Heart Disease Prediction

Data Set Info: Dataset contains information of patients including age, sex, cholestrol levels, other medical information.

Aim: Make predictions using Machine Learning whether a person is suffering from Heart Disease or not.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

For processing the data, I have split the available dataset for testing and training, I'll use the `train_test_split` method, and to scale the features, I am using `StandardScaler`.

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

I have imported all the Machine Learning algorithms I have used:

1. K Neighbors Classifier
2. Support Vector Classifier
3. Decision Tree Classifier
4. Random Forest Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
dataset = pd.read_csv('dataset.csv')
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

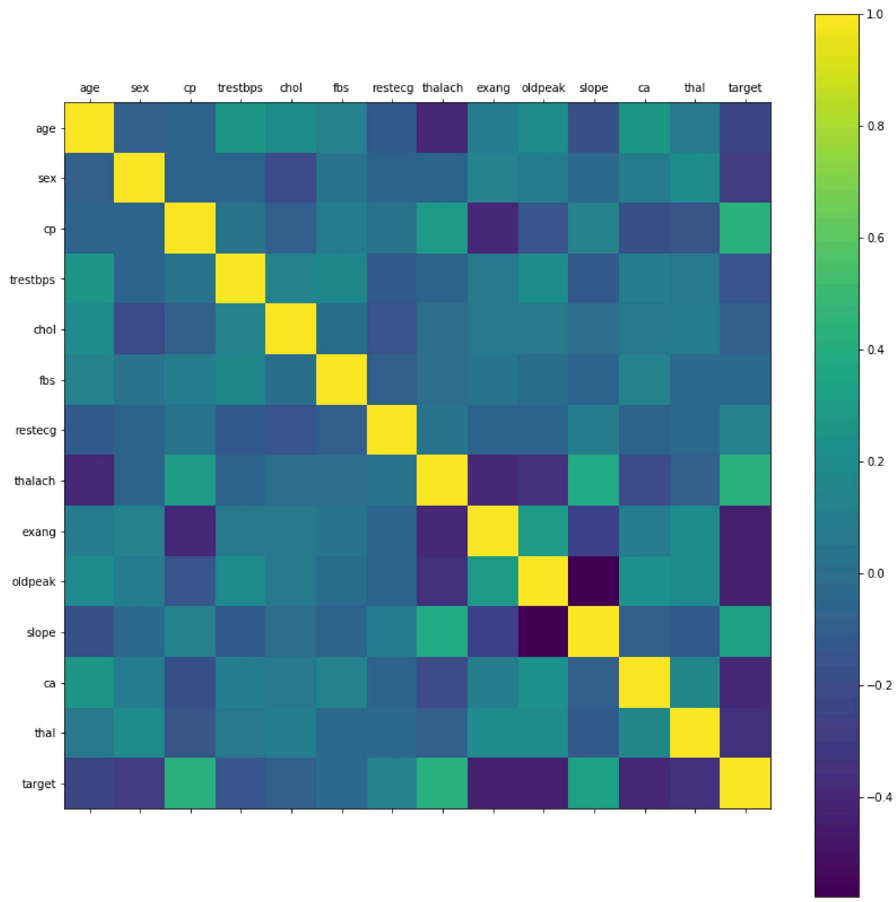
```
dataset.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

The scale of each feature column is different and quite varied as well. While the maximum for `age` reaches 77, the maximum of `chol` (serum cholestoral) is 564.

```
rcParams['figure.figsize'] = 20, 14
plt.matshow(dataset.corr())
plt.yticks(np.arange(dataset.shape[1]), dataset.columns)
plt.xticks(np.arange(dataset.shape[1]), dataset.columns)
plt.colorbar()
```

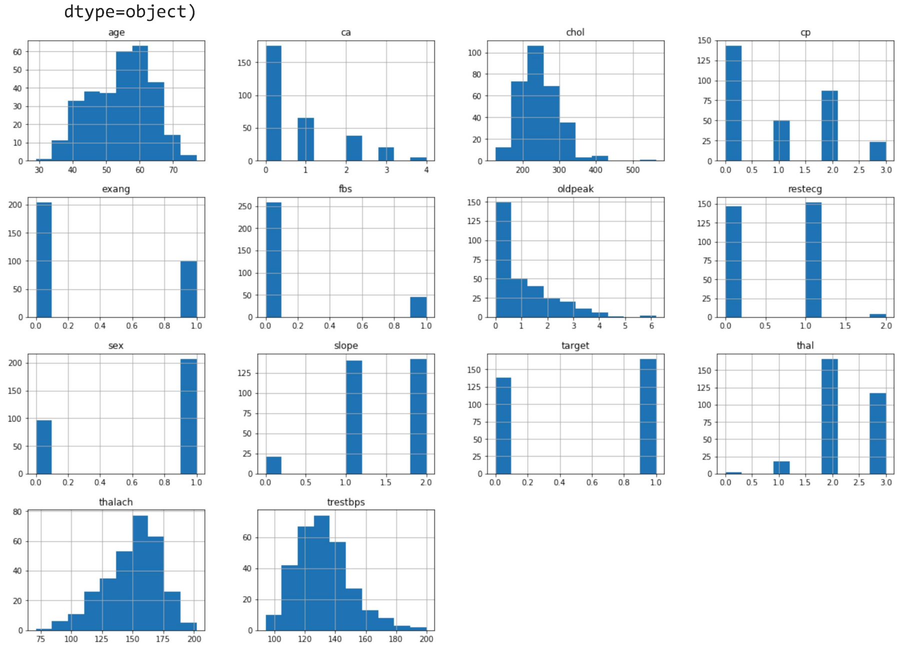
```
<matplotlib.colorbar.Colorbar at 0x1a15fc91d0>
```



A few features have negative correlation with the target value while some have positive.

```
dataset.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a15d3edd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a16d85940>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a16d0dba8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a16d37e10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a175430b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a16dd3320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a16dfc588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a1789b828>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a1789b860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a178edcc0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a17916f28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a1794b1d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a17972438>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a1799b6a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a179c7908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a179f1b70>]],
      dtype=object)
```



Scaling the features helped determining that each feature has a different range of distribution. Also, the categorical features do stand out.

```
rcParams['figure.figsize'] = 8,6
plt.bar(dataset['target'].unique(), dataset['target'].value_counts(), color = ['red', 'green'])
plt.xticks([0, 1])
plt.xlabel('Target Classes')
plt.ylabel('Count')
plt.title('Count of each Target Class')
```

```
Text(0.5, 1.0, 'Count of each Target Class')
```

Data Processing

Cretaed Dummy columns using `get_dummies` to categorize data into categorial variables and scaled the Machine Learning models.

```
dataset = pd.get_dummies(dataset, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
```

Used `StandardScaler` from `sklearn` to scale the dataset.

```
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```

Machine Learning

Importing `train_test_split` to split our dataset into training and testing datasets and importing it into all the models.

```
y = dataset['target']
X = dataset.drop(['target'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 0)
```

K Neighbors Classifier

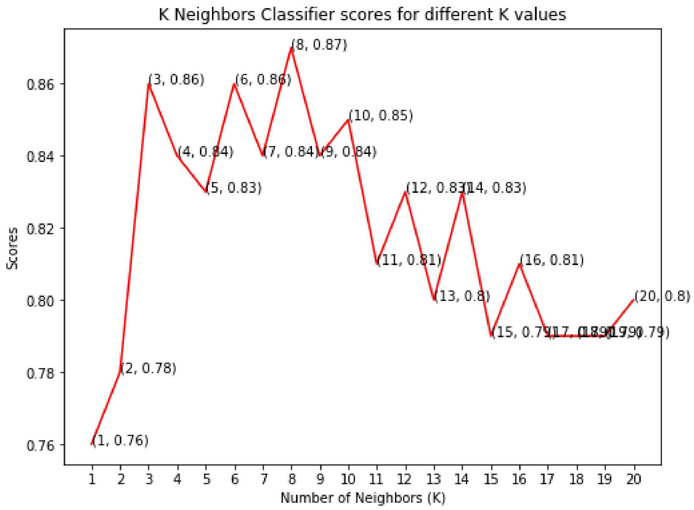
The classification score varies based on different values of neighbors that we choose. Thus, I'll plot a score graph for different values of K (neighbors) and check when do I achieve the best score.

```
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    knn_classifier.fit(X_train, y_train)
    knn_scores.append(knn_classifier.score(X_test, y_test))
```

