In [1]:

```
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
import matplotlib.pyplot as plt
filepath = r"C:\Users\WINDOWS\Desktop\hearts.csv"
data = pd.read_csv(filepath)
data.head(5)
```

Out[1]:

	age	gender	cp	restbps	schol	fbs	restele	maxhra	oldpeak	slope	nomv	thal	target
0	52	1	0	125	212	0	1	168	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	1.9	1	3	2	0

In [2]:

```
print("(Rows, columns): " + str(data.shape))
data.columns
```

```
(Rows, columns): (1025, 13)
```

Out[2]:

```
Index(['age', 'gender', 'cp', 'restbps', 'schol', 'fbs', 'restele', 'maxhr
a',
       'oldpeak', 'slope', 'nomv', 'thal', 'target'],
      dtype='object')
```

In [3]:

```
data.nunique(axis=0)# returns the number of unique values for each variable.
```

Out[3]:

```
age         41
gender       2
cp           4
restbps     49
schol      152
fbs          2
restele      3
maxhra      91
oldpeak     40
slope        3
nomv         5
thal         4
target       2
dtype: int64
```

In [4]:

```
data.describe()#summarize the count,mean,minimum and maximum for numeric variables standard
```

Out[4]:

	age	gender	cp	restbps	schol	fbs	rest
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000(
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.5297
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.5278
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000(
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000(
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000(
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000(
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000(

In [5]:

```
print(data.isna().sum())#display the missing values.
```

```
age        0
gender     0
cp         0
restbps    0
schol      0
fbs        0
restele    0
maxhra     0
oldpeak    0
slope      0
nomv       0
thal       0
target     0
dtype: int64
```

In [6]:

```
data['target'].value_counts()
```

Out[6]:

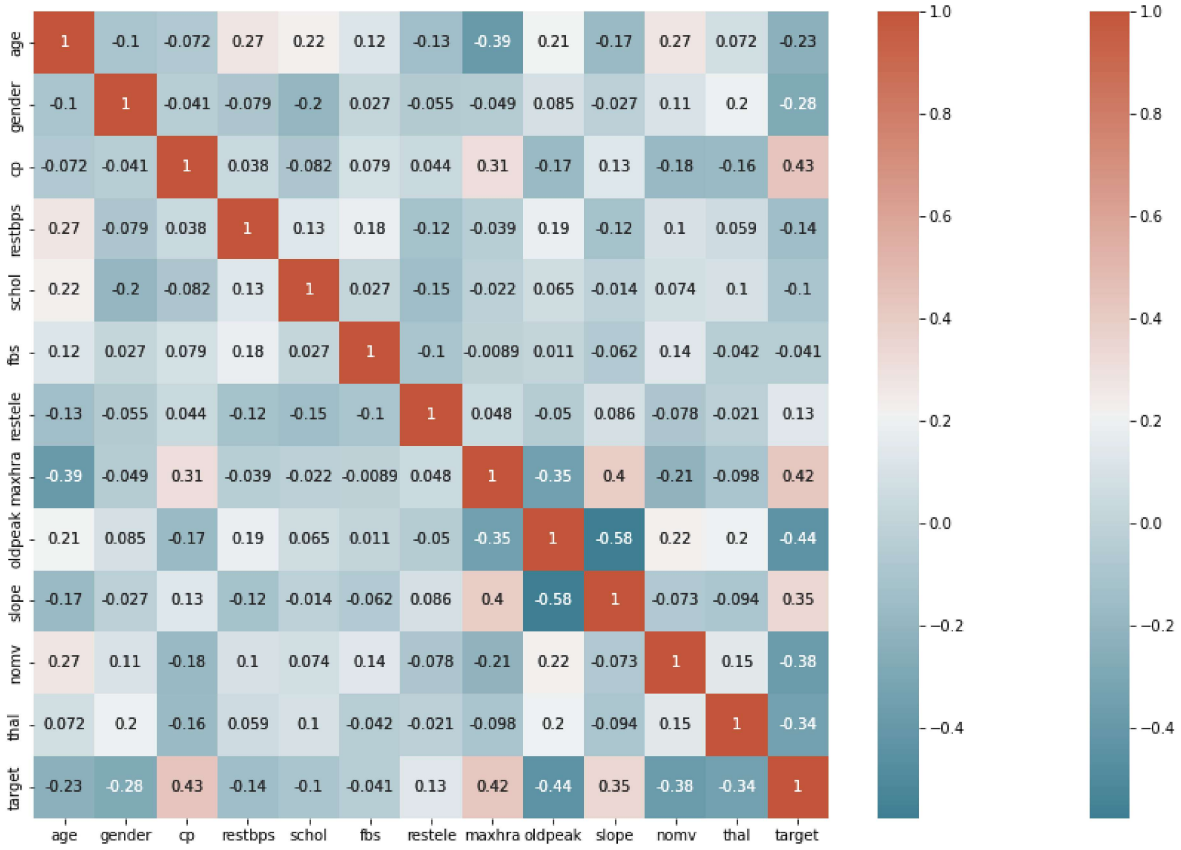
```
1    526
0    499
Name: target, dtype: int64
```

In [7]:

```
ta.corr()
ots(figsize=(15,10))
ap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=sns.diverging_
ap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=sns.diverging_
```

Out[7]:

<AxesSubplot:>



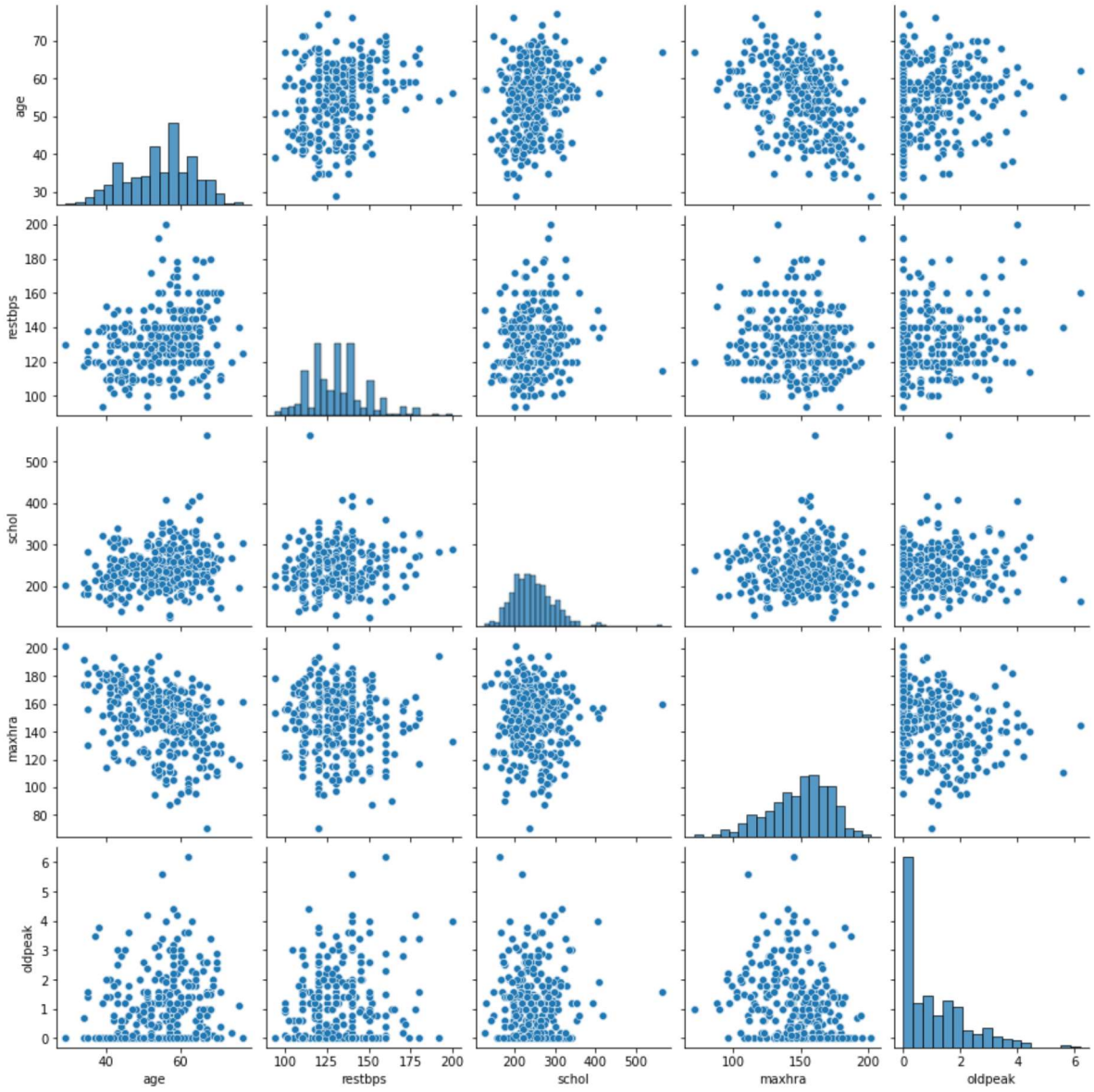
We can see there is a positive correlation between chest pain (cp) & target (our predictor). This makes sense since, the greater amount of chest pain results in a greater chance of having heart disease.

In [8]:

```
subData = data[['age','restbps','schol','maxhra','oldpeak']]
sns.pairplot(subData)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x1e000355100>
```



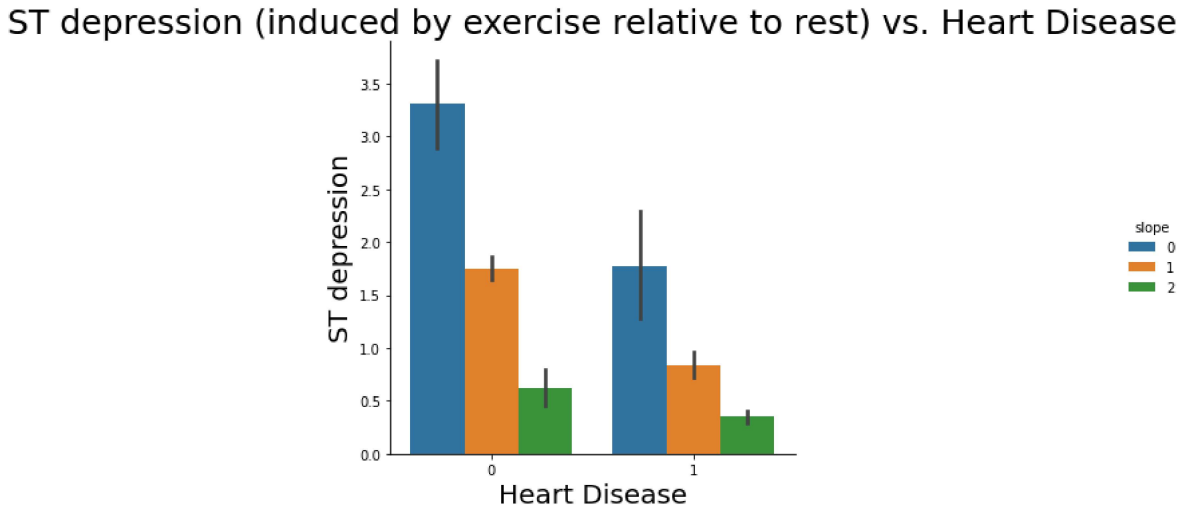
In [9]:

```
sns.catplot(x="target", y="oldpeak", hue="slope", kind="bar", data=data);

plt.title('ST depression (induced by exercise relative to rest) vs. Heart Disease',size=25)
plt.xlabel('Heart Disease',size=20)
plt.ylabel('ST depression',size=20)
```

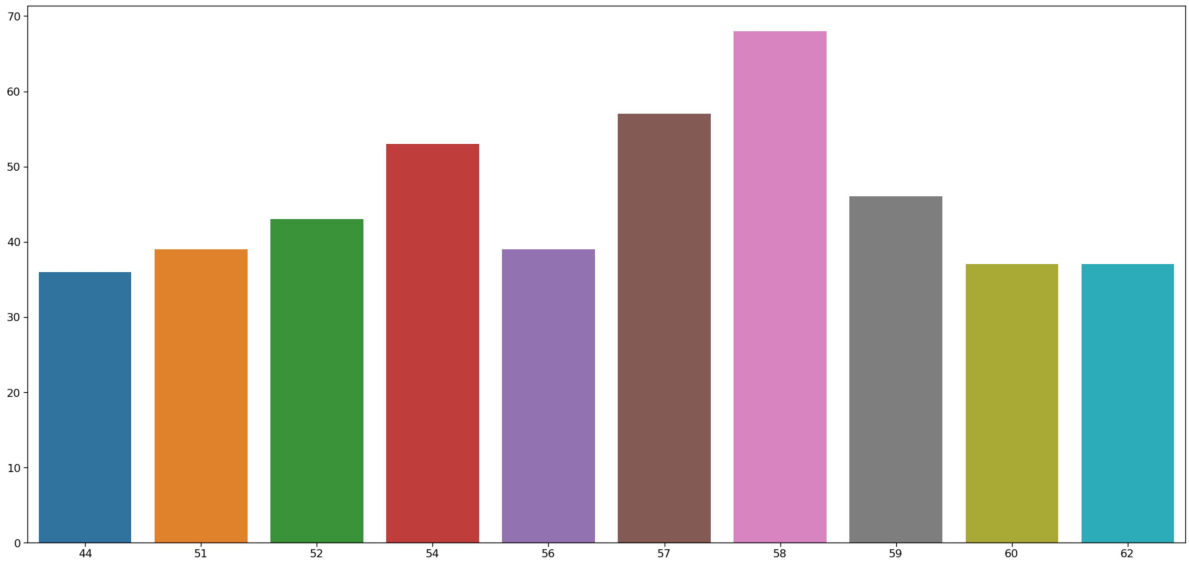
Out[9]:

```
Text(26.426458333333343, 0.5, 'ST depression')
```



In [10]:

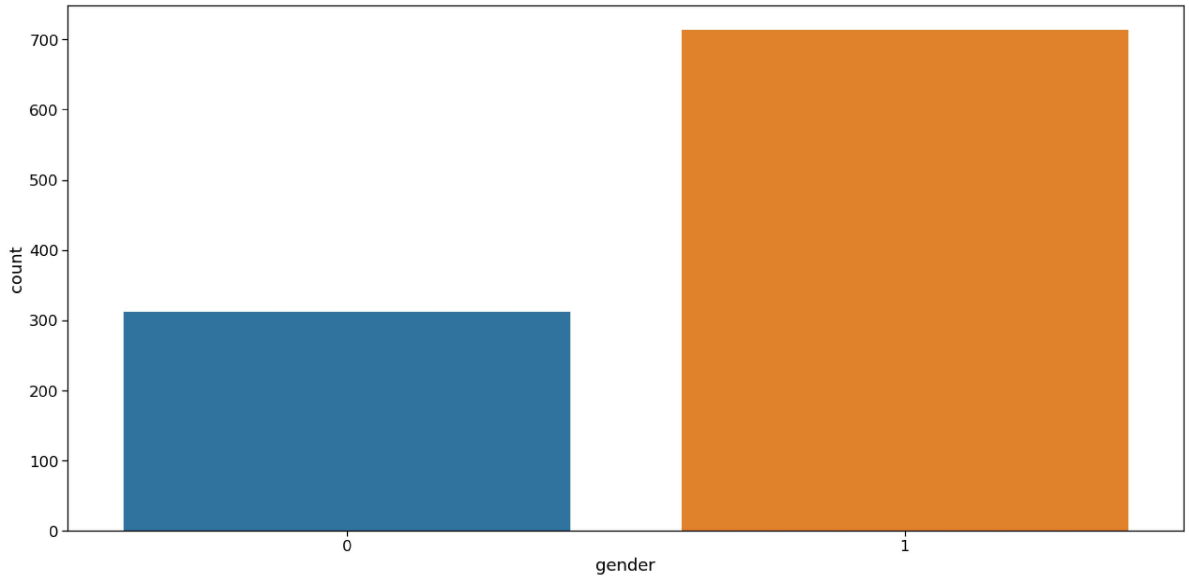
```
plt.figure(figsize=(25,12))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=data.age.value_counts()[:10].index,y=data.age.value_counts()[:10].values)
plt.tight_layout()
```



In [11]:

```
plt.figure(figsize=(18,9))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['gender'])
plt.tight_layout()
#gender feature
```

```
C:\Users\WINDOWS\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version 0.1
2, the only valid positional argument will be `data`, and passing other argu
ments without an explicit keyword will result in an error or misinterpretati
on.
  warnings.warn(
```

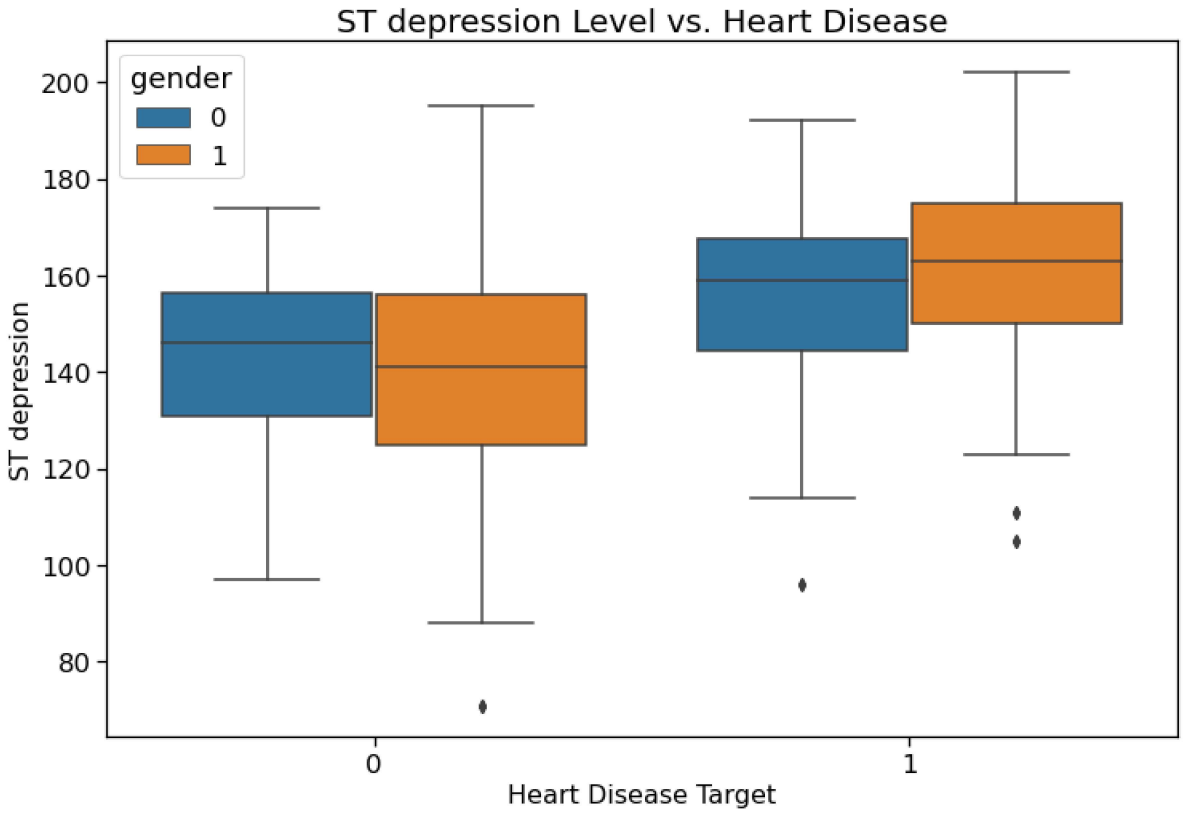


In [12]:

```
plt.figure(figsize=(12,8))
sns.boxplot(x= 'target', y= 'maxhra',hue="gender", data=data )
plt.title("ST depression Level vs. Heart Disease", fontsize=20)
plt.xlabel("Heart Disease Target",fontsize=16)
plt.ylabel("ST depression", fontsize=16)
```

Out[12]:

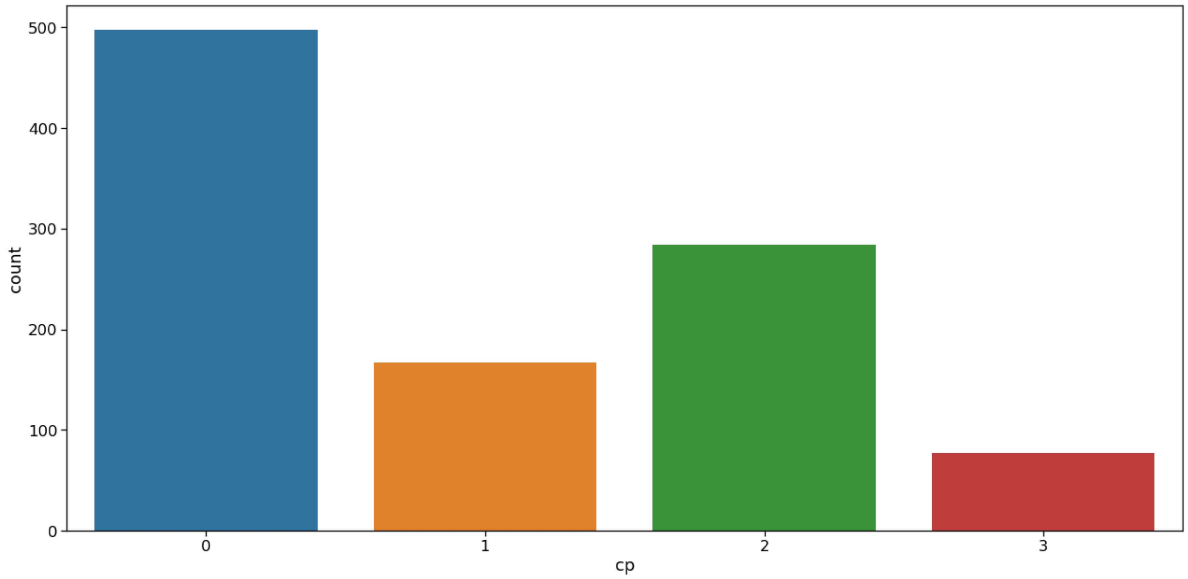
```
Text(0, 0.5, 'ST depression')
```



In [13]:

```
plt.figure(figsize=(18,9))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['cp'])
plt.tight_layout()
#chest pain type analysis
```

```
C:\Users\WINDOWS\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version 0.1
2, the only valid positional argument will be `data`, and passing other argu
ments without an explicit keyword will result in an error or misinterpretati
on.
  warnings.warn(
```



In [14]:

```
pos_data = data[data['target']==1]
pos_data.describe()
```

Out[14]:

	age	gender	cp	restbps	schol	fbs	restele	
count	526.000000	526.000000	526.000000	526.000000	526.000000	526.000000	526.000000	52
mean	52.408745	0.570342	1.378327	129.245247	240.979087	0.134981	0.598859	15
std	9.631804	0.495498	0.945881	16.112188	53.010345	0.342029	0.502109	1
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	9
25%	44.000000	0.000000	1.000000	120.000000	208.000000	0.000000	0.000000	14
50%	52.000000	1.000000	2.000000	130.000000	234.000000	0.000000	1.000000	16
75%	59.000000	1.000000	2.000000	140.000000	265.750000	0.000000	1.000000	17
max	76.000000	1.000000	3.000000	180.000000	564.000000	1.000000	2.000000	20

In [15]:

```
pos_data = data[data['target']==0]
pos_data.describe()
```

Out[15]:

	age	gender	cp	restbps	schol	fbs	restele	
count	499.000000	499.000000	499.000000	499.000000	499.000000	499.000000	499.000000	49
mean	56.569138	0.827655	0.482966	134.106212	251.292585	0.164329	0.456914	13
std	7.908153	0.378059	0.908024	18.576736	49.558924	0.370945	0.544825	2
min	35.000000	0.000000	0.000000	100.000000	131.000000	0.000000	0.000000	7
25%	52.000000	1.000000	0.000000	120.000000	217.000000	0.000000	0.000000	12
50%	58.000000	1.000000	0.000000	130.000000	249.000000	0.000000	0.000000	14
75%	62.000000	1.000000	0.000000	144.000000	284.000000	0.000000	1.000000	15
max	77.000000	1.000000	3.000000	200.000000	409.000000	1.000000	2.000000	19

In [16]:

```
print("(Positive Patients ST depression): " + str(pos_data['oldpeak'].mean()))
```

```
(Positive Patients ST depression): 1.6002004008016042
```

In [17]:

```
print("(Positive Patients thalach): " + str(pos_data['maxhra'].mean()))
```

```
(Positive Patients thalach): 139.1302605210421
```

In [20]:

```
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
```

In [21]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 1)
```

In [22]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [24]:

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

model1 = LogisticRegression(random_state=1) # get instance of model
model1.fit(x_train, y_train) # Train/Fit model

y_pred1 = model1.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred1)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.90      0.77      0.83       109
           1       0.78      0.91      0.84        96

    accuracy                           0.83       205
   macro avg       0.84      0.84      0.83       205
weighted avg       0.84      0.83      0.83       205
```

In [23]:

```
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

model6 = RandomForestClassifier(random_state=1)# get instance of model
model6.fit(x_train, y_train) # Train/Fit model

y_pred6 = model6.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred6)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       109
           1       1.00      1.00      1.00        96

    accuracy                           1.00       205
   macro avg       1.00      1.00      1.00       205
weighted avg       1.00      1.00      1.00       205
```

In [24]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred6)
print(cm)
accuracy_score(y_test, y_pred6)
```

```
[[109   0]
 [  0  96]]
```

Out[24]:

```
1.0
```

In [25]:

```
# get importance
importance = model6.feature_importances_

# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
```

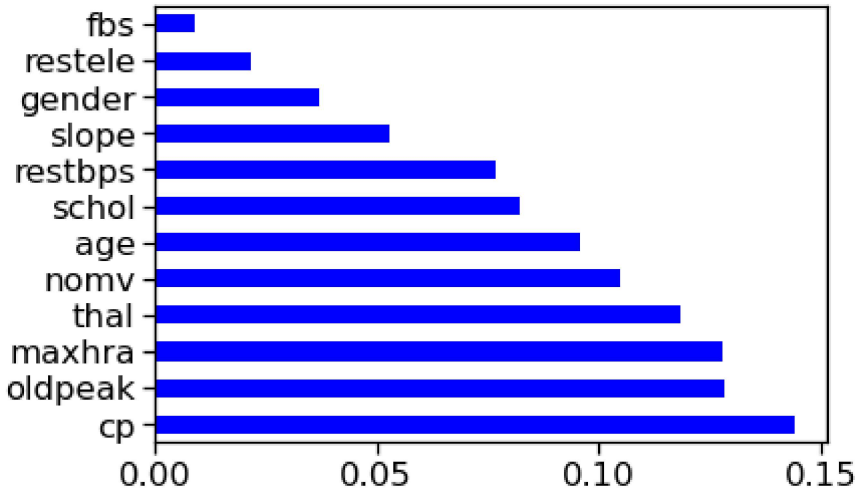
```
Feature: 0, Score: 0.09568
Feature: 1, Score: 0.03723
Feature: 2, Score: 0.14419
Feature: 3, Score: 0.07702
Feature: 4, Score: 0.08232
Feature: 5, Score: 0.00901
Feature: 6, Score: 0.02173
Feature: 7, Score: 0.12793
Feature: 8, Score: 0.12845
Feature: 9, Score: 0.05282
Feature: 10, Score: 0.10502
Feature: 11, Score: 0.11859
```

In [26]:

```
index= data.columns[:-1]
importance = pd.Series(model6.feature_importances_, index=index)
importance.nlargest(13).plot(kind='barh', colormap='winter')
```

Out[26]:

```
<AxesSubplot:>
```



In [30]:

```
y_pred = model6.predict(x_test)

print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
```

We can see that our results are very accurate