Data Glacier

Data Science Intern at Data Glacier

Week 4: Deployment on Flask

Name: Ishwarya Arulvel

Batch Code: LISUM14

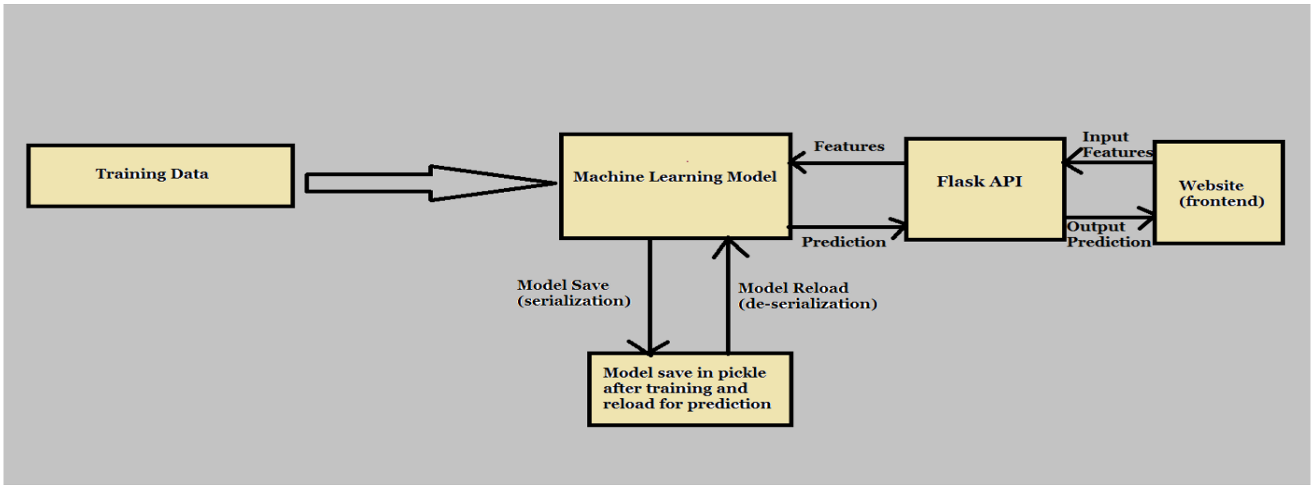
Date: 26 OCT 2022

Submitted to: Data Glacier

Table of Contents:

1. Introduction

In this project, we are going to deploying machine learning model using the Flask Framework. As a demonstration, our model helps to predict the values based on the input given in the heart disease Dataset.

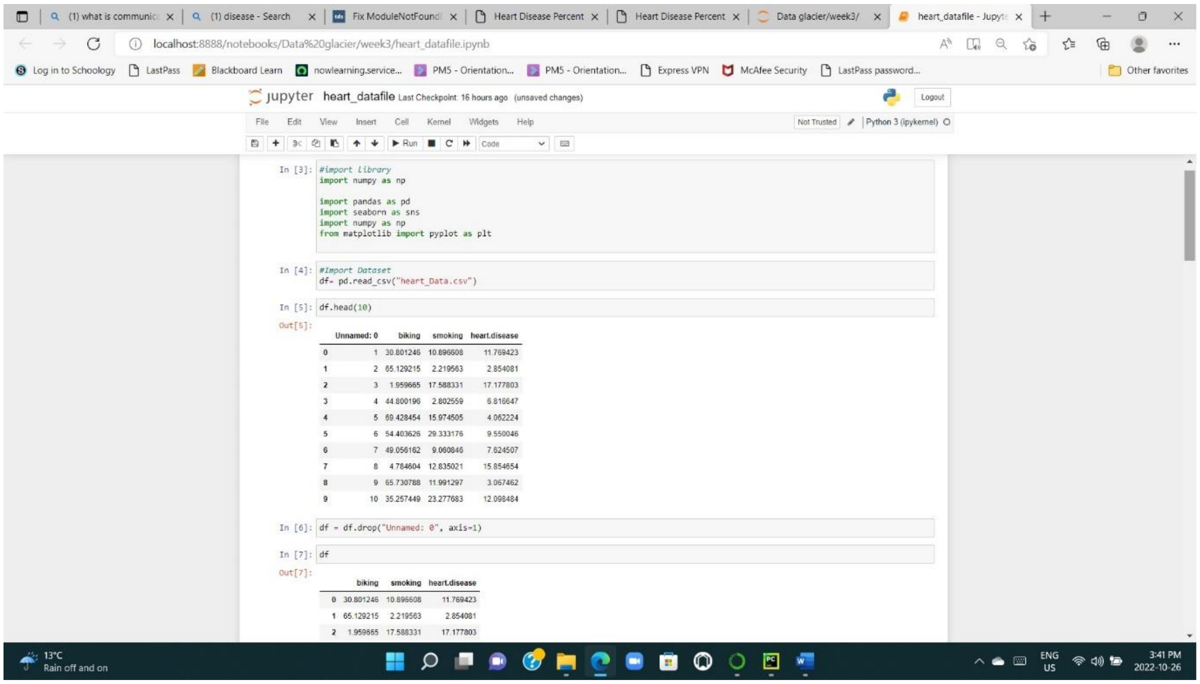

Training Data
Machine Learning Model
Features
Prediction
Flask API
Input Features
Output Prediction
Website (frontend)
Model Save (serialization)
Model Reload (de-serialization)
Model save in pickle after training and reload for prediction

We will focus on both: building a machine learning model for heart disease Dataset, then creating an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests.

2. Building a Model

2.1.1 Import Required Libraries and Dataset

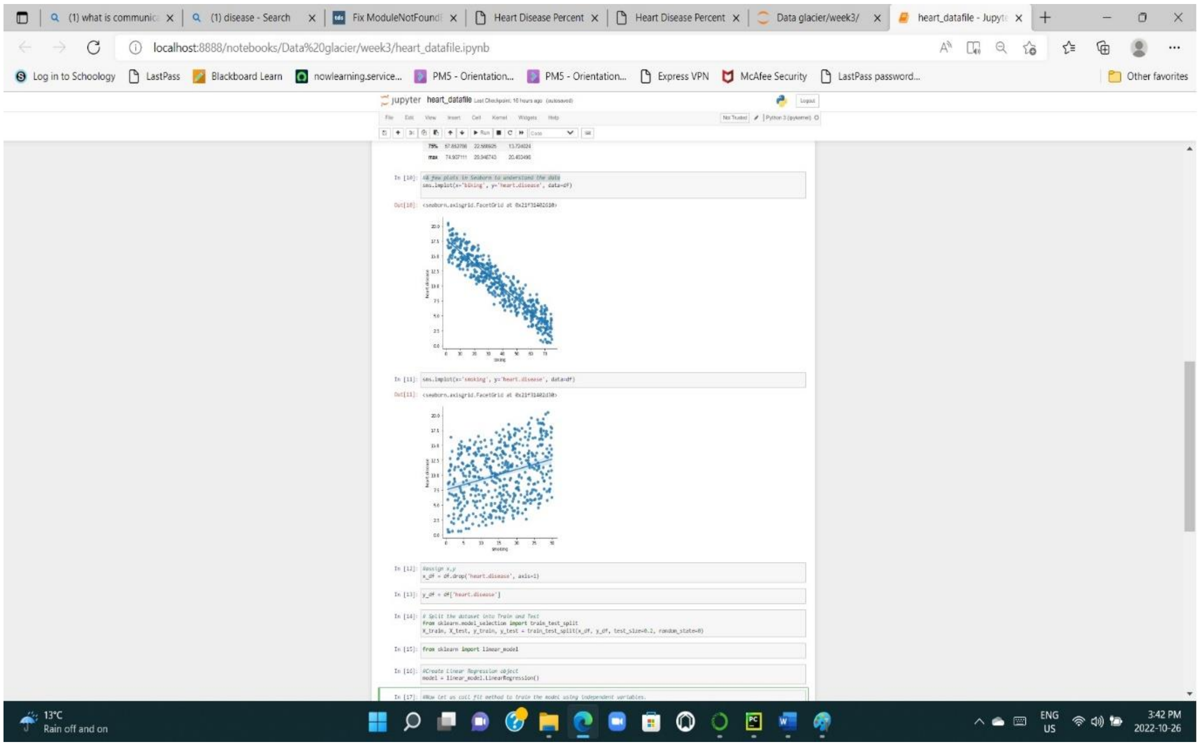
In this part, we import libraires and datasets which contain the information of heart disease dataset.



```
In [3]: #import Library
        import numpy as np

        import pandas as pd
        import seaborn as sns
        import numpy as np
        from matplotlib import pyplot as plt

In [4]: #Import Dataset
        df= pd.read_csv("heart_Data.csv")

In [5]: df.head(10)

Out[5]:
     Unnamed: 0     biking    smoking  heart.disease
  0           1  30.801246  10.896608      11.769423
  1           2  65.129215   2.219563       2.854081
  2           3   1.959665  17.588331      17.177803
  3           4  44.800196   2.802559       6.816647
  4           5  69.428454  15.974505       4.062224
  5           6  54.403626  29.333176       9.550046
  6           7  49.056162   9.060846       7.624507
  7           8   4.784604  12.835021      15.854654
  8           9  65.730788  11.991297       3.067462
  9          10  35.257449  23.277683      12.098484

In [6]: df = df.drop("Unnamed: 0", axis=1)

In [7]: df

Out[7]:
        biking    smoking  heart.disease
  0  30.801246  10.896608      11.769423
  1  65.129215   2.219563       2.854081
  2   1.959665  17.588331      17.177803
```

3.1.1 Data Preprocessing

The dataset used here is split into 80% for the training set and the remaining 20% for the test set. Use seaborn library and plots data in Seaborn to understand the data.

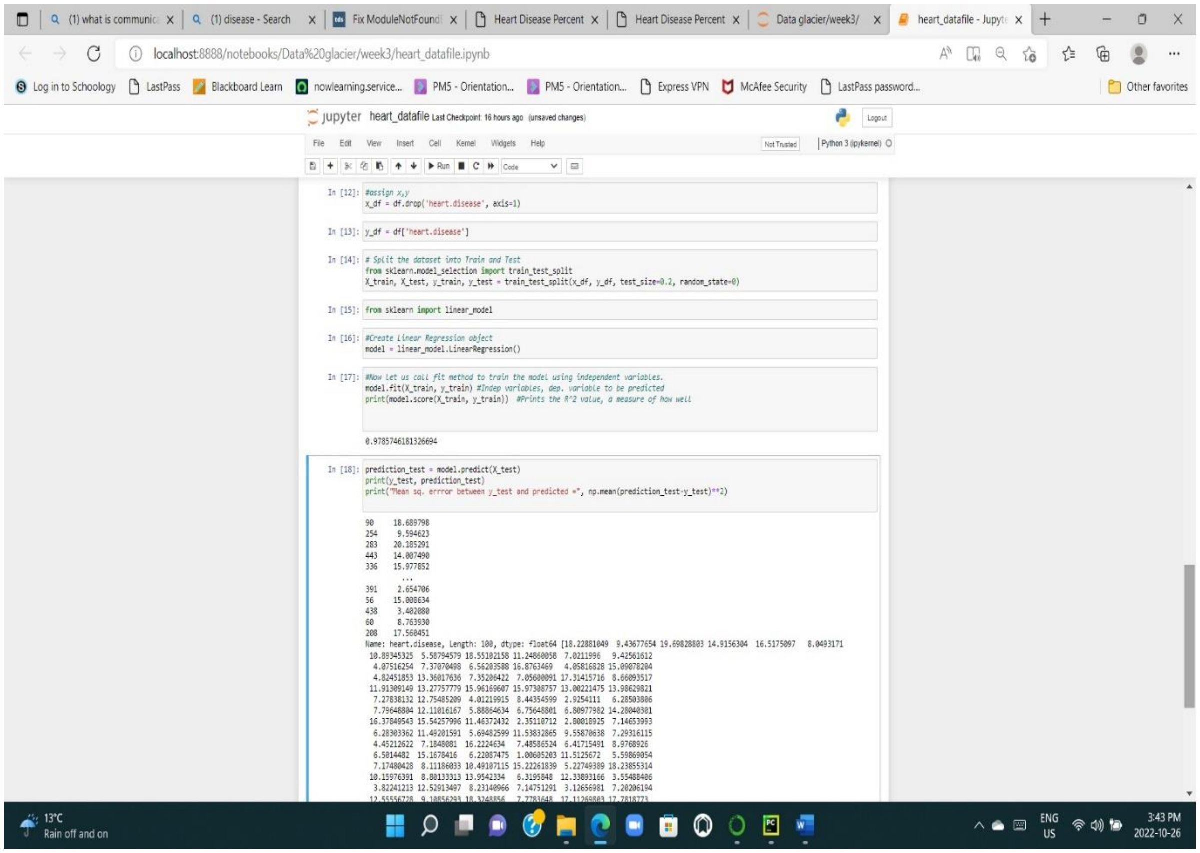


localhost:8888/notebooks/Data%20glacier/week3/heart_datafile.ipynb
jupyter heart_datafile Last Checkpoint: 16 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Logout
Run
Code
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
heart.disease
0
10
20
30
40
50
60
70
biking
0
5
10
15
20
25
30
smoking
13°C
Rain off and on
ENG
US
3:42 PM
2022-10-26

```
      75%  67.852706  22.588925  13.724024
      max  74.907111  29.946743  20.453496

In [10]: #A few plots in Seaborn to understand the data
         sns.lmplot(x='biking', y='heart.disease', data=df)

Out[10]: <seaborn.axisgrid.FacetGrid at 0x21f31482610>

In [11]: sns.lmplot(x='smoking', y='heart.disease', data=df)

Out[11]: <seaborn.axisgrid.FacetGrid at 0x21f31482d30>

In [12]: #assign x,y
         x_df = df.drop('heart.disease', axis=1)

In [13]: y_df = df['heart.disease']

In [14]: # Split the dataset into Train and Test
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=0)

In [15]: from sklearn import linear_model

In [16]: #Create Linear Regression object
         model = linear_model.LinearRegression()

In [17]: #Now let us call fit method to train the model using independent variables.
```

3.1.2 Build Model

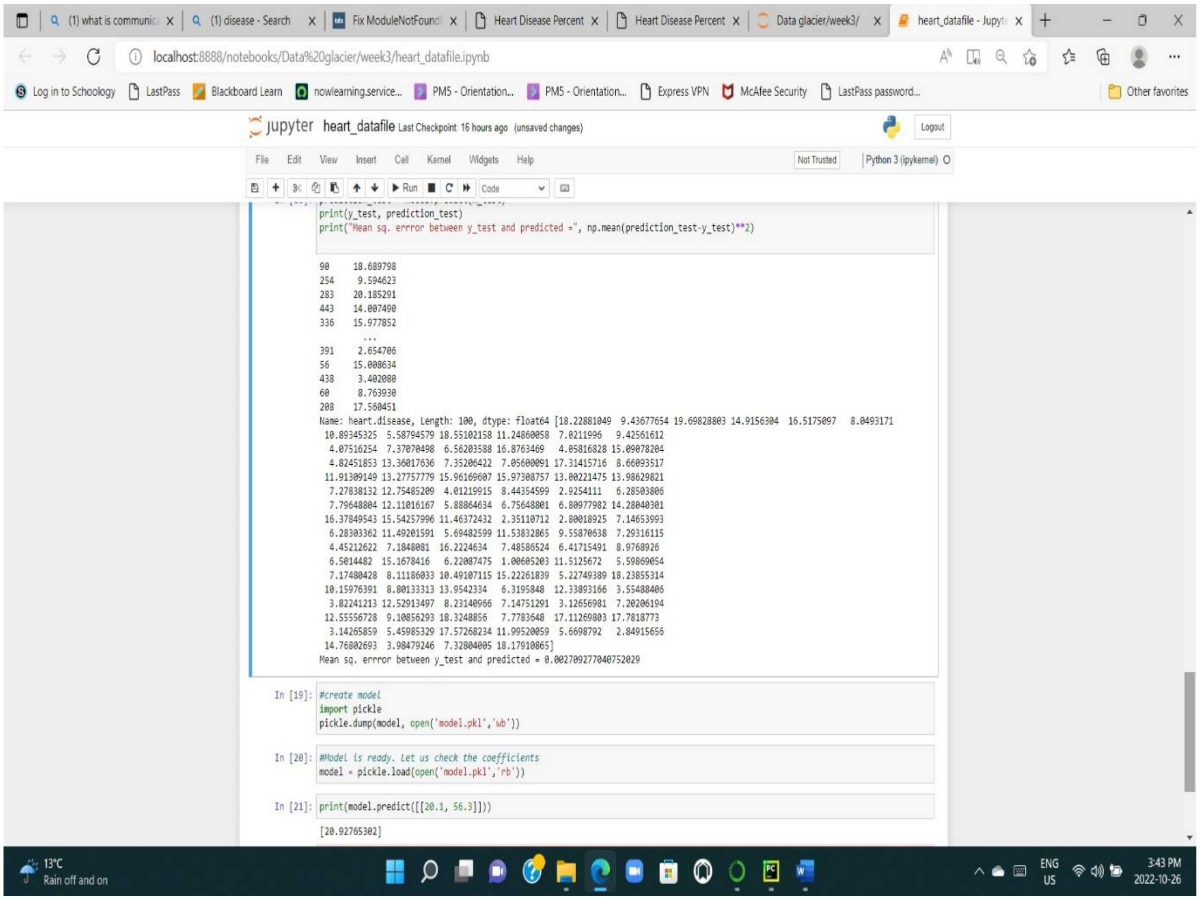
After data preprocessing, we implement machine learning model to predict the heart disease value in the dataset. For this purpose, we import sklearn. After importing and initializing linear regression we fit into the training dataset. Also, we also calculate r square value and mean square error value.



```
In [12]: #assign x,y
         x_df = df.drop('heart.disease', axis=1)

In [13]: y_df = df['heart.disease']

In [14]: # Split the dataset into Train and Test
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=0)

In [15]: from sklearn import linear_model

In [16]: #Create Linear Regression object
         model = linear_model.LinearRegression()

In [17]: #Now let us call fit method to train the model using independent variables.
         model.fit(X_train, y_train) #Indep variables, dep. variable to be predicted
         print(model.score(X_train, y_train))  #Prints the R^2 value, a measure of how well

         0.9785746181326694

In [18]: prediction_test = model.predict(X_test)
         print(y_test, prediction_test)
         print("Mean sq. errror between y_test and predicted =", np.mean(prediction_test-y_test)**2)

         90     18.689798
         254     9.594623
         283    20.185291
         443    14.007490
         336    15.977852
                  ...
         391     2.654706
         56     15.008634
         438     3.402080
         60      8.763930
         208    17.560451
         Name: heart.disease, Length: 100, dtype: float64 [18.22881049  9.43677654 19.69828803 14.9156304  16.5175097   8.0493171
          10.89345325  5.58794579 18.55102158 11.24860058  7.0211996   9.42561612
           4.07516254  7.37070498  6.56203588 16.8763469   4.05816828 15.09078204
           4.82451853 13.36017636  7.35206422  7.05600091 17.31415716  8.66093517
          11.91309149 13.27757779 15.96169607 15.97308757 13.00221475 13.98629821
           7.27838132 12.75485209  4.01219915  8.44354599  2.9254111   6.28503806
           7.79648804 12.11016167  5.88864634  6.75648801  6.80977982 14.28040301
          16.37849543 15.54257996 11.46372432  2.35110712  2.80018925  7.14653993
           6.28303362 11.49201591  5.69482599 11.53832865  9.55870638  7.29316115
           4.45212622  7.1848081  16.2224634   7.48586524  6.41715491  8.9768926
           6.5014482  15.1678416   6.22087475  1.00605203 11.5125672   5.59869054
           7.17480428  8.11186033 10.49107115 15.22261839  5.22749389 18.23855314
          10.15976391  8.80133313 13.9542334   6.3195848  12.33893166  3.55488406
           3.82241213 12.52913497  8.23140966  7.14751291  3.12656981  7.20206194
          12.55556728  9.10856293 18.3248856   7.7783648  17.11269803 17.7818773
```

3.1.3 Save the Model

After that we save our model using pickle



```
print(y_test, prediction_test)
print("Mean sq. errror between y_test and predicted =", np.mean(prediction_test-y_test)**2)
```

```
90     18.689798
254     9.594623
283    20.185291
443    14.007490
336    15.977852
         ...
391     2.654706
56     15.008634
438     3.402080
60      8.763930
208    17.560451
Name: heart.disease, Length: 100, dtype: float64 [18.22881049  9.43677654 19.69828803 14.9156304  16.5175097   8.0493171
 10.89345325  5.58794579 18.55102158 11.24860058  7.0211996   9.42561612
  4.07516254  7.37070498  6.56203588 16.8763469   4.05816828 15.09078204
  4.82451853 13.36017636  7.35206422  7.05600091 17.31415716  8.66093517
 11.91309149 13.27757779 15.96169607 15.97308757 13.00221475 13.98629821
  7.27838132 12.75485209  4.01219915  8.44354599  2.9254111   6.28503806
  7.79648804 12.11016167  5.88864634  6.75648801  6.80977982 14.28040301
 16.37849543 15.54257996 11.46372432  2.35110712  2.80018925  7.14653993
  6.28303362 11.49201591  5.69482599 11.53832865  9.55870638  7.29316115
  4.45212622  7.1848081  16.2224634   7.48586524  6.41715491  8.9768926
  6.5014482  15.1678416   6.22087475  1.00605203 11.5125672   5.59869054
  7.17480428  8.11186033 10.49107115 15.22261839  5.22749389 18.23855314
 10.15976391  8.80133313 13.9542334   6.3195848  12.33893166  3.55488406
  3.82241213 12.52913497  8.23140966  7.14751291  3.12656981  7.20206194
 12.55556728  9.10856293 18.3248856   7.7783648  17.11269803 17.7818773
  3.14265859  5.45985329 17.57268234 11.99520059  5.6698792   2.84915656
 14.76802693  3.98479246  7.32804005 18.17910865]
Mean sq. errror between y_test and predicted = 0.002709277040752029
```

```
In [19]: #create model
         import pickle
         pickle.dump(model, open('model.pkl','wb'))

In [20]: #Model is ready. Let us check the coefficients
         model = pickle.load(open('model.pkl','rb'))

In [21]: print(model.predict([[20.1, 56.3]]))

         [20.92765302]
```

4.Turning Model into Web Application

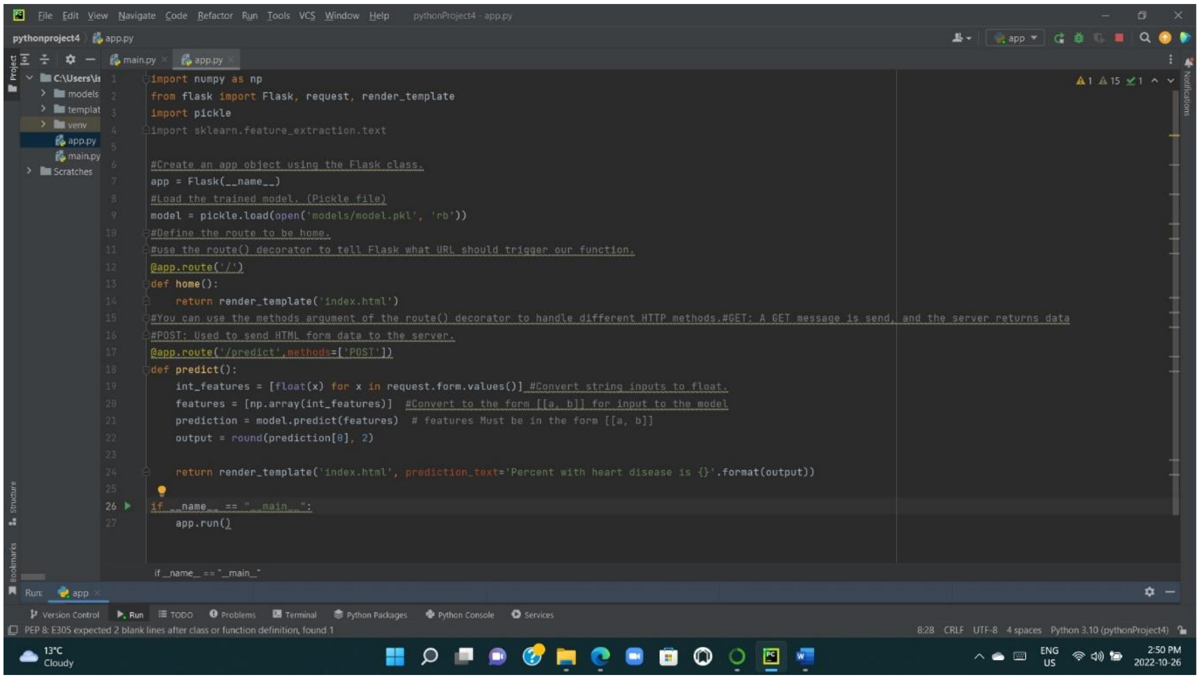
We develop a web application that consists of a simple web page with a form field with entering the values and predict. After submitting the message to the web application, it will predict the heart disease rate. First, we create a folder for this project called python project 4 , this is the directory tree inside the folder. We will explain each file.

Table 3.1: Application Folder File Directory

app.py	
Templates	**Index.html**
Model	**model.pkl**
Dataset/	**Heart_disease.csv**

3.1 App.py

The *app.py* file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SD.



```python
import numpy as np
from flask import Flask, request, render_template
import pickle
import sklearn.feature_extraction.text

#Create an app object using the Flask class.
app = Flask(__name__)
#Load the trained model. (Pickle file)
model = pickle.load(open('models/model.pkl', 'rb'))
#Define the route to be home.
#use the route() decorator to tell Flask what URL should trigger our function.
@app.route('/')
def home():
    return render_template('index.html')
#You can use the methods argument of the route() decorator to handle different HTTP methods.#GET: A GET message is send, and the server returns data
#POST: Used to send HTML form data to the server.
@app.route('/predict',methods=['POST'])
def predict():
    int_features = [float(x) for x in request.form.values()] #Convert string inputs to float.
    features = [np.array(int_features)]  #Convert to the form [[a, b]] for input to the model
    prediction = model.predict(features)  # features Must be in the form [[a, b]]
    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Percent with heart disease is {}'.format(output))

if __name__ == "__main__":
    app.run()
```

```
PEP 8: E305 expected 2 blank lines after class or function definition, found 1
```

Figure 3.1: App.py

- We ran our application as a single module; thus, we initialized a new Flask instance with the argument __*name* to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.

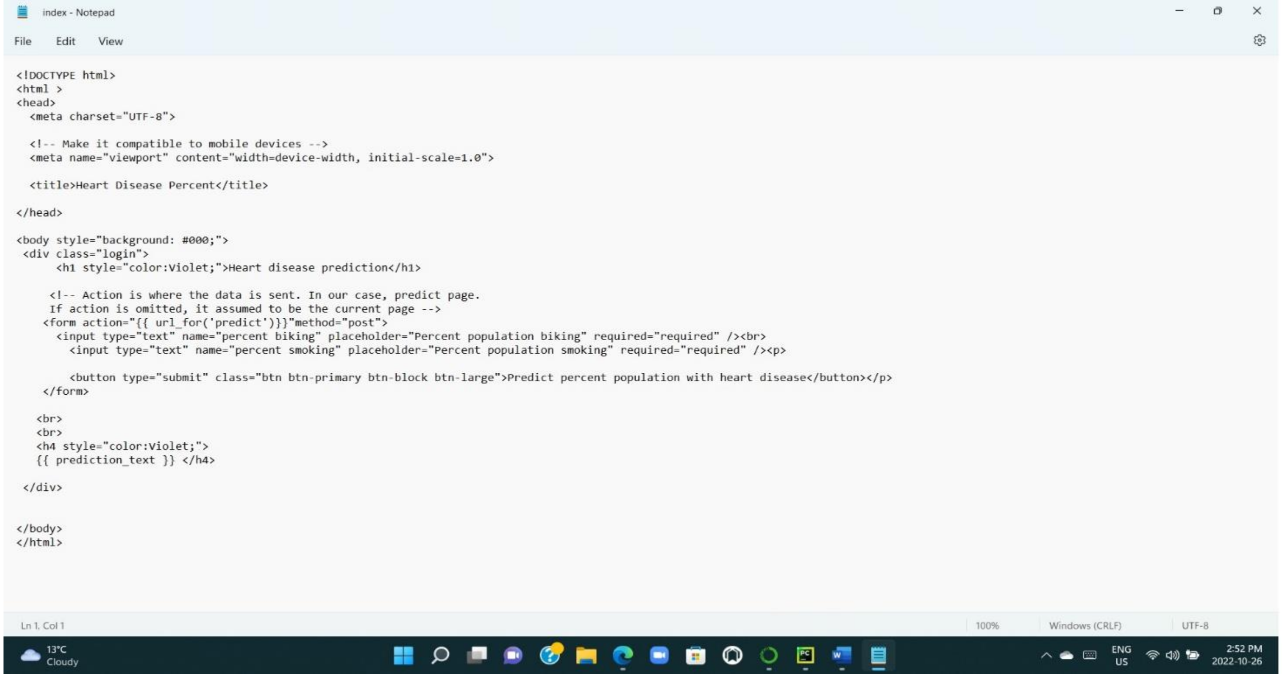
- Our *home* function simply rendered the *index.html* HTML file, which is located in the *templates* folder.

- Inside the *predict* function, we access the heart disease data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
- we used the *POST* method to transport the form data to the server in the message body.
- Lastly, we used the *run* function to only run the application on the server when this script is directly executed by the Python

interpreter, which we ensured using the *if* statement with *__name__ == '__main__'*.

- ### 3.2 Index.html

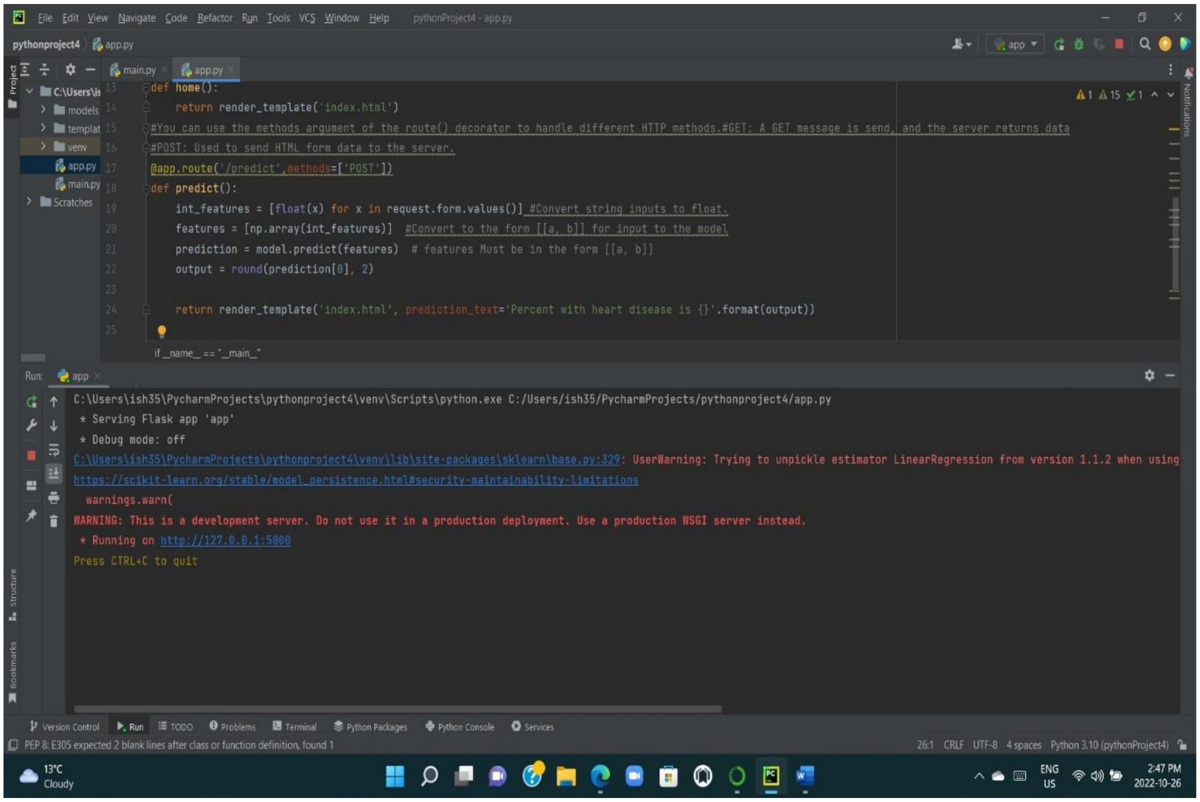
The following are the contents of the *Index.html* file that will enter the values to predict the flowers.



```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">

  <!-- Make it compatible to mobile devices -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Heart Disease Percent</title>

</head>

<body style="background: #000;">
 <div class="login">
      <h1 style="color:Violet;">Heart disease prediction</h1>

     <!-- Action is where the data is sent. In our case, predict page.
     If action is omitted, it assumed to be the current page -->
    <form action="{{ url_for('predict')}}"method="post">
      <input type="text" name="percent biking" placeholder="Percent population biking" required="required" /><br>
        <input type="text" name="percent smoking" placeholder="Percent population smoking" required="required" /><p>

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict percent population with heart disease</button></p>
    </form>

   <br>
   <br>
   <h4 style="color:Violet;">
   {{ prediction_text }} </h4>

 </div>


</body>
</html>
```

- ### 4.1.2 Running Procedure

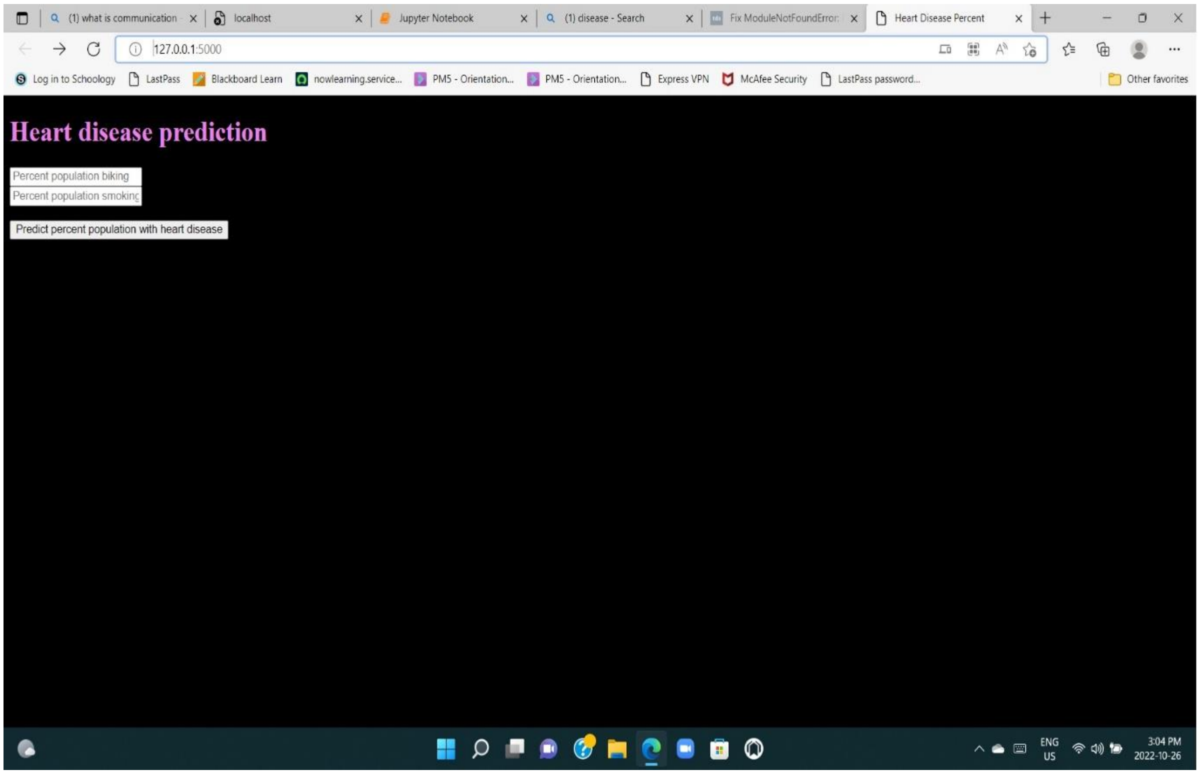
Once we have done all of the above, we can start running the API by clicking run button in *app.py* screen:



```python
def home():
    return render_template('index.html')
#You can use the methods argument of the route() decorator to handle different HTTP methods.#GET: A GET message is send, and the server returns data
#POST: Used to send HTML form data to the server.
@app.route('/predict',methods=['POST'])
def predict():
    int_features = [float(x) for x in request.form.values()] #Convert string inputs to float.
    features = [np.array(int_features)]  #Convert to the form [[a, b]] for input to the model
    prediction = model.predict(features)  # features Must be in the form [[a, b]]
    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Percent with heart disease is {}'.format(output))

if __name__ == "__main__"
```

```
C:\Users\ish35\PycharmProjects\pythonproject4\venv\Scripts\python.exe C:/Users/ish35/PycharmProjects/pythonproject4/app.py
 * Serving Flask app 'app'
 * Debug mode: off
C:\Users\ish35\PycharmProjects\pythonproject4\venv\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator LinearRegression from version 1.1.2 when using
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Now we could open a web browser and navigate to http://127.0.0.1:5000, we should see a simple website with the content like so

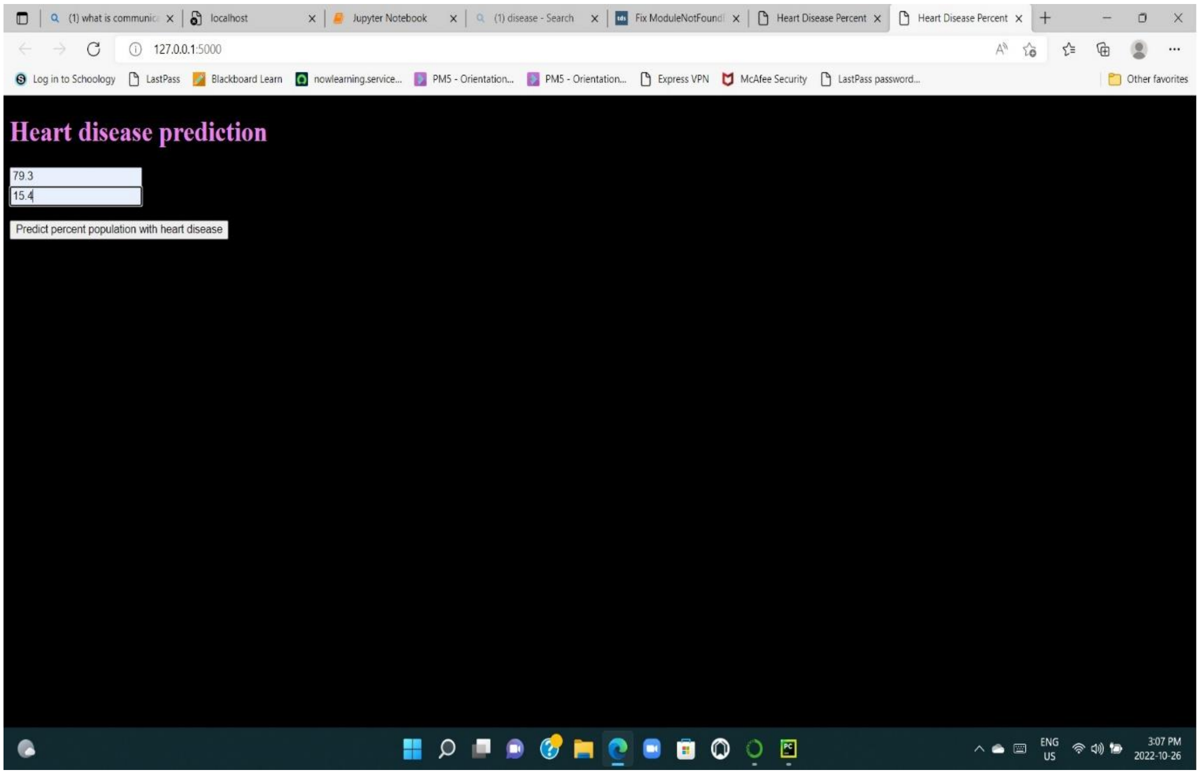


Heart disease prediction

Percent population biking

Percent population smoking

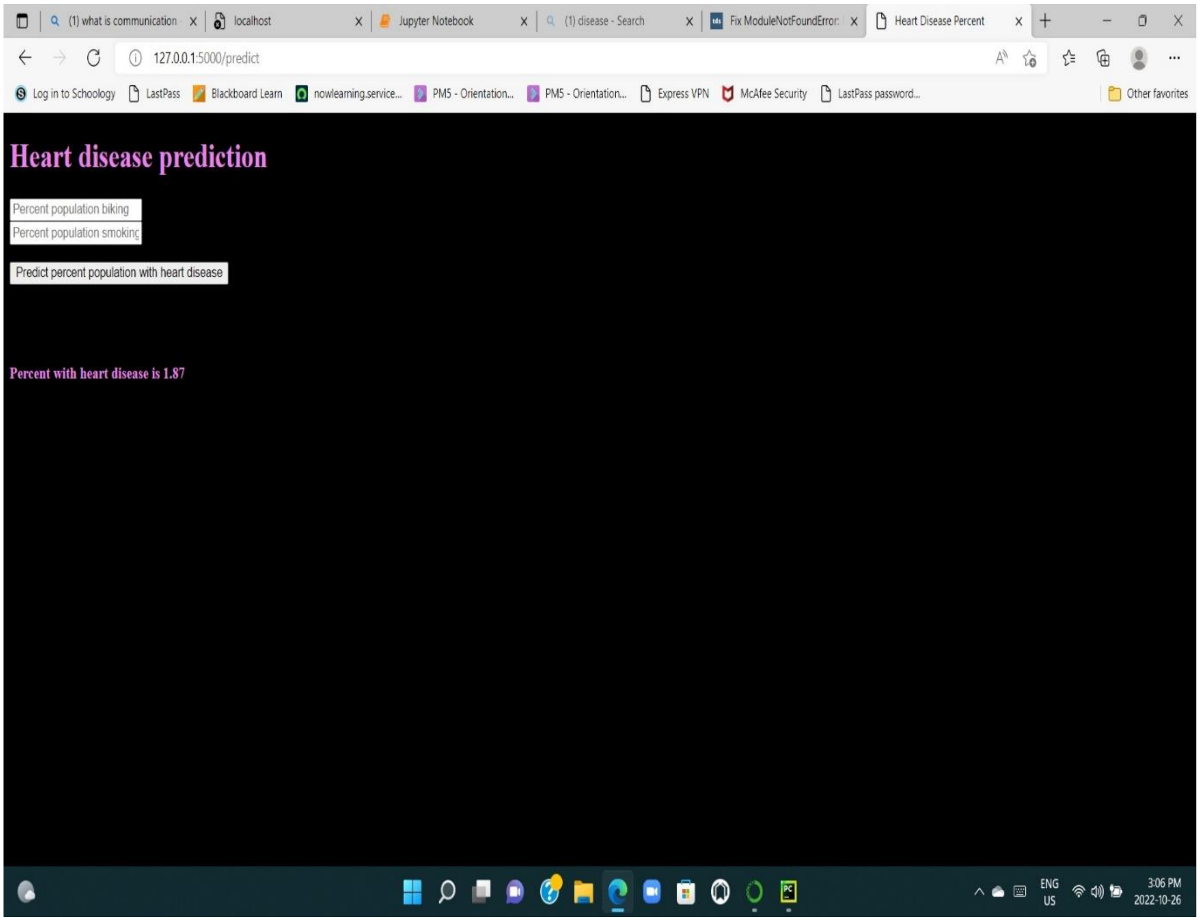
Now we enter input in the values in textbox and the value is predicted.



Heart disease prediction

79.3

15.4



Heart disease prediction

Percent population biking

Percent population smoking

Predict percent population with heart disease

Percent with heart disease is 1.87