I have the scores for different neighbor values in the array `knn_scores`. I'll now plot it and see for which value of K did I get the best scores.

```
plt.plot([k for k in range(1, 21)], knn_scores, color = 'red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1, 21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('K Neighbors Classifier scores for different K values')
```

```
Text(0.5, 1.0, 'K Neighbors Classifier scores for different K values')
```



From the plot above, it is clear that the maximum score achieved was `0.87` for the 8 neighbors.

```
print("The score for K Neighbors Classifier is {}% with {} nieghbors.".format(knn_scores[7]*100, 8))
```

```
The score for K Neighbors Classifier is 87.0% with 8 nieghbors.
```

Support Vector Classifier

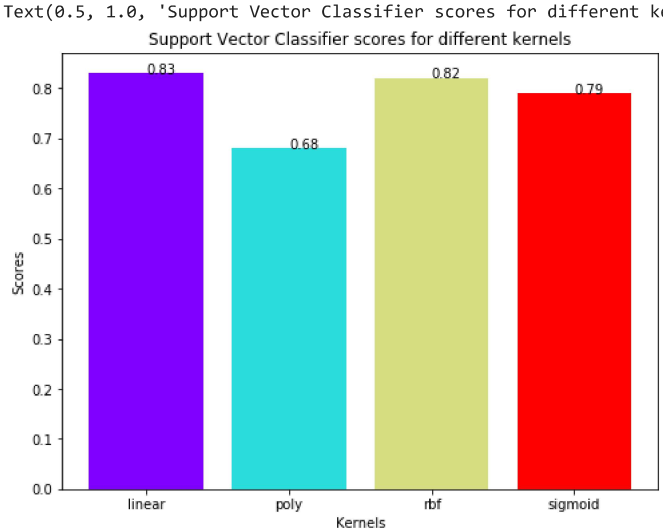
There are several kernels for Support Vector Classifier. I'll test some of them and check which has the best score.

```
svc_scores = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for i in range(len(kernels)):
    svc_classifier = SVC(kernel = kernels[i])
    svc_classifier.fit(X_train, y_train)
    svc_scores.append(svc_classifier.score(X_test, y_test))
```

I'll now plot a bar plot of scores for each kernel and see which performed the best.

```
colors = rainbow(np.linspace(0, 1, len(kernels)))
plt.bar(kernels, svc_scores, color = colors)
for i in range(len(kernels)):
    plt.text(i, svc_scores[i], svc_scores[i])
plt.xlabel('Kernels')
plt.ylabel('Scores')
plt.title('Support Vector Classifier scores for different kernels')
```

```
Text(0.5, 1.0, 'Support Vector Classifier scores for different kernels')
```



The `linear` kernel performed the best, being slightly better than `rbf` kernel.

```
print("The score for Support Vector Classifier is {}% with {} kernel.".format(svc_scores[0]*100, 'linear'))
```

```
The score for Support Vector Classifier is 83.0% with linear kernel.
```

Decision Tree Classifier

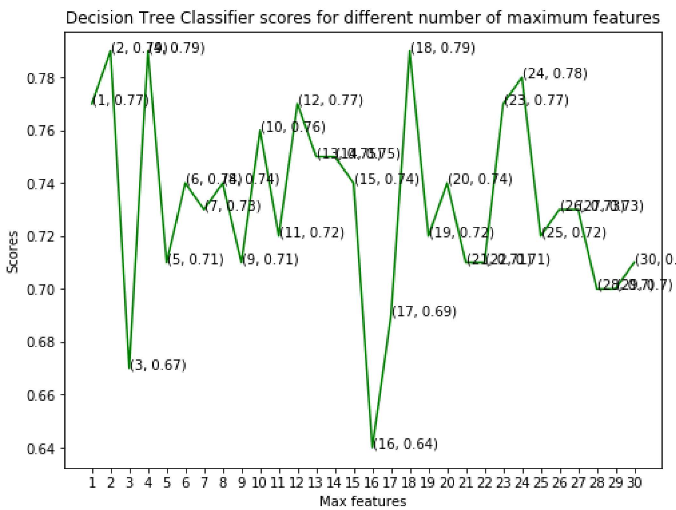
Here, I'll use the Decision Tree Classifier to model the problem at hand. I'll vary between a set of `max_features` and see which returns the best accuracy.

```
dt_scores = []
for i in range(1, len(X.columns) + 1):
    dt_classifier = DecisionTreeClassifier(max_features = i, random_state = 0)
    dt_classifier.fit(X_train, y_train)
    dt_scores.append(dt_classifier.score(X_test, y_test))
```

I selected the maximum number of features from 1 to 30 for split. Now, let's see the scores for each of those cases.

```
plt.plot([i for i in range(1, len(X.columns) + 1)], dt_scores, color = 'green')
for i in range(1, len(X.columns) + 1):
    plt.text(i, dt_scores[i-1], (i, dt_scores[i-1]))
plt.xticks([i for i in range(1, len(X.columns) + 1)])
plt.xlabel('Max features')
plt.ylabel('Scores')
plt.title('Decision Tree Classifier scores for different number of maximum features')
```

```
Text(0.5, 1.0, 'Decision Tree Classifier scores for different number of maximum
features')
```



The model achieved the best accuracy at three values of maximum features, `2`, `4` and `18`.

```
print("The score for Decision Tree Classifier is {}% with {} maximum features.".format(dt_scores[17]*100, [2,4,18]))
```

```
The score for Decision Tree Classifier is 79.0% with [2, 4, 18] maximum features.
```

Random Forest Classifier

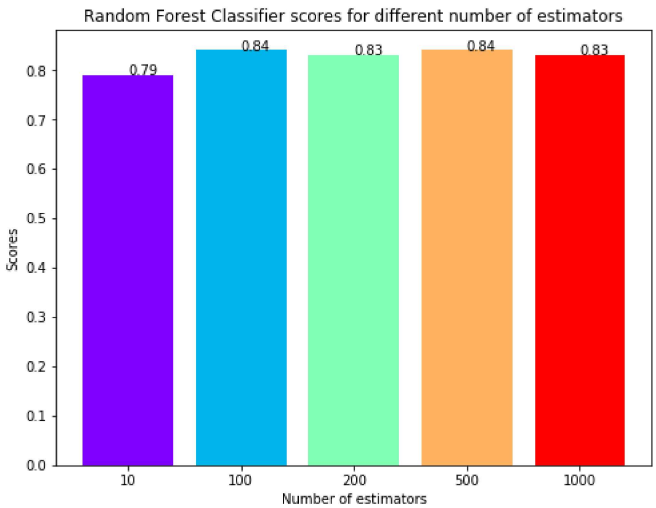
Now, I'll use the ensemble method, Random Forest Classifier, to create the model and vary the number of estimators to see their effect.

```
rf_scores = []
estimators = [10, 100, 200, 500, 1000]
for i in estimators:
    rf_classifier = RandomForestClassifier(n_estimators = i, random_state = 0)
    rf_classifier.fit(X_train, y_train)
    rf_scores.append(rf_classifier.score(X_test, y_test))
```

The model is trained and the scores are recorded. Let's plot a bar plot to compare the scores.

```
colors = rainbow(np.linspace(0, 1, len(estimators)))
plt.bar([i for i in range(len(estimators))], rf_scores, color = colors, width = 0.8)
for i in range(len(estimators)):
    plt.text(i, rf_scores[i], rf_scores[i])
plt.xticks(ticks = [i for i in range(len(estimators))], labels = [str(estimator) for estimator in estimators])
plt.xlabel('Number of estimators')
plt.ylabel('Scores')
plt.title('Random Forest Classifier scores for different number of estimators')
```

```
Text(0.5, 1.0, 'Random Forest Classifier scores for different number of estimators')
```



The maximum score is achieved when the total estimators are 100 or 500.

```
print("The score for Random Forest Classifier is {}% with {} estimators.".format(rf_scores[1]*100, [100, 500]))
```

```
The score for Random Forest Classifier is 84.0% with [100, 500] estimators.
```

Conclusion

In this project, I used Machine Learning to predict whether a person is suffering from a heart disease. After importing the data, I analysed it using plots. Then, I did generated dummy variables for categorical features and scaled other features. I then applied four Machine Learning algorithms, `K Neighbors Classifier`, `Support Vector Classifier`, `Decision Tree Classifier` and `Random Forest Classifier`. I varied parameters across each model to improve their scores. In the end, `K Neighbors Classifier` achieved the highest score of `87%` with `8 nearest neighbors`.