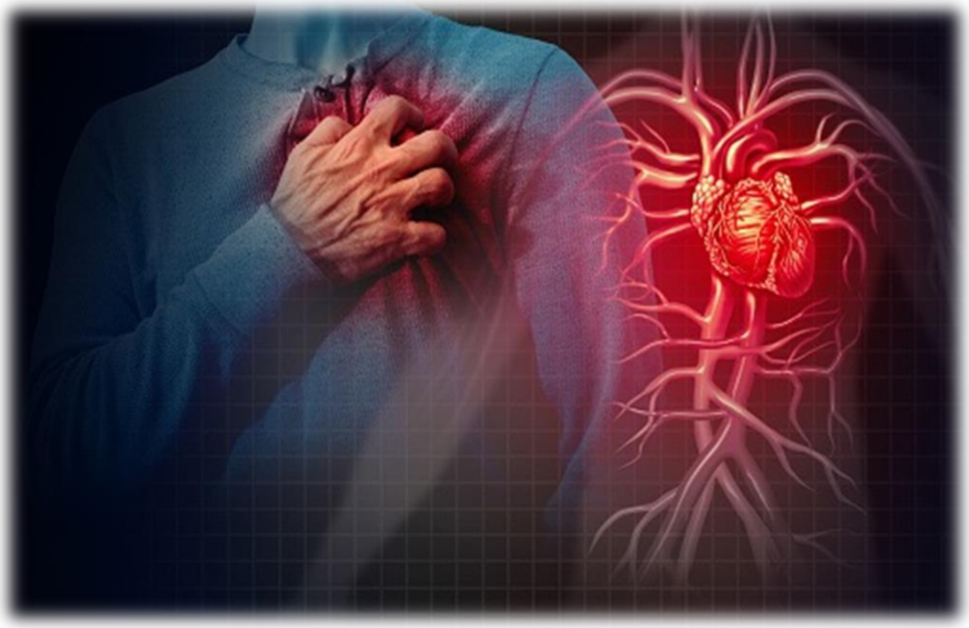


# Heart Disease Prediction

Problem Statement: Build a Machine Learning model for Heart Disease Prediction

Bikramaditya Chakraborty
ASTU Rollno- 180650007002

## Abstract

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year. CVDs are a group of disorders of the heart and blood vessels and include coronary heart disease, cerebrovascular disease, rheumatic heart disease and other conditions. Four out of 5 CVD deaths are due to heart attacks and strokes, and one third of these deaths occur prematurely in people under 70 years of age.
Heart diseases have also plagued India as it is now known to be the leading cause of death in the country. According to a report by Global Burden of Diseases in 2016, 1.7 million Indians died due to heart disease out of the world's 17.3 million deaths.

In this project, I applied four classifiers to the Heart Disease UCI dataset for heart disease prediction. The classifiers are:
1. **K Neighbours Classifier**
2. **Decision Tree Classifier**
3. **Random Forest Classifier**
4. **Support Vector Classifier**

I compared all the applied classifiers and found out the classifier which is best fit to this data for prediction of disease (based on the evaluation of each model).

The github repository containing my project is –

https://github.com/jintu7889/heart_disease_prediction

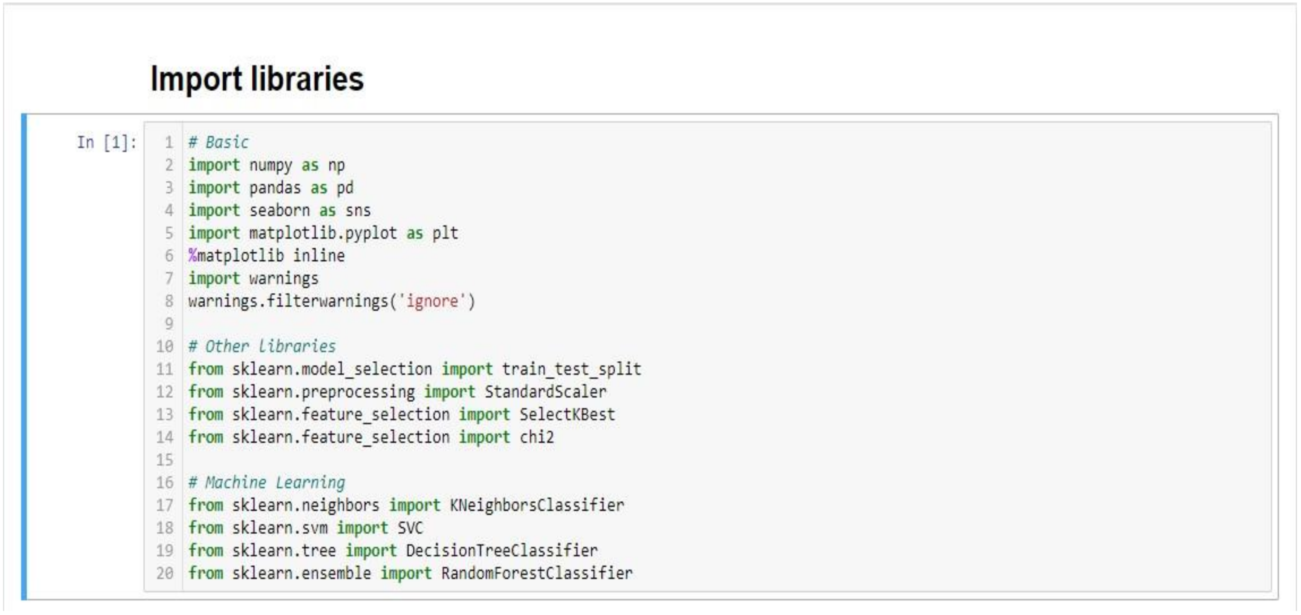
## Import libraries

I imported several libraries for the project:

1. **Numpy** : To work with arrays
2. **pandas** : To work with csv files and dataframes
3. **matplotlib** : To create charts using pyplot,
4. **seaborn :** To create an correlation matrix to get correlations of each features in dataset using ***sns.heatmap***
5. **warnings**: To ignore all warnings which might be showing up in the notebook due to past/future depreciation of a feature
6. **train_test_split**: To split the dataset into training and testing data
7. **StandardScaler**: To scale all the features, so that the Machine Learning model better adapts to the dataset
8. **SelectKBest and chi2**: To select features because irrelevant or partially relevant features can negatively impact model performance

I also imported all the necessary Machine Learning algorithms, which are:

1. KNeighborsClassifier from sklearn.neighbors
2. DecisionTreeClassifier from sklearn.tree
3. RandomForestClassifier from sklearn.ensemble
4. SVC from sklearn.svm



Import libraries

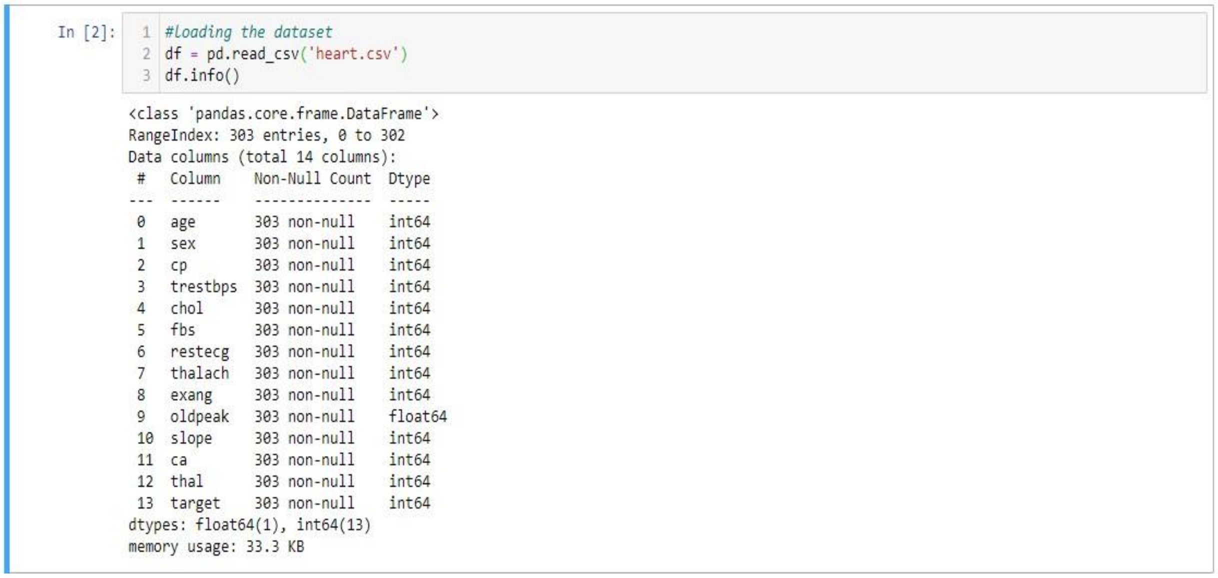
```
In [1]:
# Basic
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# Other Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

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

*Image 1: Code for importing the libraries*

## Loading Data Set

After downloading the dataset from Kaggle, I saved it to my working directory with the name heart.csv. Next, I used read_csv() to read the dataset and save it to the df variable.

## Loading dataset

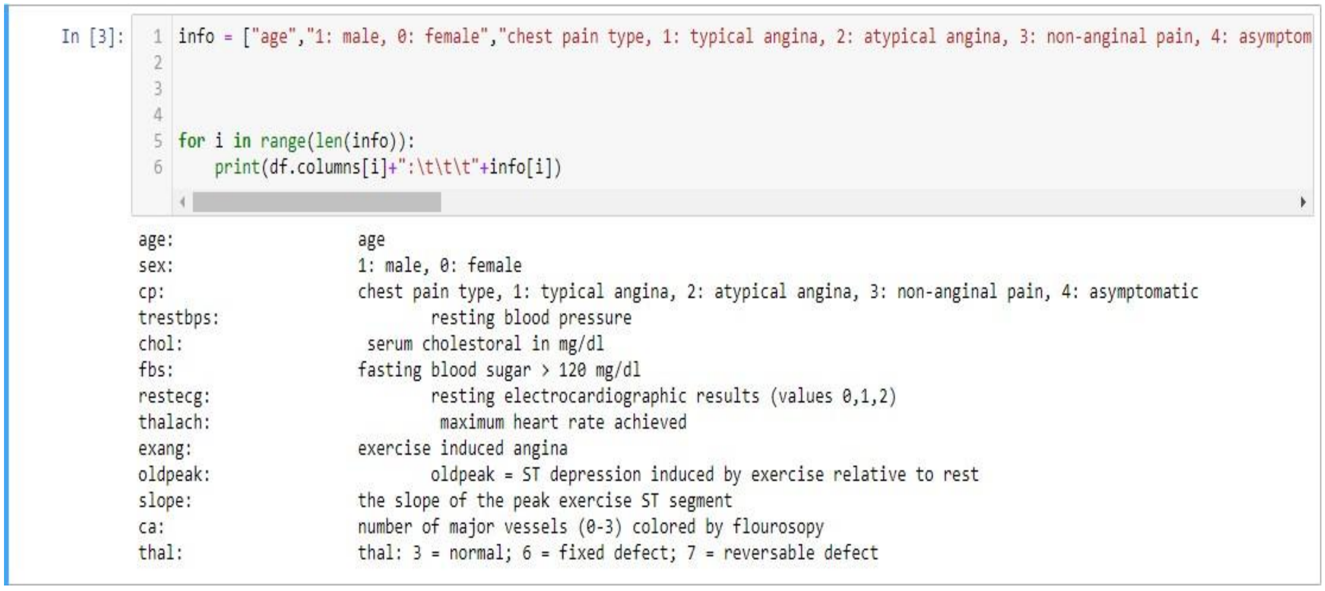


