CORONARY HEART DISEASE PREDICTION

U15CS705R - COMPREHENSION AND TECHNICAL REPORT

Activity 3

submitted by

Sundhar U M (1517102163)

Sivanandham S (1517102151)

Tamilarasan (1517102166)

VII Semester

COMPUTER SCIENCE AND ENGINEERING

SONA COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

ANNA UNIVERSITY: CHENNAI 600 025

December 2020

ANNA UNIVERSITY: CHENNAI – 600 025

BONAFIDE CERTIFICATE

This is to certify that the technical report entitled **"Coronary Heart Disease Prediction"** is the bonafide report of **Sundhar U M (1517102163), Sivanandham S (1517102151), Tamilarasan (1517102166)"** of B.E Computer Science and Engineering during the year 2020-21.

SIGNATURE
Dr.B.SATHIYABHAMA M.Tech,Ph.D.,
HEAD OF THE DEPARTMENT
Professor
Department of Computer Science and Engineering,
Sona College of Technology,
Salem.

SIGNATURE
Dr.S.SAKTHIVEL M.E,Ph.D.,
STAFF IN-CHARGE
Professor
Department of Computer Science and Engineering,
Sona College of Technology,
Salem.

Submitted for Comprehension and Technical Report **(U15CS705R)** examination held on (12/01/2021).

Examiner

INDEX

ABSTRACT

Nowadays, health disease are increasing day by day due to life style, hereditary. Especially, heart disease has become more common these days, i.e. life of people is at risk. Each individual has different values for Blood pressure, cholesterol and pulse rate. But according to medically proven results the normal values of Blood pressure is 120/90, cholesterol is and pulse rate is 72. This paper gives the survey about different classification techniques used for predicting the risk level of each person based on age, gender, Blood pressure, cholesterol, pulse rate. The patient risk level is classified using datamining classification techniques such as Naïve Bayes, KNN, Decision Tree Algorithm, Random Forest. etc., Accuracy of the risk level is high when using more number of attributes.

TECHNICAL STACK

Application Name	CHD Detection
Frontend	FLASK
Backend	Machine Learning Algorithms
Application Type	Web Application
Languages Used	Python, html
Libraries	Pandas, Numpy, sklearn
Database	NIL

SOURCE CODE DOCUMENTATION

Model building user guide:

Step 1: Importing the libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Step 2: Importing dataset

```
In [2]: df=pd.read_csv("framingham.csv")
        df.head()
```

Out[2]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	Ter
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	

Step 3: Check for null values:

```
In [5]: df.isnull().sum()
Out[5]: Gendermale          0
        age                 0
        education         105
        currentSmoker       0
        cigsPerDay         29
        BPMeds             53
        prevalentStroke     0
        prevalentHyp        0
        diabetes            0
        totChol            50
        sysBP               0
        diaBP               0
        BMI                19
        heartRate           1
        glucose           388
        TenYearCHD          0
        dtype: int64
```

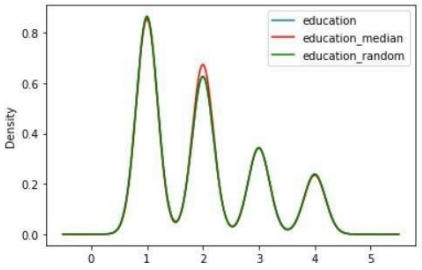
Step 4: Create a function to fill the empty values by median and random sample imputation.

In [8]:
```
def impute_nan(df,variable,median):
    df[variable+"_median"]=df[variable].fillna(median)
    df[variable+"_random"]=df[variable]
    ##It will have the random sample to fill the na
    random_sample=df[variable].dropna().sample(df[variable].isnull().sum(),random_state=0)
    ##pandas need to have same index in order to merge the dataset
    random_sample.index=df[df[variable].isnull()].index
    df.loc[df[variable].isnull(),variable+'_random']=random_sample
```

Step 5: Give the columns which has null values and plot the difference between median and random sample imputation.

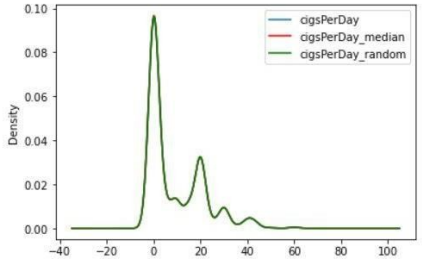
In [9]:
```
impute_nan(df,"education",median)
fig = plt.figure()
ax = fig.add_subplot(111)
df['education'].plot(kind='kde', ax=ax)
df.education_median.plot(kind='kde', ax=ax, color='red')
df.education_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[9]: <matplotlib.legend.Legend at 0x19258b2fb50>



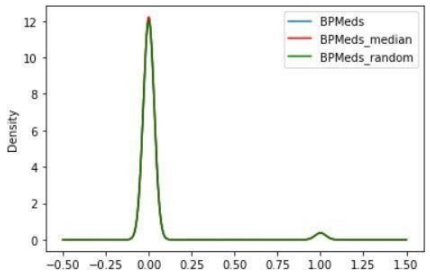
In [11]:
```
impute_nan(df,"cigsPerDay",median)
fig = plt.figure()
ax = fig.add_subplot(111)
df['cigsPerDay'].plot(kind='kde', ax=ax)
df.cigsPerDay_median.plot(kind='kde', ax=ax, color='red')
df.cigsPerDay_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[11]: <matplotlib.legend.Legend at 0x19258bfafd0>



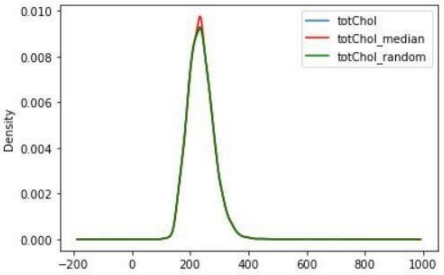
In [13]:
```
impute_nan(df,"BPMeds",median)
fig = plt.figure()
ax = fig.add_subplot(111)
df['BPMeds'].plot(kind='kde', ax=ax)
df.BPMeds_median.plot(kind='kde', ax=ax, color='red')
df.BPMeds_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[13]: <matplotlib.legend.Legend at 0x19258c59a00>



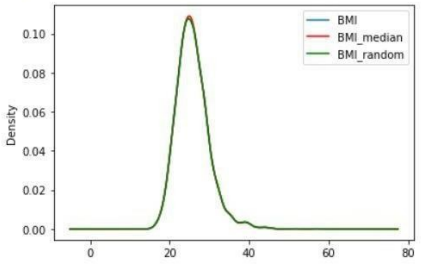
In [15]:
```
impute_nan(df,"totChol",median)
fig = plt.figure()
ax = fig.add_subplot(111)
df['totChol'].plot(kind='kde', ax=ax)
df.totChol_median.plot(kind='kde', ax=ax, color='red')
df.totChol_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[15]: <matplotlib.legend.Legend at 0x19258ce3f10>

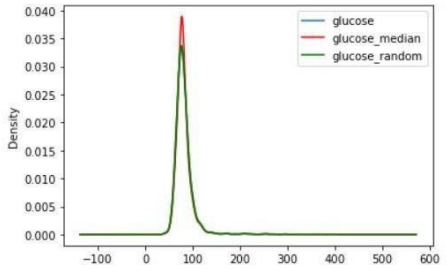


In [17]:
```
impute_nan(df,"BMI",median)
fig = plt.figure()
ax = fig.add_subplot(111)
df['BMI'].plot(kind='kde', ax=ax)
df.BMI_median.plot(kind='kde', ax=ax, color='red')
df.BMI_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[17]: <matplotlib.legend.Legend at 0x19258d3e4c0>



```
In [20]: impute_nan(df,"glucose",median)
         fig = plt.figure()
         ax = fig.add_subplot(111)
         df['glucose'].plot(kind='kde', ax=ax)
         df.glucose_median.plot(kind='kde', ax=ax, color='red')
         df.glucose_random.plot(kind='kde', ax=ax, color='green')
         lines, labels = ax.get_legend_handles_labels()
         ax.legend(lines, labels, loc='best')

Out[20]: <matplotlib.legend.Legend at 0x19258dbaca0>
```



Step 6: Drop the columns that have null values and '_median' as suffix.

```
In [22]: df.drop(["education", "education_median", "cigsPerDay", "cigsPerDay_median","BPMeds", "BPMeds_median", "totChol", "totChol_mediar
```

Step 7: Reorder the columns according to the original dataset.

```
In [23]: df.columns

Out[23]: Index(['Gendermale', 'age', 'currentSmoker', 'prevalentStroke', 'prevalentHyp',
                'diabetes', 'sysBP', 'diaBP', 'heartRate', 'TenYearCHD',
                'education_random', 'cigsPerDay_random', 'BPMeds_random',
                'totChol_random', 'BMI_random', 'glucose_random'],
               dtype='object')

In [24]: dffin = df[['age', 'Gendermale', 'currentSmoker','prevalentStroke','prevalentHyp','diabetes','sysBP','diaBP','heartRate','educati
         dffin.columns

Out[24]: Index(['age', 'Gendermale', 'currentSmoker', 'prevalentStroke', 'prevalentHyp',
                'diabetes', 'sysBP', 'diaBP', 'heartRate', 'education_random',
                'cigsPerDay_random', 'BPMeds_random', 'totChol_random', 'BMI_random',
                'glucose_random', 'TenYearCHD'],
               dtype='object')
```

Step 8: Using train_test_split method split the dataset into train and test data.

Train Test Split

```
In [25]: x=dffin.drop(['TenYearCHD'],axis=1)
         y=dffin['TenYearCHD']

In [26]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20)
```

Step 9: Train the classification models using fit method and predict the output for each algorithm separately.

Models

Naive Bayes

```
In [27]: from sklearn.naive_bayes import GaussianNB
         gnb = GaussianNB()
         gnb.fit(x_train, y_train)
         y_pred_gnb = gnb.predict(x_test)
```

Decision Tree

```
In [62]: from sklearn.tree import DecisionTreeClassifier
         dt = DecisionTreeClassifier()
         dt.fit(x_train, y_train)
         y_pred_dt = dt.predict(x_test)
```

Random Forest

```
In [102]: from sklearn.ensemble import RandomForestClassifier
          rf=RandomForestClassifier(random_state=1234)
          rf.fit(x_train,y_train)
          y_pred_rf = rf.predict(x_test)
```

K-Nearest Neighbour

```
In [30]: from sklearn.neighbors import KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=8)
         knn.fit(x_train,y_train)
         y_pred_knn = knn.predict(x_test)
```

Step 10: Evaluate the algorithms using confusion matrix and accuracy score.

Evaluation of Algorithms

In [31]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [32]:
```
mat=confusion_matrix(y_test,y_pred_gnb)
acc=accuracy_score(y_test,y_pred_gnb)
print(mat,acc)
```

```
[[660  54]
 [102  32]] 0.8160377358490566
```

In [33]:
```
mat=confusion_matrix(y_test,y_pred_dt)
acc=accuracy_score(y_test,y_pred_dt)
print(mat,acc)
```

```
[[609 105]
 [108  26]] 0.7488207547169812
```

In [103]:
```
mat=confusion_matrix(y_test,y_pred_rf)
acc=accuracy_score(y_test,y_pred_rf)
print(mat,acc)
```

```
[[708   6]
 [128   6]] 0.8419811320754716
```

In [35]:
```
mat=confusion_matrix(y_test,y_pred_knn)
acc=accuracy_score(y_test,y_pred_knn)
print(mat,acc)
```

```
[[707   7]
 [128   6]] 0.8408018867924528
```

Step 11: Pick the algorithm with highest accuracy value and create a dictionary for hyperparameter.

In [78]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 50, stop = 1000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [1,2,3,4,5,7]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1,2,4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)
```

```
{'n_estimators': [50, 155, 261, 366, 472, 577, 683, 788, 894, 1000], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [1
0, 120, 230, 340, 450, 560, 670, 780, 890, 1000], 'min_samples_split': [1, 2, 3, 4, 5, 7], 'min_samples_leaf': [1, 2, 4, 6, 8],
'criterion': ['entropy', 'gini']}
```

Step 12: Using Randomized Search CV find:

```
In [37]: rf=RandomForestClassifier()
         rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_grid,n_iter=500,cv=3,verbose=2,n_jobs=-1)
         ### fit the randomized model
         rf_randomcv.fit(x_train,y_train)

         Fitting 3 folds for each of 500 candidates, totalling 1500 fits

         [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
         [Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   16.4s
         [Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  1.2min
         [Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  2.8min
         [Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  4.8min
         [Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  7.1min
         [Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed: 10.4min
         [Parallel(n_jobs=-1)]: Done 1500 out of 1500 | elapsed: 10.7min finished

Out[37]: RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=500,
                            n_jobs=-1,
                            param_distributions={'criterion': ['entropy', 'gini'],
                                                 'max_depth': [10, 120, 230, 340, 450,
                                                               560, 670, 780, 890,
                                                               1000],
                                                 'max_features': ['auto', 'sqrt',
                                                                  'log2'],
                                                 'min_samples_leaf': [1, 2, 4, 6, 8],
                                                 'min_samples_split': [1, 2, 3, 4, 5, 7],
                                                 'n_estimators': [50, 155, 261, 366, 472,
                                                                  577, 683, 788, 894,
                                                                  1000]},
                            verbose=2)
```

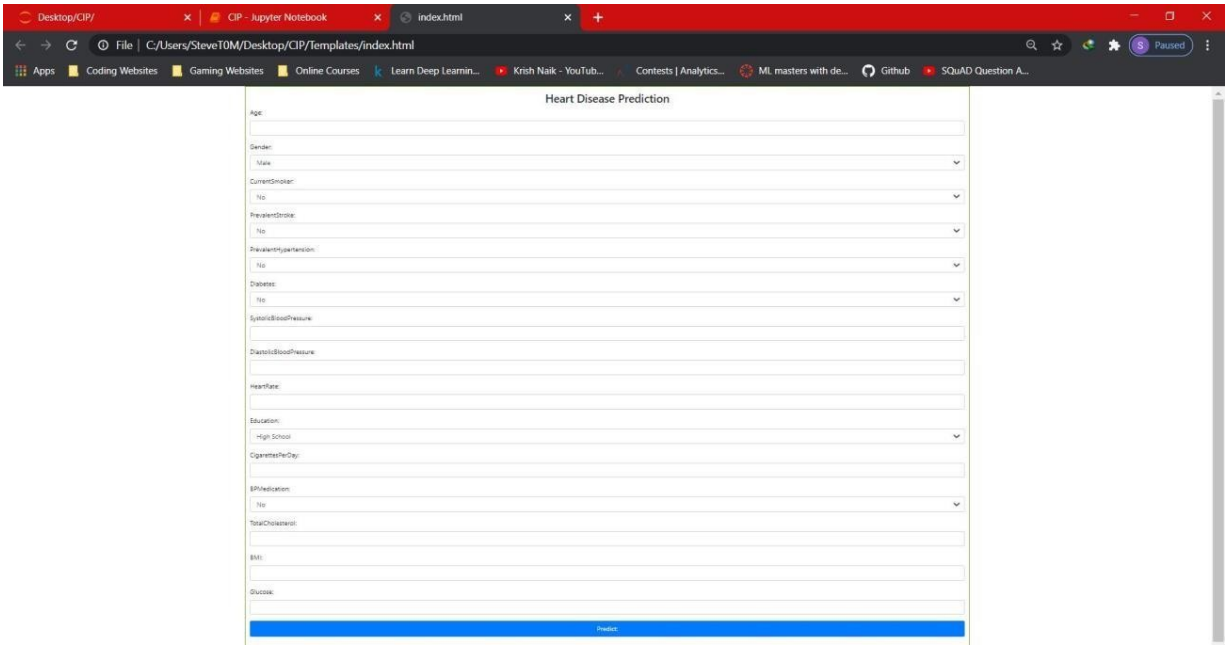
Step 13: Use the best optimal parameters fit the random forest classifier and predict for the test data.

```
In [38]: rf_randomcv.best_params_

Out[38]: {'n_estimators': 577,
          'min_samples_split': 4,
          'min_samples_leaf': 1,
          'max_features': 'auto',
          'max_depth': 780,
          'criterion': 'gini'}

In [39]: best_random_grid=rf_randomcv.best_estimator_

In [40]: y_pred=best_random_grid.predict(x_test)
         print(confusion_matrix(y_test,y_pred))
         print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))

         [[713   6]
          [124   5]]
         Accuracy Score 0.8466981132075472
```

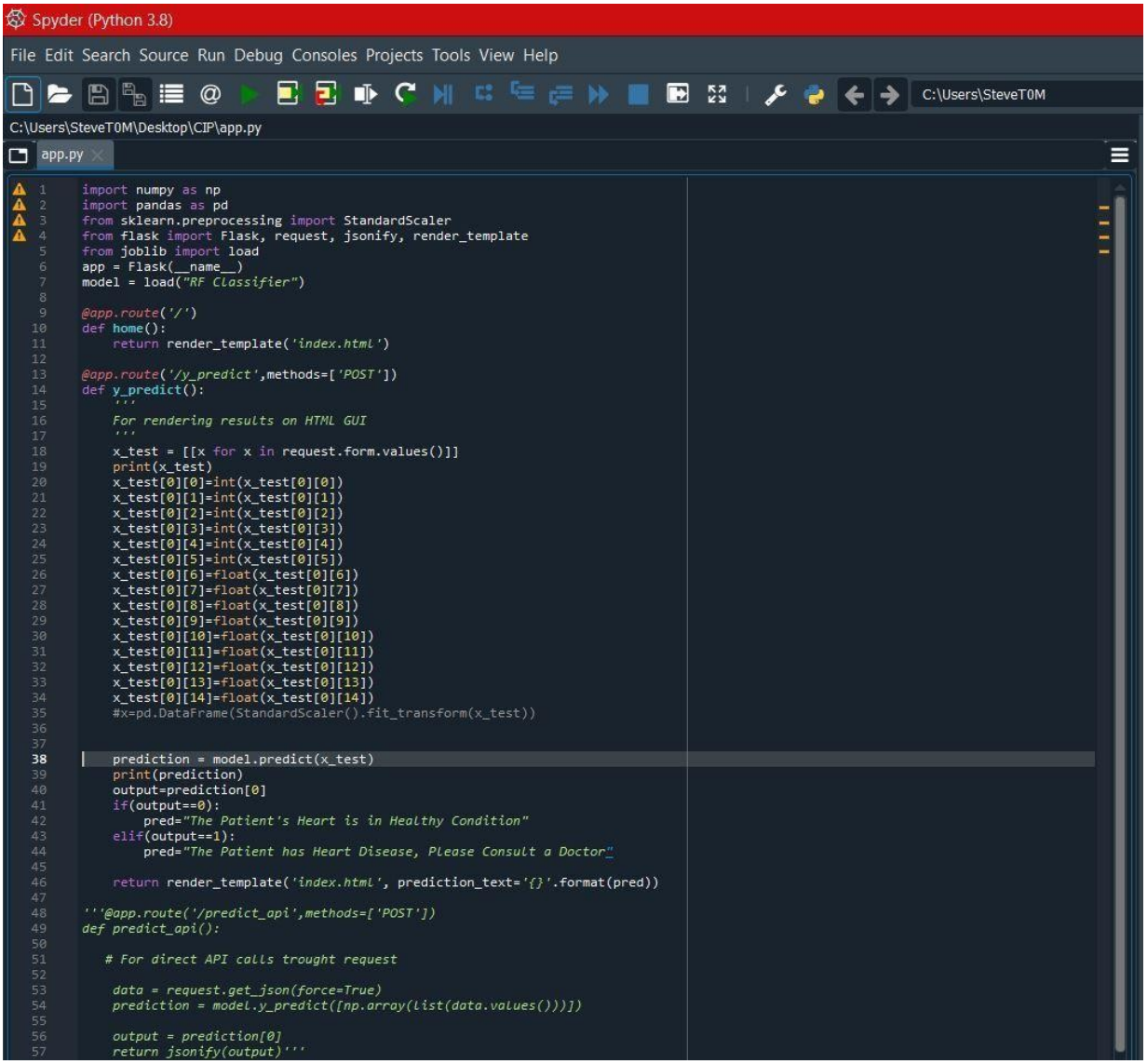
Step 14: Create an object file for using it with Front end

```
In [41]: from joblib import dump
         dump(best_random_grid,'RF Classifier')
Out[41]: ['RF Classifier']
```

Step 15: Create an html file to get the values to predict and put the file in the folder called ‘templates’ named as ‘index’.

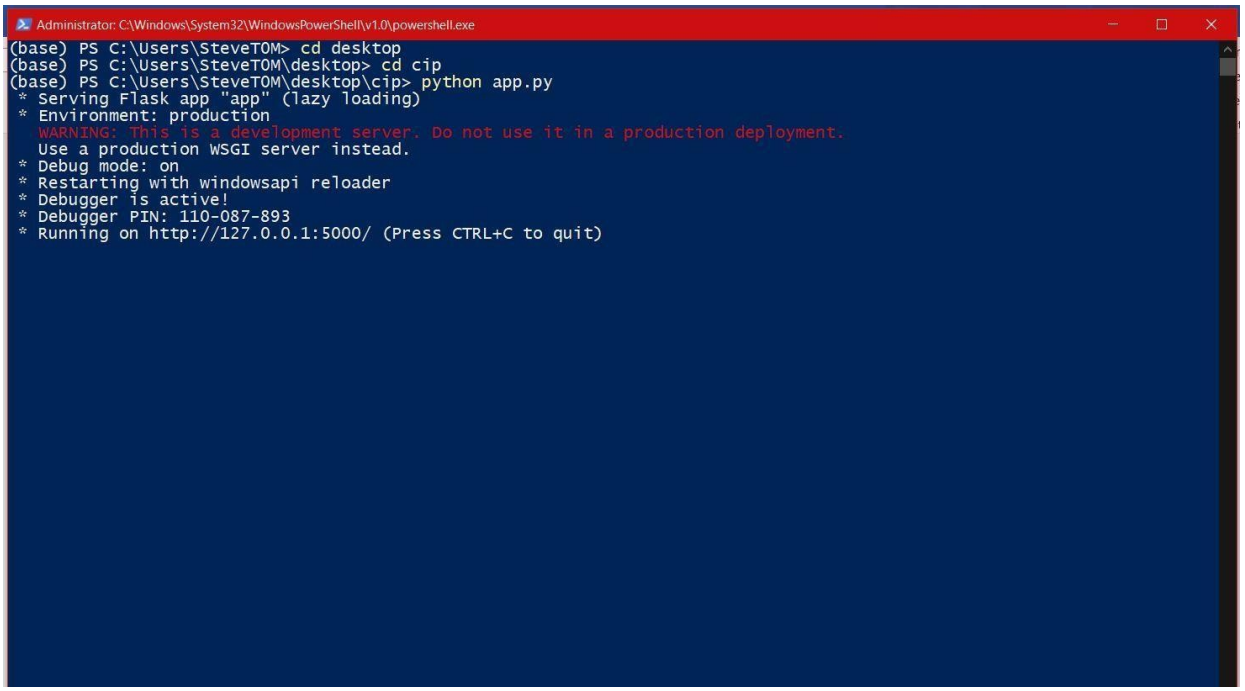


Step 16: Create a python file where we get the values from index file and predict the output.

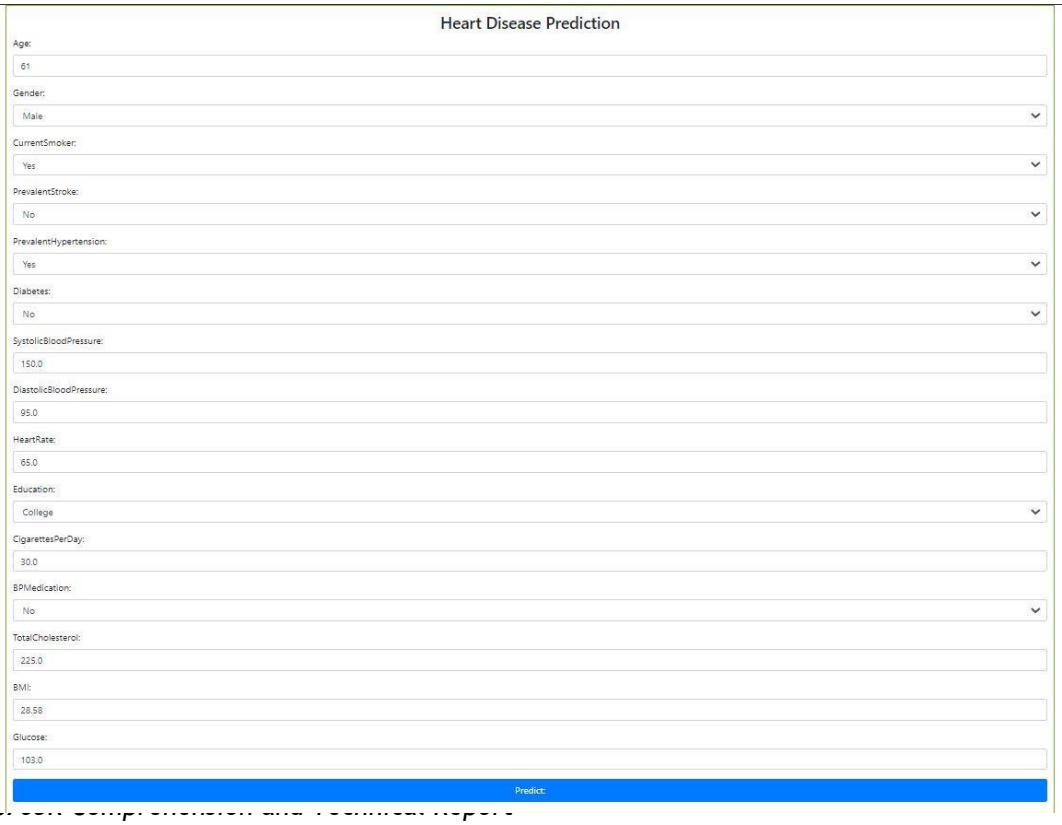


USER MANUAL

Step 1: Open Powershell prompt and traverse to the location of the python file where the Random Forest Model should also be there & Run the python file in Powershell prompt.



Step 2: Copy the URL and paste it in a browser and enter the patient details and click the predict button to get the output.



Step 3: The output will be shown below the predict button