Data Science Internship - Data Glacier

Week 4: Deployment on Flask

Name: Nikolaos Sintoris

Batch Code: LISUM10

Date: 22 June 2022

Submitted to: Data Glacier

1. Introduction

In this project, we are going to deploy a machine learning model using python Flask. First, we preprocess our data, then we create and train an SVM model and then we create an API for the model using Flask. The goal of this project is to predict if a woman has diabetes.

2. Dataset

The dataset we used is the 'The Pima Indians Diabetes Dataset' that involves predicting the onset of diabetes within 5 years in Pima Indians women given medical details. It is a binary (2-class) classification problem. The number of observations for each class is not balanced. There are 768 observations with 8 input variables and 1 output variable. Missing values are believed to be encoded with zero values. The variable names are as follows:

1. Number of times pregnant.
2. Plasma glucose concentration 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skinfold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index (weight in kg/(height in m)^2).
7. Diabetes pedigree function.
8. Age (years).
9. Class variable (0 or 1).

3. Data Preprocessing

First, we check if the attributes have right data types. The data types are correct. Secondly, we deal with the missing values. For Plasma_glucose_concentration and Body_mass_index we can delete the rows with missing values because there are not a lot. But for Diastolic_blood_pressure, Triceps_skinfold_thickness, 2h_serum_insulin, we have a lot of missing values so we are going to replace the missing value with the mean value of the attribute.

```
: # Delete the rows that have 0 in the Plasma_glucose_concentration column
  data_frame = data_frame[data_frame.Plasma_glucose_concentration != 0]
  data_frame.shape
```

```
: (763, 9)
```

```
: # Delete the rows that have 0 in the Body_mass_index column
  data_frame = data_frame[data_frame.Body_mass_index != 0]
  data_frame.shape
```

```
: (752, 9)
```

```
: attributes_names = ["Diastolic_blood_pressure", "Triceps_skinfold_thickness", "2h_serum_insulin"]
  for attribute_name in attributes_names:
      mean_value = data_frame[attribute_name].mean()
      data_frame[attribute_name].replace(0, mean_value, inplace = True)
```

```
: data_frame.describe()
```

	times_pregnant	Plasma_glucose_concentration	Diastolic_blood_pressure	Triceps_skinfold_thickness	2h_serum_insulin	Body_mass_index	Diabetes_pedig
count	752.000000	752.000000	752.000000	752.000000	752.000000	752.000000	
mean	3.851064	121.941489	72.300178	26.720695	120.291789	32.454654	
std	3.375189	30.601198	12.157628	9.648926	93.529708	6.928926	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.750000	64.000000	20.715426	81.348404	27.500000	
50%	3.000000	117.000000	72.000000	23.000000	81.348404	32.300000	
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

We can see that there are no longer missing values in our data.

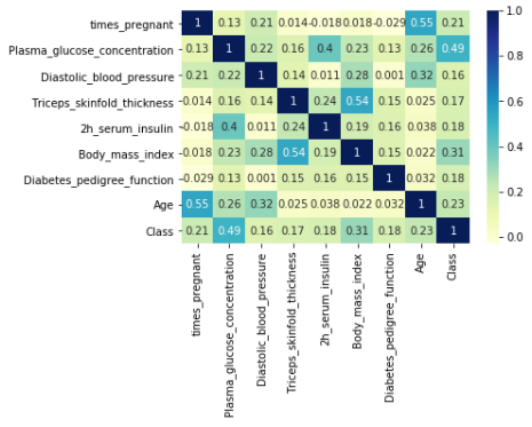
At the end, we are going to look into the correlation that attributes have. We are interested in the correlation that the Class attribute has with the other attributes, and we want to know which of them has the most strong correlation. We create a pearson correlation matrix and after we visualize the correlations with a heatmap in order to understand them better.

```
: pearson_correlation_matrix = data_frame.corr(method = "pearson")
  pearson_correlation_matrix
```

	times_pregnant	Plasma_glucose_concentration	Diastolic_blood_pressure	Triceps_skinfold_thickness	2h_serum_insulin	Body_mas
times_pregnant	1.000000	0.125717	0.205734	0.013715	-0.018230	
Plasma_glucose_concentration	0.125717	1.000000	0.219588	0.158435	0.397111	
Diastolic_blood_pressure	0.205734	0.219588	1.000000	0.135565	0.010848	
Triceps_skinfold_thickness	0.013715	0.158435	0.135565	1.000000	0.236140	
2h_serum_insulin	-0.018230	0.397111	0.010848	0.236140	1.000000	
Body_mass_index	0.018352	0.232771	0.280505	0.536120	0.189907	
Diabetes_pedigree_function	-0.029159	0.133945	0.001036	0.152481	0.157240	
Age	0.545238	0.261490	0.320725	0.025161	0.037952	
Class	0.213371	0.494190	0.159447	0.171703	0.179849	

We can visualize the correlations with a heatmap in order to understand them better.

```
: heatmap_plot = sb.heatmap(pearson_correlation_matrix, cmap = "YlGnBu", annot = True)
```



Considering the heatmap above, we can see that Plasma_glucose_concentration and Body_mass_index attributes have a modest correlation with the Class attribute. We can not come to a conclusion about correlation with these attributes.

4. Create and train model

The model we are going to use is an SVM and we are going to train it with the K-fold cross validation technic. We create 10 folders of our initial dataset. For every folder:

- One folder is the test set
- The remaining 9 folders are the training set
- Train the model with the current training set
- Test the model with the current test set
- Compute Accuracy and F1-score

After that, we compute the mean f1-score and accuracy, and we train again our model with the entire dataset.

```
kf_object = KFold(n_splits = 10)
svm_model = svm.SVC() # create SVM model

accuracy_list = []
f1_score_list = []
for current_train_indices, current_test_indices in kf_object.split(data):
    current_training_dataset = data[current_train_indices[0]:current_train_indices[-1]]
    current_training_actual_labels = actual_labels[current_train_indices[0]:current_train_indices[-1]]

    current_test_dataset = data[current_test_indices[0]:current_test_indices[-1]]
    current_test_actual_labels = actual_labels[current_test_indices[0]:current_test_indices[-1]]

    svm_model.fit(current_training_dataset, current_training_actual_labels) # Train model with current training dataset
    current_test_predicted_labels = svm_model.predict(current_test_dataset) # Test model

    current_accuracy = accuracy_score(current_test_actual_labels, current_test_predicted_labels)
    current_f1_score = f1_score(current_test_actual_labels, current_test_predicted_labels)

    accuracy_list.append(current_accuracy)
    f1_score_list.append(current_f1_score)
```

```
accuracy_np = np.array(accuracy_list)
print("Mean Accuracy: ", np.mean(accuracy_np))
```

```
Mean Accuracy:  0.7682522522522524
```

```
f1_score_np = np.array(f1_score_list)
print("Mean F1-score: ", np.mean(f1_score_np))
```

```
Mean F1-score:  0.5979985105095145
```

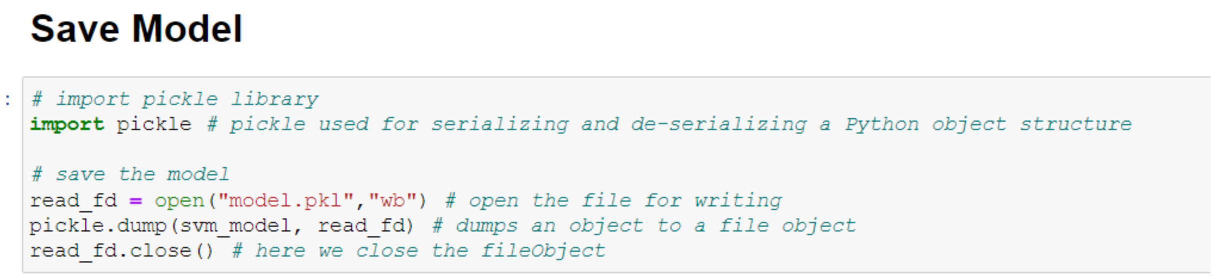
At the end we train our model again, with the whole dataset.

```
svm_model.fit(data, actual_labels) # Train model with all the data.
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

5. Save Model

After that, we save our model using pickle.



Save Model

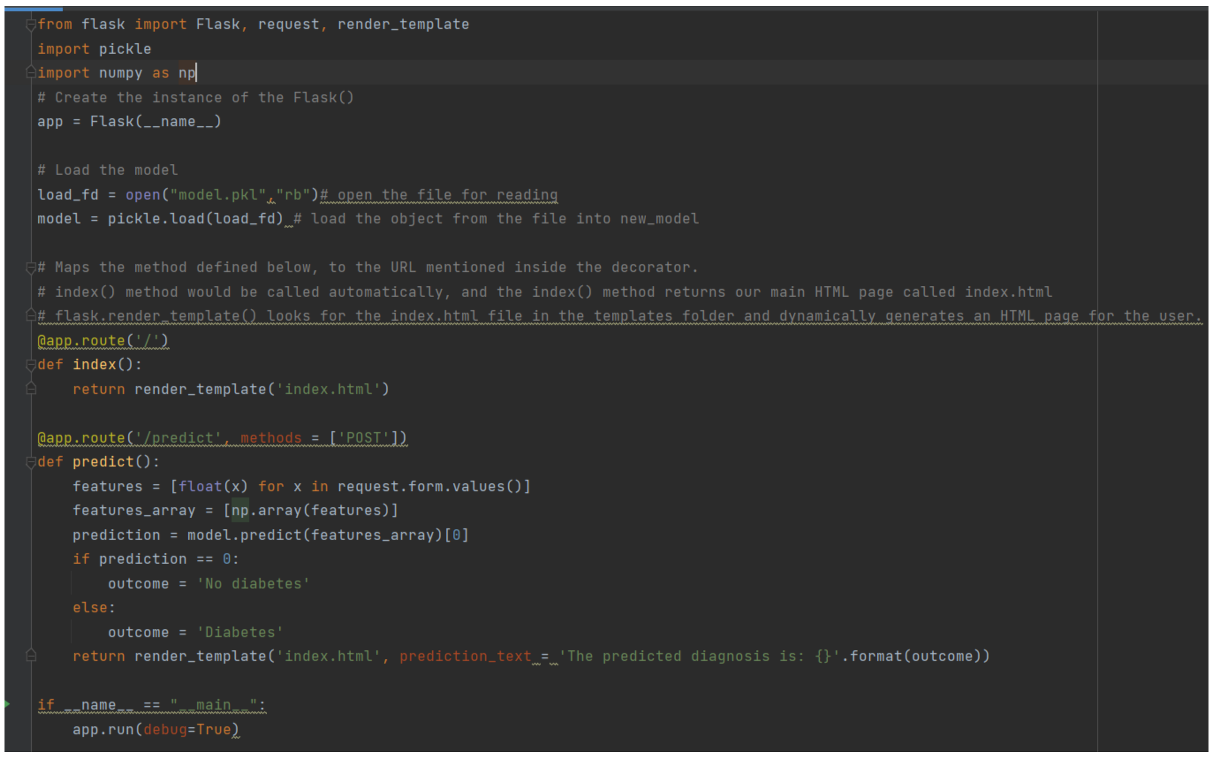
```
# import pickle library
import pickle # pickle used for serializing and de-serializing a Python object structure

# save the model
read_fd = open("model.pkl","wb") # open the file for writing
pickle.dump(svm_model, read_fd) # dumps an object to a file object
read_fd.close() # here we close the fileObject
```

6. Deploy model into a Web Application

We develop a web application that consist of a simple web page with 8 fields that let the user fill the values. After submitting the values, the user press the predict button and a message shows and informs him if it has diabetes or not.

6.1 app.py

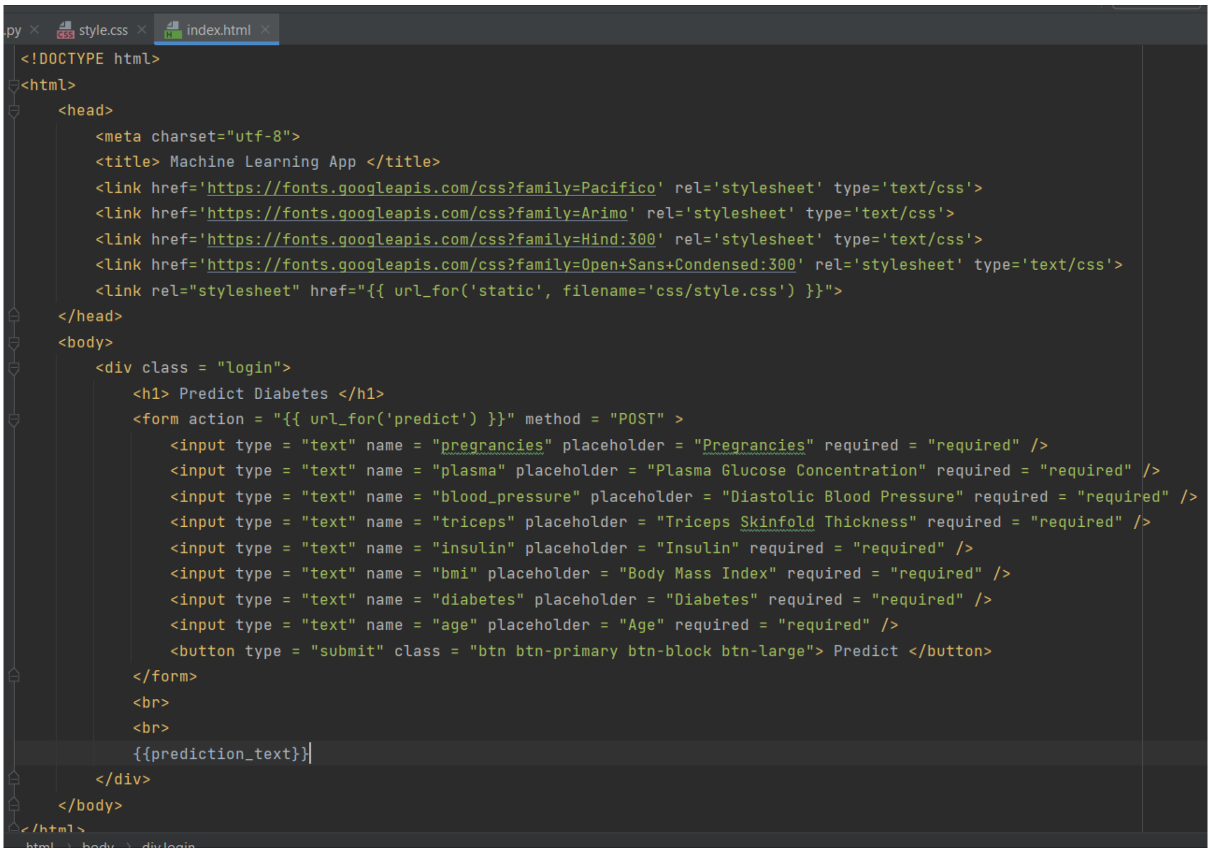
This is the main application file, where all our code resides, and it binds everything together.



```
from flask import Flask, request, render_template
import pickle
import numpy as np
# Create the instance of the Flask()
app = Flask(__name__)

# Load the model
load_fd = open("model.pkl","rb")# open the file for reading
model = pickle.load(load_fd) # load the object from the file into new_model

# Maps the method defined below, to the URL mentioned inside the decorator.
# index() method would be called automatically, and the index() method returns our main HTML page called index.html
# flask.render_template() looks for the index.html file in the templates folder and dynamically generates an HTML page for the user.
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods = ['POST'])
def predict():
    features = [float(x) for x in request.form.values()]
    features_array = [np.array(features)]
    prediction = model.predict(features_array)[0]
    if prediction == 0:
        outcome = 'No diabetes'
    else:
        outcome = 'Diabetes'
    return render_template('index.html', prediction_text = 'The predicted diagnosis is: {}'.format(outcome))

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

6.2 index.html

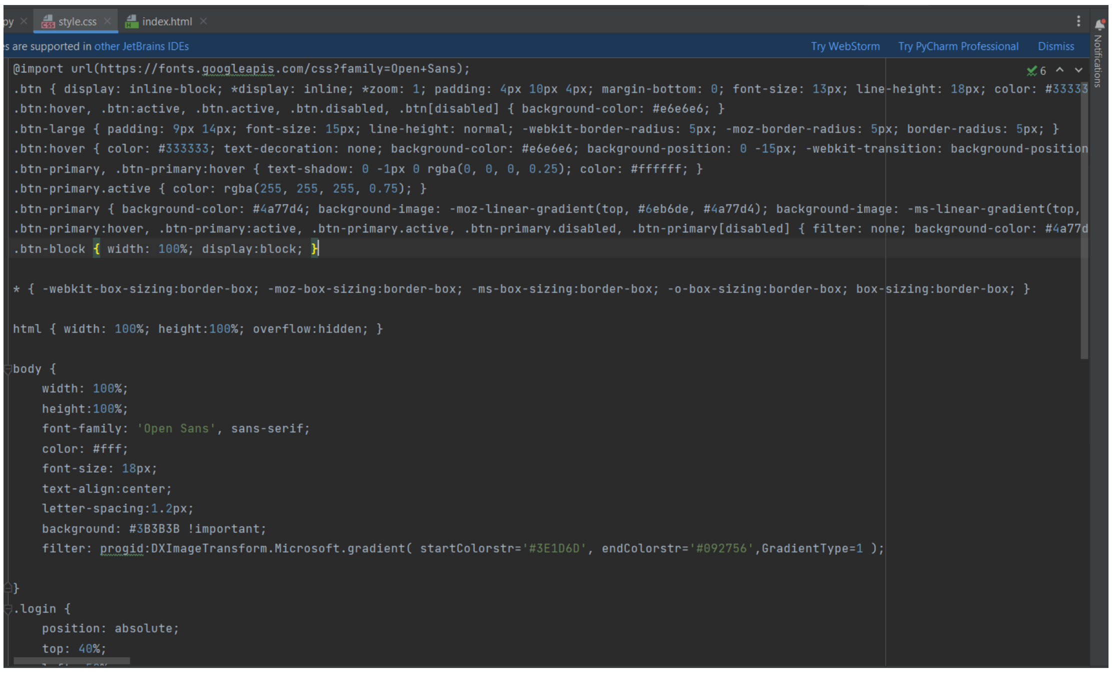
The html files is used by our main file *(app.py)* to generate the front end of our application



```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title> Machine Learning App </title>
        <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
        <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
        <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
        <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
        <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
    </head>
    <body>
        <div class = "login">
            <h1> Predict Diabetes </h1>
            <form action = "{{ url_for('predict') }}" method = "POST" >
                <input type = "text" name = "pregrancies" placeholder = "Pregrancies" required = "required" />
                <input type = "text" name = "plasma" placeholder = "Plasma Glucose Concentration" required = "required" />
                <input type = "text" name = "blood_pressure" placeholder = "Diastolic Blood Pressure" required = "required" />
                <input type = "text" name = "triceps" placeholder = "Triceps Skinfold Thickness" required = "required" />
                <input type = "text" name = "insulin" placeholder = "Insulin" required = "required" />
                <input type = "text" name = "bmi" placeholder = "Body Mass Index" required = "required" />
                <input type = "text" name = "diabetes" placeholder = "Diabetes" required = "required" />
                <input type = "text" name = "age" placeholder = "Age" required = "required" />
                <button type = "submit" class = "btn btn-primary btn-block btn-large"> Predict </button>
            </form>
            <br>
            <br>
            {{prediction_text}}
        </div>
    </body>
</html>
```

6.3 style.css

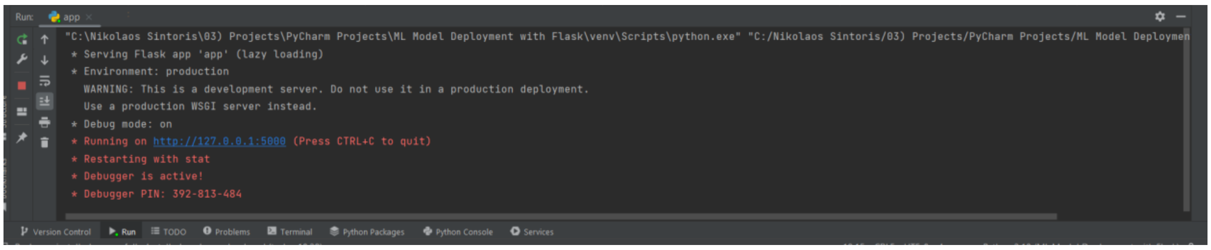
Css is to determine the look of the html.



```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: 18px; color: #33333
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }
.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transition: background-position
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: #ffffff; }
.btn-primary.active { color: rgba(255, 255, 255, 0.75); }
.btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-gradient(top,
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; background-color: #4a77d
.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:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {
    width: 100%;
    height:100%;
    font-family: 'Open Sans', sans-serif;
    color: #fff;
    font-size: 18px;
    text-align:center;
    letter-spacing:1.2px;
    background: #3B3B3B !important;
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );

}
.login {
    position: absolute;
    top: 40%;
```

