FINAL

May 30, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.mlab as mlab
import scipy.stats as st

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")

#Libraries for data processing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
#Importing kNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

#Importing Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split



#Libraries for decision tree and random forest model
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

#For performing hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV

#Evaluation metrics
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
 ↪recall_score, f1_score
```

[2]:
```
df = pd.read_csv('heart 2.csv')
```

[3]:
```
df.columns
```

[3]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

[4]:
```
# No NULL values
df.isnull().sum()
```

[4]:
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

[5]:
```
# Note: Predicted value (Heart Disease) is boolean
df.describe()
```

[5]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

[6]:
```
(df["RestingBP"] == 0).sum()
```

[6]: 1

[7]:
```
df.shape
```

[7]: (918, 12)

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

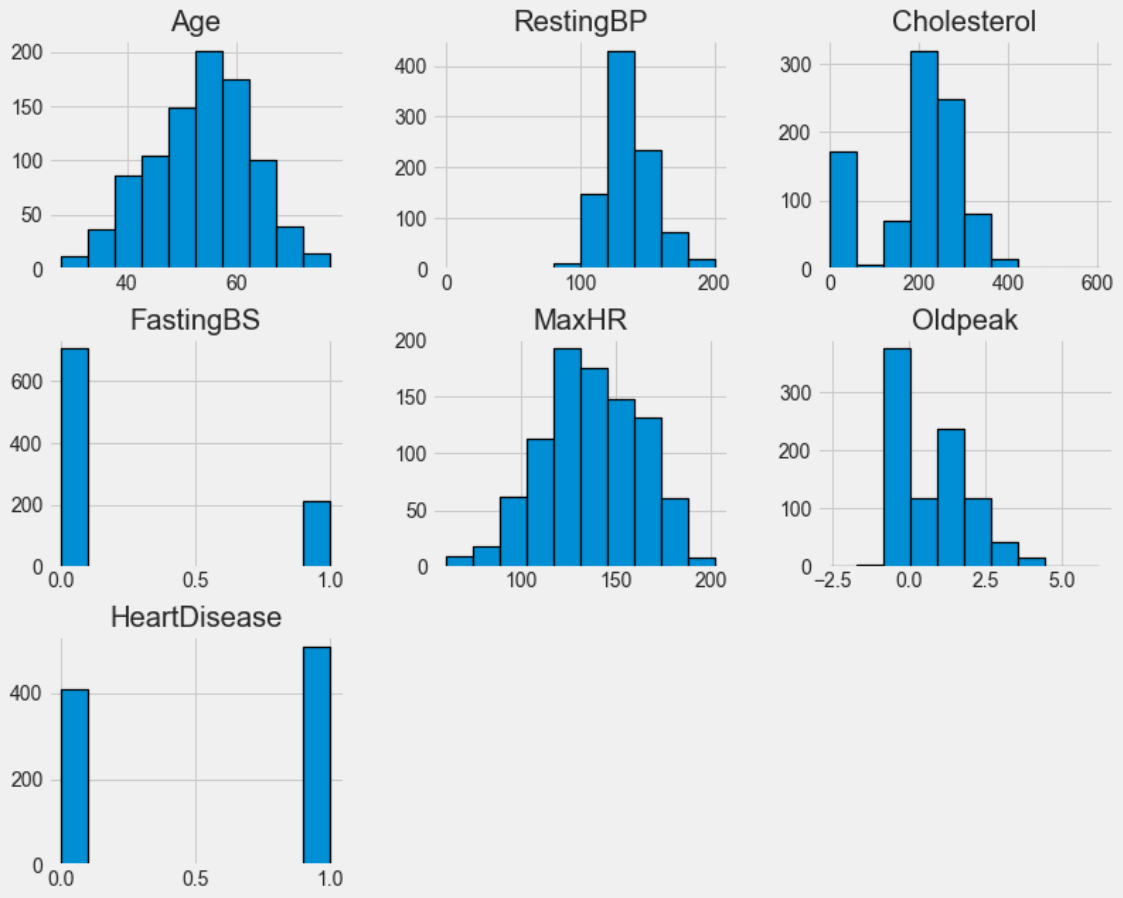
[9]:
```
df.head()
```

[9]:

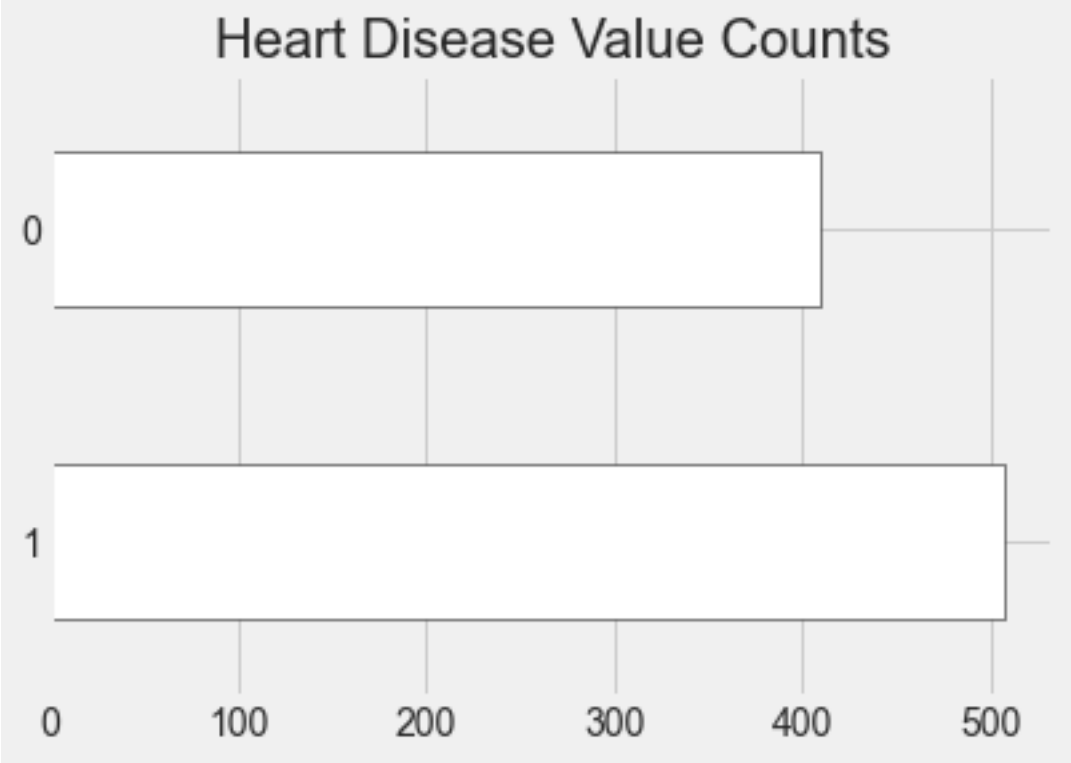
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	\
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	N	0.0	Up	0
1	N	1.0	Flat	1
2	N	0.0	Up	0
3	Y	1.5	Flat	1
4	N	0.0	Up	0

[10]:
```
df.hist(edgecolor='black', linewidth=1.2, figsize=(12, 10))
plt.show()
```



[11]:
```
df.HeartDisease.value_counts().plot(kind='barh', edgecolor = 'black', color=␣
↪'white')
plt.title('Heart Disease Value Counts')
plt.show()
```



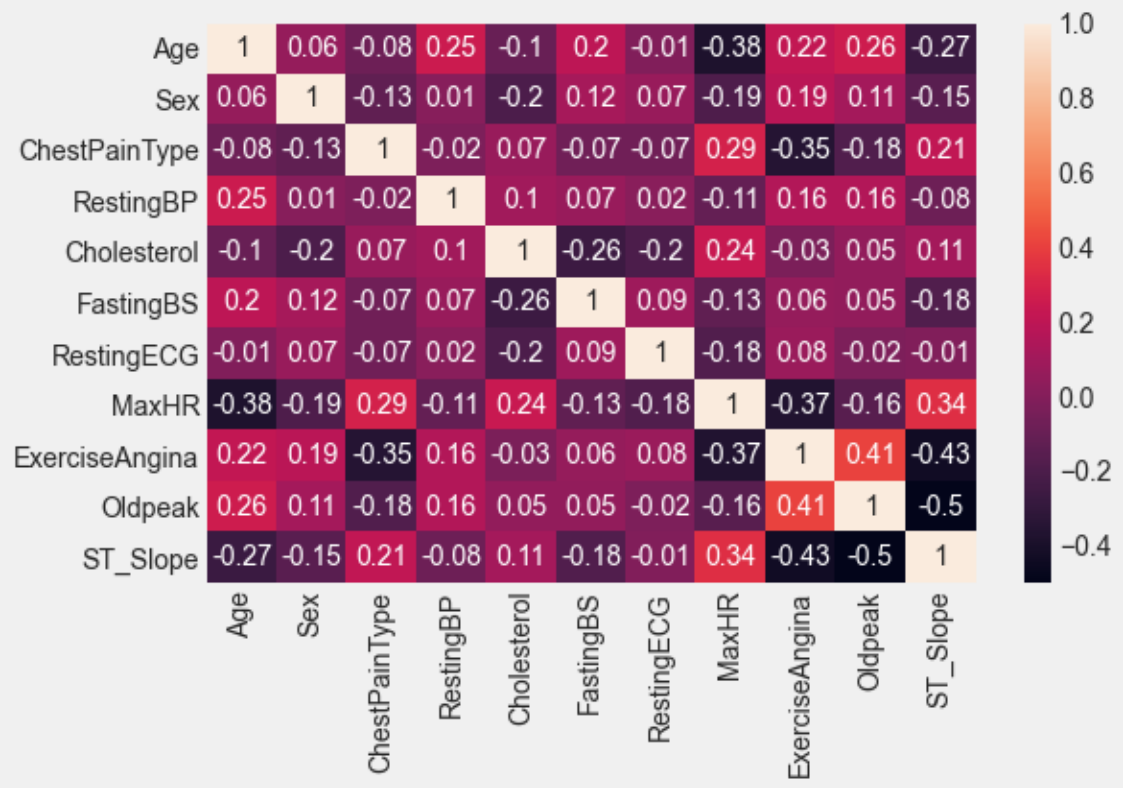
[12]:
```
categorical_col = []
for column in df.columns:
    if df[column].dtype == object and len(df[column].unique()) <= 50:
        categorical_col.append(column)
        print(f"{column} : {df[column].unique()}")

