation of all Models

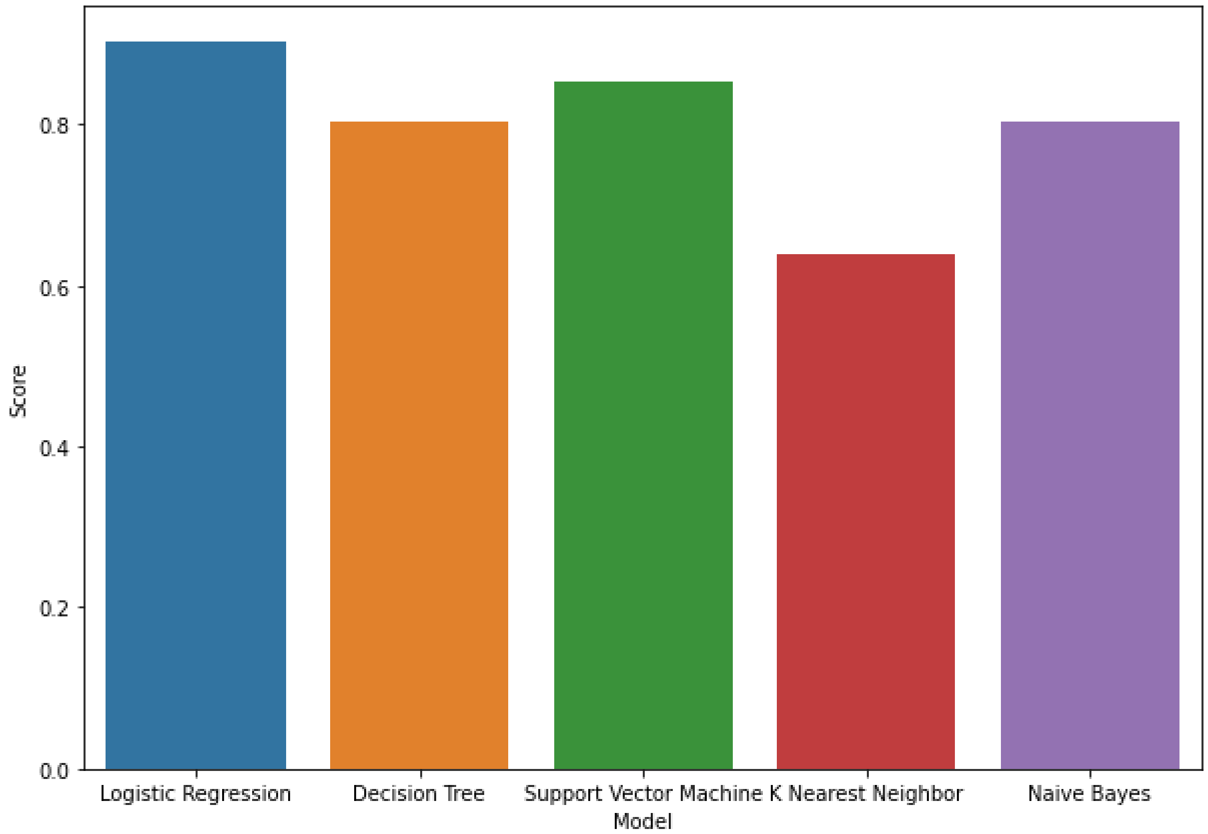
In [90]:
```
models = pd.DataFrame({ 'Model' : ['Logistic Regression', 'Decision Tree', 'Support Vector Machine','K Nearest Neighbo
                                   'Naive Bayes'],
                        'Score': [LR_accuracy, DT_accuracy, SVM_accuracy, KNN_accuracy, GNB_accuracy]})
models.sort_values(by='Score', ascending=False)
```

Out[90]:

	Model	Score
0	Logistic Regression	0.901639
2	Support Vector Machine	0.852459
1	Decision Tree	0.803279
4	Naive Bayes	0.803279
3	K Nearest Neighbor	0.639344

In [91]:
```
plt.figure(figsize = (10,7))
sns.barplot(models['Model'],models['Score']);
```

```
C:\Users\HP1\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as k
eyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [ ]: