Model Deployment Using Flask

Name : Model Deployment Using Flask with IRIS Dataset
Report date : 15-08-2021
Internship Batch : LISUM02

Model Deployment Steps

1. Choosing Dataset:

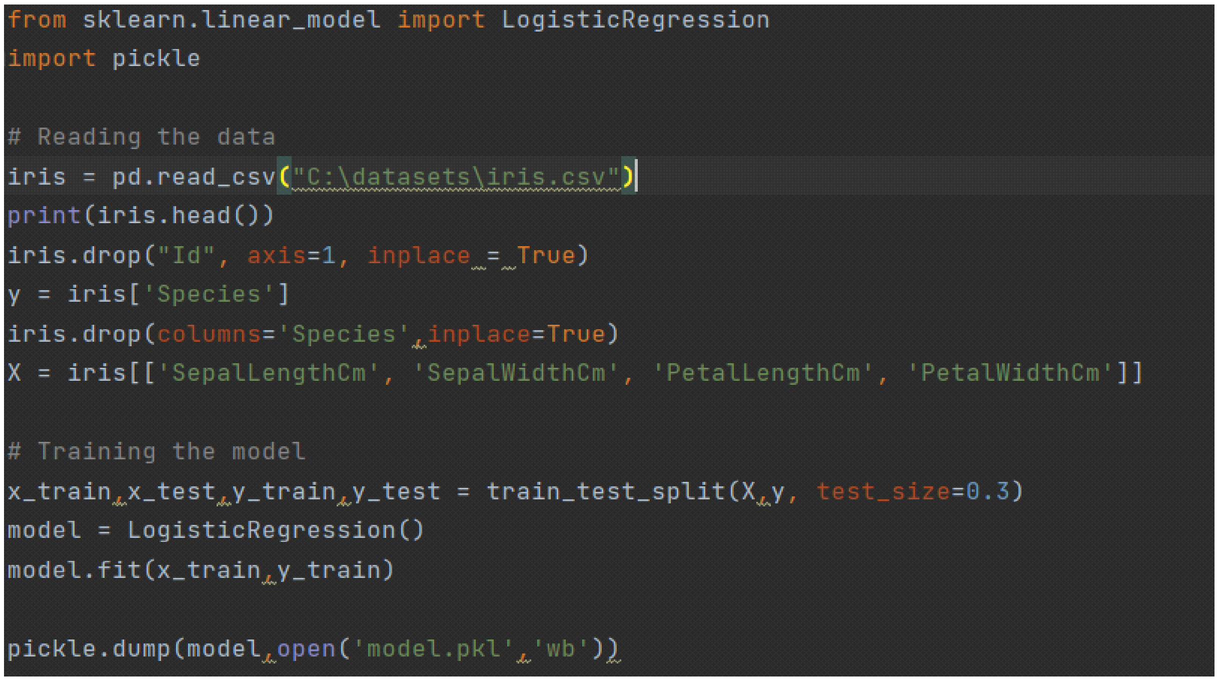
The data which is used in the project was downloaded in kaggle website. The dataset contains three iris species with 50 samples each as well as some properties about each flower.

The columns in the dataset contains:
Id
Sepal Length(cm)
Sepal Width(cm)
Petal Length (cm)
Petal Width (cm)
Species (3 types namely Iris-setosa, Iris-versicolor, Iris-virginica)

2. Building a Python Model and save using Flask:

In this step we are going to predict the type of the species by creating a python model using pycharm and import pickle for the future use.