    #sex, chestpain, resting ekg, exerciseanemia, st_slope
```

```
Sex : ['M' 'F']
ChestPainType : ['ATA' 'NAP' 'ASY' 'TA']
RestingECG : ['Normal' 'ST' 'LVH']
ExerciseAngina : ['N' 'Y']
ST_Slope : ['Up' 'Flat' 'Down']
```

[13]:
```
label = LabelEncoder()
for column in categorical_col:
    df[column] = label.fit_transform(df[column])

# Sex: M == 1, F == 0
# ChestPainType: ATA == 1, NAP == 2, ASY        == 0, TA ==3
# ExerciseAngina: N == 0, Y == 1
# RestingECG: Normal == 1, ST == 2, LVH == 0
# ST_Slope: Up == 2, Flat == 1, Down == 0
```

[14]:
```
X = df[['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']]
y = df['HeartDisease']
```

[15]:
```
corr_matrix = X.corr().round(2)
#heatmap plotting
plt.figure(figsize = (8,5))
sns.heatmap(corr_matrix, annot = True) #Annot = True is used to print values␣
 ↪inside the square
plt.show()
```



[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[17]:
```
## decision tree
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
```

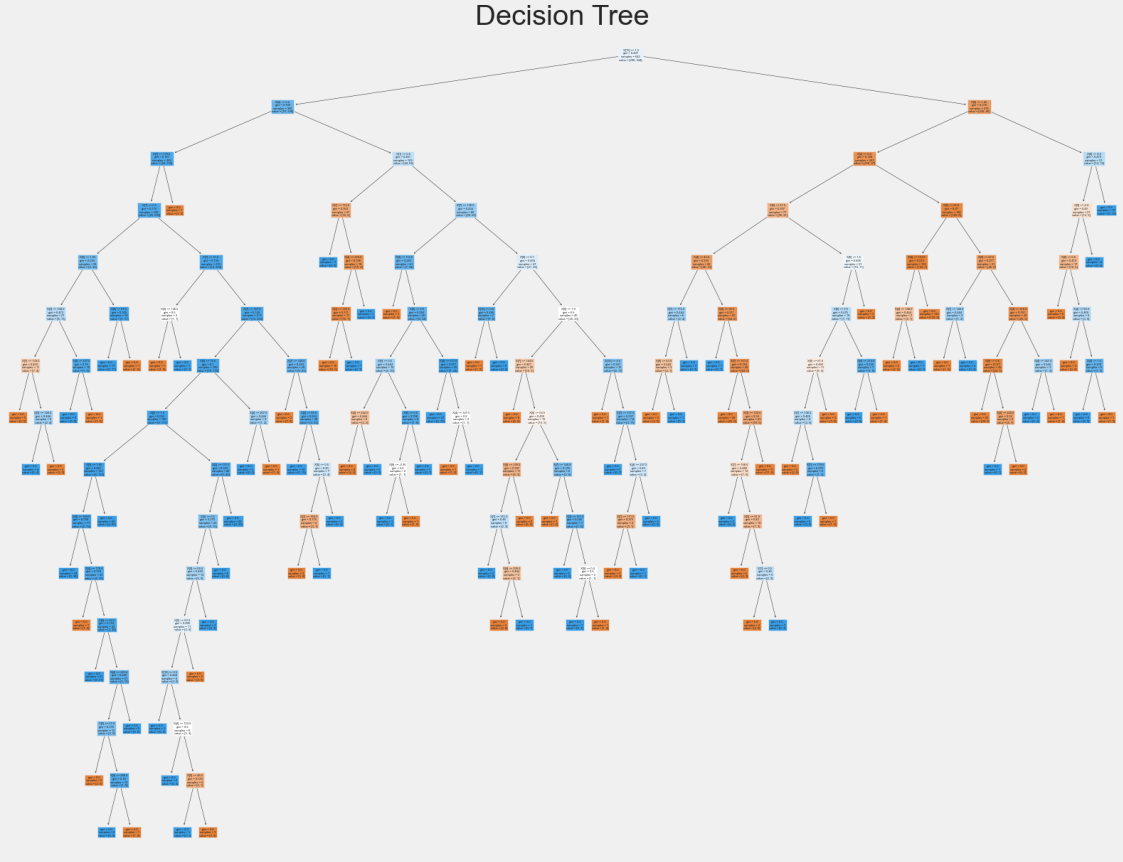
[17]:
```
DecisionTreeClassifier(random_state=42)
```

[18]:
```
def print_score(clf, X_train, y_train, X_test, y_test, train=True,␣
 ↪pos_label="Yes"):
```

```
    if train == False:
        pred = clf.predict(X_test)
        print("Test Result:\n")
        print(f"accuracy score: {accuracy_score(y_test, pred)}\n")
        print(f"Classification Report: \n \tPrecision: {precision_score(y_test,␣
↪pred)}\n\tRecall Score: {recall_score(y_test, pred)}\n\tF1 score:␣
↪{f1_score(y_test, pred)}\n")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")

print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Test Result:

accuracy score: 0.7681159420289855

Classification Report:
        Precision: 0.8623188405797102
        Recall Score: 0.725609756097561
        F1 score: 0.7880794701986755

Confusion Matrix:
 [[ 93  19]
 [ 45 119]]
```

[19]:
```
from sklearn import tree
fig = plt.figure(figsize=(25,20))
T= tree.plot_tree(tree_clf, filled = True)
plt.title('Decision Tree', size = 40)
fig.savefig('DT.png')
plt.show()
```



[284]:
```
#add
def bestpam(X_train, X_val, y_train, y_val):
    best_max_features = 0
    best_max_depth = 0
    best_min_samples_leaf = 0
    curr_pred = 0
    curr_accuracy = 0
    best_accuracy = 0
    for curr_max_features in range(1, 10):
        for curr_max_depth in range(10, 200, 10):
            for curr_min_samples in range(10, 200, 10):
                curr_clf = DecisionTreeClassifier(random_state=18,␣
↪max_features=curr_max_features, max_depth=curr_max_depth,␣
↪min_samples_leaf=curr_min_samples)
                curr_clf.fit(X_train, y_train)
                curr_pred = curr_clf.predict(X_val)
                curr_accuracy = accuracy_score(y_val, curr_pred)
                if (curr_accuracy > best_accuracy):
                    best_accuracy = curr_accuracy
                    best_min_samples_leaf = curr_min_samples
```

```
                    best_max_depth = curr_max_depth
                    best_max_features = curr_max_features
    print("Best values: max_features: "+ str(best_max_features)+"     max_depth:
 ↪ "+ str(best_max_depth)+"     min_samples_leaf: "+␣
 ↪str(best_min_samples_leaf))
    return best_accuracy

print(bestpam(X_train, X_test, y_train, y_test))
```

```
Best values: max_features: 4     max_depth: 10     min_samples_leaf: 10
0.8840579710144928
```

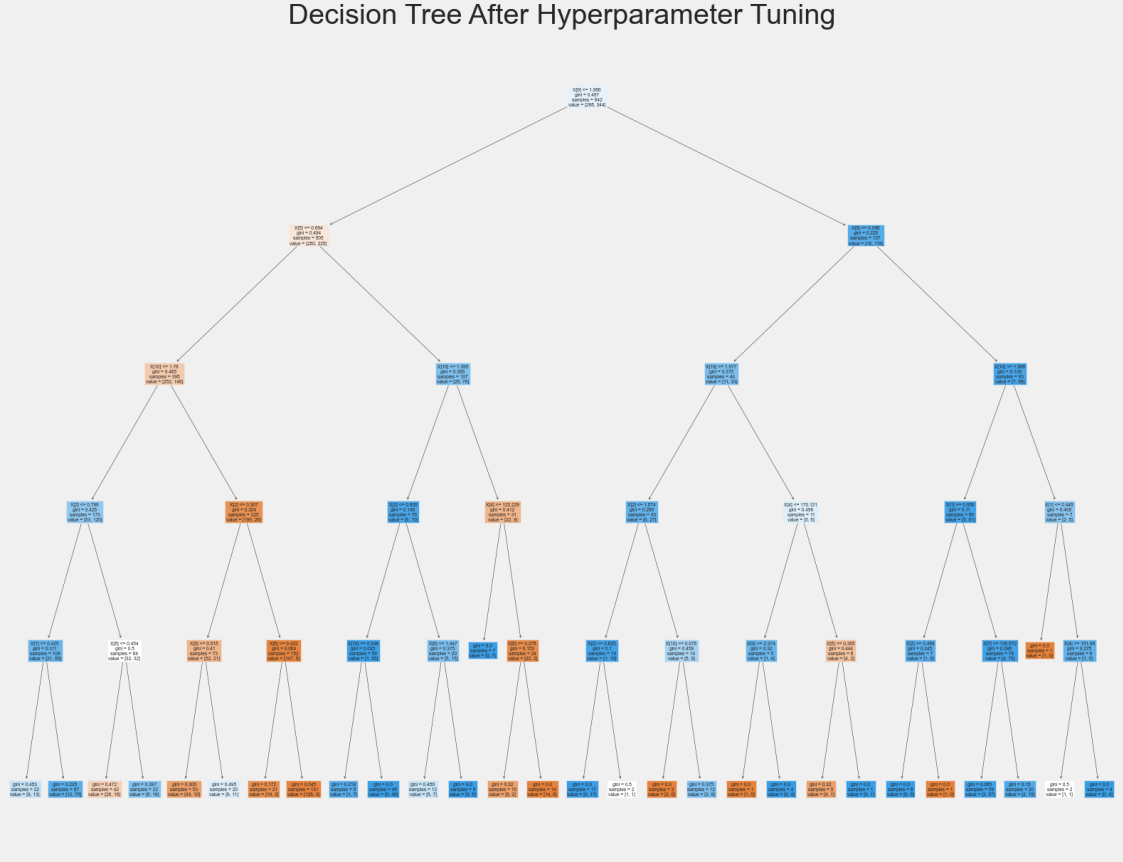
[285]:
```
params = {
    "criterion":("gini", "entropy"),
    "splitter":("best", "random"),
    "max_depth":(list(range(1, 20))),
    "min_samples_split":[2, 3, 4],
    "min_samples_leaf":list(range(1, 20)),
    'max_features': list(range(1, 20))
}


