Data Science Intern AT Data Glacier

Week 4: Deployment on Flask

Name: Rayan Yassminh

Batch Code: LISUM13: 30

Summitted to: Data Glacier

Deployment on Flask

Table of Contents:

1.Introduction

In this project I will deploy the Logistic regression model using Flask. My ML model predicts if someone is diabetic or not. The goal of this project is to create API for the ML model using Flask. Flask is a framework for building a web application.

2.Data information

Researchers at the Bio-Solutions lab want to get better understanding of diabetes disease among women and are planning to use machine learning models that will help them to identify patients who are at risk of diabetes. The "Pima Indians Diabetes.csv" dataset was collected. All patients here are females at least 21 years old of Pima Indian heritage.
Our primarily ML model shows the most important features to make the prediction; Therefore, I filtered the dataset to include only the important features then we rebuild the model on the filtered Dataset.

Pregnancies; 768 non-null		**int64**
Glucose :	**768 non-null**	**float64**
BMI :	**768 non-null**	**float64**
Pedigree:	**768 non-null**	**float64**
Class :	**768 non-null**	**int64**
dtypes:	**float64(3), int64(2)**	
memory usage: 30.1 KB		

Data Description:

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test
- BMI: Body mass index (weight in kg/(height in m)^2)
- Pedigree: Diabetes pedigree function - A function that scores likelihood of diabetes based on family history.
- Class: Class variable (0: the person is not diabetic or 1: the person is diabetic)

Index	*Pregnancies*	*Glucose*	*BMI*	*Pedigree*	*Class*
566	*1*	*99.0*	*38.6*	*0.412*	*0*
207	*5*	*162.0*	*37.7*	*0.151*	*1*
58	*0*	*146.0*	*40.5*	*1.781*	*0*
686	*3*	*130.0*	*23.1*	*0.314*	*0*
751	*1*	*121.0*	*39.0*	*0.261*	*0*

3. The Machine learning model

Import the libraries:

```
# To filter the warnings
import warnings
warnings.filterwarnings("ignore")
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np
# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Library to split data
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# To build linear model for statistical analysis and prediction
# To get diferent metric scores
from sklearn import metrics
from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix, roc_auc_score,recall_score
import pickle
```

Read the dataset

```
data = pd.read_csv("pima-indians-diabetes.csv")
# defining the most important columns and replace 0 with NaN
cols = ["Glucose", "BMI", "Pedigree"]
# replacing 0 with NaN
data[cols] = data[cols].replace(0, np.nan)
# Let's impute missing values using mean value
data[cols] = data[cols].fillna(data[cols].mean())
#Target and independent features.
X = data.drop(["Class"], axis=1)
Y = data["Class"]
# split data into training and test
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1,stratify=Y)
```

Logistic Regression model

```
lg=LogisticRegression(penalty='elasticnet',l1_ratio=0.9,max_iter=500,solver='saga',class_weight='balan
ced')
lg.fit(X_train,y_train)
# convert the model pickle.
pickle.dump(lg, open('model.pkl','wb'))
```

4. Applying ML Model to flask framework

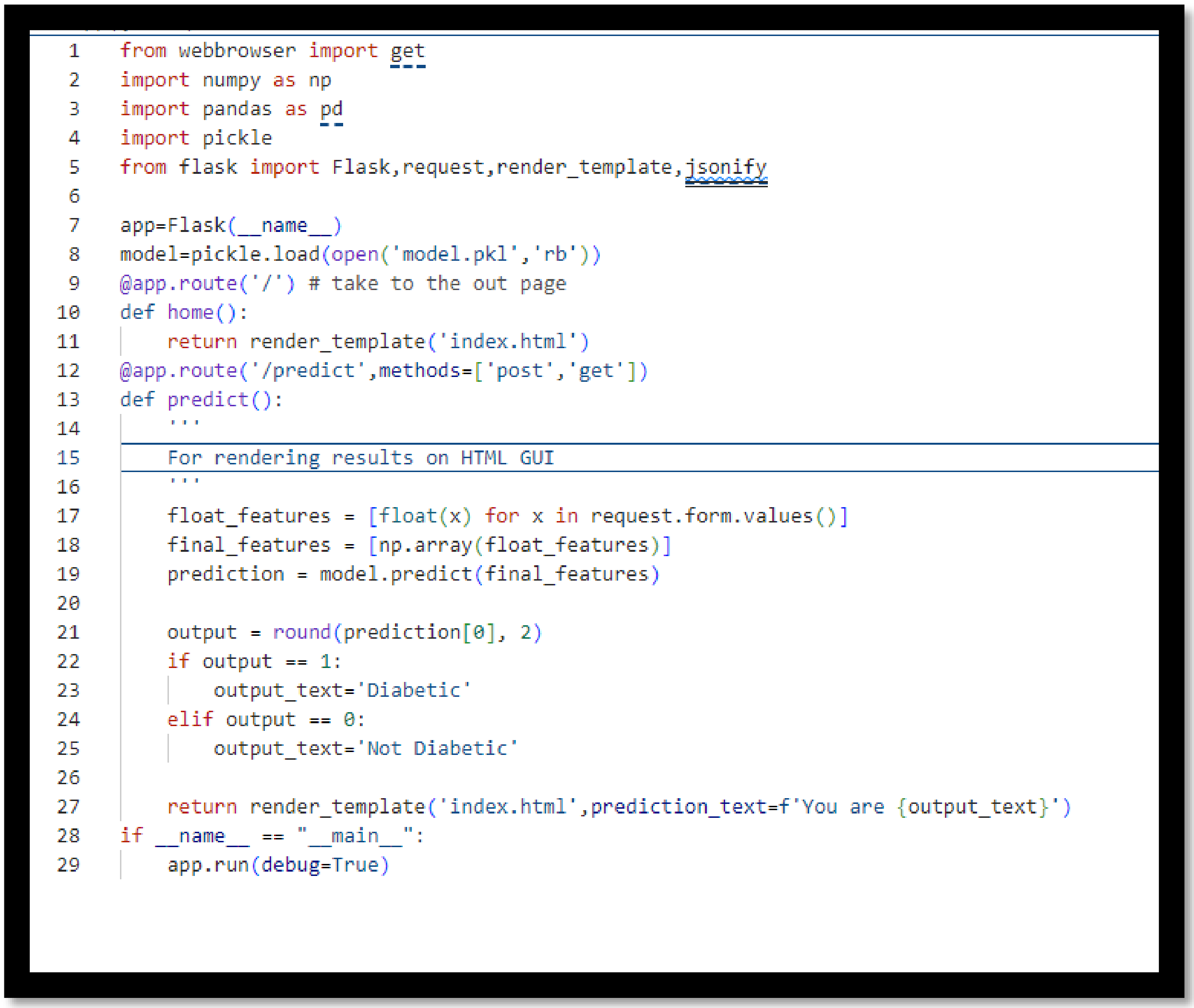
A. Creating application files

We create a folder for the application. The directory tree inside this folder as follow:

```
week4
|   app.py
|    logistic_regression_diabetes.py
|   Logistic_Regression_Hands-On.ipynb
|   model.pkl
|   pima-indians-diabetes.csv
|   request.py
|
├───static
|       script.js
|       style.css
|
└───templates
       index.html
```

○ APP.Py

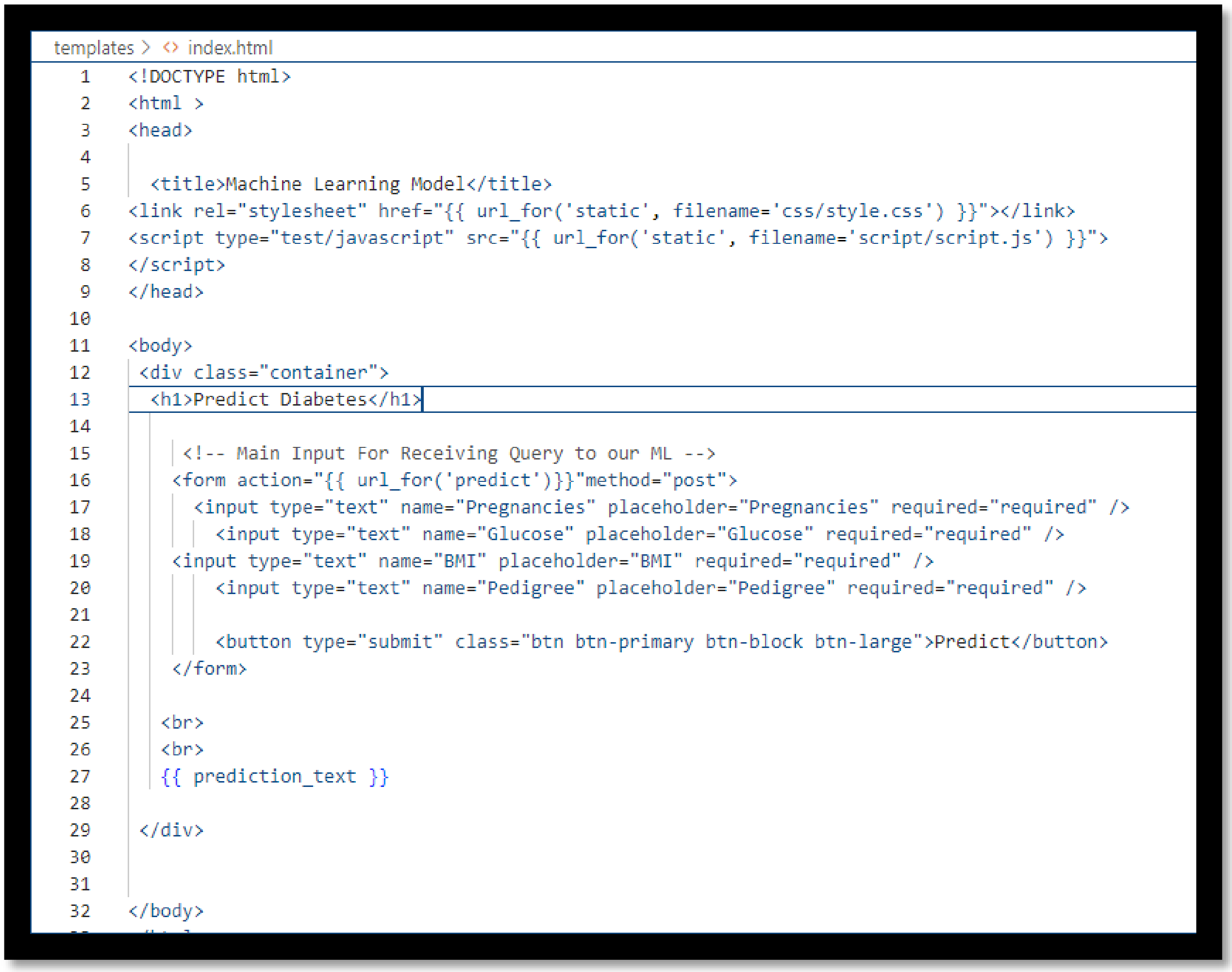
App.py contains the main code that will be run by python. It includes the Machine leaning model and route the Index.html. the route decorator used to map the URL to a return value that means connect url to return value of a function.



```python
from webbrowser import get
import numpy as np
import pandas as pd
import pickle
from flask import Flask,request,render_template,jsonify

app=Flask(__name__)
model=pickle.load(open('model.pkl','rb'))
@app.route('/') # take to the out page
def home():
    return render_template('index.html')
@app.route('/predict',methods=['post','get'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    float_features = [float(x) for x in request.form.values()]
    final_features = [np.array(float_features)]
    prediction = model.predict(final_features)

    output = round(prediction[0], 2)
    if output == 1:
        output_text='Diabetic'
    elif output == 0:
        output_text='Not Diabetic'

    return render_template('index.html',prediction_text=f'You are {output_text}')
if __name__ == "__main__":
    app.run(debug=True)
```

○ Index.html

The index file is a html file. Index.html contains HTML code or tags with some CSS and JavaScript code. It should be in templates folder. When the web site is requested, by default index.html file is returned.

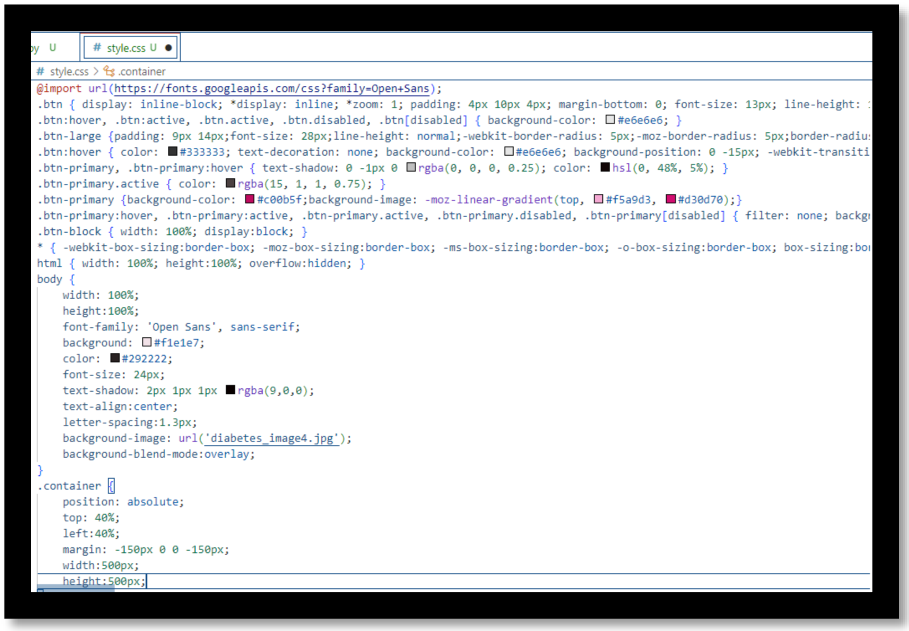


templates > <> index.html

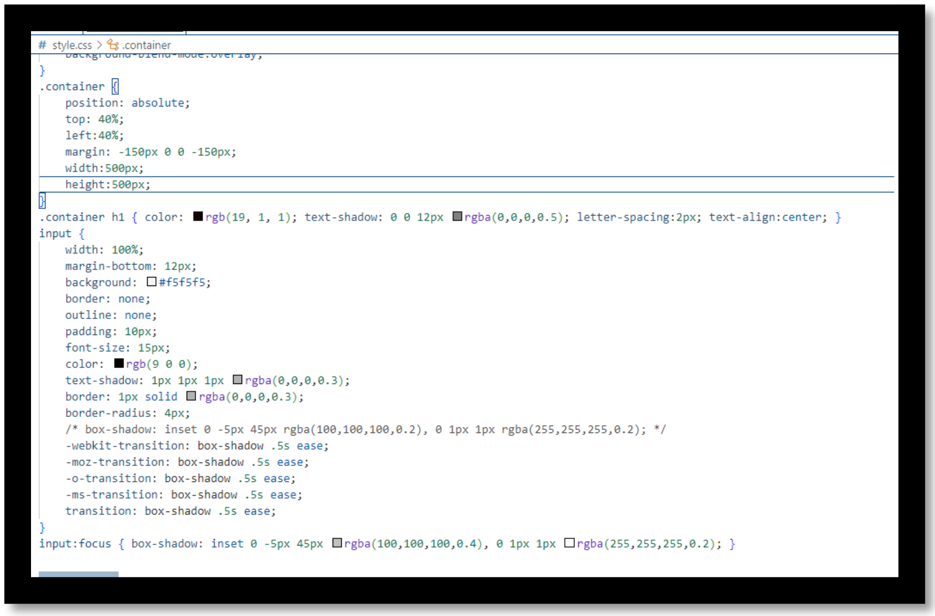
```
<!DOCTYPE html>
<html >
<head>

  <title>Machine Learning Model</title>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}"></link>
<script type="test/javascript" src="{{ url_for('static', filename='script/script.js') }}">
</script>
</head>

<body>
 <div class="container">
  <h1>Predict Diabetes</h1>

     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
      <input type="text" name="Pregnancies" placeholder="Pregnancies" required="required" />
        <input type="text" name="Glucose" placeholder="Glucose" required="required" />
    <input type="text" name="BMI" placeholder="BMI" required="required" />
        <input type="text" name="Pedigree" placeholder="Pedigree" required="required" />

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>


</body>
```

- style.css

CSS file is necessary to determine how the HTML API looks and it must be saved in static folder.



```css
@import url(https://fonts.googleapis.com/css?family=Open+Sans);
.btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; line-height: 
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
.btn-large {padding: 9px 14px;font-size: 28px;line-height: normal;-webkit-border-radius: 5px;-moz-border-radius: 5px;border-radiu
.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transiti
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: hsl(0, 48%, 5%); }
.btn-primary.active { color: rgba(15, 1, 1, 0.75); }
.btn-primary {background-color: #c00b5f;background-image: -moz-linear-gradient(top, #f5a9d3, #d30d70);}
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; backg
.btn-block { width: 100%; display:block; }
* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:bo
html { width: 100%; height:100%; overflow:hidden; }
body {
    width: 100%;
    height:100%;
    font-family: 'Open Sans', sans-serif;
    background: #f1e1e7;
    color: #292222;
    font-size: 24px;
    text-shadow: 2px 1px 1px rgba(9,0,0);
    text-align:center;
    letter-spacing:1.3px;
    background-image: url('diabetes_image4.jpg');
    background-blend-mode:overlay;
}
.container {
    position: absolute;
    top: 40%;
    left:40%;
    margin: -150px 0 0 -150px;
    width:500px;
    height:500px;
```

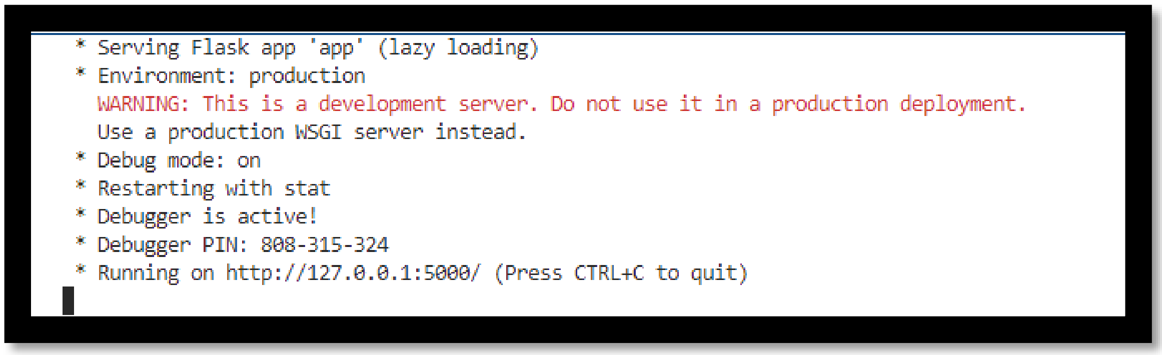


```css
}
.container {
    position: absolute;
    top: 40%;
    left:40%;
    margin: -150px 0 0 -150px;
    width:500px;
    height:500px;
}
.container h1 { color: rgb(19, 1, 1); text-shadow: 0 0 12px rgba(0,0,0,0.5); letter-spacing:2px; text-align:center; }
input {
    width: 100%;
    margin-bottom: 12px;
    background: #f5f5f5;
    border: none;
    outline: none;
    padding: 10px;
    font-size: 15px;
    color: rgb(9 0 0);
    text-shadow: 1px 1px 1px rgba(0,0,0,0.3);
    border: 1px solid rgba(0,0,0,0.3);
    border-radius: 4px;
    /* box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px rgba(255,255,255,0.2); */
    -webkit-transition: box-shadow .5s ease;
    -moz-transition: box-shadow .5s ease;
    -o-transition: box-shadow .5s ease;
    -ms-transition: box-shadow .5s ease;
    transition: box-shadow .5s ease;
}
input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0 1px 1px rgba(255,255,255,0.2); }
```

B. Running App.py

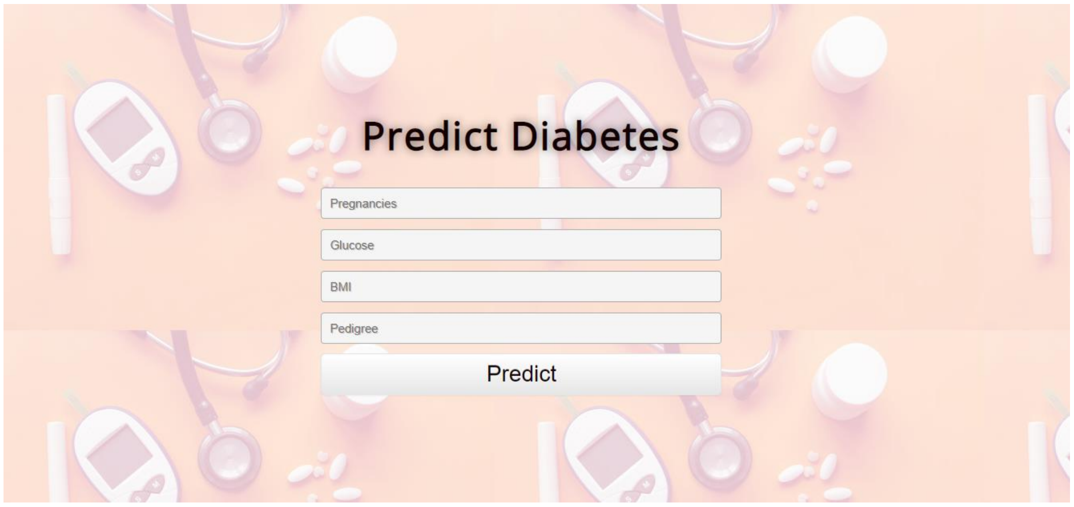
We can run the application either by double click and app.py or by run the command in terminal.

Running with debugger shows the following results.

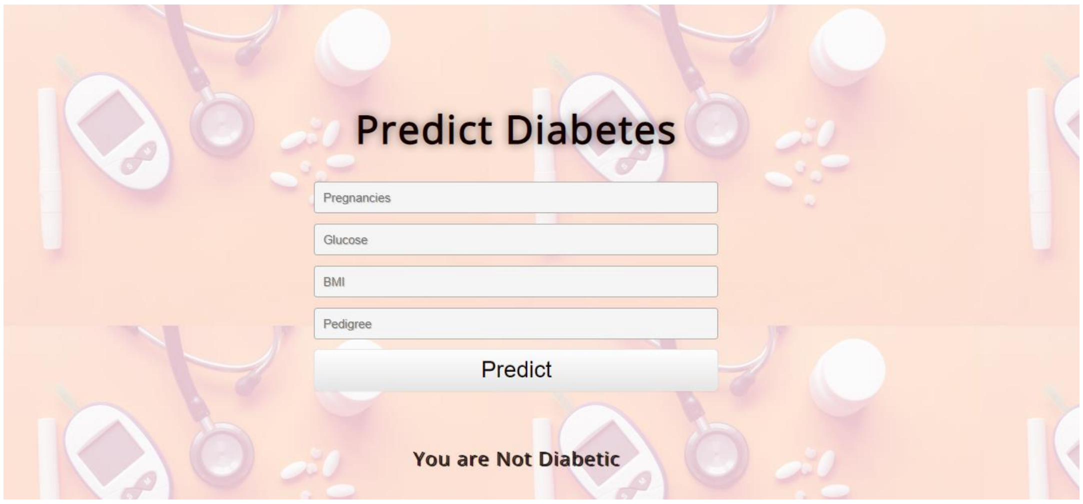


```
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 808-315-324
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- The output looks like this:



- We fill the input values, and the output will be like this.



You are Not Diabetic