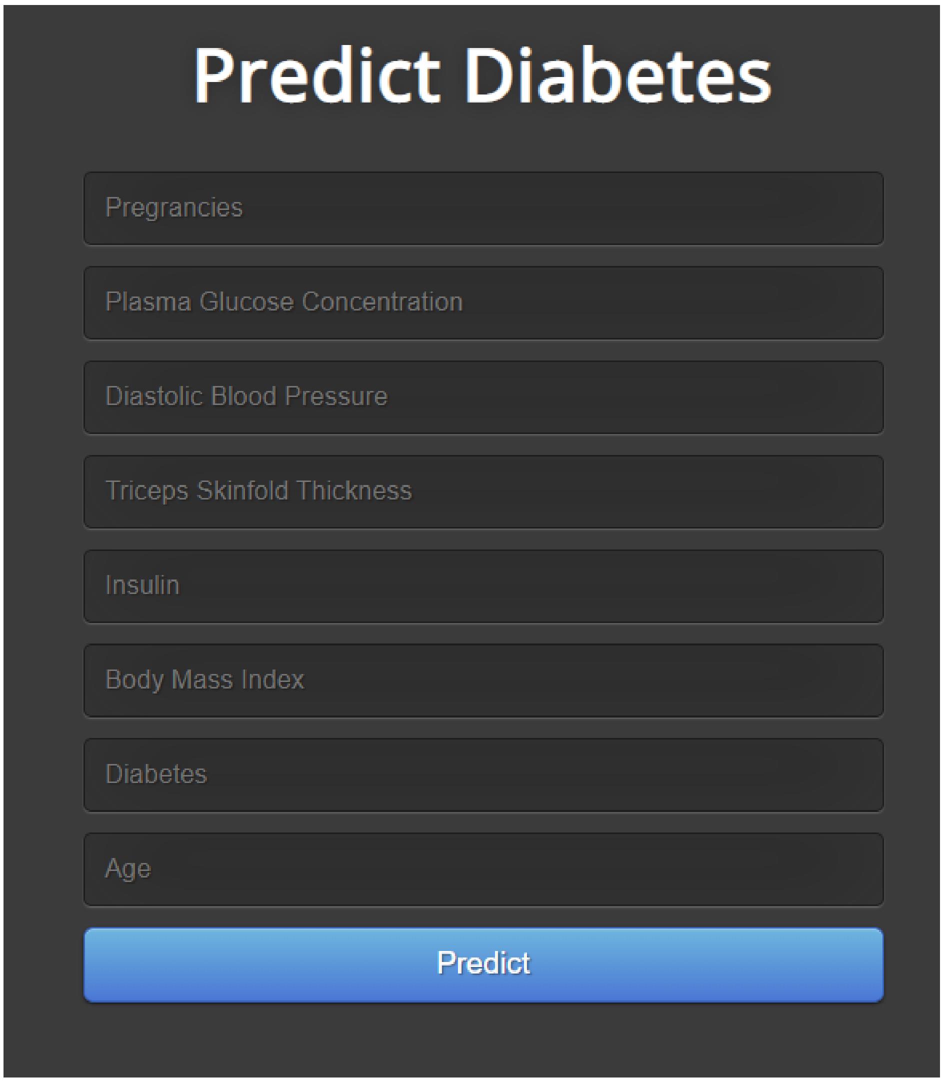
7. Run the app

We run the app.py and we check our terminal.