tree_clf = DecisionTreeClassifier(random_state=42)
tree_cv = GridSearchCV(tree_clf, params, scoring="accuracy", n_jobs=-1,␣
 ↪verbose=1, cv=3)
tree_cv.fit(X_train, y_train)
best_params = tree_cv.best_params_
print(f"Best paramters: {best_params})")

tree_clf = DecisionTreeClassifier(**best_params)
tree_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 82308 candidates, totalling 246924 fits
Best paramters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 8,
'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'})

/Users/dalithendel/opt/anaconda3/lib/python3.8/site-
packages/sklearn/model_selection/_search.py:918: UserWarning: One or more of the
test scores are non-finite: [0.7165109  0.53582555 0.7165109  …        nan
nan        nan]
  warnings.warn(
```

[285]:
```
DecisionTreeClassifier(max_depth=5, max_features=8, splitter='random')
```

[293]:
```
from sklearn import tree
fig = plt.figure(figsize=(25,20))
T= tree.plot_tree(tree_clf, filled = True)
plt.title('Decision Tree After Hyperparameter Tuning', size=40)
```

```
fig.savefig('DTHT.png')
plt.show()
```



[294]:
```
#function to find acc scores and confusion matrix
def score(clf, X_train, y_train, X_test, y_test, train=True, pos_label="Yes"):
    if train == False:
        pred = clf.predict(X_test)
        print("Test Result:\n")
        print(f"accuracy score: {accuracy_score(y_test, pred)}\n")
        print(f"Classification Report: \n \tPrecision: {precision_score(y_test,␣
 ↪pred)}\n\tRecall Score: {recall_score(y_test, pred)}\n\tF1 score:␣
 ↪{f1_score(y_test, pred)}\n")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")

score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Test Result:

accuracy score: 0.8623188405797102
```

```
Classification Report:
        Precision: 0.92
        Recall Score: 0.8414634146341463
        F1 score: 0.8789808917197452

Confusion Matrix:
 [[100  12]
 [ 26 138]]
```

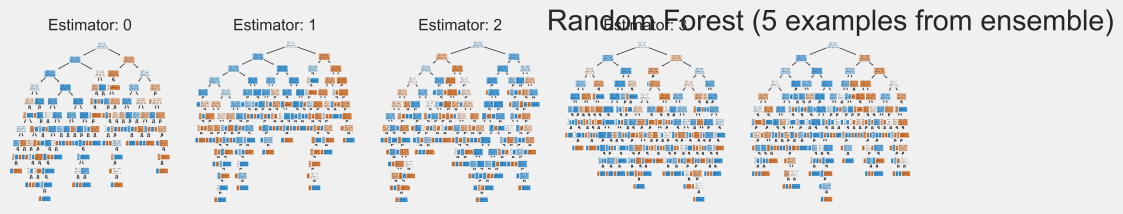
[295]:
```
#Evaluating second decision tree model
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Test Result:

accuracy score: 0.8623188405797102

Classification Report:
        Precision: 0.92
        Recall Score: 0.8414634146341463
        F1 score: 0.8789808917197452

Confusion Matrix:
 [[100  12]
 [ 26 138]]
```

[296]:
```
## random forest
rf_clf = RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)

print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Test Result:

accuracy score: 0.8876811594202898

Classification Report:
        Precision: 0.9182389937106918
        Recall Score: 0.8902439024390244
        F1 score: 0.9040247678018576