```
In [2]:
#Loading the dataset
df = pd.read_csv('heart.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

*Image 2: Code for loading the dataset and Output for df.info()*

Before any analysis, I just wanted to take a look at the data. So, I used the info() method. As we can see from the output above, there are a total of 13 features and 1 target variable. Also, there are no missing values so we don't need to take care of any null values.

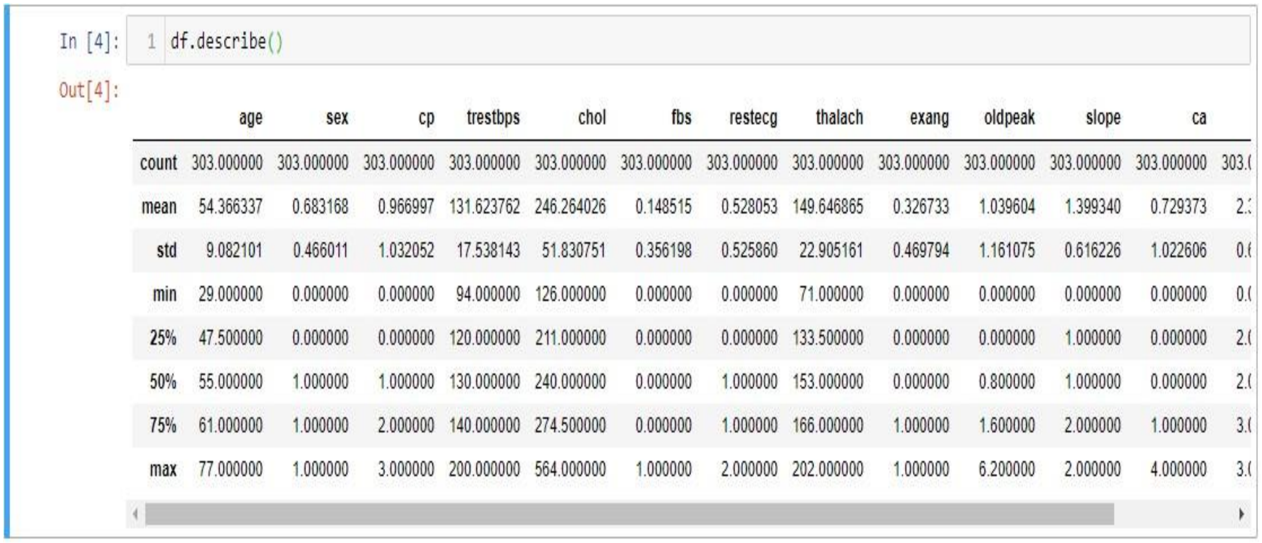


```
In [3]:
info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptom



for i in range(len(info)):
    print(df.columns[i]+":\t\t\t"+info[i])
```

```
age:                    age
sex:                    1: male, 0: female
cp:                     chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic
trestbps:                       resting blood pressure
chol:                    serum cholestoral in mg/dl
fbs:                    fasting blood sugar > 120 mg/dl
restecg:                        resting electrocardiographic results (values 0,1,2)
thalach:                         maximum heart rate achieved
exang:                  exercise induced angina
oldpeak:                        oldpeak = ST depression induced by exercise relative to rest
slope:                  the slope of the peak exercise ST segment
ca:                     number of major vessels (0-3) colored by flourosopy
thal:                   thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

*Image 3 :  code and result for all the columns data indetails*

Here I describe all the Columns information  in details.


In [4]:

```
df.describe()
```

Out[4]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.( |
| mean | 54.366337 | 0.683168 | 0.966997 | 131.623762 | 246.264026 | 0.148515 | 0.528053 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 | 2.: |
| std | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 | 0.( |
| min | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.( |
| 25% | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.( |
| 50% | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 | 2.( |
| 75% | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 | 3.( |
| max | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 | 3.( |

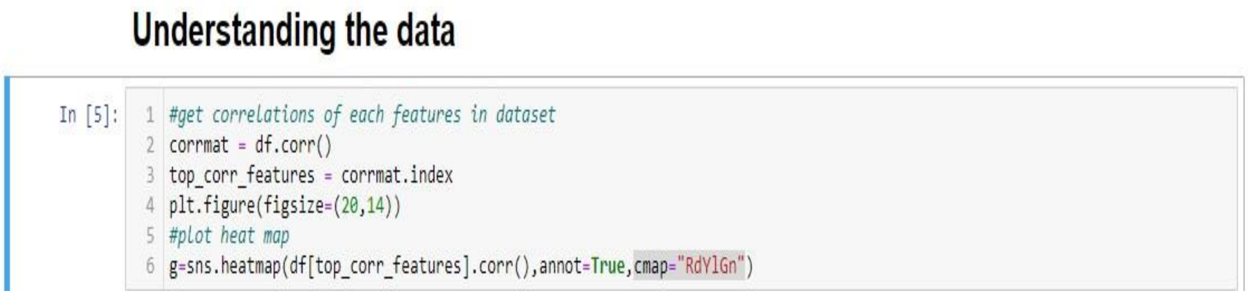
*Image 4: Output for df.describe()*

The method revealed that the range of each variable is different. The maximum value of age is 77 but for chol it is 564. Thus, feature scaling must be performed on the dataset.

## Understanding the Data

### Correlation Matrix

To begin with, let’s see the correlation matrix of features and try to analyse it. The figure size is defined to 12 x 8 by using sns.heatmap. Then, I used seaborn to show the correlation matrix. Using df[top_corr_features].corr(),annot=True, I’ve added names to the correlation matrix. cmap="RdYlGn" shows the colorbar for the matrix


**Understanding the data**

In [5]:

```
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,14))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

*Image 5: Code for showing the correlation matrix of features*

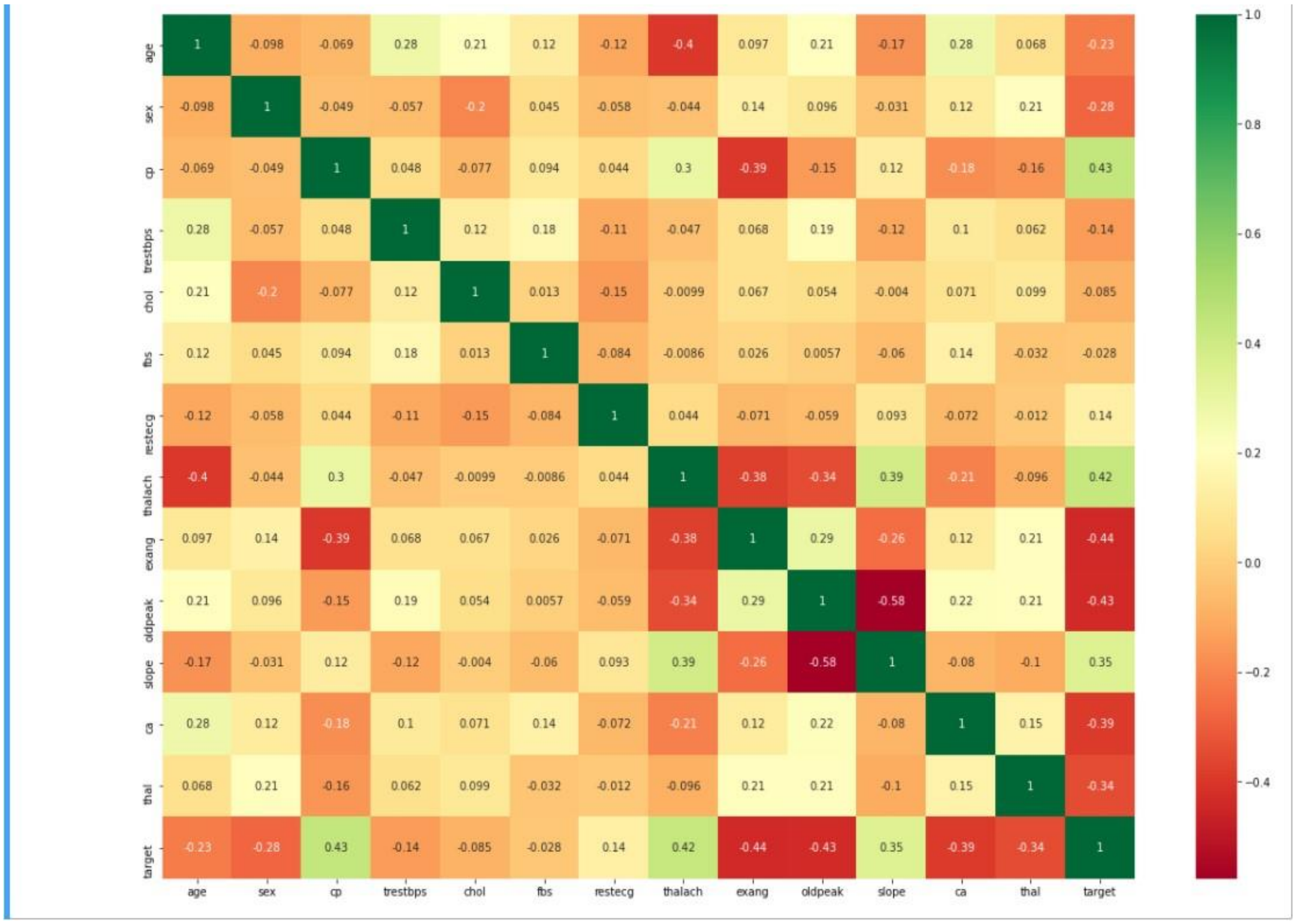


*Image 6: The correlation matrix of features*

It's easy to see that there is no single feature that has a very high correlation with our target value. Also, some of the features have a negative correlation with the target value and some have positive.

## Histogram

The best part about this type of plot is that it just takes a single command to draw the plots and it provides so much information in return. We just use

```
df.hist(figsize=(14,14))
        plt.show()
```

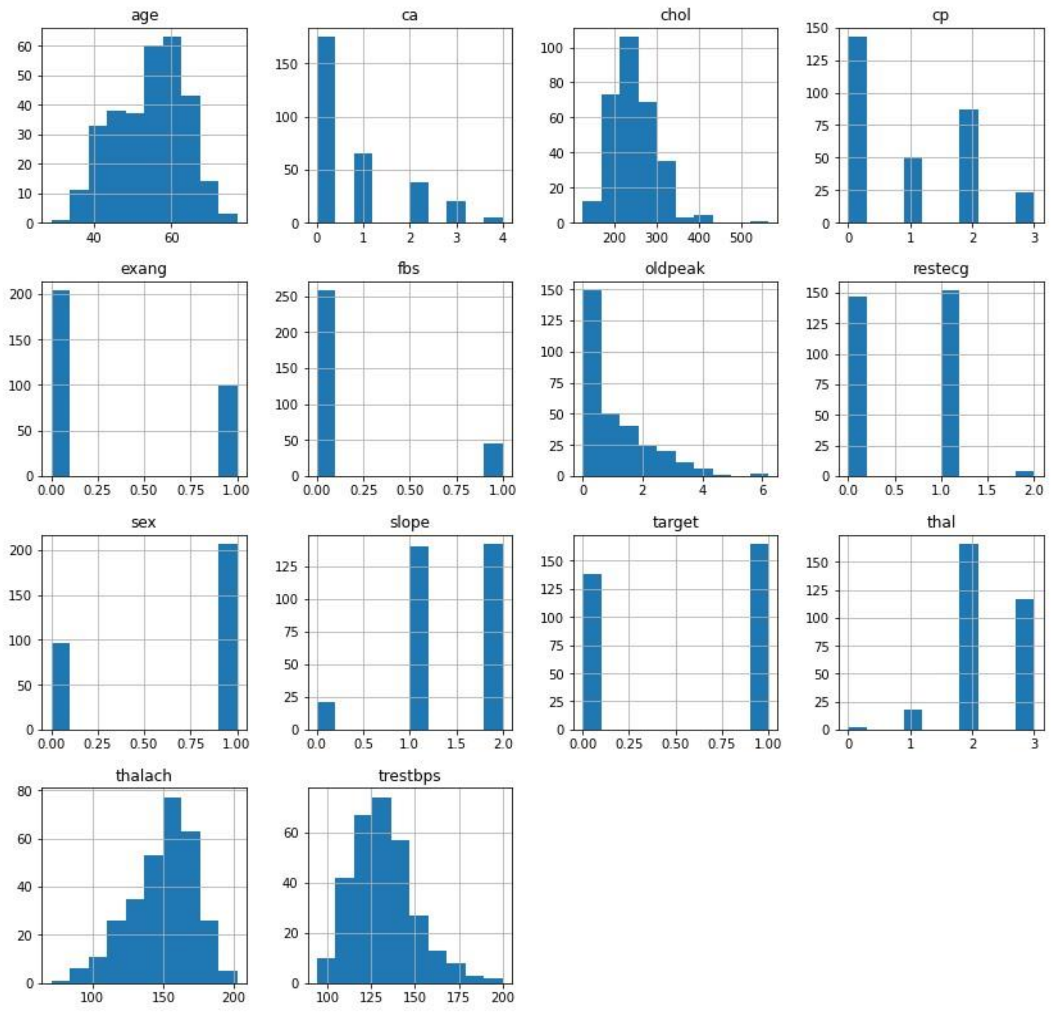
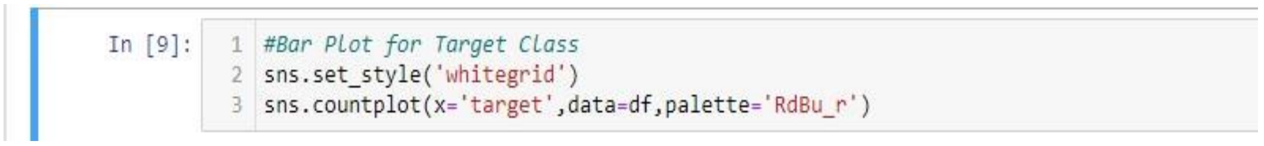


*Image 7: Histograms of features*

Let's take a look at the plots. It shows how each feature and label is distributed along different ranges, which further confirms the need for scaling. Next, wherever you see discrete bars, it basically means that each of these is actually a categorical variable. We will need to handle these categorical variables before applying Machine Learning. Our target labels have two classes, 0 for no disease and 1 for disease

## Bar Plot for Target Class

It's really essential that the dataset we are working on should be approximately balanced. An extremely imbalanced dataset can render the whole model training useless and thus, will be of no use.



```
In [9]:
#Bar Plot for Target Class
sns.set_style('whitegrid')
sns.countplot(x='target',data=df,palette='RdBu_r')
```

*Image 8: Code for plotting the bar plot of target classes*

For x-axis I used the Target values from the target column. For y-axis, I used count value to get the values for each class. We use ***whitegrid*** for the white colored background and initialize palette as ***RdBu_r*** for the Sky blue and Orange colored bars.

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x22a2843da00>
```

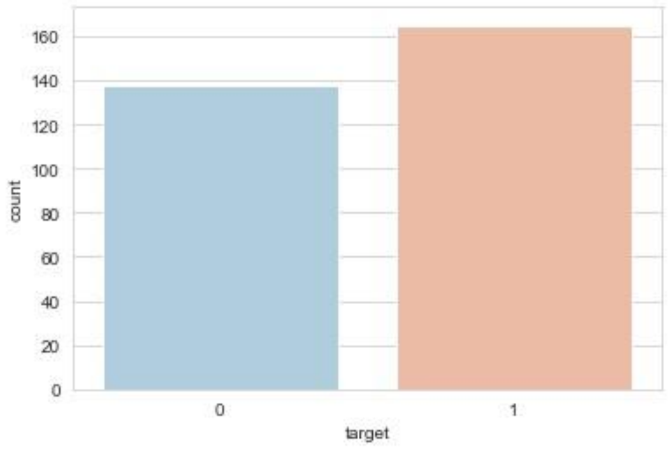
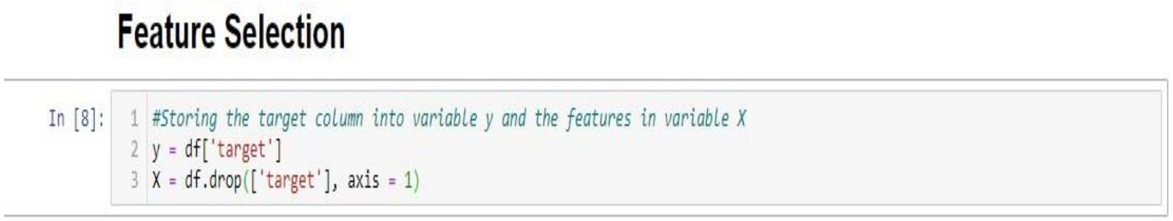


*Image 9: Bar plot of target classes*

## Feature Selection

We store the target column of the dataset into variable y and the features in variable X.

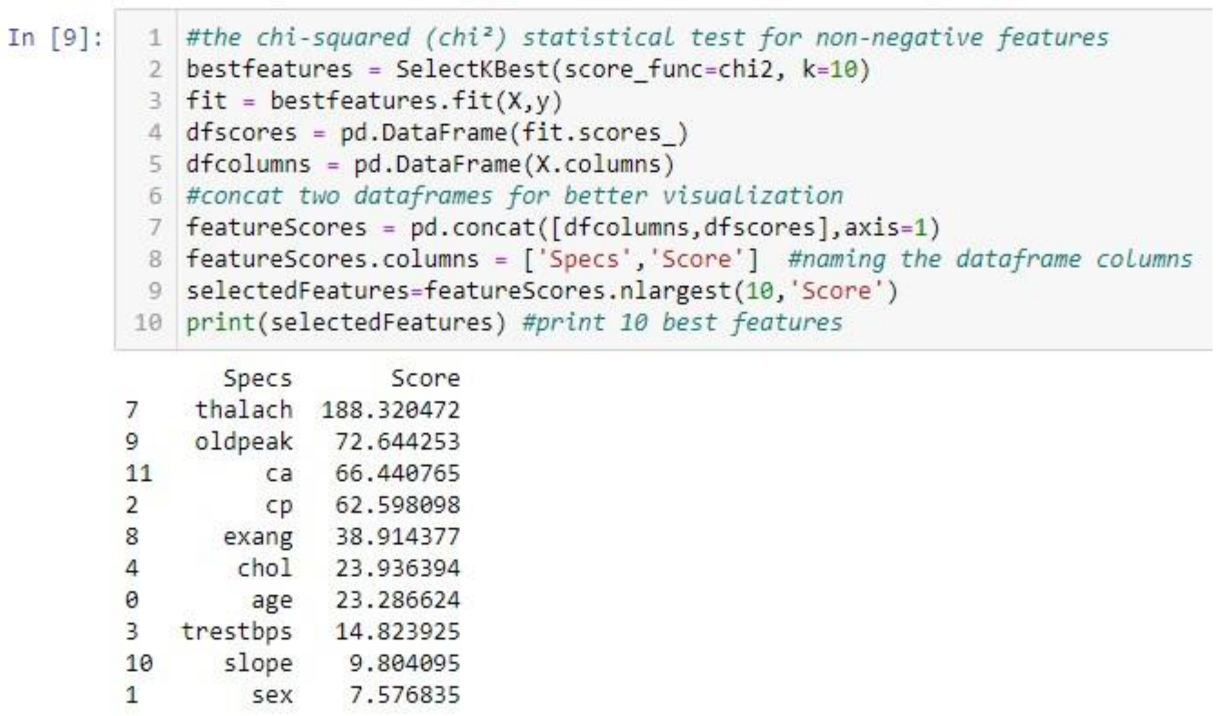


### Feature Selection

```
In [8]:
#Storing the target column into variable y and the features in variable X
y = df['target']
X = df.drop(['target'], axis = 1)
```

*Image 10: Code for storing the target column of the dataset into variable y and the features in variable X*

Feature Selection is the process where we automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make our model learn based on irrelevant features. Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features. I used the chi-squared (chi²) statistical test for non-negative features to select 10 of the best features from the dataset.

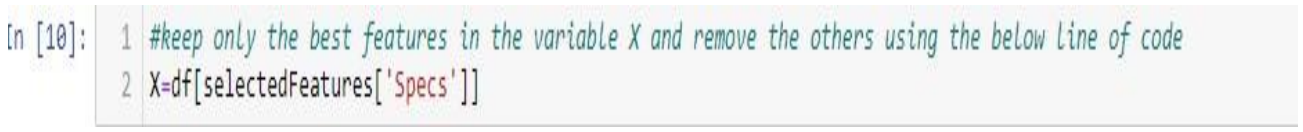


```
In [9]:
#the chi-squared (chi²) statistical test for non-negative features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
selectedFeatures=featureScores.nlargest(10,'Score')
print(selectedFeatures) #print 10 best features
```

```
       Specs       Score
7    thalach  188.320472
9    oldpeak   72.644253
11        ca   66.440765
2         cp   62.598098
8      exang   38.914377
4       chol   23.936394
0        age   23.286624
3   trestbps   14.823925
10     slope    9.804095
1        sex    7.576835
```

*Image 11: Code for feature selection and the selected features and their scores*

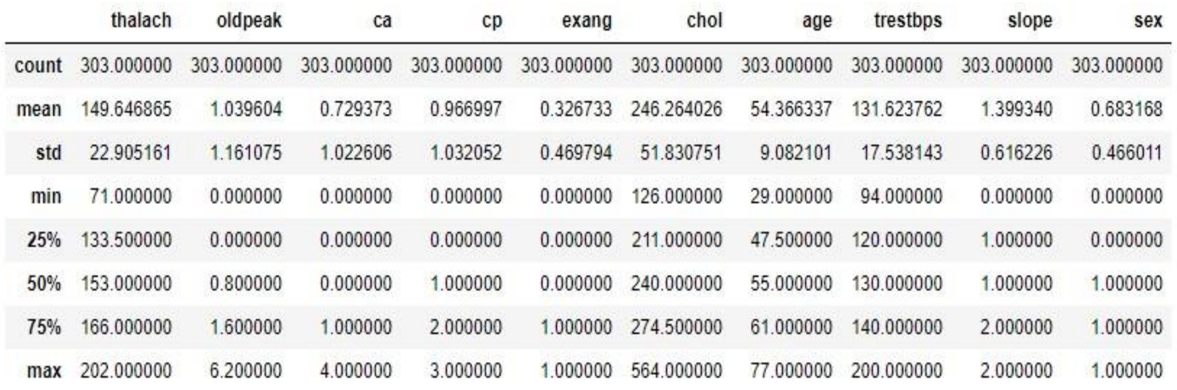
Then I keep only the best features in the variable X and remove the others using the below line of code



```
In [10]:
#keep only the best features in the variable X and remove the others using the below line of code
X=df[selectedFeatures['Specs']]
```

*Image 12: Code for keeping only the best features in the variable X and removing the others*

Now let's look at the selected features using the describe() method:

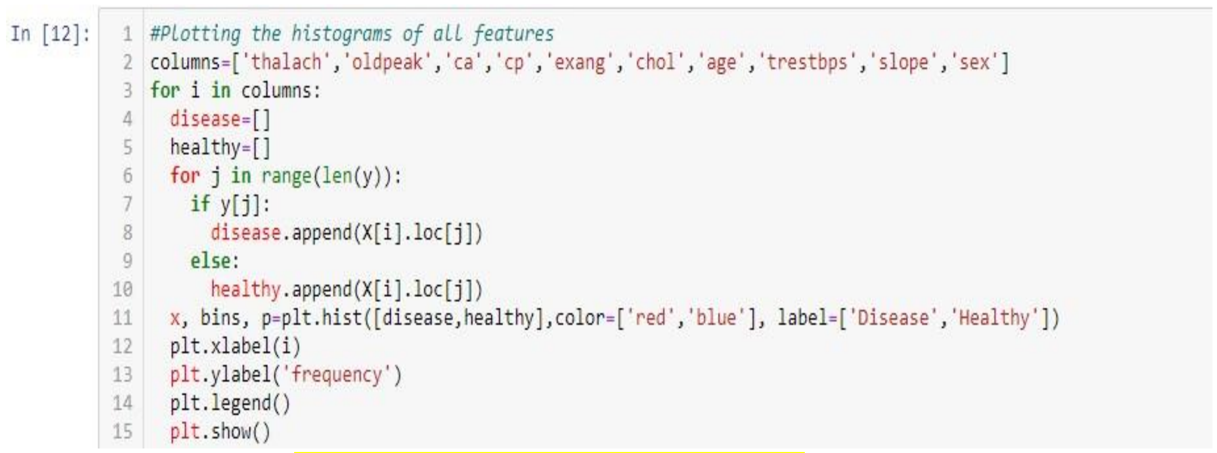


| | thalach | oldpeak | ca | cp | exang | chol | age | trestbps | slope | sex |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| mean | 149.646865 | 1.039604 | 0.729373 | 0.966997 | 0.326733 | 246.264026 | 54.366337 | 131.623762 | 1.399340 | 0.683168 |
| std | 22.905161 | 1.161075 | 1.022606 | 1.032052 | 0.469794 | 51.830751 | 9.082101 | 17.538143 | 0.616226 | 0.466011 |
| min | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 126.000000 | 29.000000 | 94.000000 | 0.000000 | 0.000000 |
| 25% | 133.500000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 211.000000 | 47.500000 | 120.000000 | 1.000000 | 0.000000 |
| 50% | 153.000000 | 0.800000 | 0.000000 | 1.000000 | 0.000000 | 240.000000 | 55.000000 | 130.000000 | 1.000000 | 1.000000 |
| 75% | 166.000000 | 1.600000 | 1.000000 | 2.000000 | 1.000000 | 274.500000 | 61.000000 | 140.000000 | 2.000000 | 1.000000 |
| max | 202.000000 | 6.200000 | 4.000000 | 3.000000 | 1.000000 | 564.000000 | 77.000000 | 200.000000 | 2.000000 | 1.000000 |

*Image 13: Output for X.describe()*

## Visualizing the data

To visualize the data, I split the dataset into two parts, one that has target value as 0 and the other with target value 1. I represented the former as 'healthy' and the latter as 'disease'. I used histograms to visualize the frequencies of the two datasets for different feature values taking a single feature at a time. I plotted the healthy dataset in blue and disease dataset in red.



```
In [12]:
#Plotting the histograms of all features
columns=['thalach','oldpeak','ca','cp','exang','chol','age','trestbps','slope','sex']
for i in columns:
  disease=[]
  healthy=[]
  for j in range(len(y)):
    if y[j]:
      disease.append(X[i].loc[j])
    else:
      healthy.append(X[i].loc[j])
  x, bins, p=plt.hist([disease,healthy],color=['red','blue'], label=['Disease','Healthy'])
  plt.xlabel(i)
  plt.ylabel('frequency')
  plt.legend()
  plt.show()
```

*Image 14: Code for visualizing the data*

Outputs are

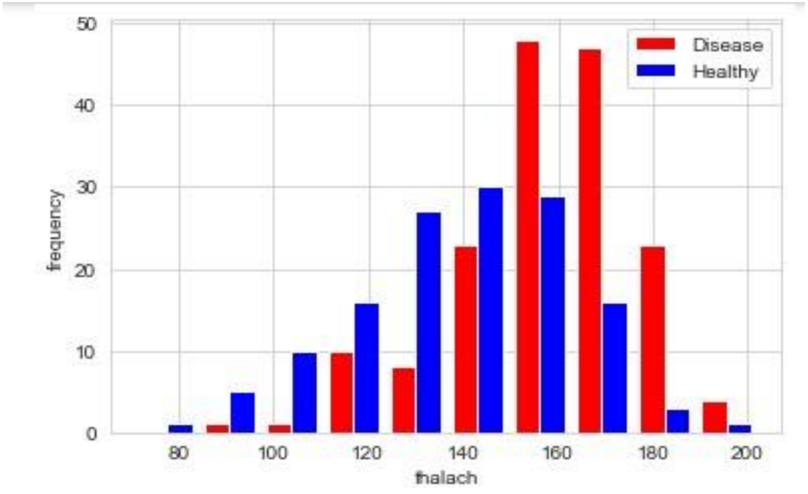


*Image 14 (a): Histogram for the feature - thalach*

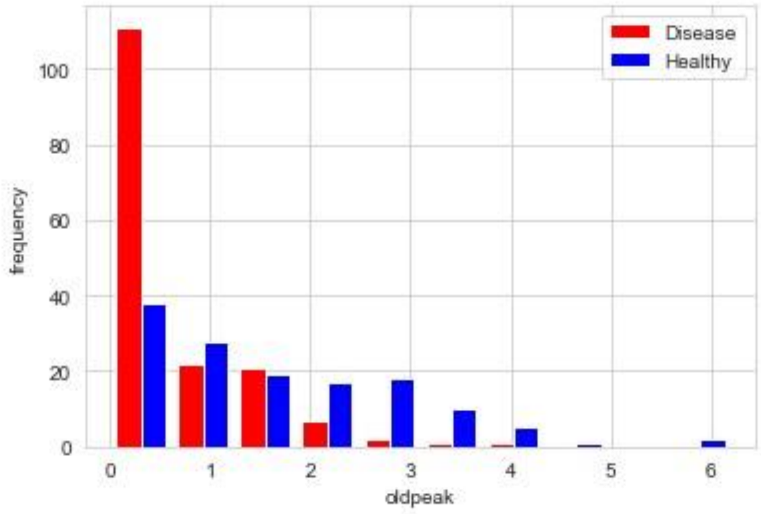


*Image 14 (b): Histogram for the feature - oldpeak*

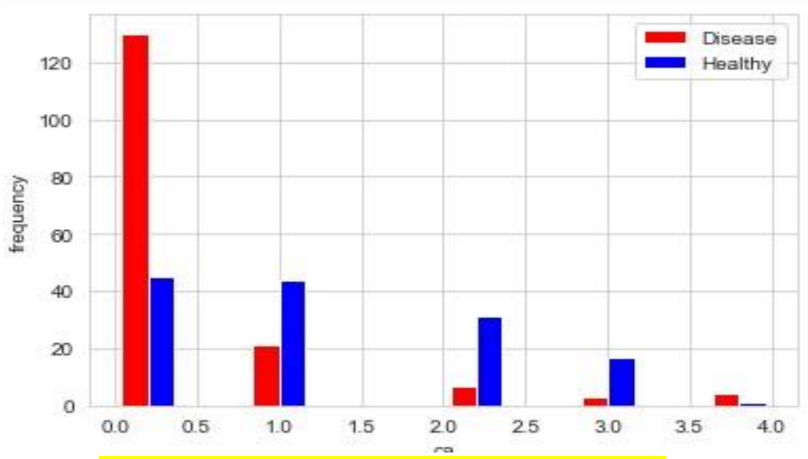


*Image 14 (c): Histogram for the feature - ca*

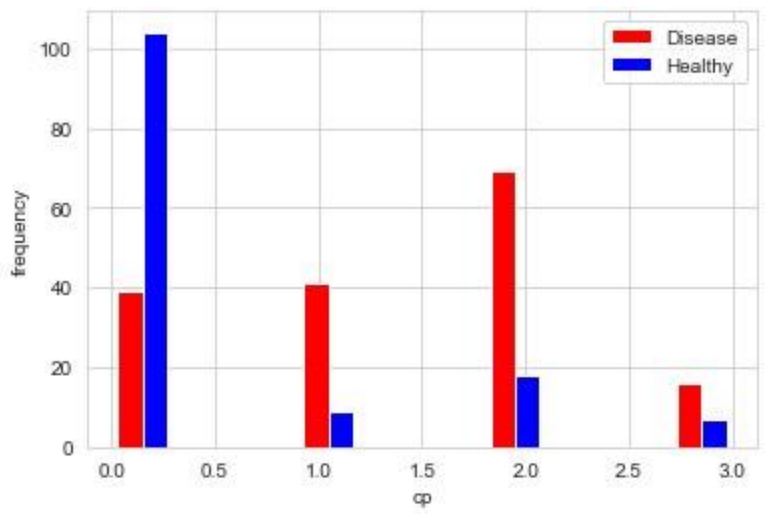


*Image 14 (d): Histogram for the feature - cp*

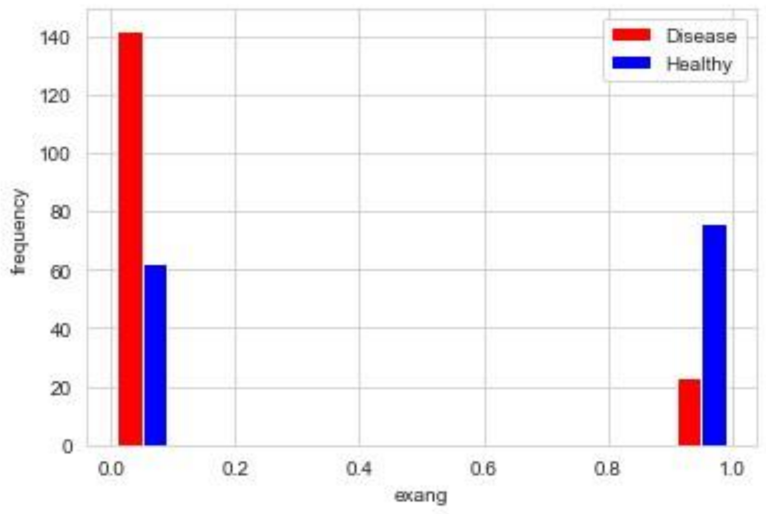


*Image 14 (e): Histogram for the feature – exang*

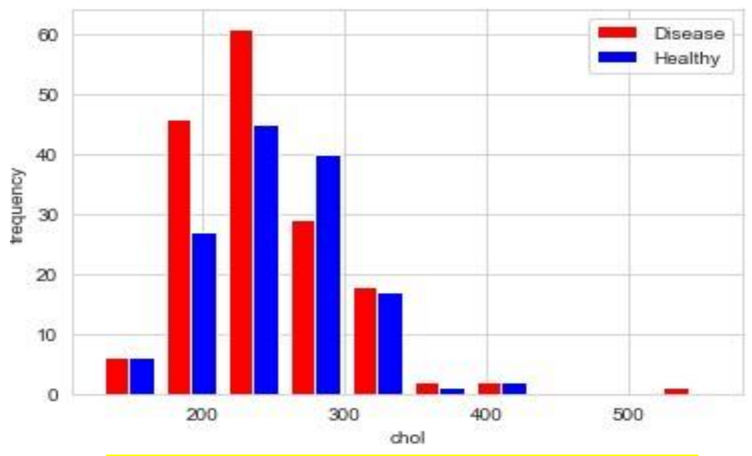


*Image 14 (f): Histogram for the feature – chol*

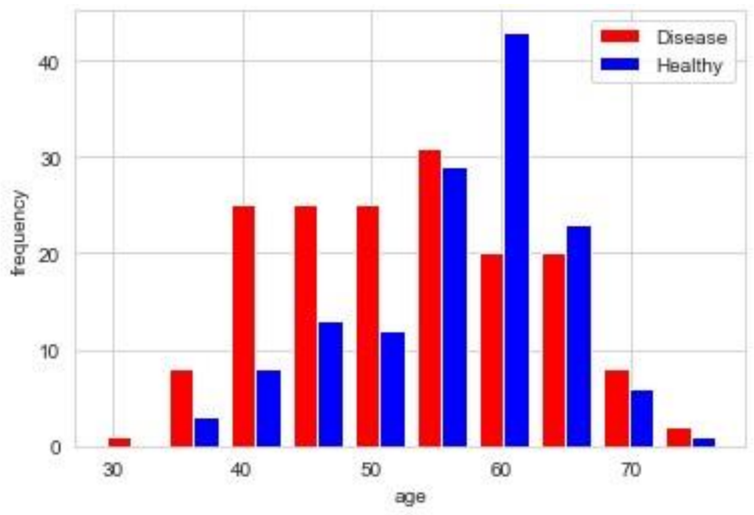


*Image 14 (g): Histogram for the feature – age*

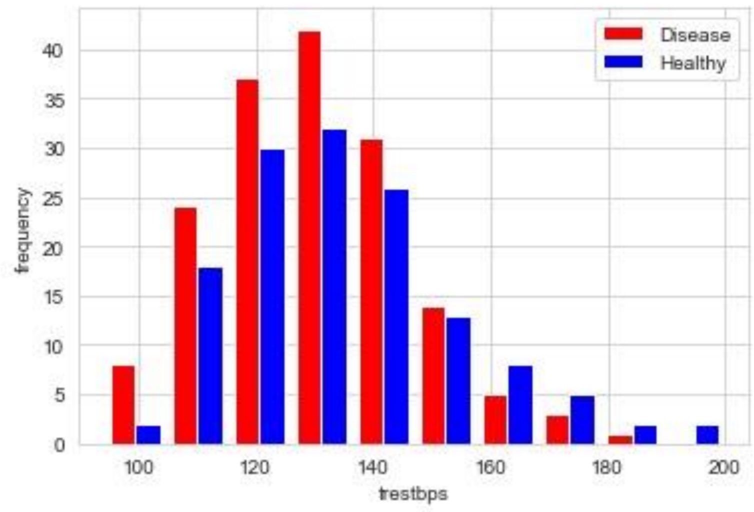


*Image 14 (h): Histogram for the feature – trestbps*

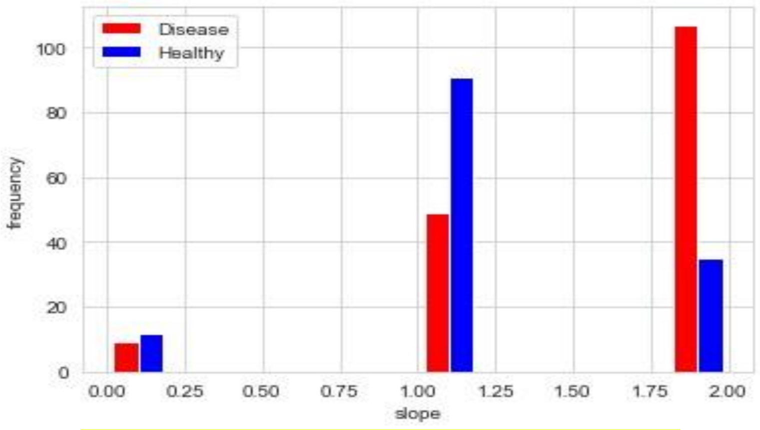


*Image 14 (i): Histogram for the feature - slope*

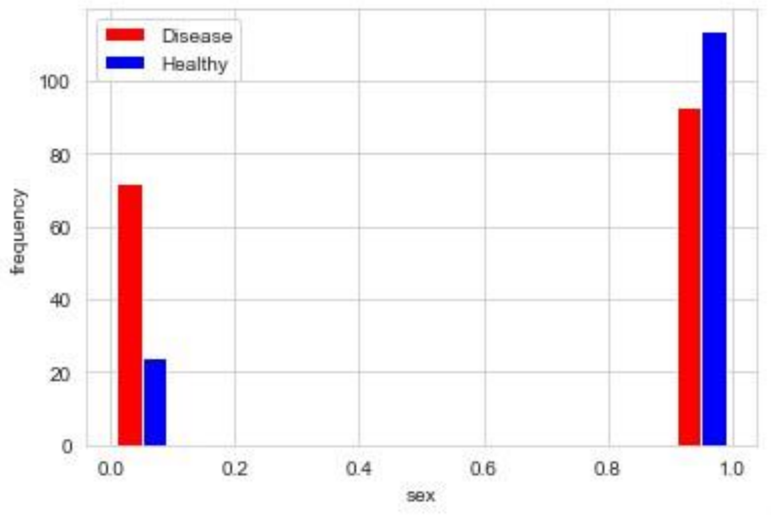
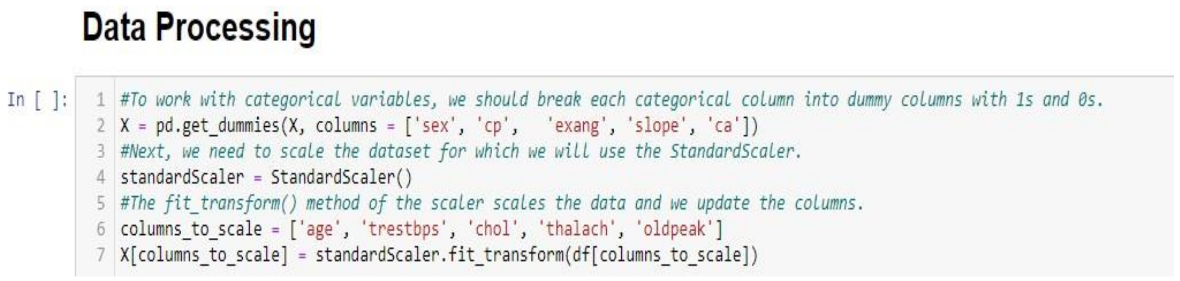


*Image 14 (j): Histogram for the feature - sex*

## Data Processing

To work with categorical variables, we should break each categorical column into dummy columns with 1s and 0s. Let's say we have a column Gender, with values 1 for Male and 0 for Female. It needs to be converted into two columns with the value 1 where the column would be true and 0 where it will be false.

To get this done, we use the get_dummies() method from pandas. Next, we need to scale the dataset for which we will use the StandardScaler. The fit_transform() method of the scaler scales the data and we update the columns.



### Data Processing

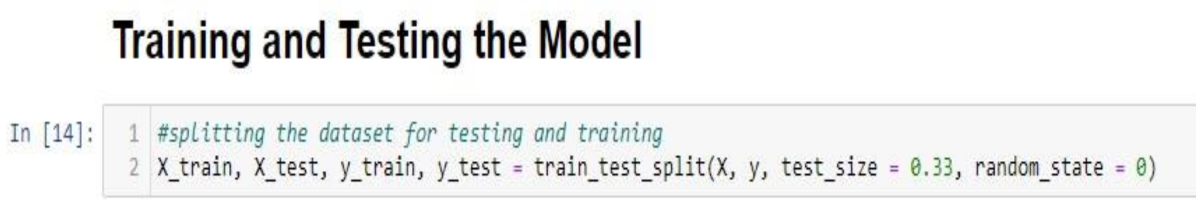
```
In [ ]:
#To work with categorical variables, we should break each categorical column into dummy columns with 1s and 0s.
X = pd.get_dummies(X, columns = ['sex', 'cp',   'exang', 'slope', 'ca'])
#Next, we need to scale the dataset for which we will use the StandardScaler.
standardScaler = StandardScaler()
#The fit_transform() method of the scaler scales the data and we update the columns.
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
X[columns_to_scale] = standardScaler.fit_transform(df[columns_to_scale])
```

*Image 15: Code for data processing*

The dataset is now ready. We can begin with training our models.

## Training and Testing the Model

In this project, I took 4 algorithms and varied their various parameters and compared the final models. I split the dataset into 67% training data and 33% testing data.



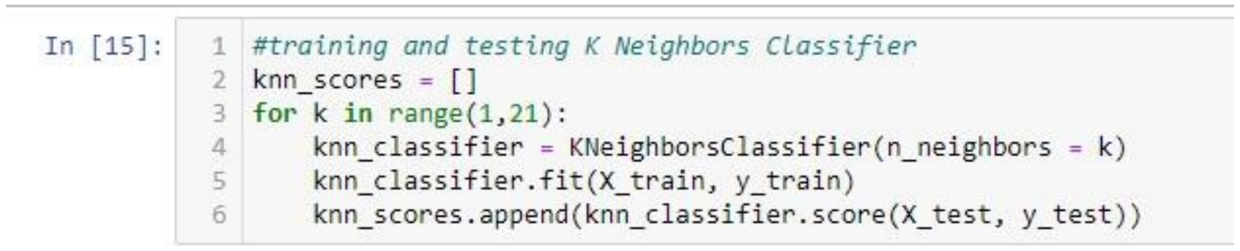
Training and Testing the Model

```
In [14]:
#splitting the dataset for testing and training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 0)
```

*Image 16: Code for splitting the dataset into 67% training data and 33% testing data*

## K Neighbors Classifier

This classifier looks for the classes of K nearest neighbors of a given data point and based on the majority class, it assigns a class to this data point. However, the number of neighbors can be varied. I varied them from 1 to 20 neighbors and calculated the test score in each case

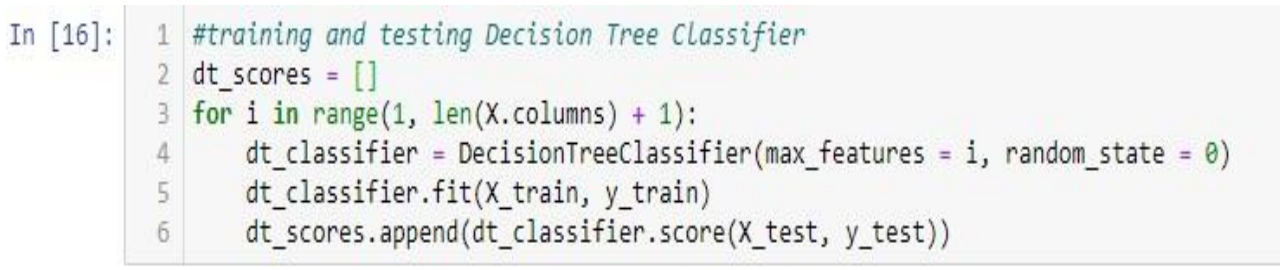


```
In [15]:
#training and testing K Neighbors Classifier
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    knn_classifier.fit(X_train, y_train)
    knn_scores.append(knn_classifier.score(X_test, y_test))
```

*Image 17: Code for training and testing of K Neighbors Classifier*

## Decision Tree Classifier

This classifier creates a decision tree based on which it assigns the class values to each data point. Here, we can vary the maximum number of features to be considered while creating the model. I range features from 1 to 30 (the total features in the dataset after dummy columns were added)

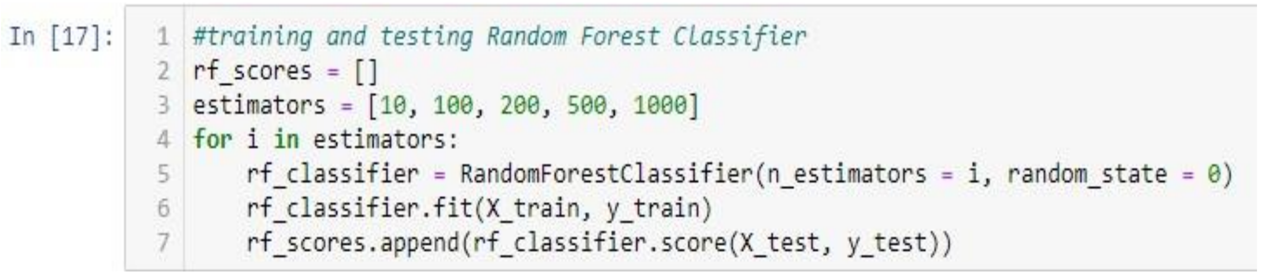


```
In [16]:
#training and testing Decision Tree Classifier
dt_scores = []
for i in range(1, len(X.columns) + 1):
    dt_classifier = DecisionTreeClassifier(max_features = i, random_state = 0)
    dt_classifier.fit(X_train, y_train)
    dt_scores.append(dt_classifier.score(X_test, y_test))
```

*Image 18: Code for training and testing of Decision Tree Classifier*

## Random Forest Classifier

This classifier takes the concept of decision trees to the next level. It creates a forest of trees where each tree is formed by a random selection of features from the total features. Here, we can vary the number of trees that will be used to predict the class. I calculate test scores over 10, 100, 200, 500 and 1000 trees.

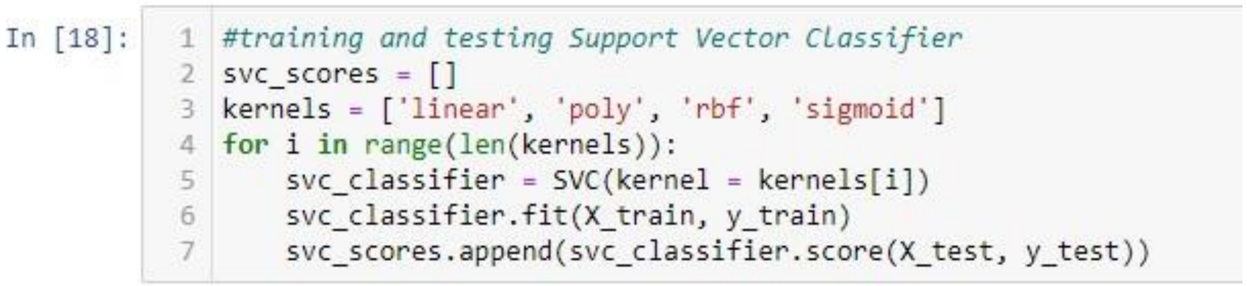


```
In [17]:
#training and testing Random Forest Classifier
rf_scores = []
estimators = [10, 100, 200, 500, 1000]
for i in estimators:
    rf_classifier = RandomForestClassifier(n_estimators = i, random_state = 0)
    rf_classifier.fit(X_train, y_train)
    rf_scores.append(rf_classifier.score(X_test, y_test))
```

*Image 19: Code for training and testing of Random Forest Classifier*

## Support Vector Classifier

This classifier aims at forming a hyperplane that can separate the classes as much as possible by adjusting the distance between the data points and the hyperplane. There are several kernels based on which the hyperplane is decided. I tried four kernels namely, linear, poly, rbf, and sigmoid



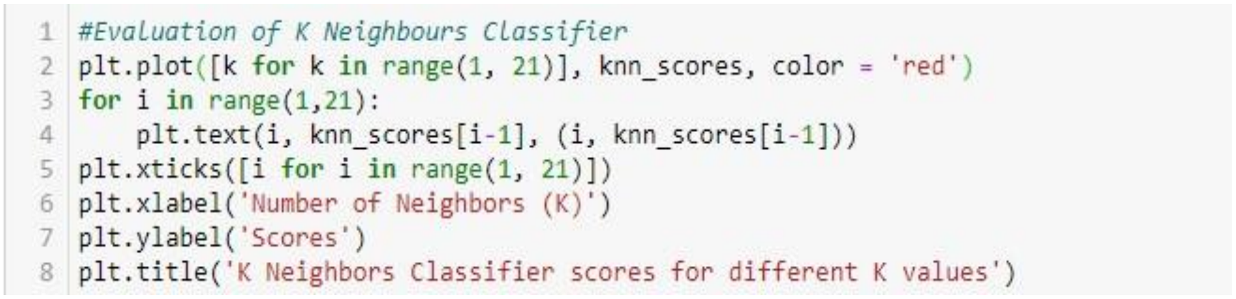
```
In [18]:
#training and testing Support Vector Classifier
svc_scores = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for i in range(len(kernels)):
    svc_classifier = SVC(kernel = kernels[i])
    svc_classifier.fit(X_train, y_train)
    svc_scores.append(svc_classifier.score(X_test, y_test))
```

*Image 21: Code for training and testing of Support Vector Classifier*

## Evaluation of Model

### K Neighbors Classifier

I plot a line graph of the number of neighbors and the test score achieved in each case



```python
#Evaluation of K Neighbours Classifier
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')
```

*Image 22: Code for evaluation of K Neighbors Classifier*

```
Out[19]:  Text(0.5, 1.0, 'K Neighbors Classifier scores for different K values')
```

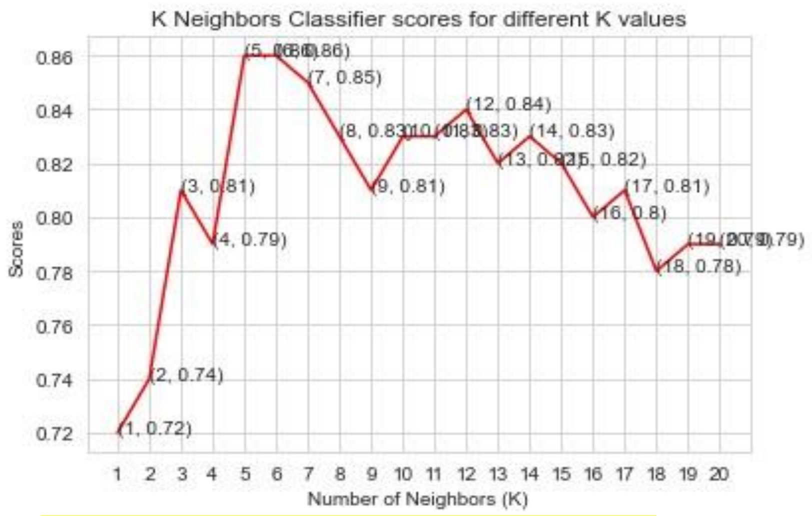
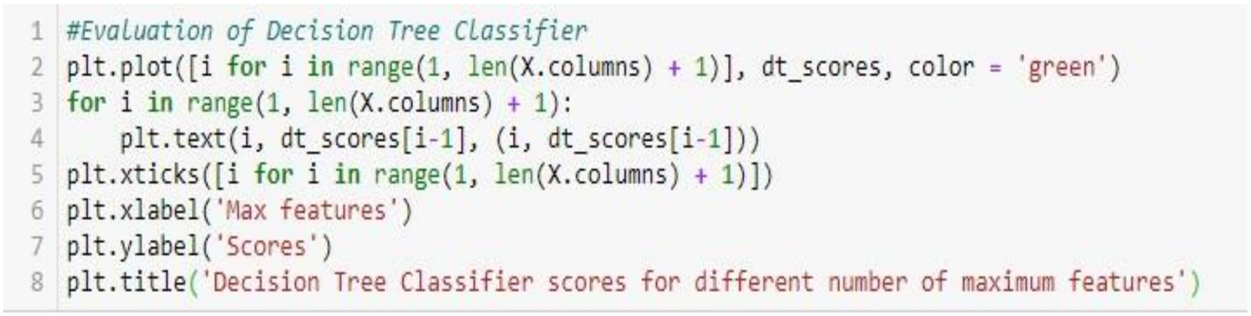


*Image 23: Evaluation of K Neighbors Classifier*

As we can see, we achieved the maximum score of 86% when the number of neighbors was chosen to be either 5 or 6.

### Decision Tree Classifier

Since we have the scores, we can now plot a line graph and see the effect of the number of features on the model scores.



```python
#Evaluation of Decision Tree Classifier
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')
```

*Image 24: Code for evaluation of Decision Tree Classifier*

```
Out[20]: Text(0.5, 1.0, 'Decision Tree Classifier scores for different number of maximum features')
```

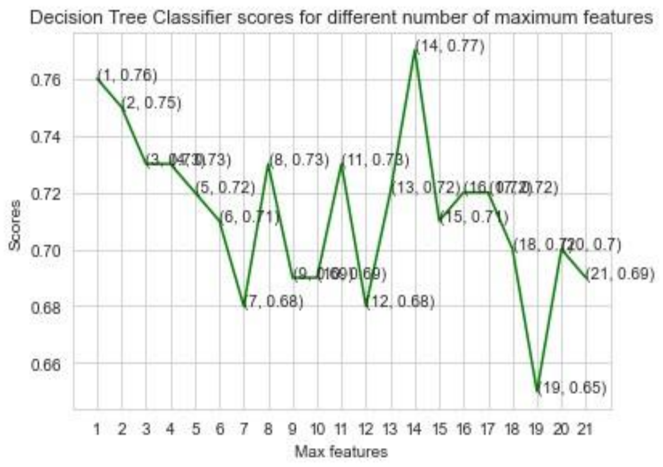
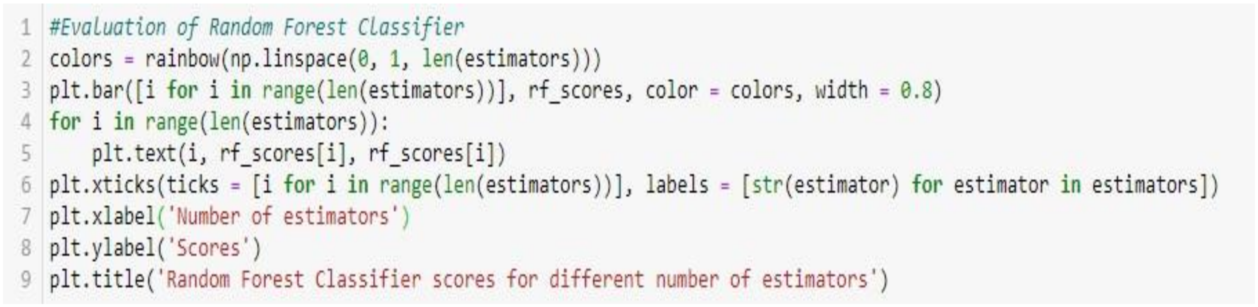


*Image 25: Evaluation of Decision Tree Classifier*

## Random Forest Classifier

Next, I plot saved scores across a bar graph to see which gave the best results. We may notice that I did not directly set the X values as the array [10, 100, 200, 500, 1000]. It will show a continuous plot from 10 to 1000, which would be impossible to decipher. So, to solve this issue, I first used the X values as [1, 2, 3, 4, 5]. Then, I renamed them using xticks.



```python
#Evaluation of Random Forest Classifier
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')
```

*Image 26: Code for evaluation of Random Forest Classifier*

```
Out[23]: Text(0.5, 1.0, 'Random Forest Classifier scores for different number of estimators')
```

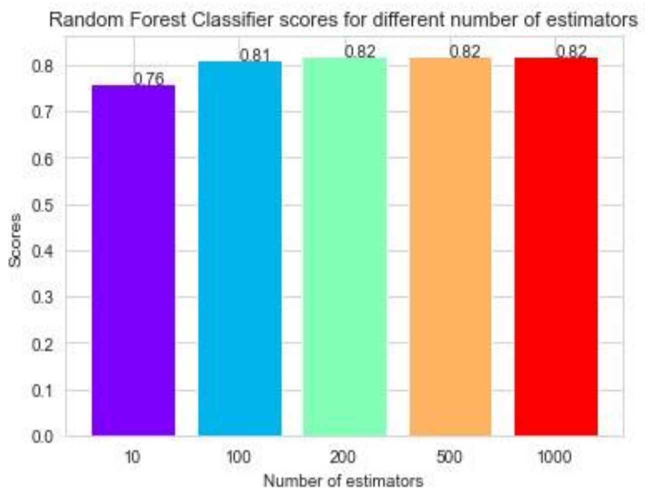
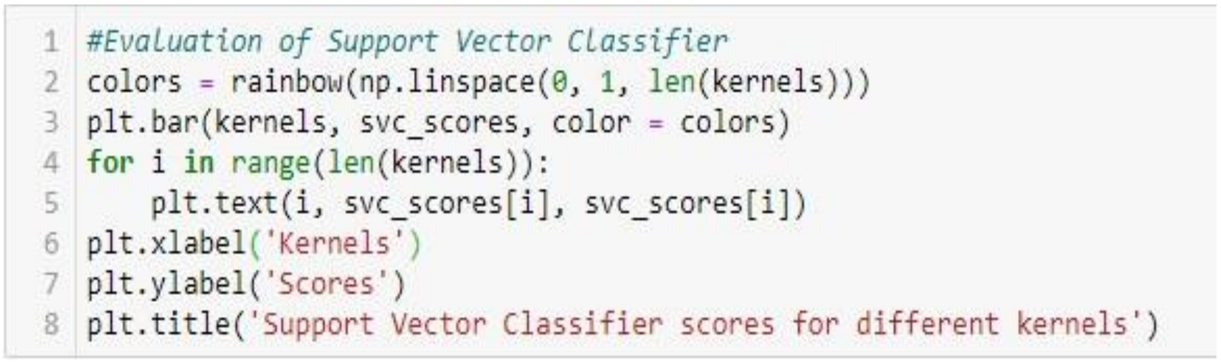


*Image 27: Evaluation of Random Forest Classifier*

Taking a look at the bar graph, we can see that the maximum score of 82% was achieved for 200, 500 and 1000 trees.

## Support Vector Classifier

Since I have the scores for each kernel, I used the rainbow method to select different colors for each bar and plot a bar graph of the scores achieved by each.



```python
#Evaluation of Support Vector Classifier
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')
```

*Image 28: Code for evaluation of Support Vector Classifier*

```
Out[24]: Text(0.5, 1.0, 'Support Vector Classifier scores for different kernels')
```

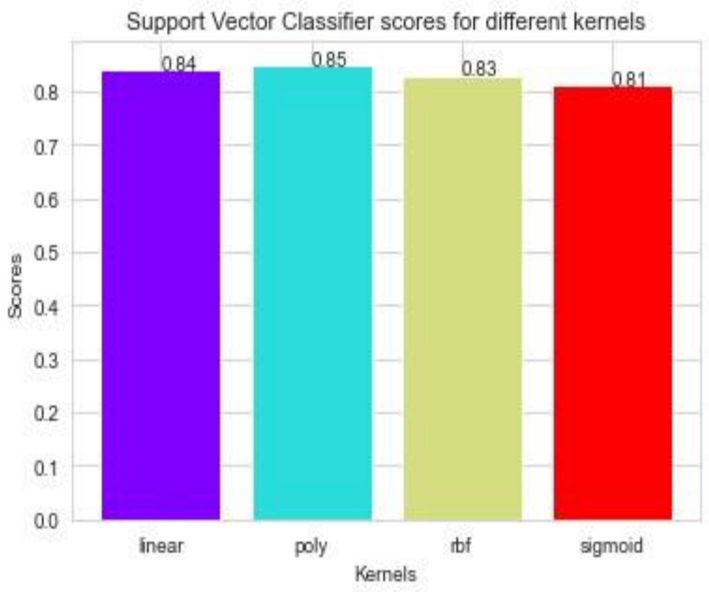


*Image 29: Evaluation of Support Vector Classifier*

As can be seen from the plot above, the poly kernel performed the best for this dataset and achieved a score of 85%

## Conclusion

In this project, I have trained and tested four models for the analysis of the heart disease patient dataset. The models have maximum scores as follows:

1. K Neighbors Classifier: 86%
2. Support Vector Classifier: 85%
3. Decision Tree Classifier: 77%
4. Random Forest Classifier: 82%

K Neighbors Classifier scored the best score of 86% with 5 or 6 neighbors.