Heart Disease Prediction System Using Machine Learning

Dissanayaka D.M.A.S.

Sri Lanka Institute of Information Technology

Table of Content

Contents

Table of Figures

Heart Disease Prediction System

1 Introduction

A range of conditions that affect your heart can consider as heart diseases. Heart blood vessel diseases, heart rhythm problems (arrhythmias), heart defects from birth (congenital heart defects) are examples of heart diseases.
Heart Disease Prediction System is designed to predict the status of the patients whether the patient could be a heart disease patient or not. This system is created using a Machine Learning algorithm and Flask framework to input user data and predicts the results from those inputs.

2 Work Environment

To build this Machine Learning Model, I used Jupyter Lab, and for the deployment, I used the Flask framework.

2.1 Building the work environment.

First, we need the JupyterLab for building the Machine Learning model. JupyterLab can be installed using **pip**, **pipenv**, **conda,** and **docker**. In this case, I'll be using **pip** to install the required software.

2.1.1 Python Installation

Before all installation, we need to install Python. Python can be installed by using the following commands.

Windows – Download the python installer and run the installer

Linux (Ubuntu) - $ apt-get update

$ apt-get install python<version>

macOS – Install Homebrew as the first step.

The second step is to install Python using the following command.

$ brew install python3

2.1.2 pip installation

After the installation of Python, we can install pip.

Windows - Download get-pip.py to a folder on your computer. Open the command prompt on that folder and run **python get-pip.py** command.

Linux (Ubuntu) – $apt install python3-pip

macOS – sudo easy_install pip

2.1.3 JupyterLab installation

After the installation of Python and pip, we can continue to install the JupyterLab. Since all the prerequisites are installed, we can continue to install JupyterLab by running the following command. I'll be using pip to install the JupyterLab.

pip install JupyterLab

To run the JupyterLab, simply enter **jupyter lab** on the terminal or command prompt.

3 Dataset

To rain and test the machine learning model that uses in this system, I downloaded a dataset of heart disease patients from the Kaggle website. The data set has 303 records with 14 columns.

Column Descriptions:

- **Age** – Age of the patient
- **Sex** – Gender of the patient
- **cp** – Chest Pain level
- **trestbps** – Resting blood pressure (in mm Hg on admission to the hospital)
- **chol** – Cholesterol level in mg/dl
- **fbs** – Fasting blood sugar Level
- **restecg** – E.S.G. status
- **thalach** – Maximum heart rate archived
- **exang** – Chest pain when exercising
- **oldpeak** – S.T. depression induced by exercise relative to rest
- **slope** – The slope of the peak exercise S.T. segment
- **ca** – Number of major vessels (0-3) colored by fluoroscopy
- **thal** – Thalassemia
- **target** – Whether the patient has heart disease or not.

In some datasets, we can select the columns that we are using to train the model by looking at the values of each column and what they represent. But in this case, all the columns are valuable to the prediction.

4 Jupyter Notebook

When opening the JupyterLab through Linux terminal or windows command prompt, we can get a web application with various selections. There are three main selections, such as Notebook, Console, and Other. The best way to create the model and train it, the Notebook.

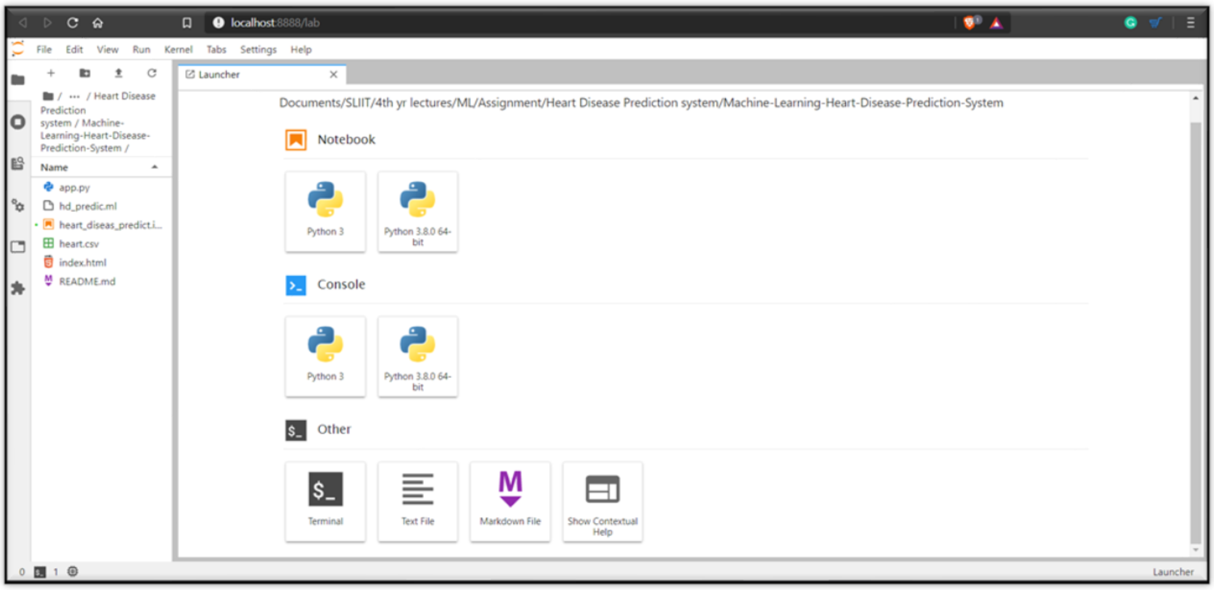


Figure 4:1: JupyterLab Interface

In the Notebooks section, we can select a notebook according to our python version.

Figure 4:2: Jupyter Notebook Interface

Jupyter Notebook is what we are using to import, analyze the dataset, and according to the imported dataset, we create, test, and train the machine learning model in Jupyter Notebook.

In Jupyter Notebook, there are specific tools to use when using the Notebook.

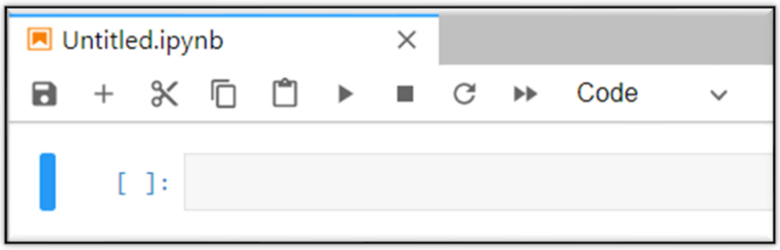


Figure 4:3: Jupyter Notebook controls

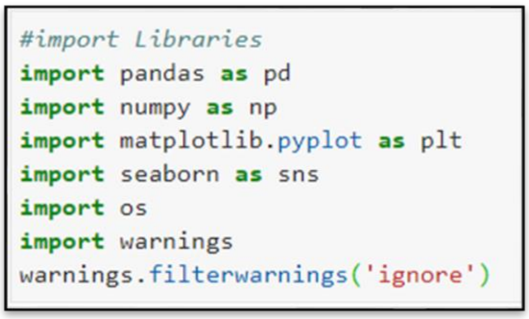
Jupyter Notebook controls are used to save the file, insert a new cell, cut the selected cell, copy selected cells, paste cells on the clipboard, run the selected cells and advance, interrupt the kernel, restart the kernel and, restart the kernel and run the whole Notebook.

5 Creating the Machine Learning Model

The first thing we should do is import the required libraries. There are several libraries we required to create this model. We just cannot import these libraries from just a code. These libraries should install in the P.C. before import them to the jupyter Notebook.

5.1 Importing libraries

- **Pandas** - Pandas is for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- **NumPy -** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
- **Matplotlib** - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose G.U.I. toolkits like Tkinter, wxPython, Qt, or GTK+.
- **Seaborn -** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
- **os** – os provides a way of using operating system dependent functionality

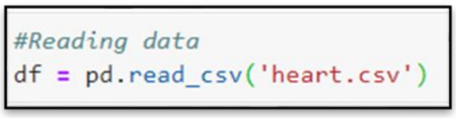


```
#import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
warnings.filterwarnings('ignore')
```

Figure 5:1: Importing Libraries

5.2 Reading the Dataset

Reading the dataset is essential to build the machine learning model. Our dataset is saved on the folder as a CSV file. In jupyter, it possible to read the dataset as the following image.



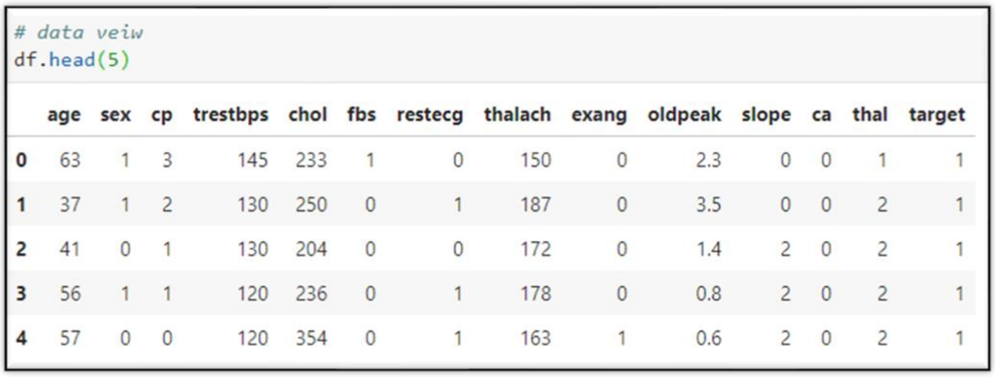
```
#Reading data
df = pd.read_csv('heart.csv')
```

Figure 5:2: Reading the dataset

5.3 Exploring the data

We can explore the data using several ways. First, we can view the imported dataset. Looking for how many columns and rows we have to deal with, what are the column names are, and described data, which means the mean, median, maximum numbers, minimum numbers, and averages of the datasets.

Viewing the dataset table:

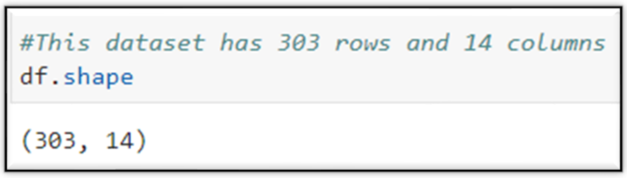


```
# data veiw
df.head(5)
```

	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

Figure 5:3: View the dataset table

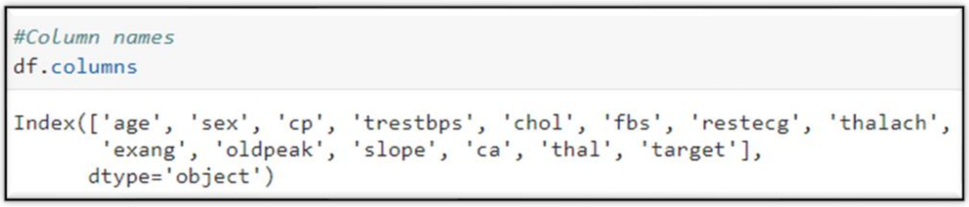
Exploring the columns and rows:



```
#This dataset has 303 rows and 14 columns
df.shape

(303, 14)
```

Figure 5:4: Viewing the column numbers and raws

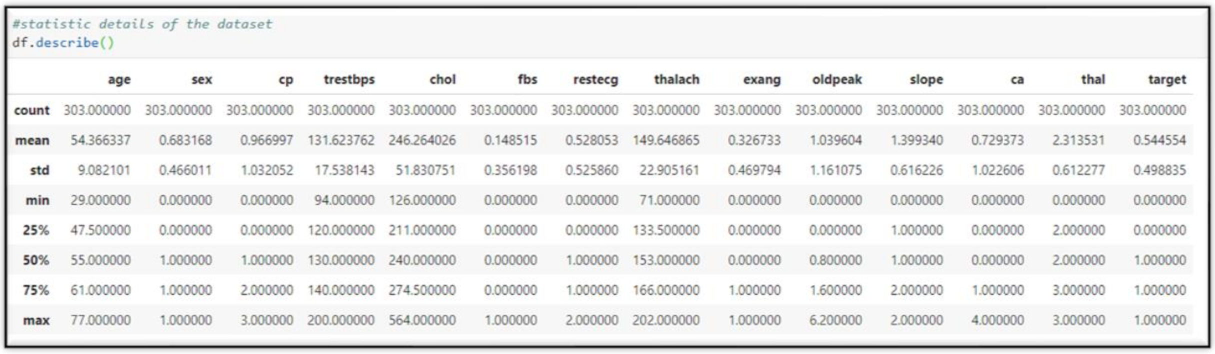
As in Figure 5:4: Viewing the column numbers and raws using the code in the above picture, we can see how columns and rows in the dataset. As in the below image, we can see the column names.



```
#Column names
df.columns

Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

Figure 5:5: Column Names

To get statistics such as mean median, averages of the dataset, we can use the below image's code.



```
#statistic details of the dataset
df.describe()
```

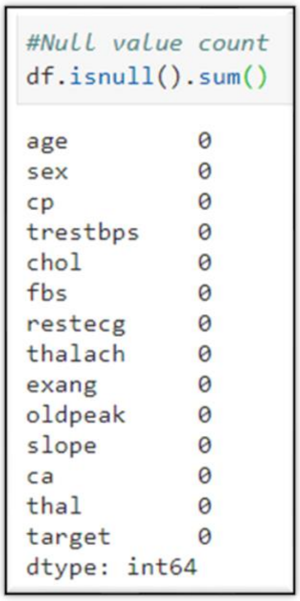
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
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	0.612277	0.498835
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	0.000000	0.000000
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	2.000000	0.000000
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	2.000000	1.000000
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	3.000000	1.000000
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	3.000000	1.000000

Figure 5:6: Statistics of the dataset

5.4 Data cleaning

In a dataset, data cleaning is a critical part when it comes to machine learning. When we are testing and training the machine learning model, we use the dataset to split the data. In the dataset, there might be duplicated data. We need to clean those data before using in a machine learning model.

Using the code in the below image, we can see the duplicated data in the dataset.



```
#Null value count
df.isnull().sum()

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
```

Figure 5:7: Data Duplication Check

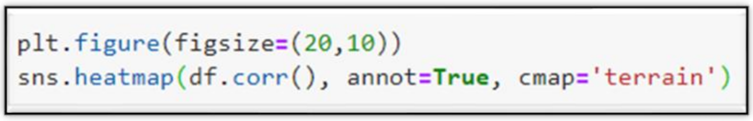
According to Figure 5:7: Data Duplication Check, there are no duplicated data in the dataset. So, there is no need for cleaning data in the dataset.

5.5 Data correlation among the attributes

There are 14 columns in this dataset, and we can compare those columns to get a better idea of the dataset and how this heart disease spread through the patients. Jupyter Notebook has inbuilt libraries to create and view various charts, which makes it easy to do.

5.5.1 Heat Map:

Using the following codes in Figure 5:8: Codes for the heatmap, we can create a heatmap to view attributes compared to each other.



```python
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True, cmap='terrain')
```

Figure 5:8: Codes for the heatmap

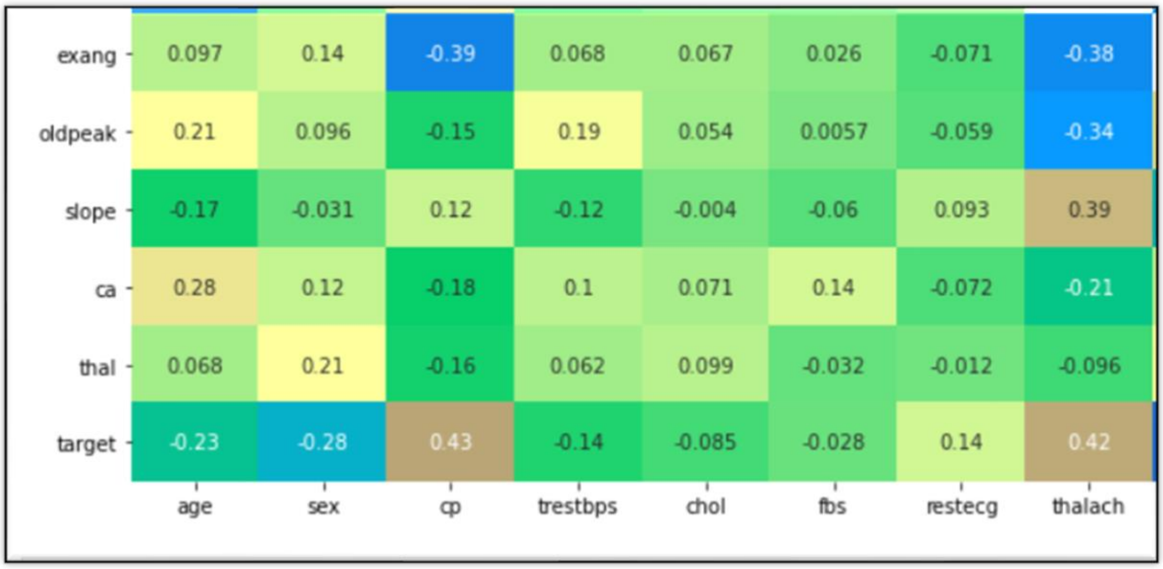
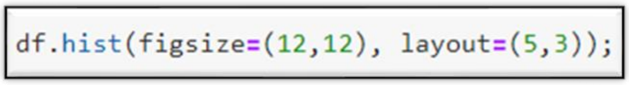


Figure 5:9: Part of the heatmap

As you can see in Figure 5:9: Part of the heatmap, we can observe a positive correlation between target and cp, thalach, slope, and also a negative correlation between target and sex, exang, ca, thal, oldpeak.

5.5.2 Histogram Chart:

As like the heatmap in, Jupyter Notebook is capable of making histogram charts by using various attributes. To make histogram charts, the code as follows.



```python
df.hist(figsize=(12,12), layout=(5,3));
```

Figure 5:10: Code to build histograms

Using the above command in Figure 5:10: Code to build histograms, we can create histograms for each column in the dataset as following Figure 5:11: Histograms for some of the columns

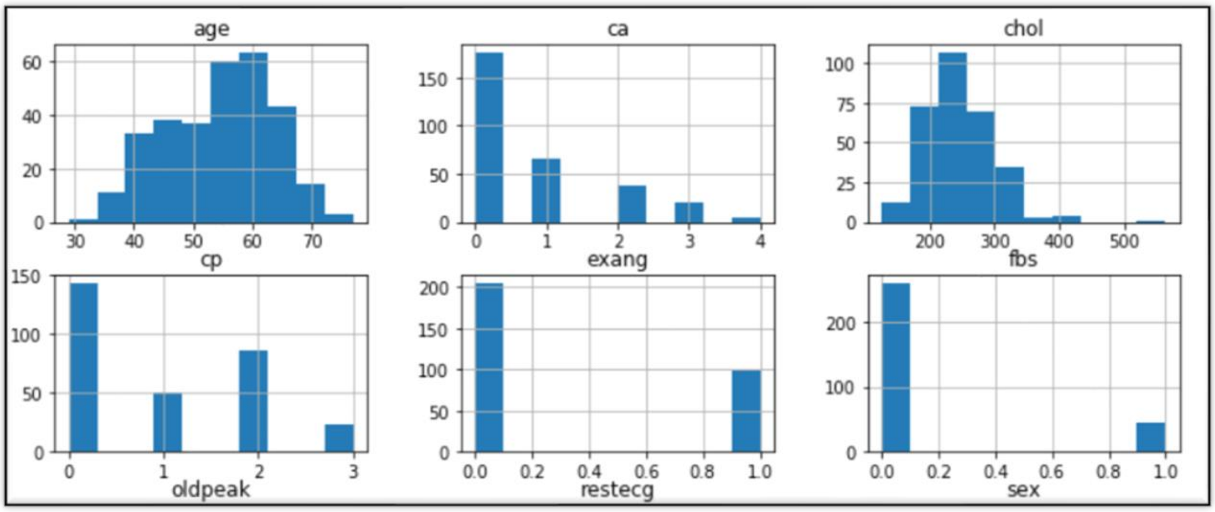
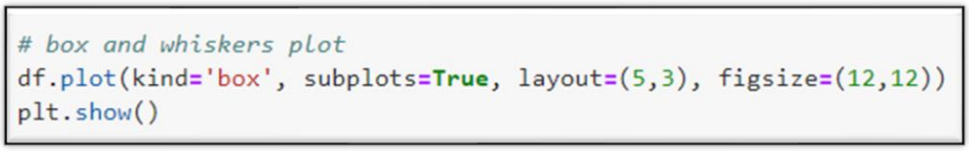


Figure 5:11: Histograms for some of the columns

5.5.3 Box and Whisker Plot:

Jupyter Notebook is also capable of creating box and whisker plots using the attributes of the dataset. It can be done by using the code in following Figure 5:12: Code for build Box and Whisker Plots.



```
# box and whiskers plot
df.plot(kind='box', subplots=True, layout=(5,3), figsize=(12,12))
plt.show()
```

Figure 5:12: Code for build Box and Whisker Plots

The result of the above code as follows.

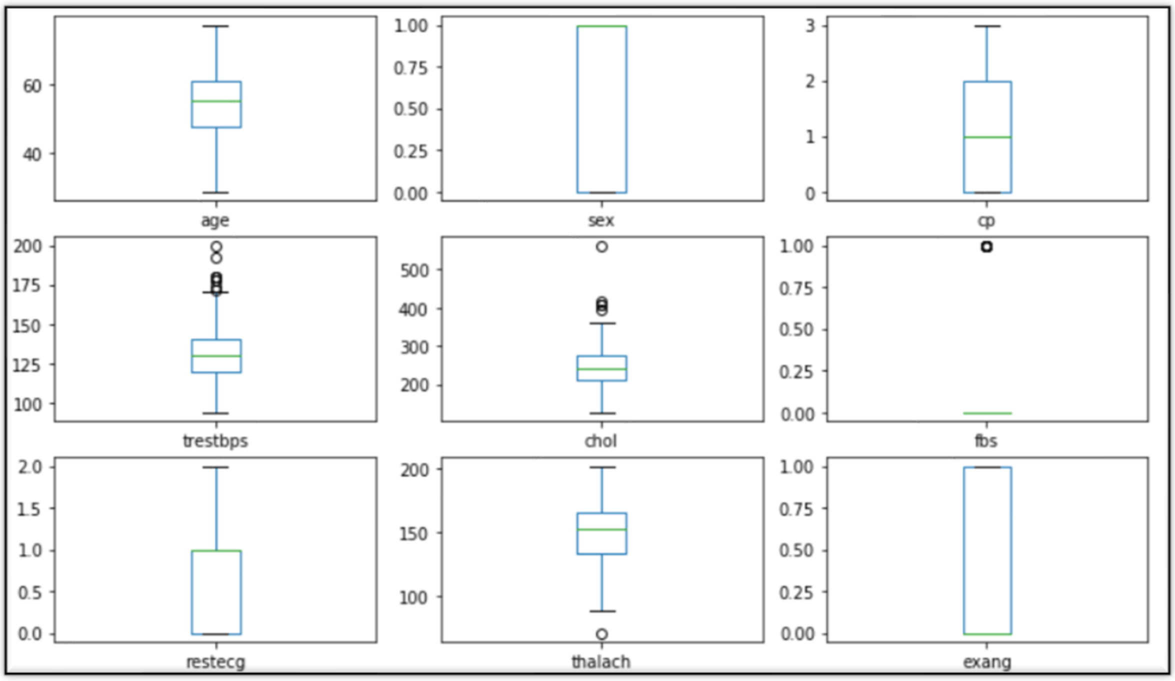
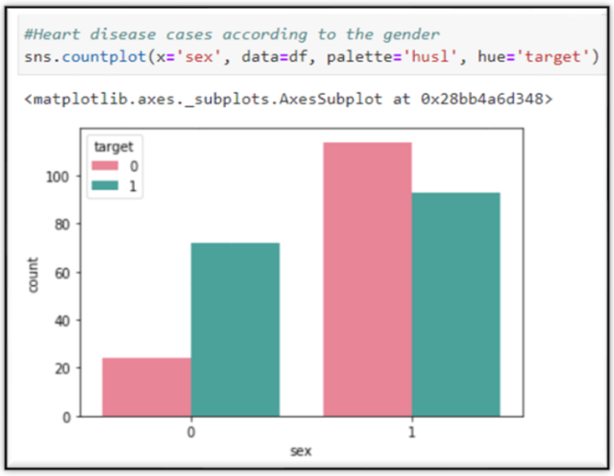


Figure 5:13: Box and Whisker Plots

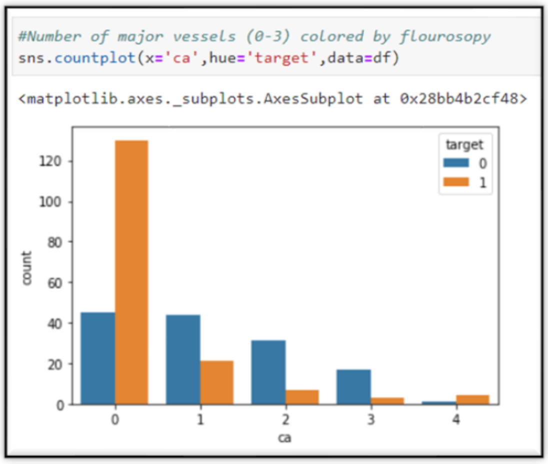
5.5.4 Bar Charts:

Furthermore, Jupyter Notebook is capable of making charts to get details of every column. To make charts code and the columns are as follows.



```
#Heart disease cases according to the gender
sns.countplot(x='sex', data=df, palette='husl', hue='target')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28bb4a6d348>
```



```
#Number of major vessels (0-3) colored by flourosopy
sns.countplot(x='ca',hue='target',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28bb4b2cf48>
```

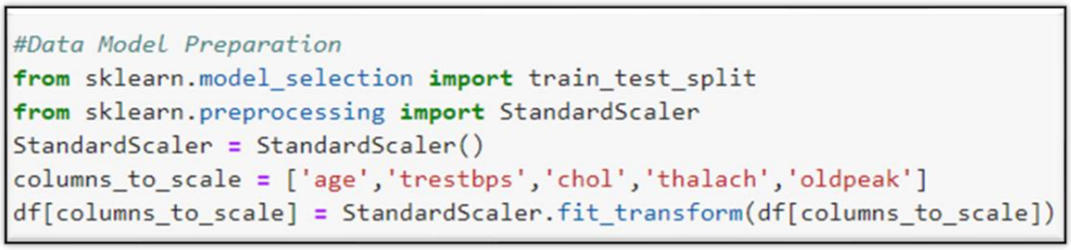
Figure 5:14: Bar Charts

As I mentioned earlier, JupyterLab is capable of doing many things with the data sets, such as getting counts, comparing columns, and many more.

6 Building the Model

6.1 Data Transformation

When using the dataset to build the machine learning model, the first step would be preparing the data and spit them. We must transform the data into a readable format, so it is easier to predict the result using algorithms. Also, we must split the data into a training data set and testing data set. The transformation can be done by using StandardScaler library as follows in Figure 6:1: Data Scaling

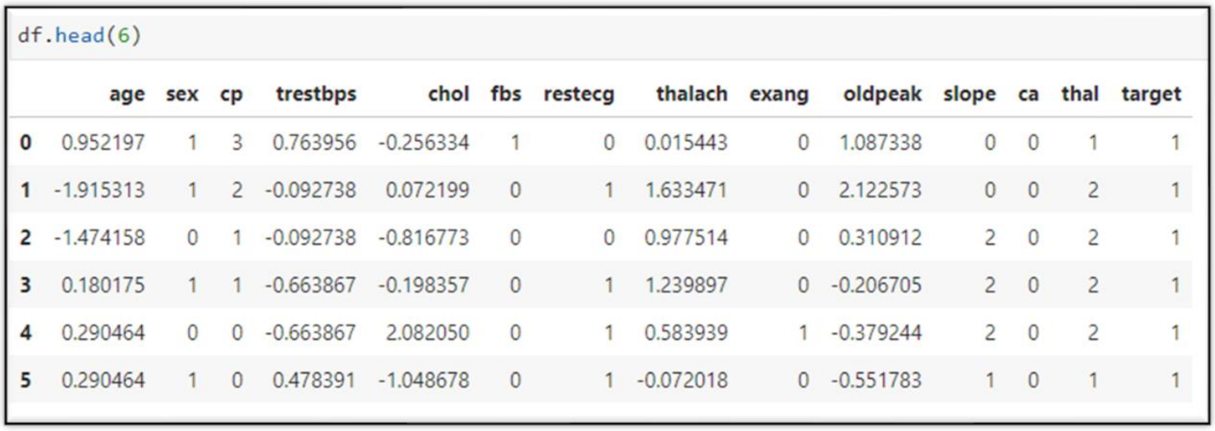


```
#Data Model Preparation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()
columns_to_scale = ['age','trestbps','chol','thalach','oldpeak']
df[columns_to_scale] = StandardScaler.fit_transform(df[columns_to_scale])
```

Figure 6:1: Data Scaling

What this code segment does is importing the library by using **sklearn.preprocessing import StandardScaler** and preprocess the data that in age, trestbps, chol, thalach, and oldpeak columns and equals it to **df**.

From sklearn.model_selection import train_test_split is to use later in the data splitting stage. Now we can see the newly transformed data by using the following command in the below figure.

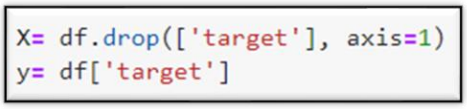


```
df.head(6)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	0.952197	1	3	0.763956	-0.256334	1	0	0.015443	0	1.087338	0	0	1	1
1	-1.915313	1	2	-0.092738	0.072199	0	1	1.633471	0	2.122573	0	0	2	1
2	-1.474158	0	1	-0.092738	-0.816773	0	0	0.977514	0	0.310912	2	0	2	1
3	0.180175	1	1	-0.663867	-0.198357	0	1	1.239897	0	-0.206705	2	0	2	1
4	0.290464	0	0	-0.663867	2.082050	0	1	0.583939	1	-0.379244	2	0	2	1
5	0.290464	1	0	0.478391	-1.048678	0	1	-0.072018	0	-0.551783	1	0	1	1

Figure 6:2: Veiwing the dataset

In this dataset, we must drop the column **target** because that is the column we are trying to predict using this machine learning model. The code to do that as follows



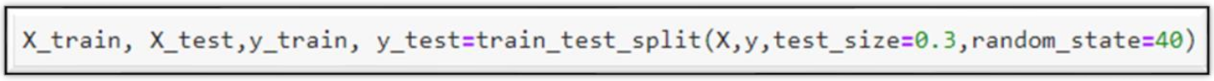
```
X= df.drop(['target'], axis=1)
y= df['target']
```

Figure 6:3: Dropping target column

In Figure 6:3: Dropping target column, I only drop the target column in X data. The reason for that is that I use the X data to train the data. By training the data, it will predict the target column's value. To Y data, I am only selecting the **target** column.

6.2 Data Splitting

The next part is selecting the size of the data. The code for the splitting and selecting size as follows.

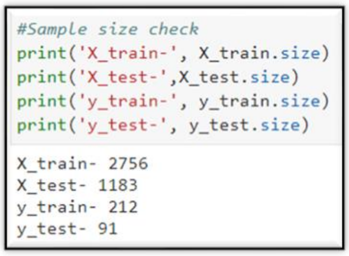


```
X_train, X_test,y_train, y_test=train_test_split(X,y,test_size=0.3,random_state=40)
```

Figure 6:4: Data Splitting and Selecting Size

In this code segment, X_train, X_test, y_train, y_test are the objects, and that is the sequence to follow. The rest of the code is splitting the data to x and y. And the test size that I chose is 30% and a random state of 40.

Now we can print the split dataset values as follows.



```
#Sample size check
print('X_train-', X_train.size)
print('X_test-',X_test.size)
print('y_train-', y_train.size)
print('y_test-', y_test.size)
```

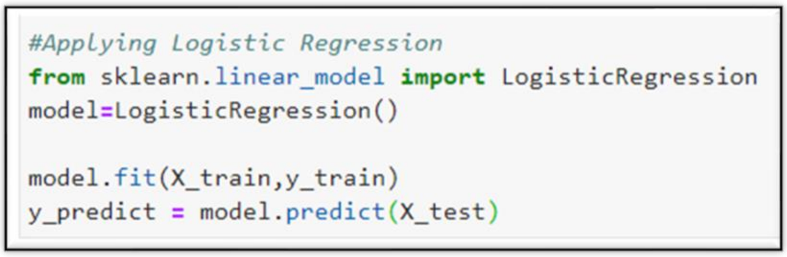
```
X_train- 2756
X_test- 1183
y_train- 212
y_test- 91
```

Figure 6:5: Split data values

In the split data, we can see training data have got larger values because I have allocated 70% to training data and only 30% to the testing data.

6.3 Applying the Algorithm

There are several algorithms available to predict the results of this model. I used the Logistic Regression algorithm because it had the best prediction percentage so far.



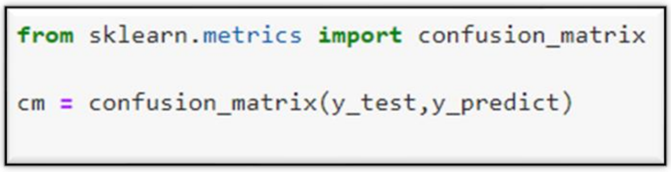
```
#Applying Logistic Regression
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()

model.fit(X_train,y_train)
y_predict = model.predict(X_test)
```

Figure 6:6: Applying the Algorithm

In the above code segment, first I have imported the algorithm by using the command of **from sklearn.linera_model import LogisticRegression**. Then I assigned an object name **model** to that algorithm.

Then, I fit the **model** object with X_train and y_train data so I can define another object named y_predict and predict using X-test data.

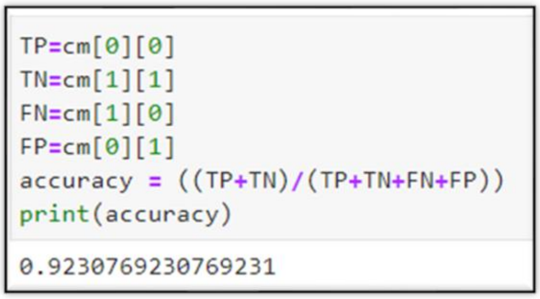
To virtualize the performance of the algorithm, I used the confusion matrix.



```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_predict)
```

Figure 6:7: Virtualization using confusion matrix

Using those virtualizations, I tested the accuracy of the algorithm using True Positive values, True Negative values, False Positive values, False Negative values, and calculating them as follows.



```
TP=cm[0][0]
TN=cm[1][1]
FN=cm[1][0]
FP=cm[0][1]
accuracy = ((TP+TN)/(TP+TN+FN+FP))
print(accuracy)
```

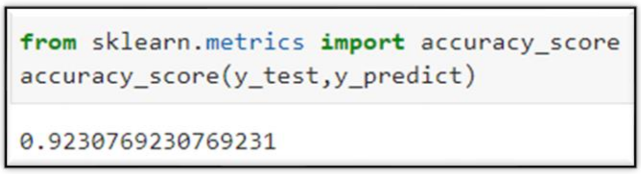
```
0.9230769230769231
```

Figure 6:8: Testing Accuracy

As in Figure 6:8: Testing Accuracy, we can see we have got 92% accuracy, which is not bad. But it could be improved and it should.

6.4 Predicting the Result

Now all the building and testing are done. So, we can predict the first output by using y_test data and y_predict object that used from fitting the X_test data.

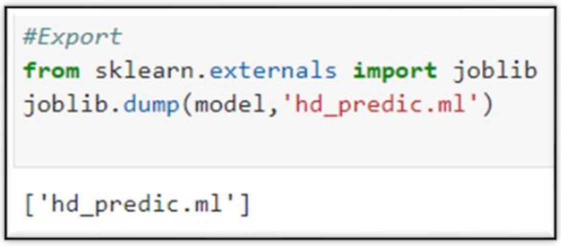


```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_predict)

0.9230769230769231
```

Figure 6:9: Prediction

According to Figure 6:9: Prediction, we can see our prediction result is 92% and the same as the test result. Now we can use this model to predict the patients' status.

6.5 Export of the Model



```
#Export
from sklearn.externals import joblib
joblib.dump(model,'hd_predic.ml')

['hd_predic.ml']
```

Figure 6:10: Export of the Model

As in Figure 6:10: Export of the Model, we can dump this model as **hd_predict.ml** by using the imported library of joblib. The dumped **hd_predict.ml** file is to use in the python web application.

7 Building the Website

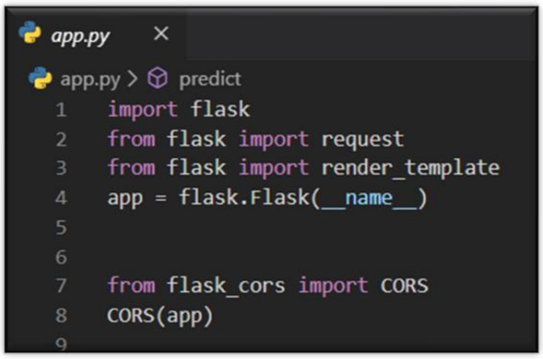
The machine learning model is complete now. To predict the result from this model, we have to get dynamic data from users. In this case, I build an HTML file with a form to get user inputs and pass those data using the GET method to my python backend file.

7.1 Building the HTML File

The HTML file includes a form of getting the user inputs. It also includes the GET method to send the user inputs to the **app.py** file.

7.2 Building the Flask Framework

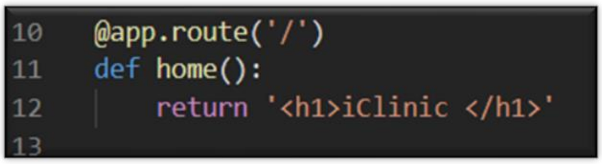
The first file we should build is the **app.py** file. First, we should import the libraries to build the application. If you don't familiar with building a web application using the Flask framework, you can refer to this website to learn.



```python
import flask
from flask import request
from flask import render_template
app = flask.Flask(__name__)


from flask_cors import CORS
CORS(app)
```

Figure 7:1: Importing Flask Libraries

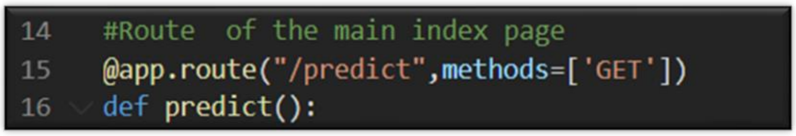
Next, we have to set the default route to the application as follows.



```python
@app.route('/')
def home():
    return '<h1>iClinic </h1>'
```

Figure 7:2: Setting the default route

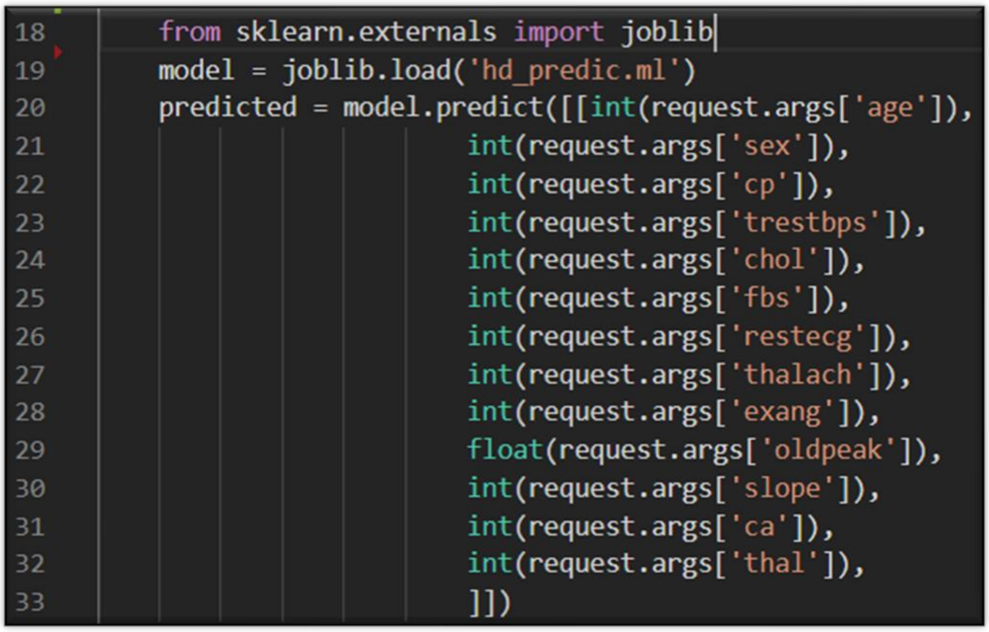
Then we can build the GET method to get the data from the HTML file and predict the result. Before request the data fro the HTML file, we have to set another route to define the GET method as follows.



```python
#Route  of the main index page
@app.route("/predict",methods=['GET'])
def predict():
```

Figure 7:3: Setting route and defining function

As in Figure 7:3: Setting route and defining function, I created the route and added the GET method request and defined the function to make the machine learning prediction. Next, we can get the data to the function to predict the result.

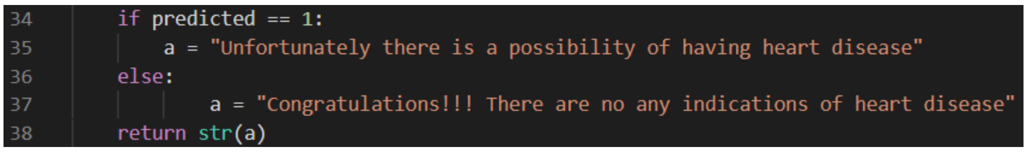


```python
    from sklearn.externals import joblib
    model = joblib.load('hd_predic.ml')
    predicted = model.predict([[int(request.args['age']),
                                int(request.args['sex']),
                                int(request.args['cp']),
                                int(request.args['trestbps']),
                                int(request.args['chol']),
                                int(request.args['fbs']),
                                int(request.args['restecg']),
                                int(request.args['thalach']),
                                int(request.args['exang']),
                                float(request.args['oldpeak']),
                                int(request.args['slope']),
                                int(request.args['ca']),
                                int(request.args['thal']),
                                ]])
```

Figure 7:4: Getting data from HTML file

Figure 7:4: Getting data from HTML file shows how to import the hd_predict.ml file that previously dumped from Jupyter Notebook and getting the HTML user inputs to the prediction function, it is possible to predict the result.

Then we can return the prediction result to the HTML file as follows.



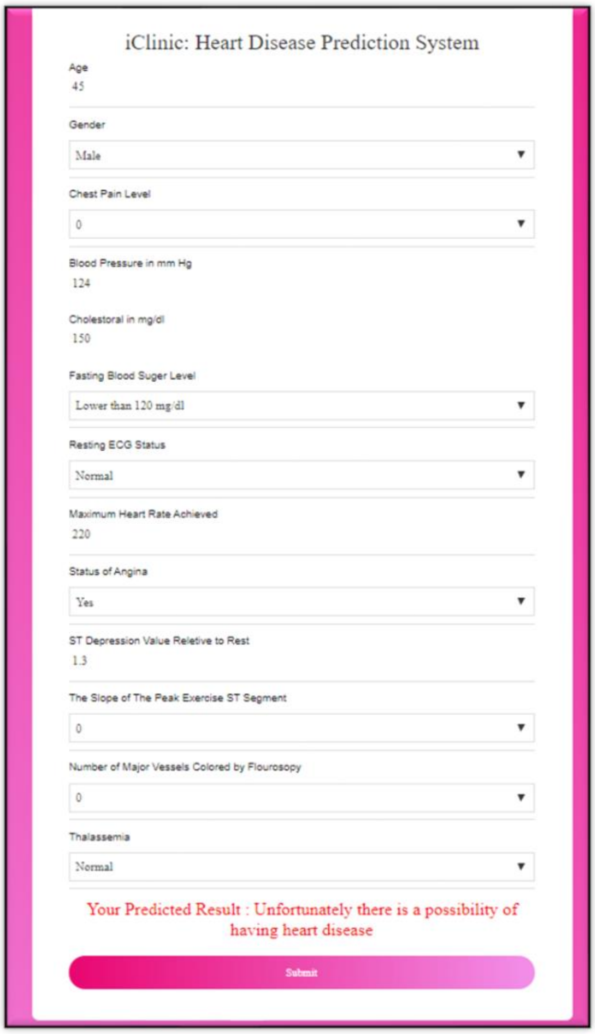
```
    if predicted == 1:
        a = "Unfortunately there is a possibility of having heart disease"
    else:
            a = "Congratulations!!! There are no any indications of heart disease"
    return str(a)
```

Figure 7:5: Returning the result

8 Running the Application

To run the application, all we have to do is open a terminal or command prompt from the file location and run the **app.py** file using **python app.py** command.

Then we can open the HTML file separately and insert the data and submit the data to predict the results.



iClinic: Heart Disease Prediction System

Age	45
Gender	Male
Chest Pain Level	0
Blood Pressure in mm Hg	124
Cholestoral in mg/dl	150
Fasting Blood Suger Level	Lower than 120 mg/dl
Resting ECG Status	Normal
Maximum Heart Rate Achieved	220
Status of Angina	Yes
ST Depression Value Reletive to Rest	1.3
The Slope of The Peak Exercise ST Segment	0
Number of Major Vessels Colored by Flourosopy	0
Thalassemia	Normal

Your Predicted Result : Unfortunately there is a possibility of having heart disease

Figure 8:1: Final Output