Final Assignment

The Purpose of this analysis is to find out, for a given number of variables of a heart patient, if the patient could have a potential heart failure or not.

This is a classfication type problem. For this, we will first perform analysis, pre-process the data and find out the accuracy of 3 different classifiers.

In [129]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from plotly.subplots import make_subplots
import plotly.express as px
import plotly.graph_objs as go

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.metrics import accuracy_score

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [130]:
```
df = pd.read_csv('heart.csv')
```

1) Data exploration: Here we analyse the data look for probable causes and relations to our target variable.

In [131]:
```
df.head()
```

Out[131]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

In [132]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
Age               918 non-null int64
Sex               918 non-null object
ChestPainType     918 non-null object
RestingBP         918 non-null int64
Cholesterol       918 non-null int64
FastingBS         918 non-null int64
RestingECG        918 non-null object
MaxHR             918 non-null int64
ExerciseAngina    918 non-null object
Oldpeak           918 non-null float64
ST_Slope          918 non-null object
HeartDisease      918 non-null int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.1+ KB
```

Looking for Null or undefined fields

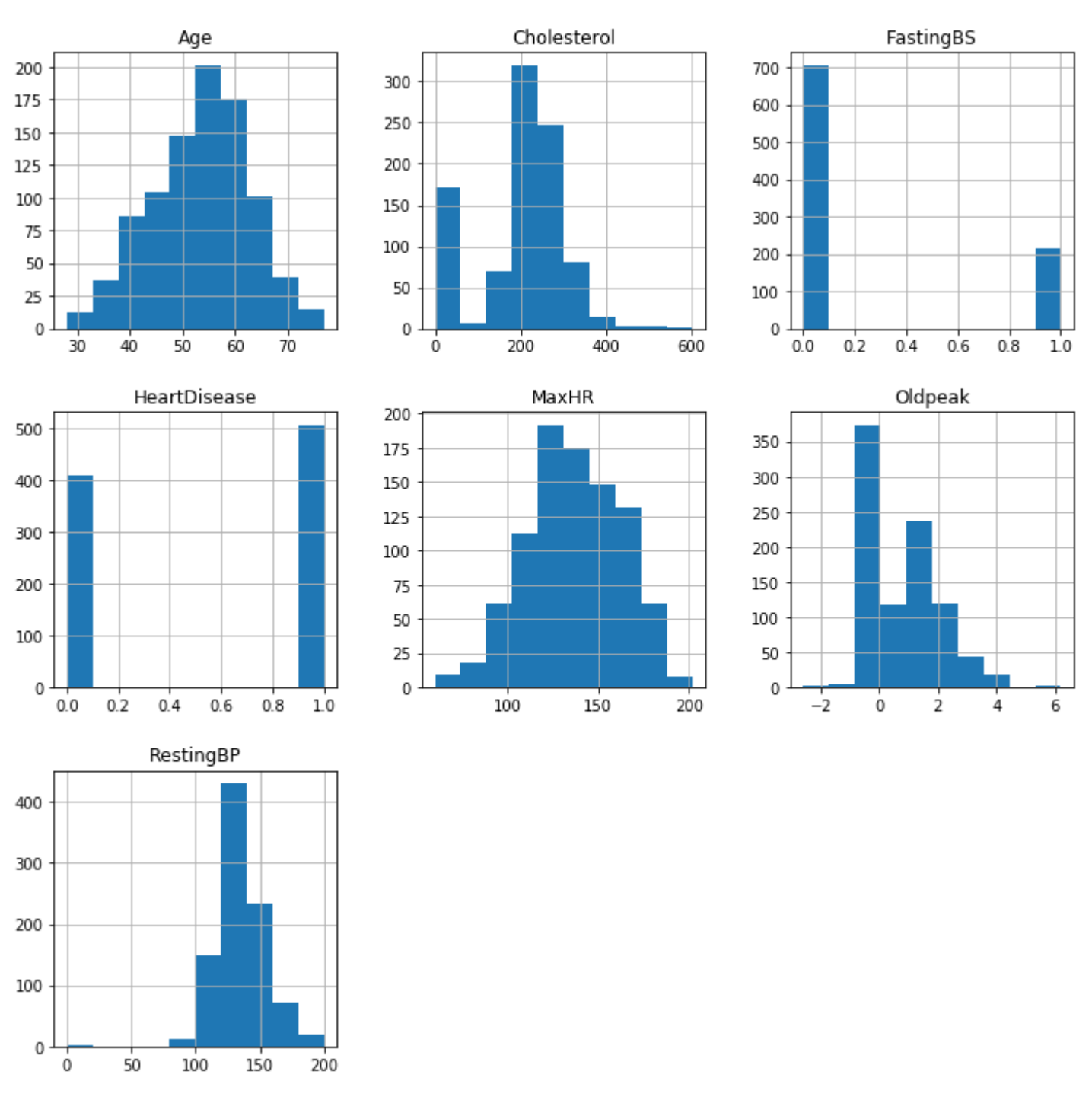
In [133]:
```
df.isna().sum()
```

Out[133]:
```
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
```

Manual analysis: Looking manually for correlation of data

In [134]:
```
df.hist(figsize=(12,12))
```

Out[134]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000211919053C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000211916357B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021191658A20>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021191682EF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000211916AF4A8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000211916D5A20>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000211916FEF98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002119172C588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002119172C5C0>]],
      dtype=object)
```



In [135]:
```
hd = df.corr()["HeartDisease"]
hd = pd.DataFrame(hd)
hd
```

Out[135]:

	HeartDisease
Age	0.282039
RestingBP	0.107589
Cholesterol	-0.232741
FastingBS	0.267291
MaxHR	-0.400421
Oldpeak	0.403951
HeartDisease	1.000000

In [136]:
```
pd.pivot_table(df, index='HeartDisease', values='Cholesterol', aggfunc='mean')
```

Out[136]:

HeartDisease	Cholesterol
0	227.121951
1	175.940945

In [137]:
```
df1 = pd.pivot_table(df, index='ChestPainType', values=['RestingBP', 'Cholesterol', 'MaxHR'], aggfunc=
'mean')
df1
```

Out[137]:

ChestPainType	Cholesterol	MaxHR	RestingBP
ASY	186.645161	128.477823	133.229839
ATA	233.046243	150.208092	130.624277
NAP	197.438424	143.236453	130.960591
TA	207.065217	147.891304	136.413043

In [138]:
```
px.bar(df1, x=df1.index, y=['RestingBP', 'Cholesterol', 'MaxHR'], barmode='group')
```

In [139]:
```
df2 = pd.pivot_table(df, index='ChestPainType', columns='Sex', values=['RestingBP', 'Cholesterol', 'Max
HR'], aggfunc='mean')
df2
```

Out[139]:

	Cholesterol		MaxHR		RestingBP	
Sex	F	M	F	M	F	M
ChestPainType						
ASY	235.200000	178.666667	139.128571	126.727700	136.242857	132.734742
ATA	249.350000	224.389381	152.366667	149.061947	128.650000	131.672566
NAP	245.603774	180.420000	148.528302	141.366667	129.264151	131.560000
TA	210.900000	206.000000	145.200000	148.638889	141.000000	135.138889

Looks like there are a lot more males with disease than females.

In [140]:
```
px.bar(df, x='Sex', y='HeartDisease')
```

In [141]:
```
df_angina = pd.pivot_table(df, index='HeartDisease', columns='ExerciseAngina', values='MaxHR', aggfunc=
'count')

fig = make_subplots(rows=1, cols=2, specs=[[{'type':'pie'},{'type':'pie'}]], subplot_titles=('No', 'Ye
s'))

fig.add_trace(go.Pie(values=df_angina['N'], labels=df_angina.index),row=1, col=1)

fig.add_trace(go.Pie(values=df_angina['Y'], labels=df_angina.index),row=1, col=2)

fig.update_layout(title_text='Exercise Angina', title_x=0.5, title_font_size=40, showlegend=False)
```

Converting object and string type data to numerical data for better analysis and later, better models

In [142]:
```
print(df['Sex'].unique())
print(df['ChestPainType'].unique())
print(df['RestingECG'].unique())
print(df['ExerciseAngina'].unique())
print(df['ST_Slope'].unique())
```

```
['M' 'F']
['ATA' 'NAP' 'ASY' 'TA']
['Normal' 'ST' 'LVH']
['N' 'Y']
['Up' 'Flat' 'Down']
```

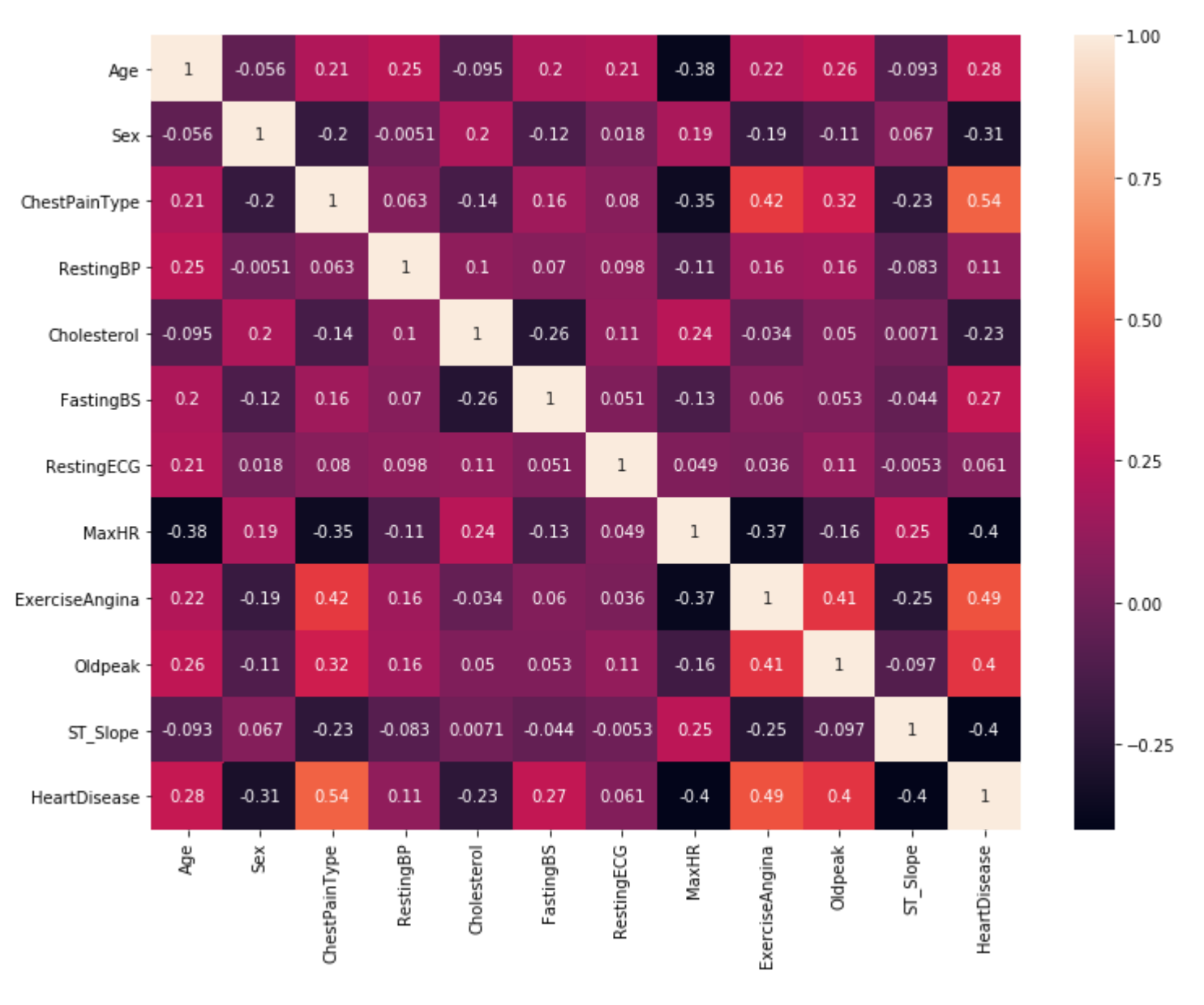
In [143]:
```
df['ChestPainType'] = df['ChestPainType'].replace({'ATA': 0, 'NAP' : 1, 'TA' : 2, 'ASY' : 3})
df['RestingECG'] = df['RestingECG'].replace({'Normal': 0, 'ST' : 1, 'LVH' : 2})
df['Sex'] = df['Sex'].replace({'M': 0, 'F' : 1})
df['ExerciseAngina'] = df['ExerciseAngina'].replace({'N': 0, 'Y' : 1})
df['ST_Slope'] = df['ST_Slope'].replace({'Up': 1, 'Flat' : 0, 'Down' : 2})
df.head()
```

Out[143]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	0	0	140	289	0	0	172	0	0.0	1	0
1	49	1	1	160	180	0	0	156	0	1.0	0	1
2	37	0	0	130	283	0	1	98	0	0.0	1	0
3	48	1	3	138	214	0	0	108	1	1.5	0	1
4	54	0	1	150	195	0	0	122	0	0.0	1	0

In [144]:
```
fig = plt.figure(figsize=(12, 9))
sns.heatmap(df.corr(),annot=True)
```

Out[144]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21191944978>
```



2) Data-Preprocessing: Here data is looked at for anomalies and split into training set and test set

In [154]:
```
df_train = df.copy()

lbl_encode = LabelEncoder()
for i in df_categorical.columns:
    df_train[i] = df_train[[i]].apply(lbl_encode.fit_transform)
```

In [155]:
```
df_train = df_train.drop(columns='HeartDisease')
target = df['HeartDisease'].copy()

x_train, x_test, y_train, y_test = train_test_split(df_train, target, test_size=0.15, random_state=42)

scaled_features_train = StandardScaler().fit_transform(x_train.values)
scaled_features_test = StandardScaler().fit_transform(x_test.values)
x_train_scaled = pd.DataFrame(scaled_features_train, index=x_train.index, columns=df_train.columns)
x_test_scaled = pd.DataFrame(scaled_features_test, index=x_test.index, columns=df_train.columns)
```

In [156]:
```
def clf_performance(classifier, model_name):
    print(model_name)
    print('Best Score: ' + str(classifier.best_score_))
    print('Best Parameters: ' + str(classifier.best_params_))
```

3) Machine Learning: Here we compare 3 different classifiers and their result

First we look at logistic regression

In [147]:
```
lr = LogisticRegression(max_iter=2000)
cv = cross_val_score(lr, x_train_scaled, y_train, cv=5)
print(cv)
print(cv.mean())
```

```
[0.89808917 0.83333333 0.82051282 0.83974359 0.8516129 ]
0.8486583637580146
```

In [160]:
```
lr = LogisticRegression()
param_grid = {'max_iter': [2000],
              'penalty': ['l1', 'l2'],
              'C': np.logspace(-4, 4, 20),
              'solver': ['liblinear']}
clf_lr = GridSearchCV(lr, param_grid=param_grid, cv=5, verbose=True, n_jobs=-1)
best_clf_lr = clf_lr.fit(x_train_scaled,y_train)
clf_performance(best_clf_lr, 'LogisticRegression')
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
LogisticRegression
Best Score: 0.8525641025641025
Best Parameters: {'C': 0.23357214690901212, 'max_iter': 2000, 'penalty': 'l1', 'solver': 'liblinear'}
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    1.0s finished
```

Second we look at logistic KNN classifier

In [148]:
```
knn = KNeighborsClassifier(n_neighbors=5)
cv = cross_val_score(knn,x_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.92356688 0.84615385 0.89102564 0.8525641  0.86451613]
0.875565319551348
```

In [159]:
```
knn = KNeighborsClassifier()
param_grid = {'n_neighbors': [3, 5, 7, 9],
              'weights': ['uniform', 'distance'],
              'algorithm': ['auto', 'ball_tree', 'kd_tree'],
              'p': [1, 2]}
clf_knn = GridSearchCV(knn, param_grid = param_grid, cv = 5, verbose = True, n_jobs = -1)
best_clf_knn = clf_knn.fit(x_train_scaled,y_train)
clf_performance(best_clf_knn,'KNN')
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   12.6s
KNN
Best Score: 0.8833333333333333
Best Parameters: {'algorithm': 'auto', 'n_neighbors': 9, 'p': 2, 'weights': 'distance'}
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed:   14.7s finished
```

Lastlly we look at Random forest classifier

In [157]:
```
dt = RandomForestClassifier(random_state = 1)
cv = cross_val_score(dt,x_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.88535032 0.83974359 0.83333333 0.85897436 0.84516129]
0.8525125781690402
```

In [158]:
```
rf = RandomForestClassifier(random_state = 1)
param_grid = {'n_estimators': [50, 100, 200],
               'criterion': ['gini', 'entropy'],
              'min_samples_leaf': [1, 2, 3],
              }
clf_rf = GridSearchCV(rf, param_grid=param_grid, cv=5, verbose=True, n_jobs=-1)
best_clf_rf = clf_rf.fit(x_train_scaled, y_train)
clf_performance(best_clf_rf, 'RandomForestClassifier')
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   13.2s
RandomForestClassifier
Best Score: 0.8743589743589744
Best Parameters: {'criterion': 'gini', 'min_samples_leaf': 3, 'n_estimators': 50}
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:   16.0s finished
```