HeartDisease prediction

0.1 1 : Indtroduction

[67]:
```python
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
import matplotlib.pyplot as plt
```

0.2 2 : Data Wrangling

[68]:
```python
data = pd.read_csv("SL_data.csv")
data.head()
```

[68]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

[69]:
```python
print("(Rows, columns): " + str(data.shape))
data.columns
```

```
(Rows, columns): (303, 14)
```

[69]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[70]:
```
data.nunique(axis=0)# returns the number of unique values for each variable.
```

[70]:
```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

1 #summarizes the count, mean, standard deviation, min, and max for numeric variables.

data.describe()

[71]:
```
# Display the Missing Values

print(data.isna().sum())
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

[72]:
```
data['target'].value_counts()
```

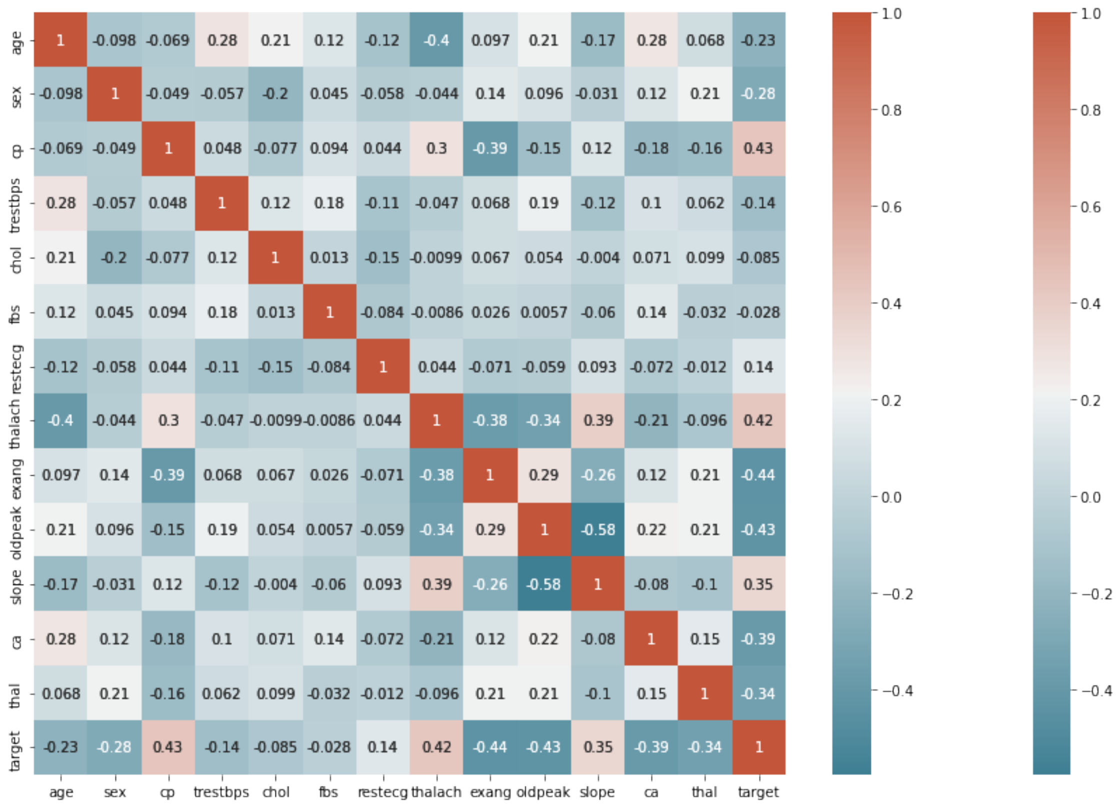
```
[72]: 1    165
      0    138
      Name: target, dtype: int64
```

1.1 3. Exploratory Data Analysis

3.1 : Correlation Matrix- correlations between all variables.

```python
[73]: corr = data.corr()
plt.subplots(figsize=(15,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,␣
 ↪annot=True, cmap=sns.diverging_palette(220, 20, as_cmap=True))
sns.heatmap(corr, xticklabels=corr.columns,
            yticklabels=corr.columns,
            annot=True,
            cmap=sns.diverging_palette(220, 20, as_cmap=True))
```

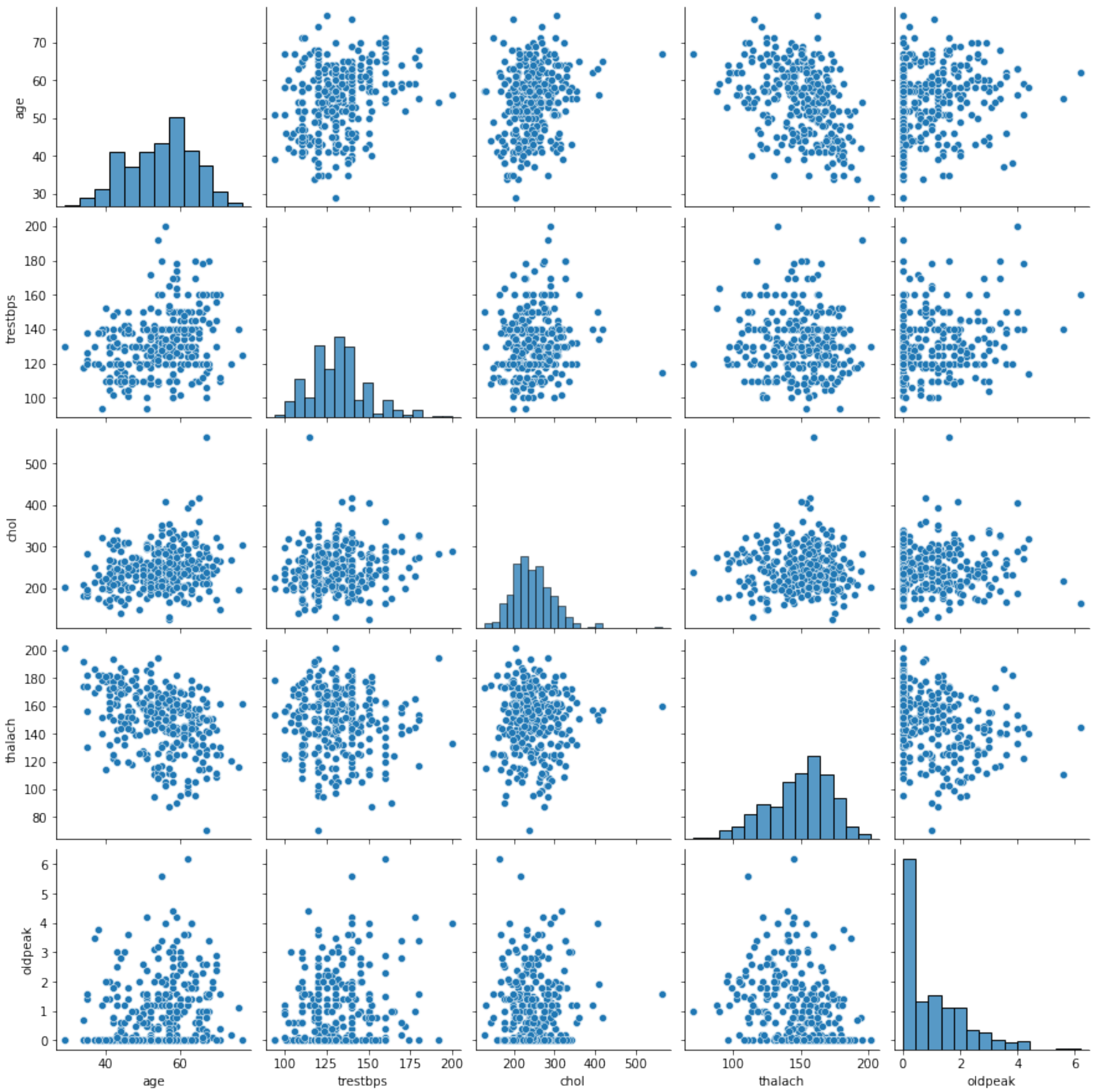
```
[73]: <AxesSubplot:>
```



1.1.1 3.2 : A pairplot with only our continuous features.

[74]:
```
subData = data[['age','trestbps','chol','thalach','oldpeak']]
sns.pairplot(subData)
```

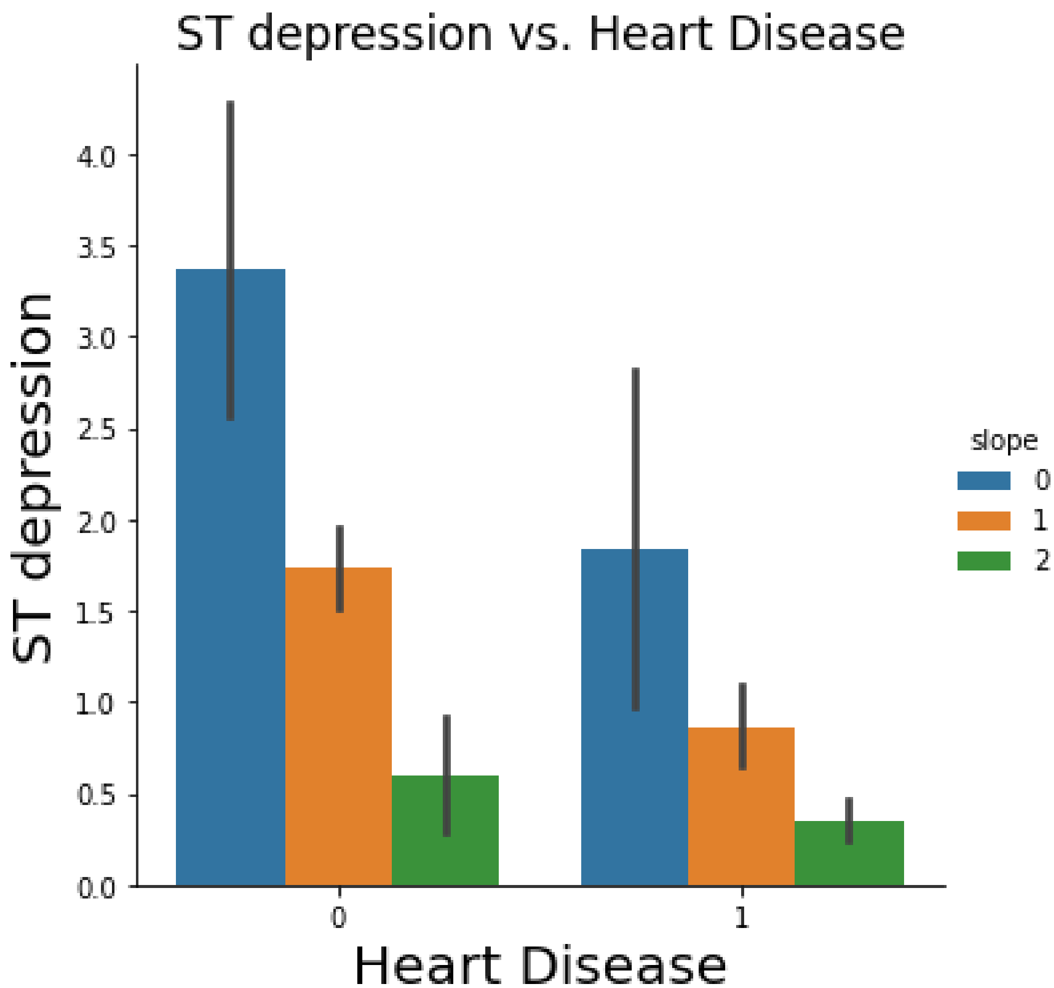
[74]: <seaborn.axisgrid.PairGrid at 0x7f4ab045bf10>



[75]:
```
sns.catplot(x="target", y="oldpeak", hue="slope", kind="bar", data=data);

plt.title('ST depression vs. Heart Disease',size = 17)
plt.xlabel('Heart Disease',size=20)
plt.ylabel('ST depression',size=20)
```

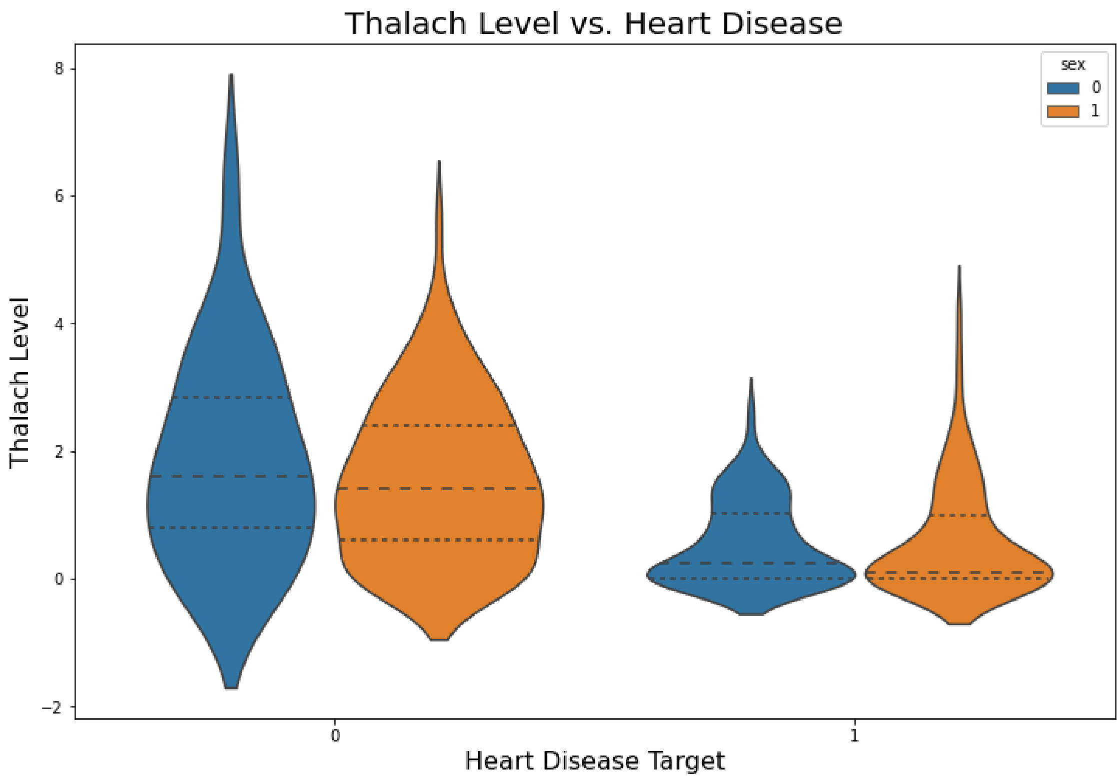
[75]: Text(26.426458333333343, 0.5, 'ST depression')



1.1.2 3.3 : Violin & Box Plots

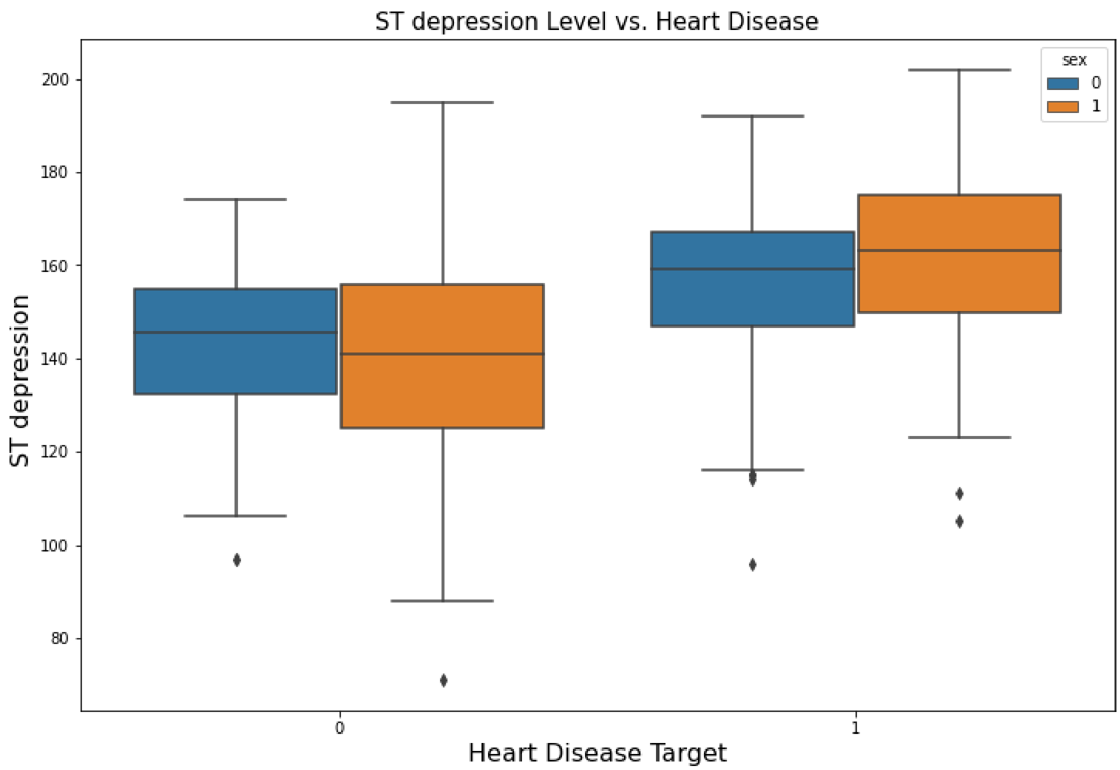
```
[76]: plt.figure(figsize=(12,8))
sns.violinplot(x= 'target', y= 'oldpeak',hue="sex", inner='quartile',data= data␣
↪)
plt.title("Thalach Level vs. Heart Disease",fontsize=20)
plt.xlabel("Heart Disease Target", fontsize=16)
plt.ylabel("Thalach Level", fontsize=16)
```

[76]: Text(0, 0.5, 'Thalach Level')



[77]:
```
plt.figure(figsize=(12,8))
sns.boxplot(x= 'target', y= 'thalach',hue="sex", data=data )
plt.title("ST depression Level vs. Heart Disease", fontsize = 15)
plt.xlabel("Heart Disease Target",fontsize=16)
plt.ylabel("ST depression ", fontsize=16)
```

[77]: Text(0, 0.5, 'ST depression ')



1.1.3 3.4 : Filtering data by positive Heart Disease patient

[78]:
```
pos_data = data[data['target']==1]
pos_data.describe()
```

[78]:

	age	sex	cp	trestbps	chol	fbs	\
count	165.000000	165.000000	165.000000	165.000000	165.000000	165.000000	
mean	52.496970	0.563636	1.375758	129.303030	242.230303	0.139394	
std	9.550651	0.497444	0.952222	16.169613	53.552872	0.347412	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	
25%	44.000000	0.000000	1.000000	120.000000	208.000000	0.000000	
50%	52.000000	1.000000	2.000000	130.000000	234.000000	0.000000	
75%	59.000000	1.000000	2.000000	140.000000	267.000000	0.000000	
max	76.000000	1.000000	3.000000	180.000000	564.000000	1.000000	

	restecg	thalach	exang	oldpeak	slope	ca	\
count	165.000000	165.000000	165.000000	165.000000	165.000000	165.000000	
mean	0.593939	158.466667	0.139394	0.583030	1.593939	0.363636	
std	0.504818	19.174276	0.347412	0.780683	0.593635	0.848894	
min	0.000000	96.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	149.000000	0.000000	0.000000	1.000000	0.000000	

```
50%      1.000000  161.000000    0.000000    0.200000    2.000000    0.000000
75%      1.000000  172.000000    0.000000    1.000000    2.000000    0.000000
max      2.000000  202.000000    1.000000    4.200000    2.000000    4.000000

             thal  target
count  165.000000   165.0
mean     2.121212     1.0
std      0.465752     0.0
min      0.000000     1.0
25%      2.000000     1.0
50%      2.000000     1.0
75%      2.000000     1.0
max      3.000000     1.0
```

1.1.4 3.5 : Filtering data by Negative Heart Disease patient

[79]:
```
neg_data = data[data['target']==0]
neg_data.describe()
```

[79]:
```
              age         sex          cp    trestbps        chol         fbs  \
count  138.000000  138.000000  138.000000  138.000000  138.000000  138.000000
mean    56.601449    0.826087    0.478261  134.398551  251.086957    0.159420
std      7.962082    0.380416    0.905920   18.729944   49.454614    0.367401
min     35.000000    0.000000    0.000000  100.000000  131.000000    0.000000
25%     52.000000    1.000000    0.000000  120.000000  217.250000    0.000000
50%     58.000000    1.000000    0.000000  130.000000  249.000000    0.000000
75%     62.000000    1.000000    0.000000  144.750000  283.000000    0.000000
max     77.000000    1.000000    3.000000  200.000000  409.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  138.000000  138.000000  138.000000  138.000000  138.000000  138.000000
mean     0.449275  139.101449    0.550725    1.585507    1.166667    1.166667
std      0.541321   22.598782    0.499232    1.300340    0.561324    1.043460
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  125.000000    0.000000    0.600000    1.000000    0.000000
50%      0.000000  142.000000    1.000000    1.400000    1.000000    1.000000
75%      1.000000  156.000000    1.000000    2.500000    1.750000    2.000000
max      2.000000  195.000000    1.000000    6.200000    2.000000    4.000000

             thal  target
count  138.000000   138.0
mean     2.543478     0.0
std      0.684762     0.0
min      0.000000     0.0
25%      2.000000     0.0
50%      3.000000     0.0
```

```
75%       3.000000     0.0
max       3.000000     0.0
```

[80]:
```
print("(Positive Patients ST depression): " + str(pos_data['oldpeak'].mean()))
print("(Negative Patients ST depression): " + str(neg_data['oldpeak'].mean()))
```

```
(Positive Patients ST depression): 0.5830303030303029
(Negative Patients ST depression): 1.5855072463768118
```

[81]:
```
print("(Positive Patients thalach): " + str(pos_data['thalach'].mean()))
print("(Negative Patients thalach): " + str(neg_data['thalach'].mean()))
```

```
(Positive Patients thalach): 158.46666666666667
(Negative Patients thalach): 139.1014492753623
```

1.2 4. Machine Learning and Predictive Analytics

1.2.1 4.1 : Preparing Data for Modeling

[82]:
```
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
```

1.2.2 4.2 : Splitting data into the Training and Test sets

[83]:
```
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)
```

[84]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

1.2.3 4.3 : Modeling and Training

Model 1: Logistic Regression

[85]:
```
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       0.77      0.67      0.71        30
           1       0.71      0.81      0.76        31

    accuracy                           0.74        61
   macro avg       0.74      0.74      0.74        61
weighted avg       0.74      0.74      0.74        61
```

Model 2: K-NN (K-Nearest Neighbors)

[86]:
```
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier

model2 = KNeighborsClassifier() # get instance of model
model2.fit(x_train, y_train) # Train/Fit model

y_pred2 = model2.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred2)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.78      0.70      0.74        30
           1       0.74      0.81      0.77        31

    accuracy                           0.75        61
   macro avg       0.76      0.75      0.75        61
weighted avg       0.76      0.75      0.75        61
```

Model 3: SVM

[87]:
```
from sklearn.metrics import classification_report
from sklearn.svm import SVC

model3 = SVC(random_state=1) # get instance of model
model3.fit(x_train, y_train) # Train/Fit model

y_pred3 = model3.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred3)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73        30
           1       0.72      0.84      0.78        31

    accuracy                           0.75        61
```

```
   macro avg       0.76      0.75      0.75        61
weighted avg       0.76      0.75      0.75        61
```

Model 4: Naives Bayes Classifier

[88]:
```
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB

model4 = GaussianNB() # get instance of model
model4.fit(x_train, y_train) # Train/Fit model

y_pred4 = model4.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred4)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.79      0.73      0.76        30
           1       0.76      0.81      0.78        31

    accuracy                           0.77        61
   macro avg       0.77      0.77      0.77        61
weighted avg       0.77      0.77      0.77        61
```

Model 5: Decision Trees

[89]:
```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier

model5 = DecisionTreeClassifier(random_state=1) # get instance of model
model5.fit(x_train, y_train) # Train/Fit model

y_pred5 = model5.predict(x_test) # get y predictions
print(classification_report(y_test, y_pred5)) # output accuracy
```

```
              precision    recall  f1-score   support

           0       0.68      0.70      0.69        30
           1       0.70      0.68      0.69        31

    accuracy                           0.69        61
   macro avg       0.69      0.69      0.69        61
weighted avg       0.69      0.69      0.69        61
```

Model 6: Random Forest

[90]:
```python
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       0.88      0.70      0.78        30
           1       0.76      0.90      0.82        31

    accuracy                           0.80        61
   macro avg       0.82      0.80      0.80        61
weighted avg       0.81      0.80      0.80        61
```

4.4 : Confusion Matrix

[91]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred6)
print(cm)
accuracy_score(y_test, y_pred6)
```

```
[[21  9]
 [ 3 28]]
```

[91]: 0.8032786885245902

4.4.1 : Confusion Matrix: Interpretation

1 : True Positive = 21

2 : True negative = 28

3 : 9 is False positive and its false

4 : 3 is False Negative and it is false

1.2.4 Accuracy = (TP + TN)/(TP + TN + FP + FN).

Accuracy = (21 + 28) /(21 + 28 + 9 + 3) = 0.80 = 80% accuracy

1.3 5: Which Feature is more Important

[92]:
```
# get importance
importance = model6.feature_importances_

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

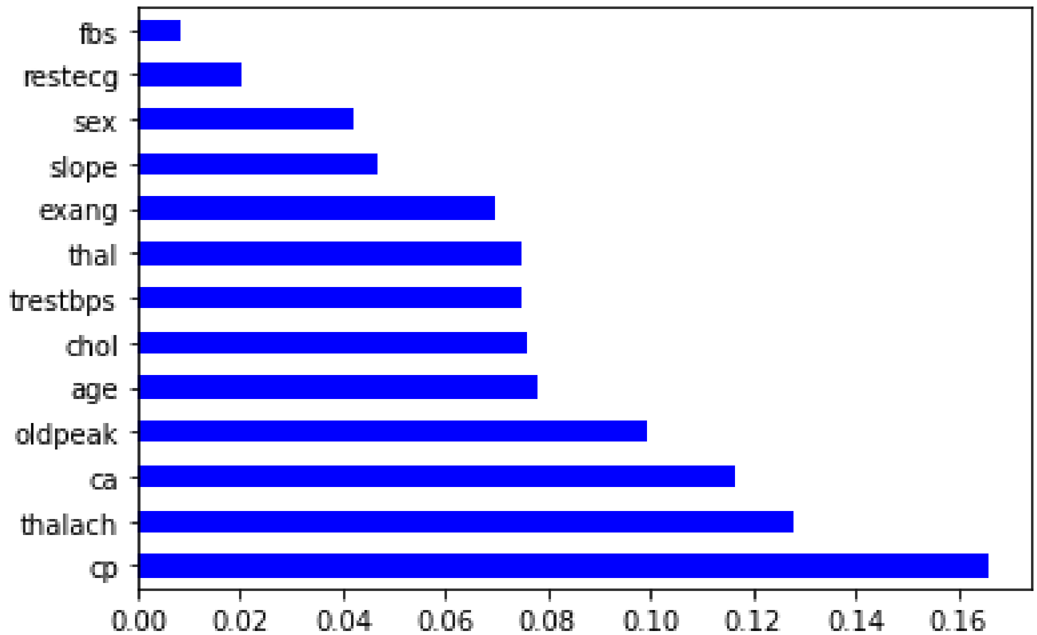
```
Feature: 0, Score: 0.07814
Feature: 1, Score: 0.04206
Feature: 2, Score: 0.16580
Feature: 3, Score: 0.07477
Feature: 4, Score: 0.07587
Feature: 5, Score: 0.00828
Feature: 6, Score: 0.02014
Feature: 7, Score: 0.12772
Feature: 8, Score: 0.06950
Feature: 9, Score: 0.09957
Feature: 10, Score: 0.04677
Feature: 11, Score: 0.11667
Feature: 12, Score: 0.07473
```

1.3.1 Since Feature : 7 has highest score (0.16580)

1.3.2 Featue :7 is more Important

[93]:
```
index= data.columns[:-1]
importance = pd.Series(model6.feature_importances_, index=index)
importance.nlargest(13).plot(kind='barh', colormap='winter')
```

[93]: <AxesSubplot:>



from the graph above, we can conclude that following Features are more Important

1 : chest pain type (cp)

2 : maximum heart rate achieved (thalach)

3 : number of major vessels (ca)

4 : ST depression (oldpeak)

1.4 6 : Predictions

1.4.1 6.1 : Test set results: Predictions

[94]:
```
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]
 [0 0]
 [0 0]
```

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

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

1 : First value is predicted value, Second value is actual value.

2 : If the predicted value = actual value then prediction correct

1.5 7 : Conclusions

7.1 : Random Forest algorithm accuracy = 80%

7.2 : A ccuracy above 70% is considered good

7.3 : Above 80% acccuracy chances of Overfitting

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```