



# **Data Science Intern at Data Glacier**

## **Week 5: Cloud API Development**

**Name:** Ishwarya Arulvel

**Batch Code:** LISUM14

**Date:** 3 NOV 2022

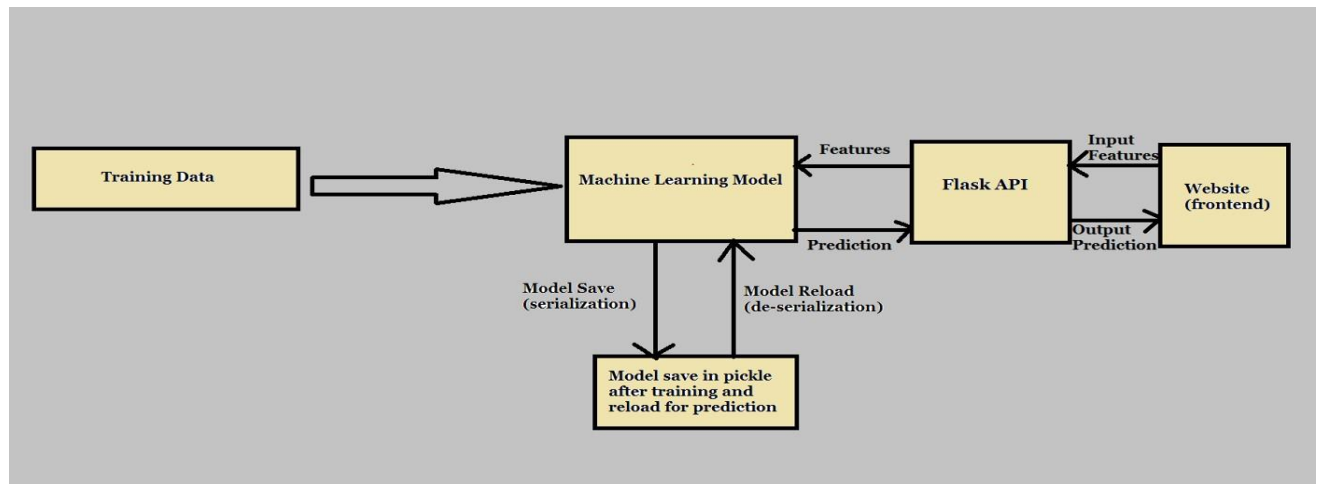
**Submitted to:** Data Glacier

### **Table of Contents:**

1. Introduction .....	3
2. Build Machine Learning Model .....	5
3. Turning Model into Flask Framework .....	6
3.1.App.py .....	7
3.2.Index.html .....	8
4.Running Procedure .....	9
5.Model Development using Heroku .....	12
5.1Steps for Model Development using Heroku .....	12

## 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.

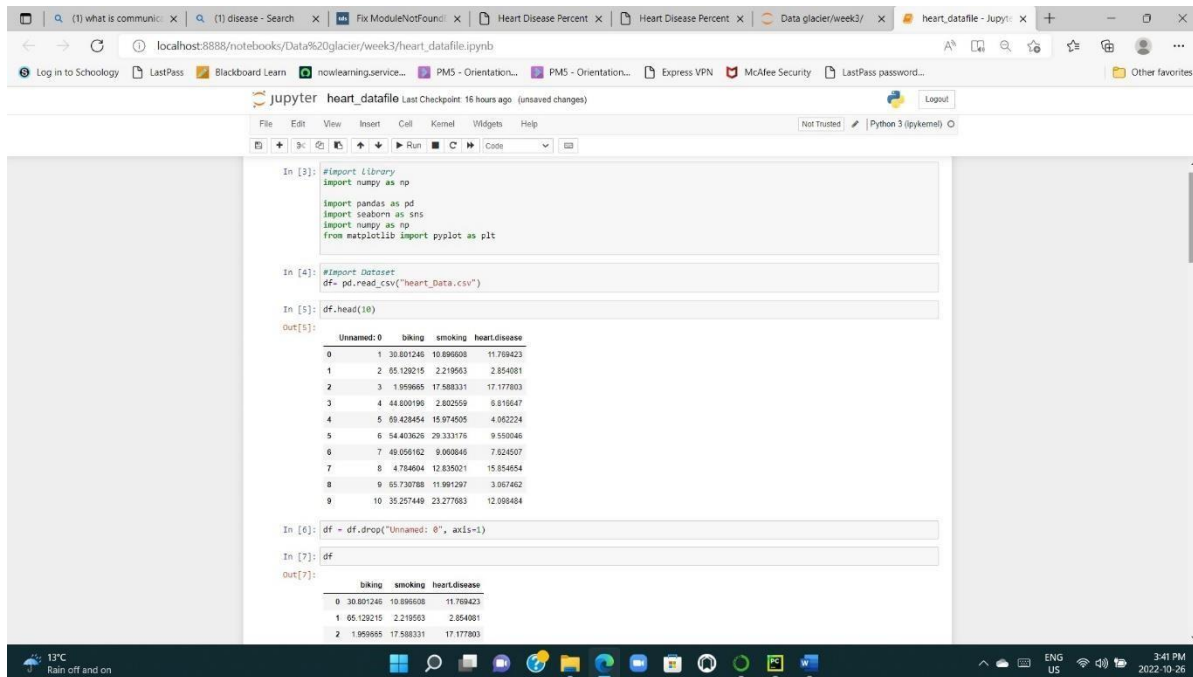


We will focus on both: building a machine learning model for heart disease Dataset, then creating an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests.

## 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.



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
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	30.801246	10.896608	11.768423
1	65.129215	2.219563	2.854081
2	1.959665	17.588331	17.177803
3	44.800196	2.802559	6.816647
4	69.428454	15.974505	4.952224
5	64.403626	29.333176	9.550046
6	49.056182	9.060646	7.824507
7	4.784604	12.835021	15.854654
8	65.730788	11.991297	3.067462
9	35.257449	23.277983	12.096464

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

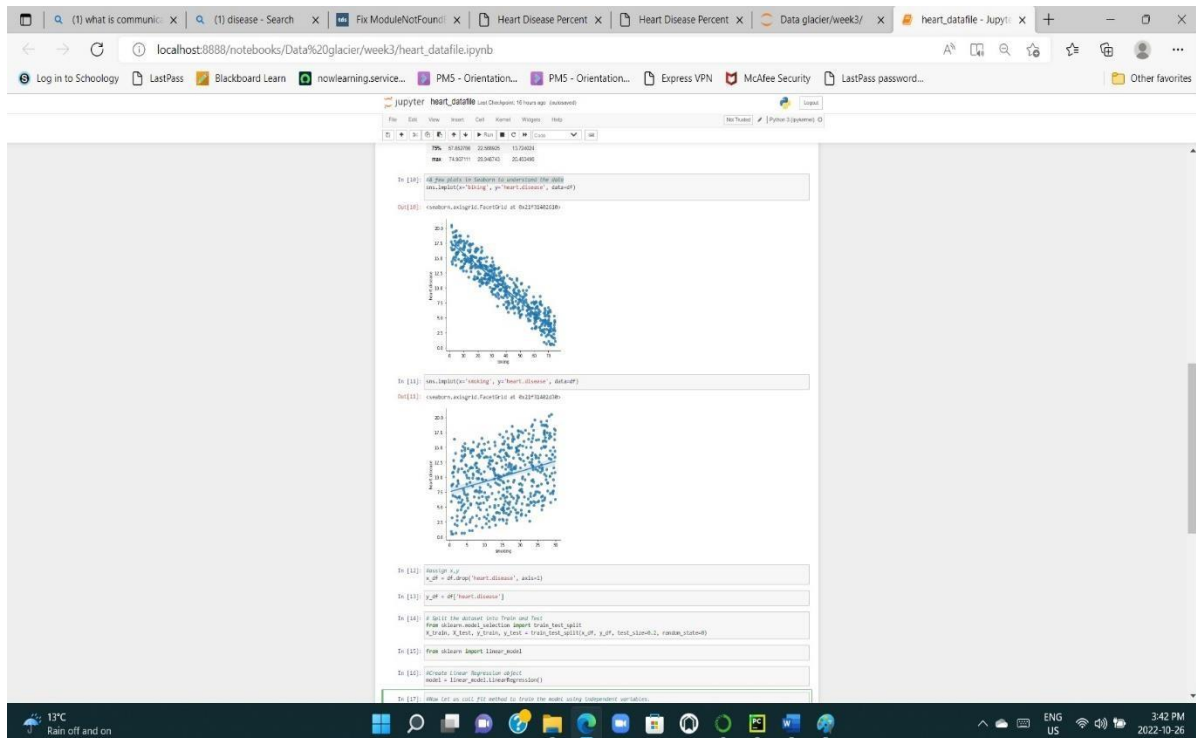
In [7]: df

Out[7]:
```

biking	smoking	heart.disease
0 30.801246	10.896608	11.768423
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.



### 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) #Hide variables, dep. variable to be predicted
print(model.score(X_train, y_train)) #Prints the R^2 value, a measure of how well

0.9785746181126684

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

90 16.689788
254 9.594623
283 20.185291
443 14.007490
336 15.977852
...
391 2.654706
56 15.006634
438 3.402800
60 8.763930
208 17.560451
Name: heart.disease, Length: 180, dtype: float64 [10.22801840 9.43677054 19.69828883 14.9156304 16.5175897 8.0493171
10.89345325 5.58794579 18.55102150 11.24868858 7.0211396 9.42561612
4.07516254 7.37870488 6.56283588 16.8763469 4.85816828 15.89878284
4.12431893 13.26817638 7.25209422 7.85608891 17.25421718 6.60893517
11.91805446 13.27157779 15.95168807 15.97380787 15.00221475 15.98620821
7.27838132 12.75485289 4.01219515 8.44354599 2.9254111 6.28503886
7.79488884 12.11816167 5.88054634 6.75648801 6.80977882 14.28040381
16.17869543 15.54257996 11.46373432 2.35110712 2.80039325 7.14653393
6.28363362 11.49201591 5.68482999 11.53832865 9.5878638 7.28316115
4.45121622 7.1848081 16.2224634 7.48058524 6.41715491 6.9788926
6.3914462 15.1678616 6.22877479 1.06045283 11.5125672 5.59689854
7.17488428 8.11186803 10.49107115 15.22261839 5.22749389 18.2385314
10.15976391 8.88133313 13.9542104 6.31295848 12.33893166 3.54884806
3.82241213 12.52913407 8.23149966 7.14751291 3.12656581 7.28208104
17.45156725 6.38816301 18.7438066 7.7833488 17.11263881 17.701871]
```

### 3.1.3 Save the Model

After that we save our model using pickle

The screenshot displays a Jupyter Notebook environment with the following content:

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

The output shows a list of predicted values for the heart disease rate, such as 18.680798, 9.594623, 20.185291, etc., followed by the mean squared error calculation.

```
Name: heart_disease, Length: 100, dtype: float64 [18.22881849  9.43677654 19.69828883 14.9156384 16.5175097  8.0493171
10.89345325  5.58794579 18.55102158 11.24860858  7.0211896  9.42561612
 4.07516254  7.37078408  6.56203588 16.8763469  4.05816828 15.09878284
 4.82451853 13.36817636  7.35206422  7.05600091 17.31415716  8.66093517
11.91309149 13.27757779 15.96169807 15.97308757 13.00221475 13.98629821
 7.27830132 12.75485209  4.01210915  8.44354599  2.9254111  6.28503086
 7.79648884 12.11016167  5.88864654  6.75648801  6.98977982 14.28840381
16.37849543 15.54257996 11.46372432  2.35110712  2.08018925  7.14653993
 6.20393362 11.49201591  5.69482599 11.53832065  9.55870638  7.29316115
 4.45212622  7.18480881 16.2224634  7.48586524  6.41715491  8.9780926
 6.5014482  15.1678416  6.22807475  1.06085203 11.5125472  5.59890854
 7.17480428  8.11186033 10.49107115 15.22261839  5.22749380 18.23855314
10.15976391  8.80133313 13.9542334  6.3195848 12.33893166  3.55488406
 3.82241213 12.52913407  8.23140966  7.14751291  3.12656981  7.26206194
12.55556728  9.10856293 18.3248856  7.7783648 17.11269803 17.7818773
 3.14265859  5.45985329 17.57268234 11.99520059  5.6698792  2.84915656
14.76882693  3.98479246  7.32804805 18.17910865]
Mean sq. error between y_test and predicted = 0.062789277040752029
```

```
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]]))

[20.92765382]
```

## 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

<b>app.py</b>	
<b>Templates</b>	<b>Index.html</b>
<b>Model</b>	<b>model.pkl</b>
<b>Dataset/</b>	<b>Heart_disease.csv</b>

### 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.

```
1 import numpy as np
2 from flask import Flask, request, render_template
3 import pickle
4 import sklearn.feature_extraction.text
5
6 #Create an app object using the Flask class.
7 app = Flask(__name__)
8 #Load the trained model. (Pickle file)
9 model = pickle.load(open('models/model.pkl', 'rb'))
10 #Define the route to be home.
11 #Use the route() decorator to tell Flask what URL should trigger our function.
12 @app.route('/')
13 def home():
14     return render_template('index.html')
15 #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
16 #POST: Used to send HTML form data to the server.
17 @app.route('/predict', methods=['POST'])
18 def predict():
19     int_features = [float(x) for x in request.form.values()] #Convert string inputs to float.
20     features = [np.array(int_features)] #Convert to the form [[a, b]] for input to the model
21     prediction = model.predict(features) # features must be in the form [[a, b]]
22     output = round(prediction[0], 2)
23
24     return render_template('index.html', prediction_text='Percent with heart disease is {}'.format(output))
25
26 if __name__ == '__main__':
27     app.run()
```

Figure 3.1: App.py

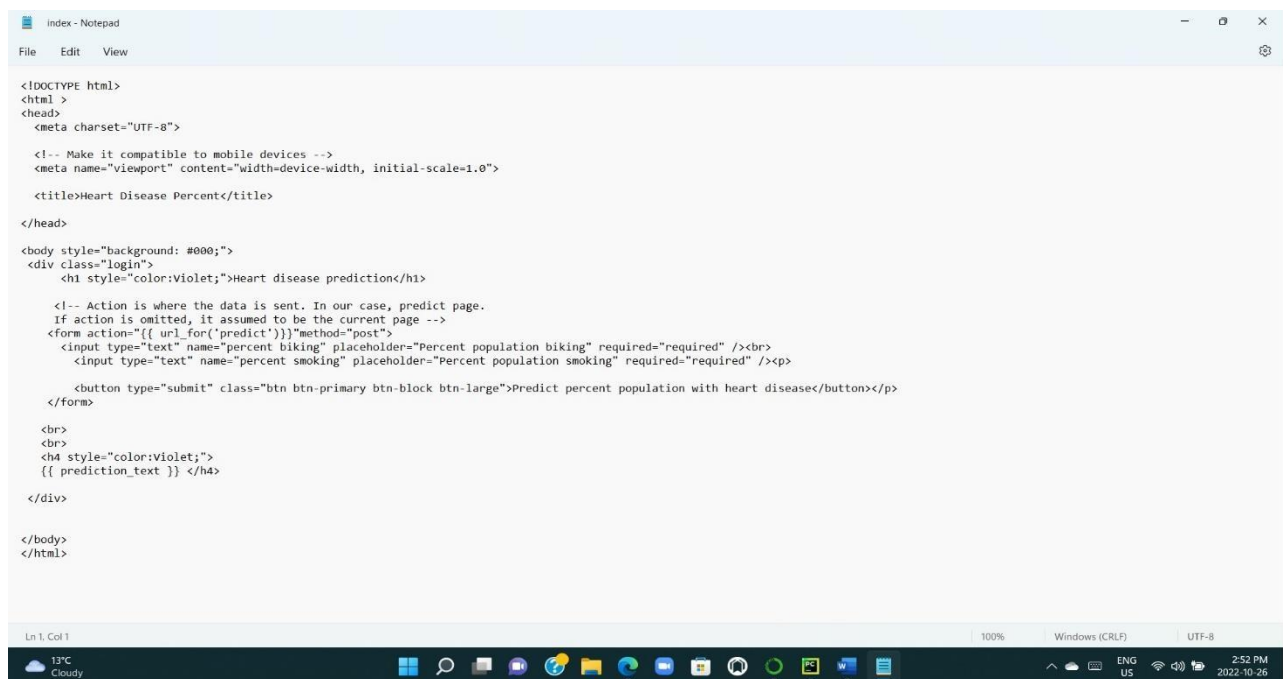
- We ran our application as a single module; thus, we initialized a new Flask instance with the argument `__name__` to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.
- Our *home* function simply rendered the *index.html* HTML file, which 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>

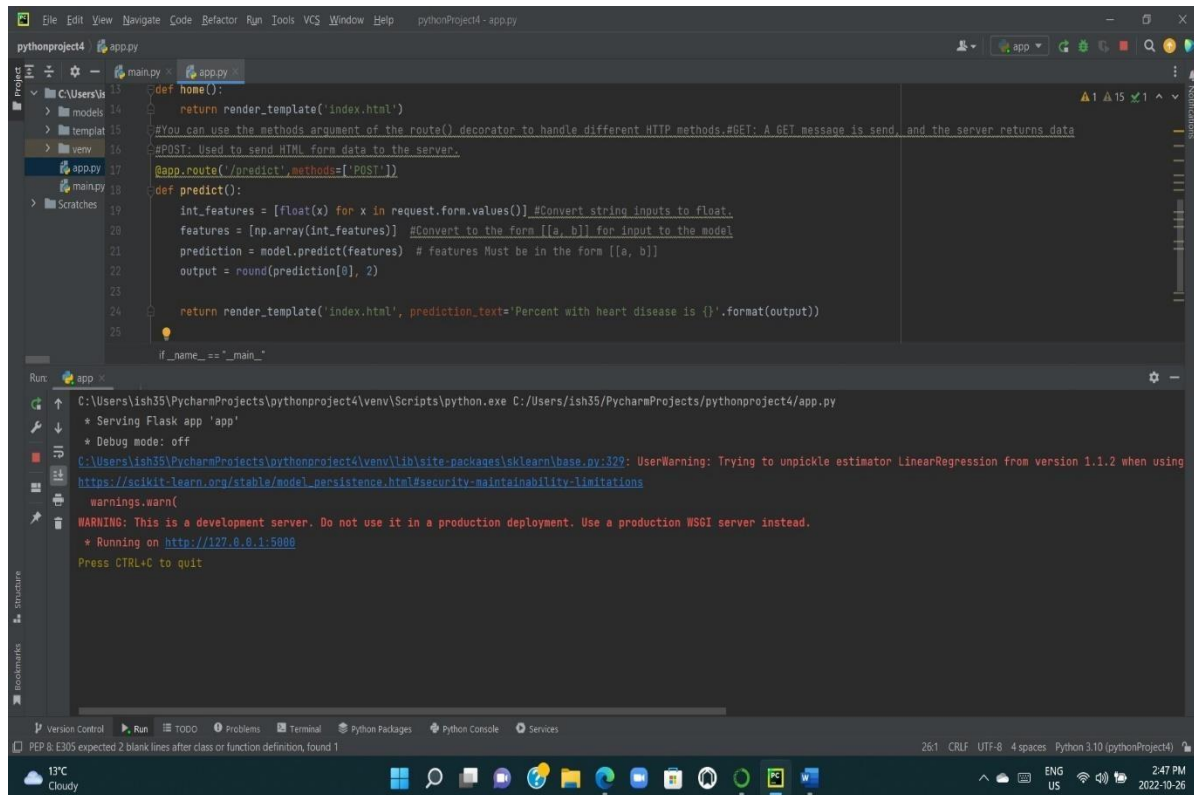
<!-- Action is where the data is sent. In our case, predict page.
If action is omitted, it assumed to be the current page -->
<form action="{{ url_for('predict')}}"method="post">
<input type="text" name="percent biking" placeholder="Percent population biking" required="required" /><br>
<input type="text"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

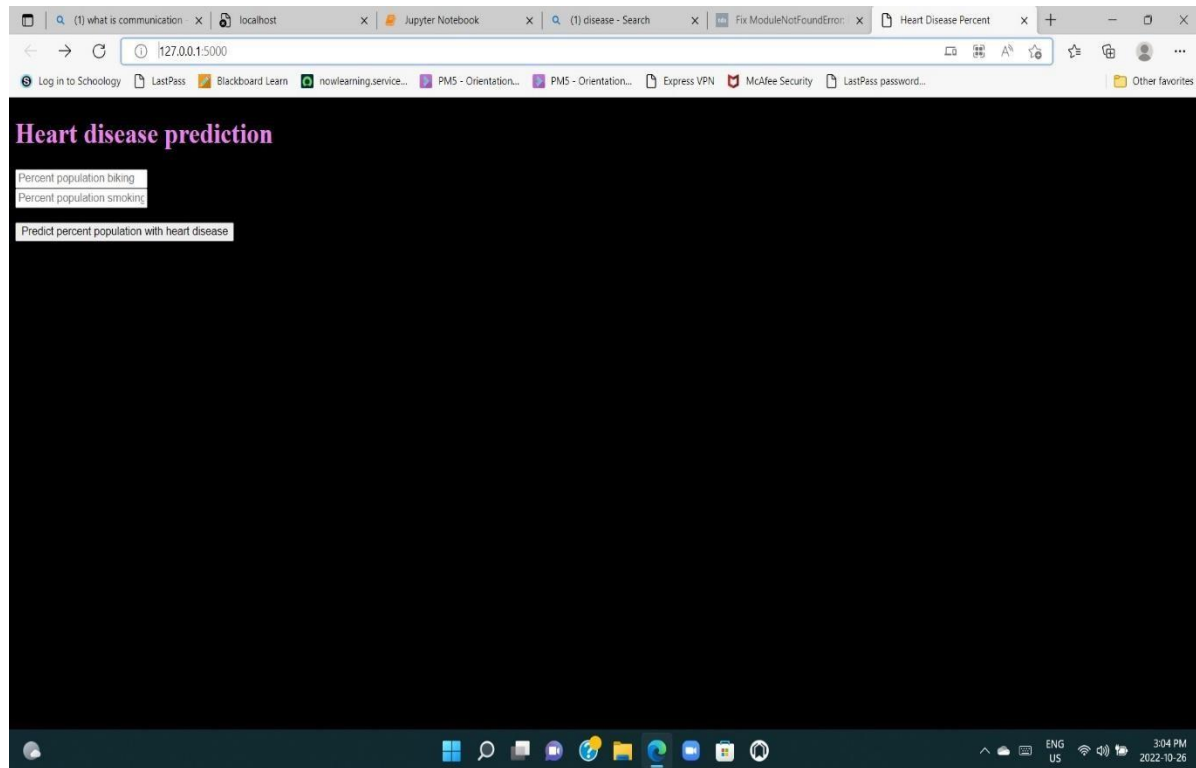
</body>
</html>
```

- **4.1.2 Running Procedure**

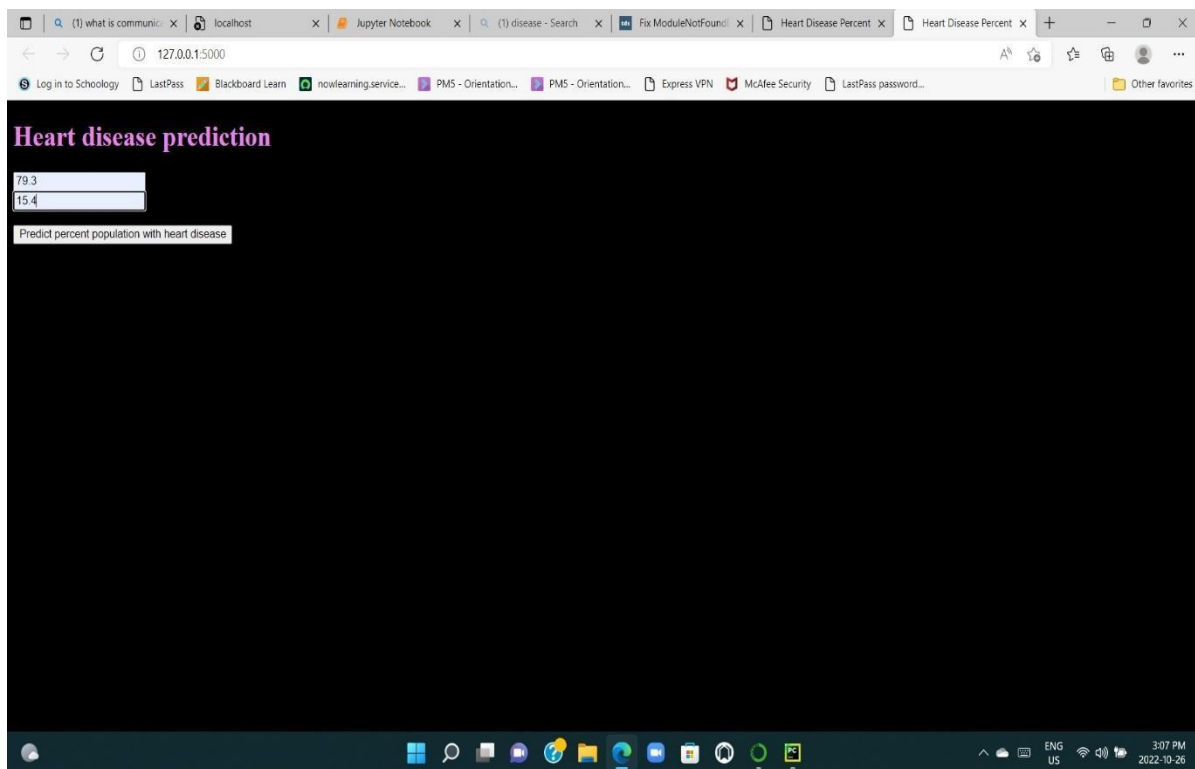
Once we have done all of the above, we can start running the API by clicking run button in *app.py* screen:



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



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



Heart disease prediction

Percent population biking

Percent population smoking

Predict percent population with heart disease

Percent with heart disease is 1.87

## **5.Model deployment using Heroku**

We're ready to start our Heroku deployment now that our model has been trained, the machine learning pipeline has been set up, and the application has been tested locally. There are a few ways to upload the application source code onto Heroku. The easiest way is to link a GitHub repository to your Heroku account.

### **Requirement.txt**

It is a text file containing the flask, NumPy, request, pickles, kit-learn library required to execute the application.

### **Runtime.txt**

It is a text file containing the python packages required to execute the application.

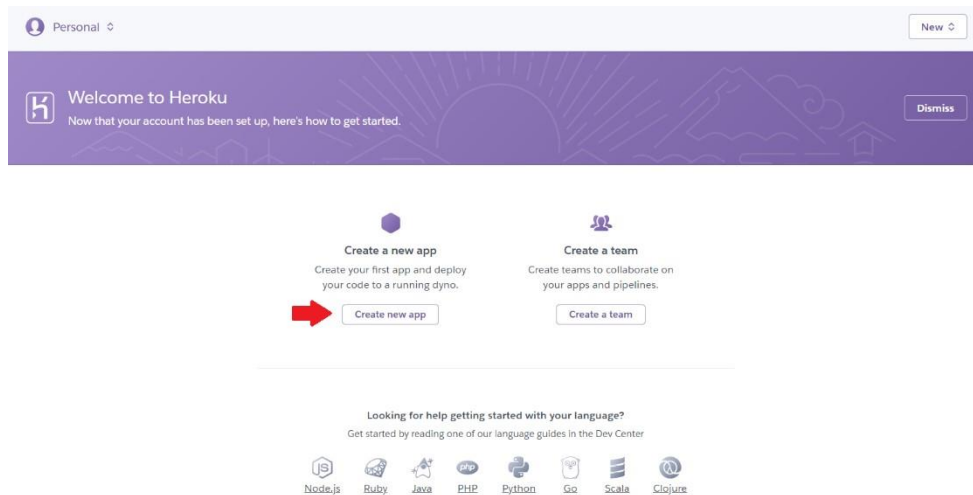
### **Procfile**

Procfile is a file that specifies the commands that are executed by an Heroku app on Startup.

## **5.1Steps for Model Deployment Using Heroku**

Once we uploaded files to the GitHub repository, we are now ready to start deployment on Heroku. Follow the steps below:

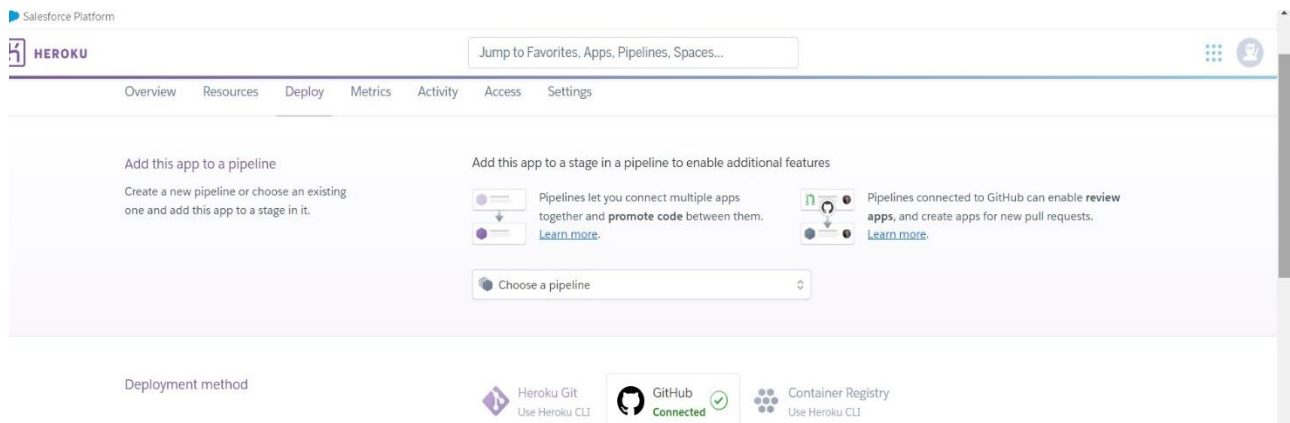
1. After sign up on **heroku.com** then click on **Create new app**.



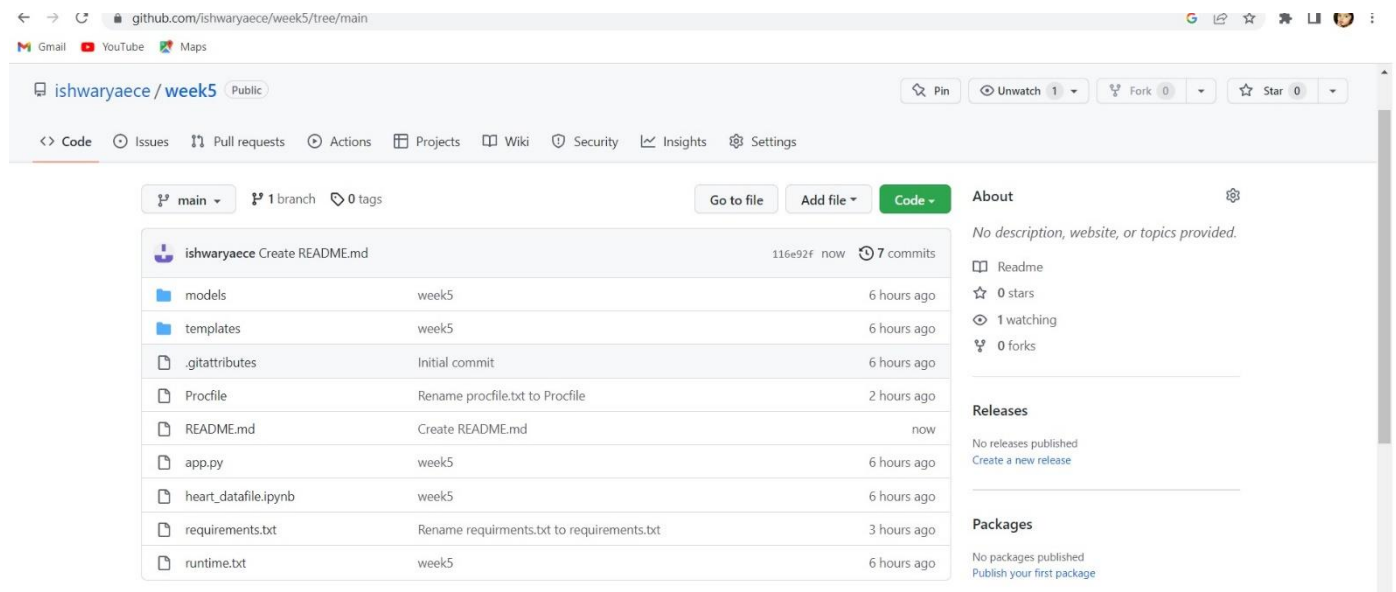
## 2. Enter App name and region

The image shows the 'Create New App' form in the Heroku dashboard. The form has a header 'Create New App' and a search bar 'Jump to Favorites, Apps, Pipelines, Spaces...'. The main form fields are 'App name' and 'Choose a region'. The 'App name' field contains the text 'heart'. The 'Choose a region' dropdown menu is set to 'United States'. Below these fields are two buttons: 'Add to pipeline...' and 'Create app'.

2. Connect to GitHub repository where code is I uploaded.



After that I choose the repository where I upload the code.



Deploy branch

heroku

Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

App connected to GitHub  
Code diffs, manual and auto deploys are available for this app.

Connected to [ishwaryaece/week5](#) by [ishwaryaece](#) [Disconnect...](#)

Releases in the [activity feed](#) link to GitHub to view commit diffs

Automatic deploys  
Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from 'master' to 'main' for both manual and automatic deploys, please follow the instructions [here](#).

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#)

Choose a branch to deploy

main

☐ Wait for CI to pass before deploy  
Only enable this option if you have a Continuous Integration service configured on your repo.

**Enable Automatic Deploys**

Manual deploy  
Deploy the current state of a branch to this app.

Deploy a GitHub branch  
This will deploy the current state of the branch you specify below. [Learn more](#)

Choose a branch to deploy

main **Deploy Branch**

After waiting 5 to 15 minutes our application is Ready

Manual deploy  
Deploy the current state of a branch to this app.

Deploy a GitHub branch  
This will deploy the current state of the branch you specify below. [Learn more](#)

Choose a branch to deploy

main **Deploy Branch**

Receive code from GitHub ☒

Build main `ff1c93bf` ☒

Release phase ☒

Deploy to Heroku ☒

Your app was successfully deployed.

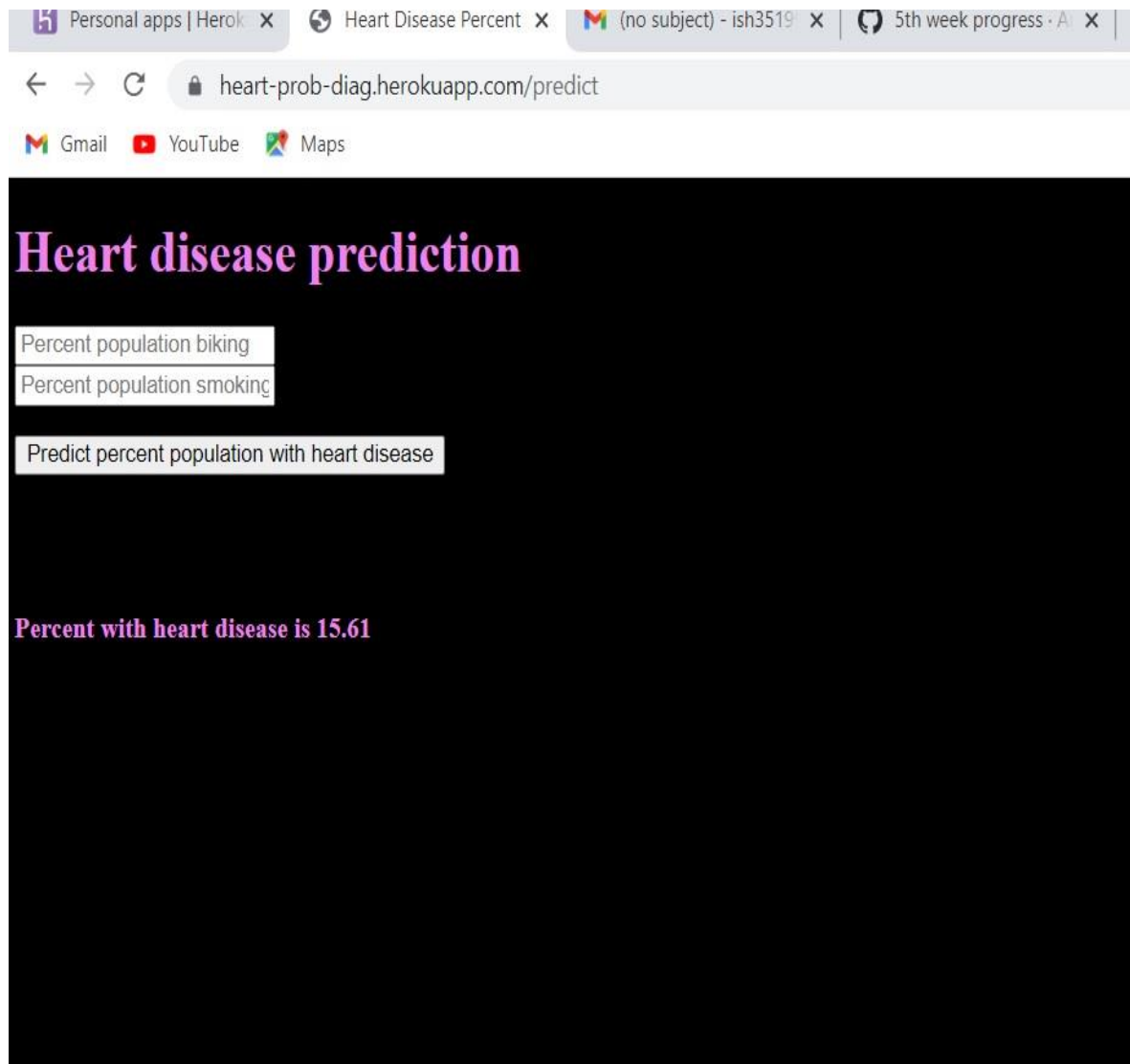
[View](#)

heroku.com Blogs Careers Documentation **Support**

Terms of Service Privacy Cookies © 2022 Salesforce.com

Finally click view then we get the expected result





The app is published at <https://heart-prob-diag.herokuapp.com/predict>