# HEART DISEASE PREDICTION UISNG MACHINE LEARNING WITH PYTHON

## Overview

Machine Learning is one of the most widely used concepts around the world. It will be essential in the healthcare sectors which will be useful for doctors to fasten the diagnosis. In this article, we will be dealing with the **Heart disease dataset** and will analyze, predict the result whether the patient has heart disease or normal, i.e. Heart disease prediction using Machine Learning. This prediction will make it faster and more efficient in healthcare sectors which will be a time-consuming process.

***Key Takeaways***

- This process involves data cleaning, data statistics, getting insights from the dataset.
- This involves four machine learning algorithms which will result in performance metrics of the model.

## Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import rcParams
import seaborn as sns
```

This workflow demonstrates a basic machine learning pipeline using logistic regression for binary classification. These lines of code set up the environment for creating and customizing plots in a Jupyter notebook, using both matplotlib and seaborn libraries.

## Reading the Data from CSV file

In [3]:
```
# loading the csv data

heart_data = pd.read_csv('heart_disease_data.csv')
```

We are loading the heart dataset , taken from the Kaggle.

Link of Dataset:- https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

# Data collection and processing

In [4]:
```
#first 5 rows
heart_data.head()
```

Out[4]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | 1 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | 1 |

In [5]:
```
#last 5 rows

heart_data.tail()
```

Out[5]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **298** | 57 | 0 | 0 | 140 | 241 | 0 | 1 | 123 | 1 | 0.2 | 1 | 0 | 3 | 0 |
| **299** | 45 | 1 | 3 | 110 | 264 | 0 | 1 | 132 | 0 | 1.2 | 1 | 0 | 3 | 0 |
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 | 0 |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 | 0 |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 | 0 |

In [6]:
```
#data information

heart_data.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
```

In [7]:
```
#no. of rows and columns in dataset

heart_data.shape
```

Out[7]:
```
(303, 14)
```

In [8]:
```
# checking for missing values

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

Out[8]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [9]:
```
# statistical measures

heart_data.describe()
```

Out[9]:

| | 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 | 303.0 |
| 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 | 2.31 |
| 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 | 0.61 |
| 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 | 0.0 |
| 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 | 2.0 |
| 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 | 2.0 |
| 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 | 3.0 |
| 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 | 3.0 |

In [10]:
```
#checking the distribution for target varibale
# 1 for heart diseases
# 0 for not heart diseases
```

In [11]:
```
heart_data['target'].value_counts()
```

Out[11]:
```
1    165
0    138
Name: target, dtype: int64
```

In [12]:
```
#splitting the features and target
```

In [13]:
```
X = heart_data.drop(columns = 'target',axis = 1)
Y = heart_data['target']
```

In [14]:
```
print(X)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal
0        0   0     1
1        0   0     2
2        2   0     2
3        2   0     2
4        2   0     2
..     ...  ..   ...
298      1   0     3
299      1   0     3
300      1   2     3
301      1   1     3
302      1   1     2

[303 rows x 13 columns]
```

In [15]:
```
print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

In [16]:
```
#Splitting the data into training data and test data
```

In [17]:
```
X_train , X_test, Y_train , Y_test = train_test_split(X, Y, test_size=0.2, stratify = Y, random_state=2)
```

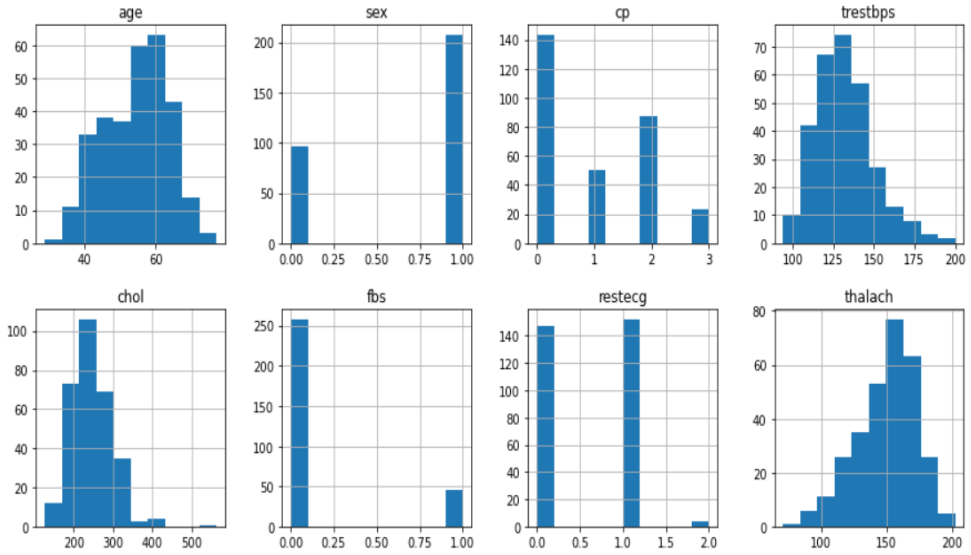
In [18]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(303, 13) (242, 13) (61, 13)
```

# Data Visualization

In [19]:
```
# Plotting histogram for the entire dataset
fig = plt.figure(figsize = (15,15))
ax = fig.gca()
g = heart_data.hist(ax=ax)
```

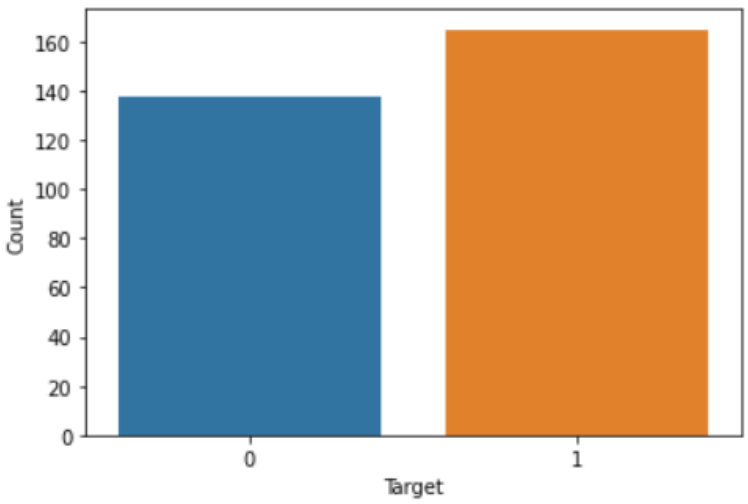
```
C:\Users\USER\AppData\Local\Temp\ipykernel_6452\2492204175.py:4: UserWarning: To output multiple subplots, the figure containin
g the passed axes is being cleared.
  g = heart_data.hist(ax=ax)
```




Together, this code will generate a grid of histograms for each numerical feature in the heart_data DataFrame, all within a single large figure of size 15x15 inches.

In [20]:
```
# Visualization to check if the dataset is balanced or not
g = sns.countplot(x='target', data= heart_data)
plt.xlabel('Target')
plt.ylabel('Count')
```

Out[20]: Text(0, 0.5, 'Count')



Together, this code produces a bar plot that shows the number of occurrences of each unique value in the target column. This visualization helps to check if the dataset is balanced, i.e., if the different classes in the target variable are represented approximately equally. If the bars are of similar height, the dataset is balanced. If there is a significant difference in the height of the bars, the dataset is imbalanced.

```
In [2]:
#Feature Selection
#To get correlation of each feature in the data set
```

```
In [21]:
corrmat = heart_data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(16,16))
#plot heat map
g=sns.heatmap(heart_data[top_corr_features].corr(),annot=True)
plt.show()
```


This code generates a heatmap that visualizes the correlation matrix of the heart_data DataFrame, with annotations showing the correlation values. This helps in identifying which features are strongly correlated with each other, which can be useful for feature selection and understanding relationships in the data.

# Model Training

In [22]:
```
# Model training
```

In [23]:
```
#logistic regression model
```

In [22]:
```
model = LogisticRegression()
```

In [23]:
```
#training thelogistic Regression modelwith training data

model.fit(X_train,Y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[23]:
```
LogisticRegression()
```

In [24]:
```
#model evaluation
```

In [25]:
```
#accuracy score
```

In [26]:
```
#accuracy on training data

X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

```

In [27]:
```
print(training_data_accuracy)
```

```
0.8512396694214877
```

In [28]:
```
#accuracy on test data

X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

```

In [29]:
```
print(test_data_accuracy)
```

```
0.819672131147541
```

In [30]:
```
#Building a Predictive System
```

In [31]:
```
input_data=(41,0,1,130,204,0,0,172,0,1.4,2,0,2)

#change the input data to a numpy array
input_data_as_numpyarray = np.asarray(input_data)

```

In [32]:
```
#to reshape the numpy arrayforpredicting for only one instances

input_data_reshaped = input_data_as_numpyarray.reshape(1,-1)

prediction = model.predict(input_data_reshaped)

print(prediction)

if(prediction[0] == 0):
    print("The Person does not have Heat Disease")
else:
    print("The Person have Heart Disease")
```

```
[1]
The Person have Heart Disease
```

In [33]:

```
input_data=(70,1,0,130,322,0,0,109,0,2.4,1,3,2)

#change the input data to a numpy array
input_data_as_numpyarray = np.asarray(input_data)

input_data_reshaped = input_data_as_numpyarray.reshape(1,-1)

prediction = model.predict(input_data_reshaped)

print(prediction)

if(prediction[0] == 0):
    print("The Person does not have Heat Disease")
else:
    print("The Person have Heart Disease")
```

```
[0]
The Person does not have Heat Disease
```

# Conclusion

Finally, we can conclude that real-time predictors will be essential in the healthcare sector nowadays. From this project, we will be able to predict real-time heart disease using the patient's data from the model using the Logistic Regression Algorithm, thereby making accurate heart disease prediction using machine learning.