## Heart Disease Prediction

### 1. Importing the Dependencies

```
import numpy as np
# to make some numpy arrays
import pandas as pd
# useful for creating dataframe (dataframe are nothing but structured table)
from sklearn.model_selection import train_test_split
# to split the original dataset into training and testing data
from sklearn.linear_model import LogisticRegression
# Importing logistic regression model
from sklearn.metrics import accuracy_score
# to evaluate our accuracy
```

---

### 2. Data Collection and Processing

```
#loading csv data to pandas dataframe
heart_data = pd.read_csv('/content/data.csv')

#printing first 5 rows of df
heart_data.head()

#printing last 5 rows of df
heart_data.tail()

# number of rows and columns
heart_data.shape

#getting some info
heart_data.info()

#statistical Measures  about the data
heart_data.describe()
```

---

### 3.checking for missing values

```
# there is no missing or null values if present then we can use like imputation

heart_data.isnull().sum()
```

---

**4.checking the distribution of Target variables**

```
heart_data['target'].value_counts()

# .value_counts tells us the total counts present for the resp column
#we need to have almost equal distribution in both classes
#if the distribution is very unequal then we have to do some processing
```

---

**5.Splitting the Features and Target**

```
#features can be used to predict the target
#all the features are stored in variable X and targets are stored in variable Y

X = heart_data.drop(columns = 'target', axis = 1)
Y = heart_data['target']

#Whenever dropping any column we have to mention the axis. axis=1 for column and axis=0 for rows
#storing the dropped target into Y
```

---

**6.Splitting the data into training data and test data**

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify = Y, random_state = 2)

#X_train = the features of all the training data
#X_test = the features of all the test data
#Y_train = the target of all these features present in the X_train
#Y_test = the target of all these features present in the X_test
#test_size = how much percent of data you want as the test data , here 0.2 refers 20%
#stratify = similar proportion of 0 and 1 in train and test data as it was in the original dataset (even distribution throughout training and testing data)
#random_state = controls the shuffling of data

print (X.shape, X_train.shape, X_test.shape)
#as we can see 80% of data has gone to X_train and 20% to X_test
```

---

**7.Model Training**

*#here we are using Logistic regression model bcoz it is very useful for binary classification*

```
model = LogisticRegression()
```

*#training the model using training data -> (.fit function is used)*
*# this will find the relationship between features of X_train and target of Y_train*
*#now we can use this trained model to predict new values*

```
model.fit(X_train, Y_train)
```

---

**8.Model Evaluation and Accuracy Score**

*#accuracy on training data using prediction*

*#in this we will be giving the X_train(features) alone and ask the model to predict the target*
*#.predict function predicts the target value*
*#storing the target value in variable X_train_prediction*

*#accuracy score function helps to find the accuracy*
*#we will be storing the accuracy value in this variable training_data_accuracy*

```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print ("Accuracy on training data:", training_data_accuracy)
```

---

*#accuracy on testing data using prediction*

```
X_test_prediction = model.predict(X_test)
testing_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print ("Accuracy on testing data:", testing_data_accuracy)
```

*#the accuracy score on testing and training data should be almost the same. if the differe is way too large then it means our model is overfitted*

---

**9.Building a Predictive System**

```
# change the input data to a numpy array

input_data = (37,1,2,130,250,0,1,187,0,3.5,0,0,2)

#asarray converts the tuple into numpy array

input_data_as_numpy_array = np.asarray(input_data)

#we need to reshape the array it is imp to tell our model that we want to predict for only one instance or else the model will consider the initial shape of dataset

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshaped)

print(prediction)

if (prediction[0] == 0):
  print ("healthy")
else:
  print ("has heart disease")
```