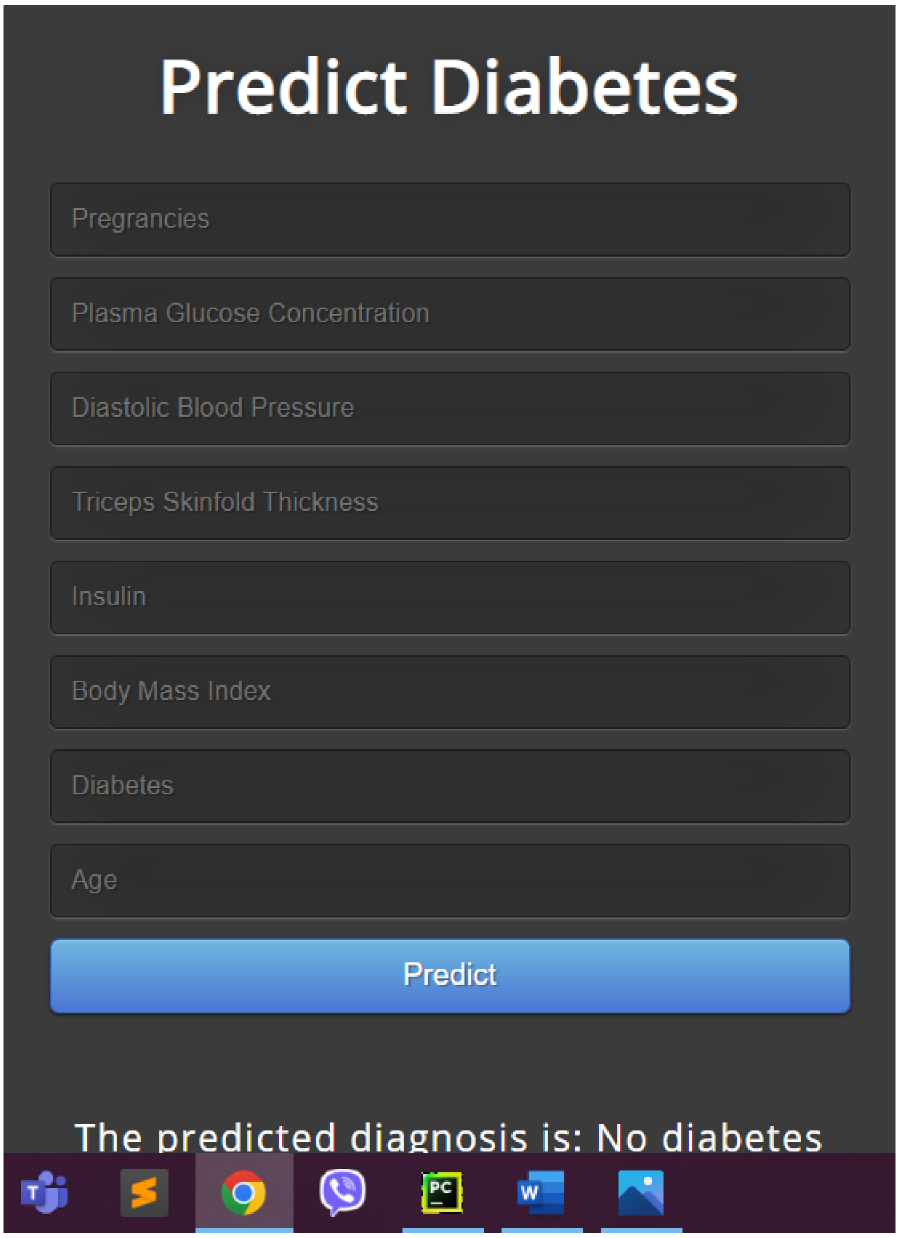


```
"C:\Nikolaos Sintoris\03) Projects\PyCharm Projects\ML Model Deployment with Flask\venv\Scripts\python.exe" "C:/Nikolaos Sintoris/03) Projects/PyCharm Projects/ML Model Deploymen
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 392-813-484
```

Now we double click on the web address http://127.0.0.1:5000/ and we can see our web app.



Then, we insert the values of the observation we want to check and press the button Predict.



Predict Diabetes

Pregrancies

Plasma Glucose Concentration

Diastolic Blood Pressure

Triceps Skinfold Thickness

Insulin

Body Mass Index

Diabetes

Age

Predict

The predicted diagnosis is: No diabetes