Confusion Matrix:
 [[ 99  13]
 [ 18 146]]
```

[297]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (10,2), dpi=900)
for index in range(0, 5):
    tree.plot_tree(rf_clf.estimators_[index],
                   filled = True,
                   ax = axes[index]);

    axes[index].set_title('Estimator: ' + str(index), fontsize = 11)
fig.savefig('rf_5trees.png')
plt.title('Random Forest (5 examples from ensemble)', size = 20)
plt.show()
```



[303]:
```
#finding feature importance for RF
importance = rf_clf.feature_importances_
importance_feats = X_train.columns
```

[310]:
```
#same as above and printing a DF
df_importance = pd.DataFrame({'Feature':importance_feats, 'Percentage':
 ↪importance})
df_importance.sort_values('Percentage', ascending=False, inplace=True)
df_importance.Percentage = df_importance.Percentage.apply(lambda x: x * 100)
```

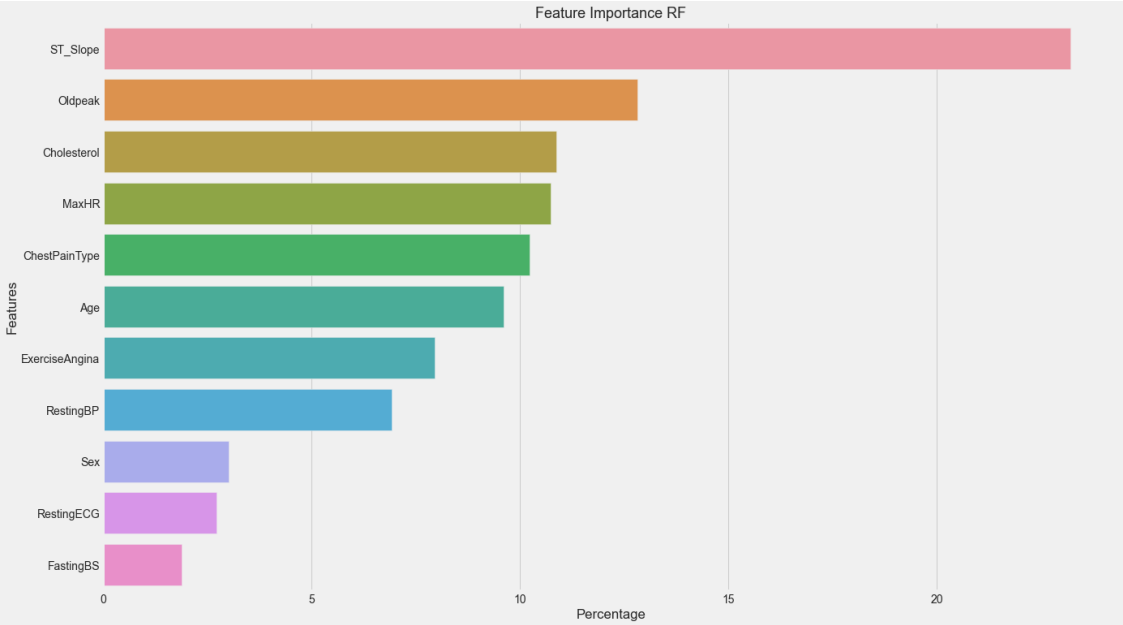
[311]:
```
df_importance
```

[311]:

	Feature	Percentage
10	ST_Slope	23.202805
9	Oldpeak	12.814285
4	Cholesterol	10.882240
7	MaxHR	10.737578
2	ChestPainType	10.237212
0	Age	9.619771
8	ExerciseAngina	7.955635
3	RestingBP	6.930208
1	Sex	3.015218
6	RestingECG	2.720309
5	FastingBS	1.884739

[312]:
```
#plotting Feature importance for RF
plt.figure(figsize=(20,12))
```

```
sns.barplot(x='Percentage', y='Feature', data=df_importance)

plt.title('Feature Importance RF', fontsize=18)
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
plt.xlabel('Percentage')
plt.ylabel('Features')
fig.savefig('feat.png')
plt.show()

## ST_Slope is most important feature
```



[ ]: