Heart Disease Prediction

Group Name: Python Squad

Group Members:

First name	Last Name	Student number
Albar	Vasi	C0854302
Ammu	Prakash	C0853579
Taj Baba	Syed	C0852768
Noorul Amin	Syed	C0847495

Submission date: ***21st April 2022***

Contents

Lambton College | In Toronto

AML-1204
Term Project

Abstract

With the increasing pace of the world towards technology, working from home has increased, and social interaction in real life has significantly reduced. Not only this, an unhealthy diet has been accepted by most people to keep up with the fast-paced surroundings. Undoubtedly, taking this life environment has resulted in health concerns, with heart disease being one of the major concerns. In the early days, it was observed that only people above 60 years were contracting heart disease, but in today's world, kids, teenagers to adults in their mid and late 20s are also contracting heart disease. It also carries a phenomenon where every other illness, big or small, can cause a cardiac arrest/heart attack. So it is evident what a significant concern heart disease is for the human population.

This concern brought us to pick up this topic as our term project. Here, we would be implementing three machine algorithms. The one with the best accuracy would be used to create the final prediction model where a user can enter their stats in real time and check if they have a possibility of heart disease or not.

Introduction

Heart disease is one of the leading causes of death of men, women and most of the other ethnic and racial groups in United States. It is seen that the one person in every 30 – 40 seconds dies due to a cardiovascular disease in United States. The increasing rate of heart diseases have caused United states about $360 billion each year in 2016 and 2017. This huge amount of money includes health care services, medicines, and productivity loss due to death.

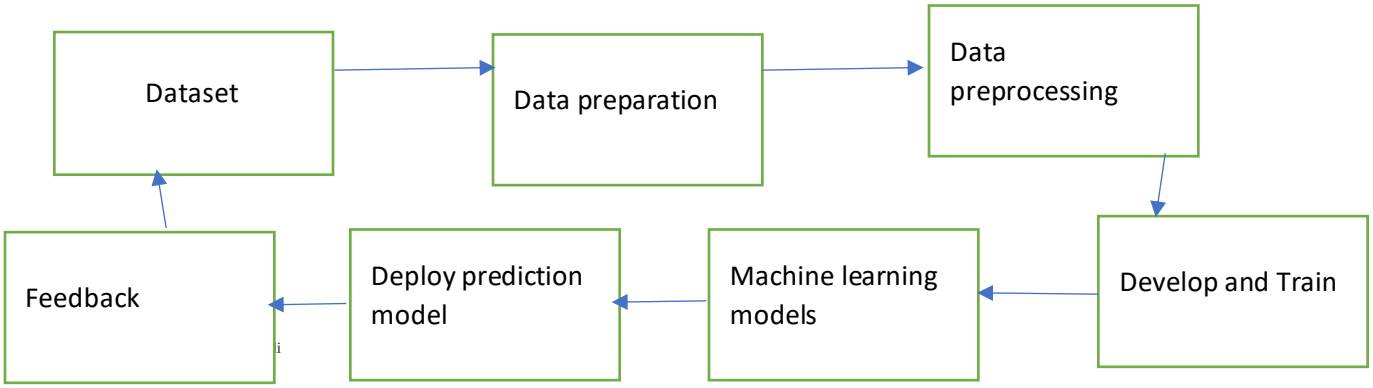
It is very important that an early prognosis for heart disease is done so those appropriate decisions can be made like changing lifestyle, dietary changes, etc., for the patients who are at risk.

To be able to implement our project, it was important to have a dataset to train our prediction algorithms to get the desired result. For this, we have taken the dataset from Kaggle, which consists of 1025 rows and 14 columns. Out of the 14 columns, 13 columns contribute highly toward a person suffering from heart disease or not. The 14th column, or the target column, is the final value based on the initial 13 columns, explaining whether that particular patient is suffering from heart disease or not.

To ensure that no patient data is leaked and abide by PII, we have ensured to pickup the dataset that doesn't have any patient's identity revealing pieces of information.

To implement our project to complete fruition, we have mainly used pandas, sckitlearn, Tkinter, and seaborn libraries.

Flow Chart



Methods

Dataset information –

We downloaded the dataset from Kaggle. The dataset is a sample dataset, and sample dataset is drawn randomly from the population. This dataset contains the information of patients showcasing both the patients that have a heart problem and the one's that do not have a heart problem. As we are using supervised learning, this dataset becomes ideal for us to implement in our project.

Below are the detailed value of the dataset –

- Age

- Sex

- Chest pain type (4 values)

— Value 0: typical angina

— Value 1: atypical angina

— Value 2: non-anginal pain

— Value 3: asymptomatic

- trestbps: resting blood pressure (in mm Hg on admission to the hospital)

- chol: serum cholestoral in mg/dl

- fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

- restecg: resting electrocardiographic results

— Value 0: normal

— Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)

— Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria

- thalach: maximum heart rate achieved

- exang: exercise induced angina (1 = yes; 0 = no)

- oldpeak = ST depression induced by exercise relative to rest

- slope: the slope of the peak exercise ST segment

— Value 1: up-sloping

— Value 2: flat

— Value 3: down-sloping

- ca: number of major vessels (0–3) colored by fluoroscope

- thal: 3 = normal; 6 = fixed defect; 7 = reversible defect

- target : 0=low risk of heart attack, 1=high risk of heart attack

We start the project with importing the libraries that are necessary –

```
In [72]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from sklearn import metrics
         from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

In [95]: from sklearn.ensemble import RandomForestClassifier

In [102]: from sklearn.ensemble import GradientBoostingClassifier

In [122]: from tkinter import *
```

Pandas – We are loading Pandas to ensure that the dataset is converted to structured table or dataframe from comma separated values. This is being done due to feeding csv files to machine learning algorithms can result in complications

Train Test Split – Train test split is a module of scikitlearn that is used to split the data into two variables which represest test and train respectively.

Metrics – We are importing metrics as we need to understand the accuracy score of the machine learning models using the dataset.

StandardScaler – StandardScaler is used in preprocessing making sure that the iterations done on the data set is scalable and we do not run in to max no of iterations reached error.

Logistic Regression, Random Forest Classifier and Gradient Boosting Classifier – These are three supervised learning classification algorithms that we are working with.

Tkinter – We have used tkinkter library to create a gui view where a user can enter their values as per the dataset columns in real time and check the possibility of heart disease.

Loading the dataset –

We start first with loading the dataset to pandas –

```
In [73]: dataset = 'heart.csv'
         heart_dataset = pd.read_csv(dataset)
```

After loading the dataset we use the head and tail function to view the first 5 and last 5 values,

```
In [74]: heart_dataset.head()
```

Out[74]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

```
In [75]: heart_dataset.tail()
```

Out[75]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

We use the info() function to check the data type and value in each column of the dataset –

```
In [77]: heart_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Describe() function is being used to display a detailed analysis of the data set –

```
In [80]: heart_dataset.describe()
```

Out[80]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

As part of data prepration step we need to check if there are any null values or not in the data set, if there are any then we would have to apply techniques to make sure those empty spaces are filled, but in our case we do not have any null values -

```
In [79]: heart_dataset.isnull().sum()

Out[79]: 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
```

As a last step in data preparation, we view the target values present in the dataset –

```
In [81]: heart_dataset['target'].value_counts()

Out[81]: 1    526
         0    499
         Name: target, dtype: int64
```

We start with data preprocessing, by dividing the dataset in to two variables, X and Y. Out of the 14 columns, we have entered 13 columns in variable X as the prediction will be done based on the 13 columns and to the Y variable we have given the predetermined target column. The Y variable will be used against the X variable to check for accuracy for prediction.

```
In [82]: X = heart_dataset.drop(columns='target', axis=1)
         Y = heart_dataset['target']
```

We print values of X and Y to ensure that the dataset division is successful.

```
In [83]: print(X)

        age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0        52    1   0       125   212    0        1      168      0      1.0
1        53    1   0       140   203    1        0      155      1      3.1
2        70    1   0       145   174    0        1      125      1      2.6
3        61    1   0       148   203    0        1      161      0      0.0
4        62    0   0       138   294    1        1      106      0      1.9
...     ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
1020     59    1   1       140   221    0        1      164      1      0.0
1021     60    1   0       125   258    0        0      141      1      2.8
1022     47    1   0       110   275    0        0      118      1      1.0
1023     50    0   0       110   254    0        0      159      0      0.0
1024     54    1   0       120   188    0        1      113      0      1.4

        slope  ca  thal
0           2   2     3
1           0   0     3
2           0   0     3
3           2   1     3
4           1   3     2
...       ...  ..   ...
1020        2   0     2
1021        1   1     3
1022        1   1     2
1023        2   0     2
1024        1   1     3

[1025 rows x 13 columns]
```

```
In [84]: print(Y)

0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

Next step is to ensure that we do not run in to iteration issue when we implement the machine learning algorithms and thus to avoid that we use StandardScalar module on X.

```
In [85]: sc= StandardScaler()
         X = sc.fit_transform(X)
```

We next start to implement the scikitlearn module train_test_split to split the dataset in X and Y, in to training and testing. We have provided 80% dat to training while 20% has gone to test. We have also implemented, Startify and random state functions, to ensure that each time train_test_split is run, the dataset splits in the same manner.

```
In [86]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

Classification Algorithms –

We are implementing classification algorithms for this project as we are trying to predict the disease and not just perform analysis and visualization. So when we say prediction, our final value will be 1 or 0, where 0 would denote no heart disease problems and 1 would denote possibility of heart disease.

As we are end value will be 1 or 0, in problems like these classification algorthims provide the best accuracies.

We are implementing Logistic Regression, Random Forest Classifier and Gradient Boosting Classifier.

tic Regression –

We load the logistic regression module from sciktilearn library and save the module in a variable.

Logistic Regression

```
In [88]: from sklearn.linear_model import LogisticRegression

In [89]: log_reg = LogisticRegression()
```

We implemented the fit() function using the logistic regression on X and Y train respectively, to complete model training.

```
In [90]: log_reg.fit(X_train, Y_train)
Out[90]: LogisticRegression()
```

After we complete the model training, we implement the predict() function on both X train and test, and check for the accuracy score by running it against the Y train and test which contain the original target values.

```
In [91]: X_train_prediction = log_reg.predict(X_train)
         training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

In [92]: print('Accuracy on Training data : ', training_data_accuracy)

         Accuracy on Training data :  0.8585365853658536

In [93]: X_test_prediction = log_reg.predict(X_test)
         test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

In [94]: print('Accuracy on Test data : ', test_data_accuracy)

         Accuracy on Test data :  0.8048780487804879
```

We get 0.85 and 0.80 as the accuracy score for train and test respectively. This is a good accuracy score. However, we will implement two more algorithms to confirm the best fit model to run using the dataset at hand.

Random Forest Classifier –

We start with loading the algorithm from scikitlearn library and assigning it to a variale

```
In [95]: from sklearn.ensemble import RandomForestClassifier

In [96]: rf_class = RandomForestClassifier()
```

Next we use the fit() function to complete the training of the model,

```
In [97]: rf_class.fit(X_train,Y_train)
Out[97]: RandomForestClassifier()
```

Lastly, we perform the prediction on train and test and check the accuracy score.

```
In [98]: rf_class_train_prediction = rf_class.predict(X_train)

In [99]: accuracy_score(Y_train,rf_class_train_prediction)
Out[99]: 1.0

In [100]: rf_class_test_prediction = rf_class.predict(X_test)

In [101]: accuracy_score(Y_test,rf_class_test_prediction)
Out[101]: 1.0
```

We get both train and test accuracy score as 1.0 that is equivalent to 100% and random forest classifier appears to be the best fit model for our dataset.

We however, complete the predicition for Gradient Boosting Classifier as well and we follow the same steps as above.

```
In [102]: from sklearn.ensemble import GradientBoostingClassifier

In [103]: gbc = GradientBoostingClassifier()

In [104]: gbc.fit(X_train,Y_train)
Out[104]: GradientBoostingClassifier()

In [105]: gbc_train_prediction = gbc.predict(X_train)

In [106]: accuracy_score(Y_train,gbc_train_prediction)
Out[106]: 0.9878048780487805

In [107]: gbc_test_prediction = gbc.predict(X_test)

In [108]: accuracy_score(Y_test,gbc_test_prediction)
Out[108]: 0.9853658536585366
```

The accuracy we received for Gradient Boosting Classifier is 0.987 and 0.985 for train and test, respectively.

For better comparison, we fed all the accuracy scores to pandas data frame and displayed and also used seaborn function of python to visualize those tables.

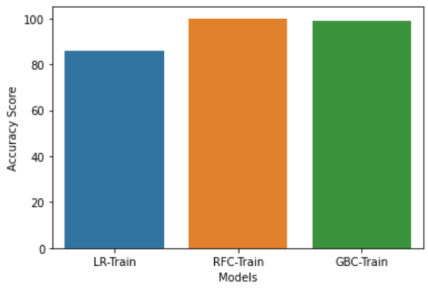
```
In [109]: train_final_data = pd.DataFrame({'Models':['LR-Train','RFC-Train','GBC-Train'],
                                           'Accuracy Score':[training_data_accuracy*100,
                                                             accuracy_score(Y_train,rf_class_train_prediction)*100,
                                                             accuracy_score(Y_train,gbc_train_prediction)*100]})

In [110]: train_final_data
Out[110]:
```

	Models	Accuracy Score
0	LR-Train	85.853659
1	RFC-Train	100.000000
2	GBC-Train	98.780488

```
In [42]: import seaborn as sns

In [54]: sns.barplot(train_final_data['Models'],train_final_data['Accuracy Score'])
C:\Users\albar\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
Out[54]: <AxesSubplot:xlabel='Models', ylabel='Accuracy Score'>
```



```
In [111]: test_final_data = pd.DataFrame({'Models':['LR-Test','RFC-Test','GBC-Test'],
                                          'Accuracy Score':[test_data_accuracy*100,
                                                            accuracy_score(Y_test,rf_class_test_prediction)*100,
                                                            accuracy_score(Y_test,gbc_test_prediction)*100]})
```

```
In [112]: test_final_data
```

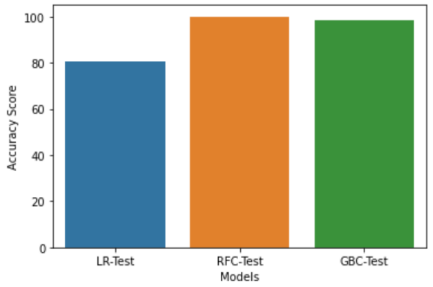
Out[112]:

	Models	Accuracy Score
0	LR-Test	80.487805
1	RFC-Test	100.000000
2	GBC-Test	98.536585

```
In [44]: sns.barplot(test_final_data['Models'],test_final_data['Accuracy Score'])
```

```
C:\Users\albar\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[44]: <AxesSubplot:xlabel='Models', ylabel='Accuracy Score'>



As per the accuracy score and bar plots it is evident that Random Forest Classifier is the best fit model for our dataset to predict heart disease.

To test that, we recall the Random Forest Classifier and split the dataset again in to X and Y. This time around we do not do a splitting of data set in train and test, but rather feed the whole data set to Random Forest Classifier and prepare it for prediction.

```
In [45]: X=heart_dataset.drop('target',axis=1)
         Y=heart_dataset['target']
```

Model Prediction

```
In [46]: from sklearn.ensemble import RandomForestClassifier
```

```
In [47]: rf_class = RandomForestClassifier()
         rf_class.fit(X,Y)
```

Out[47]: RandomForestClassifier()

We have entered data as per the dataset columns using pandas data frame from the existing dataset,

In [48]:
```
new_data = pd.DataFrame({
    'age':34,
    'sex':0,
    'cp':1,
    'trestbps':118,
    'chol':210,
    'fbs':0,
    'restecg':1,
    'thalach':192,
    'exang':0,
    'oldpeak':0.7,
     'slope':2,
    'ca':0,
    'thal':2,
},index=[0])
```

In [49]:
```
new_data
```

Out[49]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	34	0	1	118	210	0	1	192	0	0.7	2	0	2

We do the prediction on this data and get the desired value,

In [50]:
```
predictor = rf_class.predict(new_data)
if predictor[0]==0:
    print("No Disease")
else:
    print("Disease")
```

```
Disease
```

Results

As per the accuracy score and bar plots it is evident that Random Forest Classifier is the best fit model for our dataset to predict heart disease.

To test that, we recall the Random Forest Classifier and split the dataset again in to X and Y. This time around we do not do a splitting of data set in train and test, but rather feed the whole data set to Random Forest Classifier and prepare it for prediction.

In [45]:
```
X=heart_dataset.drop('target',axis=1)
Y=heart_dataset['target']
```

Model Prediction

In [46]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [47]:
```
rf_class = RandomForestClassifier()
rf_class.fit(X,Y)
```

Out[47]:
```
RandomForestClassifier()
```

We have entered data as per the dataset columns using pandas data frame from the existing dataset,

```
In [48]: new_data = pd.DataFrame({
             'age':34,
             'sex':0,
             'cp':1,
             'trestbps':118,
             'chol':210,
             'fbs':0,
             'restecg':1,
             'thalach':192,
             'exang':0,
             'oldpeak':0.7,
              'slope':2,
             'ca':0,
             'thal':2,
         },index=[0])
```

```
In [49]: new_data
```

Out[49]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	34	0	1	118	210	0	1	192	0	0.7	2	0	2

We do the prediction on this data and get the desired value,

```
In [50]: predictor = rf_class.predict(new_data)
         if predictor[0]==0:
             print("No Disease")
         else:
             print("Disease")
```

```
Disease
```

GUI

The GUI that we have used, serves the purpose of a patient/user/provider able to enter the features or column values in real time and find out based on the features if there is a possibility of a heart disease.

We have combined our GUI with our best fit model i.e. Random Forest Classifier to complete the prediction.

For implementing GUI, we are using python's Tkinter library and PIL for loading the background image for the GUI.

We start by loading the library and all its functionalities –

```
In [51]: from tkinter import *
         from PIL import Image, ImageTk
```

We defined a function show_entry_fields(), and have created 13 variables depicting the 13 features/columns of the dataset.

```
def show_entry_fields():
    p1=int(e1.get())
    p2=int(e2.get())
    p3=int(e3.get())
    p4=int(e4.get())
    p5=int(e5.get())
    p6=int(e6.get())
    p7=int(e7.get())
    p8=int(e8.get())
    p9=int(e9.get())
    p10=float(e10.get())
    p11=int(e11.get())
    p12=int(e12.get())
    p13=int(e13.get())
    result=rf_class.predict([[p1,p2,p3,p4,p5,p6,p7,p8,p8,p10,p11,p12,p13]])

    if result == 0:
        Label(master, text="No Heart Disease").grid(row=31)
    else:
        Label(master, text="Possibility of Heart Disease").grid(row=31)
```

In the result variable, we are also calling our best fit model Random Forest Classifier variable with the predict function on the feature variables so that whenever user enters the value and hits the predict button our model is able to predict the presence of heart disease or not.

```
master = Tk()
master.geometry('800x500')
master.title("Heart Disease Prediction")

#background image loading

load = Image.open('pic.jpg')
render = ImageTk.PhotoImage(load)
img = Label(master, image = render)
img.place(x = 0, y = 0)



label = Label(master, text = "Heart Disease Prediction"
                    , bg = "red", fg = "black",font = 'Ariel 15 bold italic'). \
                              grid(row=1,columnspan=2)


Label(master,bg ='PeachPuff4').grid()
Label(master, text="Enter Your Age",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=3)
Label(master, text="Male Or Female [1/0]",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=4)
Label(master, text="Enter Value of Chest Pain",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=5)
Label(master, text="Enter Value of Resting Blood Pressure",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=6)
Label(master, text="Enter Value of Cholestrol",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=7)
Label(master, text="Enter Value of Fasting Blood Sugar",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=8)
Label(master, text="Enter Value of Resting ECG",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(row=9)
Label(master, text="Enter Value of thalach(Maximum Heart Rate Achieved)",bg = 'PeachPuff4',fg = 'white',font = 'bold').grid(r
```

We are storing the tkinter function in a variable called master and call it everytime we are using the label function. The Label function is being used to display the text box and enter() function is being used to enter the text in to the text box.

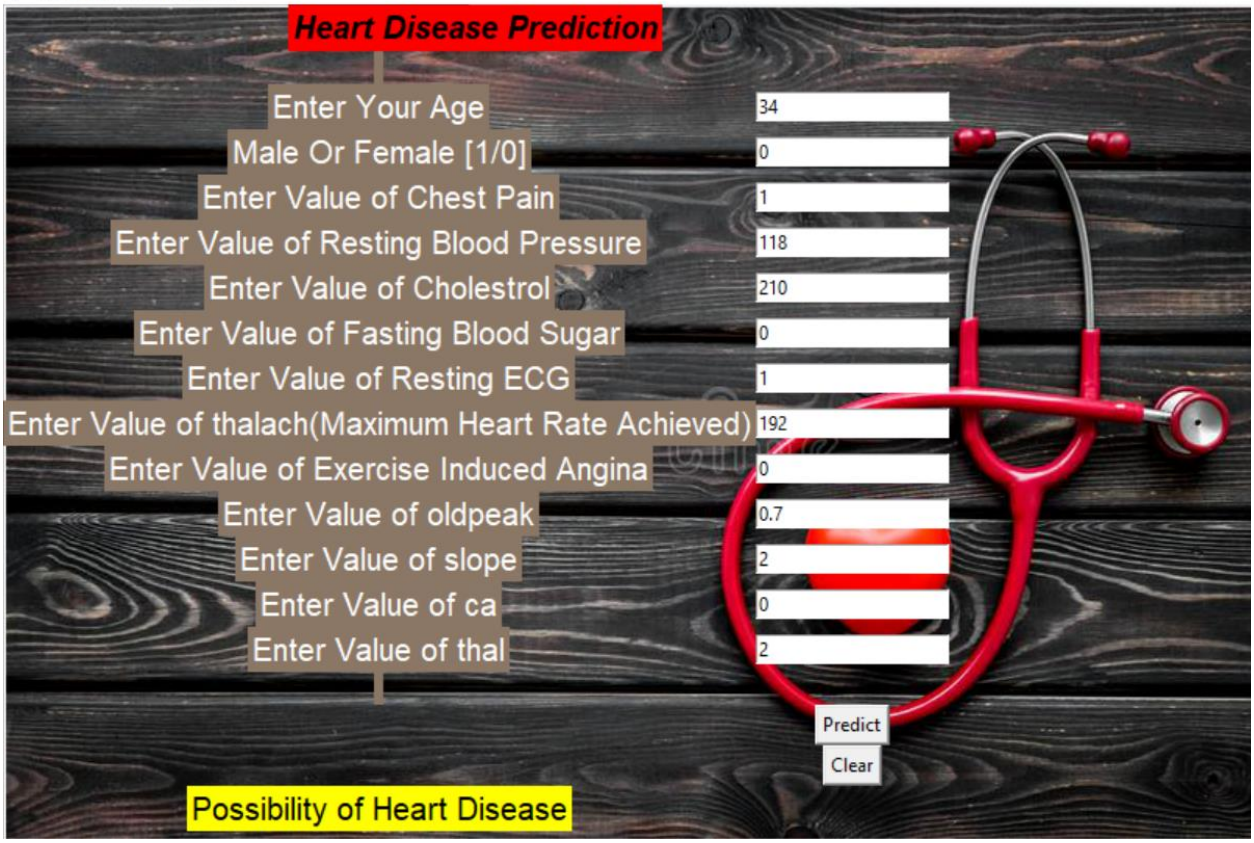
```
e1 = Entry(master)
e2 = Entry(master)
e3 = Entry(master)
e4 = Entry(master)
e5 = Entry(master)
e6 = Entry(master)
e7 = Entry(master)
e8 = Entry(master)
e9 = Entry(master)
e10 = Entry(master)
e11 = Entry(master)
e12 = Entry(master)
```

The mainloop() function of Tkinter is implemented, for event listening that is going to occur after we click the predict and clear button.

```
Button(master, text='Predict', command=show_entry_fields).grid(column=1)
Button(master, text='Clear', command=clear_text).grid(column=1)


master.mainloop()
```

The final output is a popup where a user can enter real time values and check if they have the possibility of the heart disease or not.



Conclusions and Future Work

We are able to successfully implement Heart Disease Prediction with 100% accuracy using Random Forest Classifier while using real time values as input. This project can be of great help for home users who want to test themselves with regards to the health of their heart. The implementation done here is no doubt a demonstration of a good team effort but there is a lot of future work that can be done on this project.

This project can be converted in to a full fledged disease prediction if multiple datasets of different diseases i.e. Pneumonia, Diabetes, COVID19 etc are used. A database can be implemented to keep the record of the patient's health and symptoms and predict disease based on real life symptoms instead of randomly generated datasets. A database can also be used to implement admin, provider and user privileges. We as a team are planning to explore these options and hopefully would try implement them in the near future as our expertise with python, machine learning and database grows.

References

- IRJET-V5I3930.pdf
- Heart Disease Dataset | Kaggle
- 2021 Heart Disease and Stroke Statistics Update Fact Sheet At-a-Glance
- A Guide on Splitting Datasets With Train_test_split Function (bitdegree.org)
- Scikit Learn - Logistic Regression (tutorialspoint.com)
- Python Tutorial (w3schools.com)
- albarvasi/heart_disease_prediction (github.com)