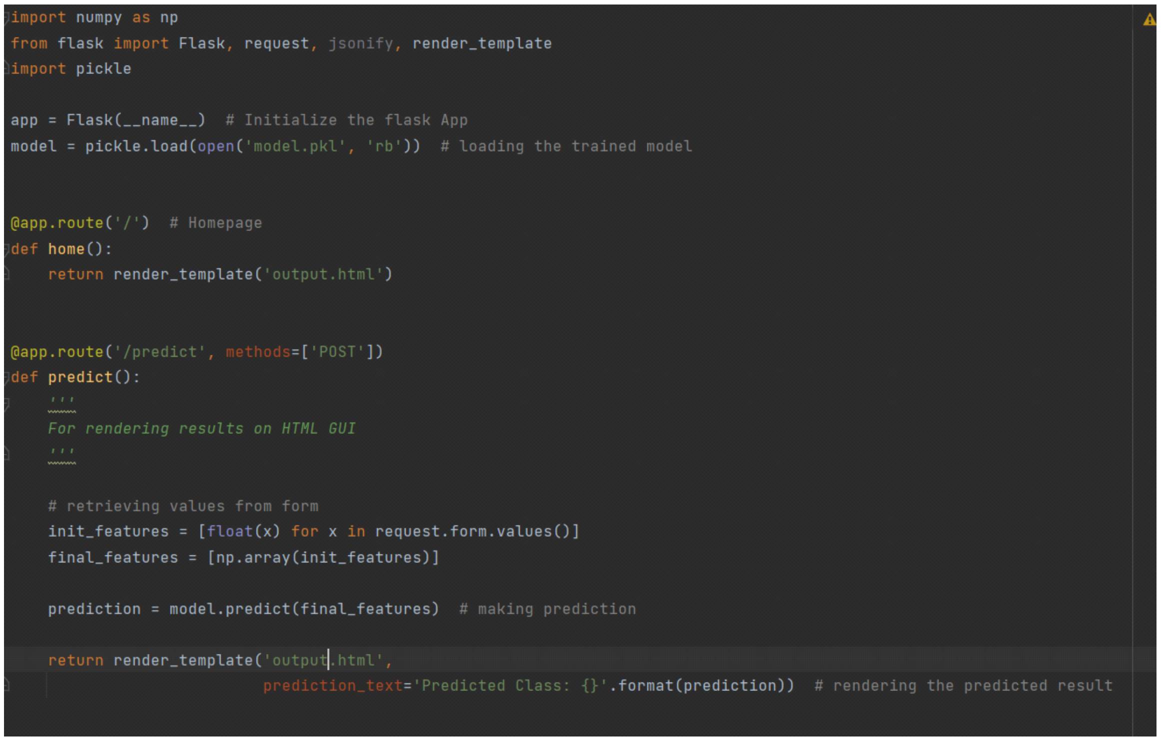
iris.py



```
from sklearn.linear_model import LogisticRegression
import pickle

# Reading the data
iris = pd.read_csv("C:\datasets\iris.csv")
print(iris.head())
iris.drop("Id", axis=1, inplace = True)
y = iris['Species']
iris.drop(columns='Species',inplace=True)
X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]

# Training the model
x_train,x_test,y_train,y_test = train_test_split(X,y, test_size=0.3)
model = LogisticRegression()
model.fit(x_train,y_train)

pickle.dump(model,open('model.pkl','wb'))
```

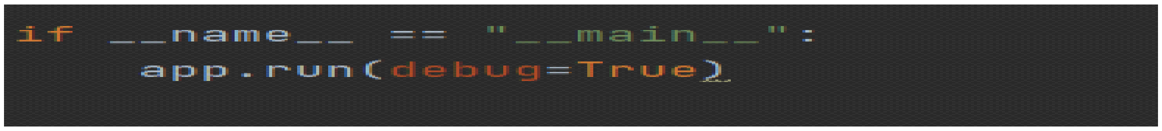
3. Flask Deployment:

In this step we use app.py file to predit the species of the flower by taking the data.



```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)  # Initialize the flask App
model = pickle.load(open('model.pkl', 'rb'))  # loading the trained model


@app.route('/')  # Homepage
def home():
    return render_template('output.html')


@app.route('/predict', methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''

    # retrieving values from form
    init_features = [float(x) for x in request.form.values()]
    final_features = [np.array(init_features)]

    prediction = model.predict(final_features)  # making prediction

    return render_template('output.html',
                           prediction_text='Predicted Class: {}'.format(prediction))  # rendering the predicted result
```

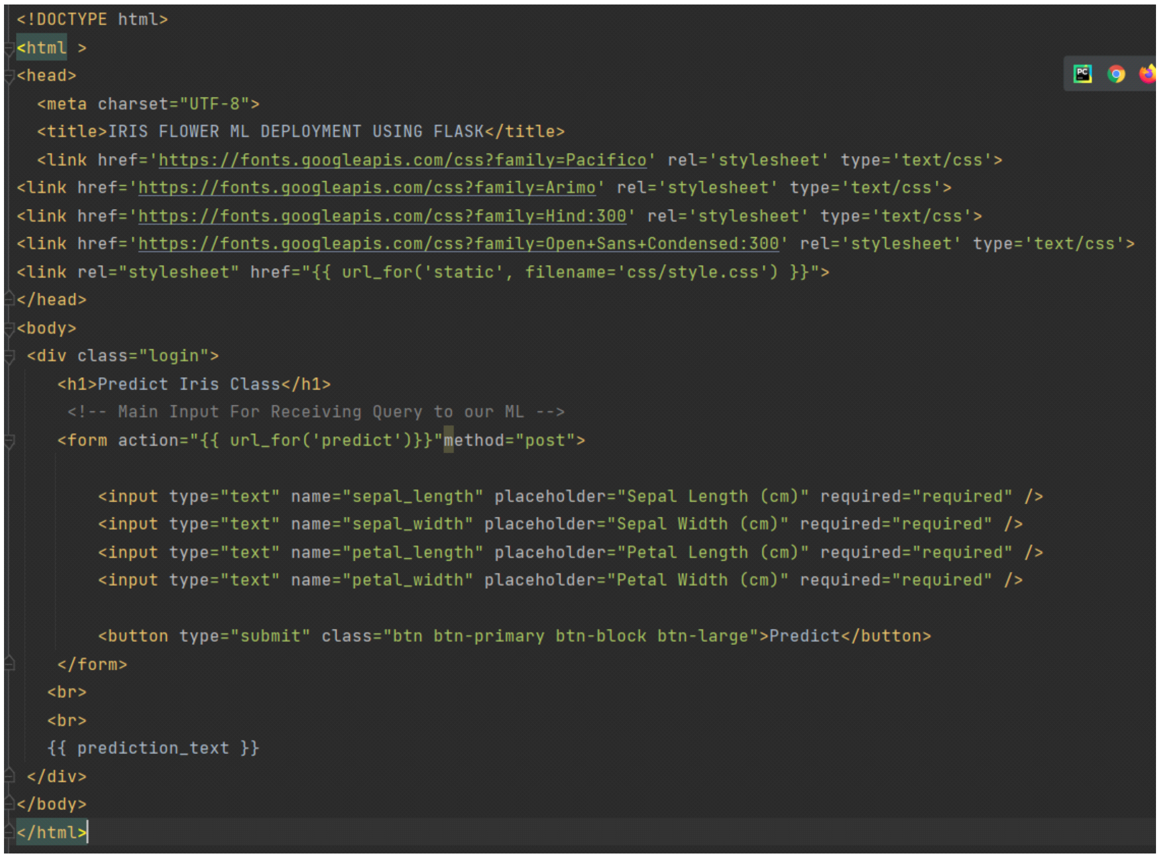
In the above screen shot there is prediction_text accepts the data and send the predicted species result as an output.



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

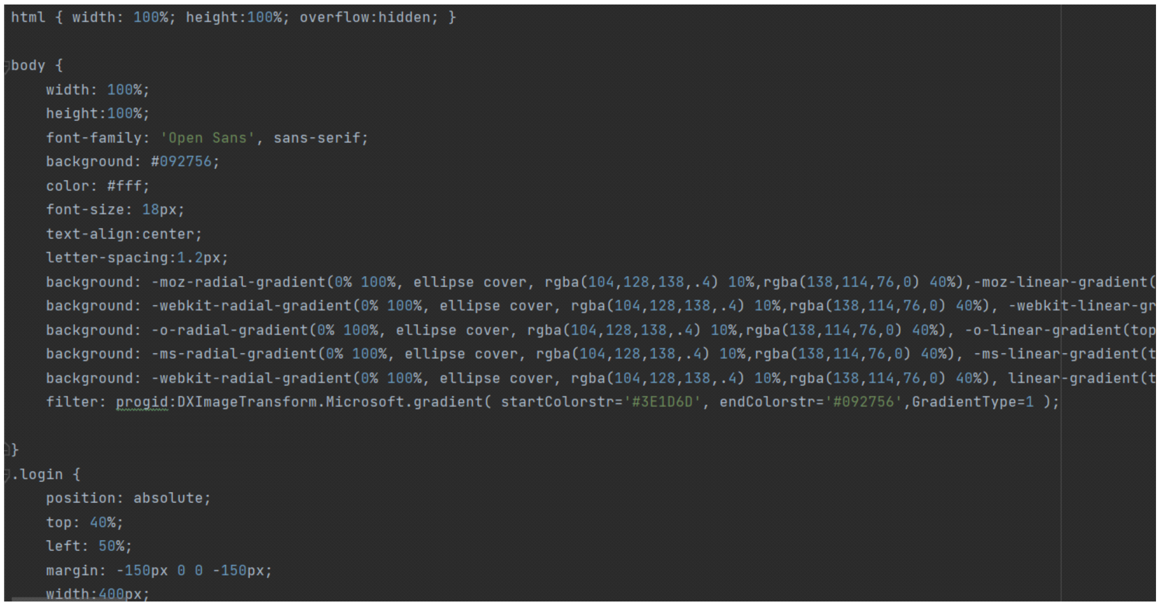
4. Creating Files for web-application:

Create Template and Static folder in the respective directory. In template folder contains the HTML file for web-application and in static folder contains the required css file for HTML file.

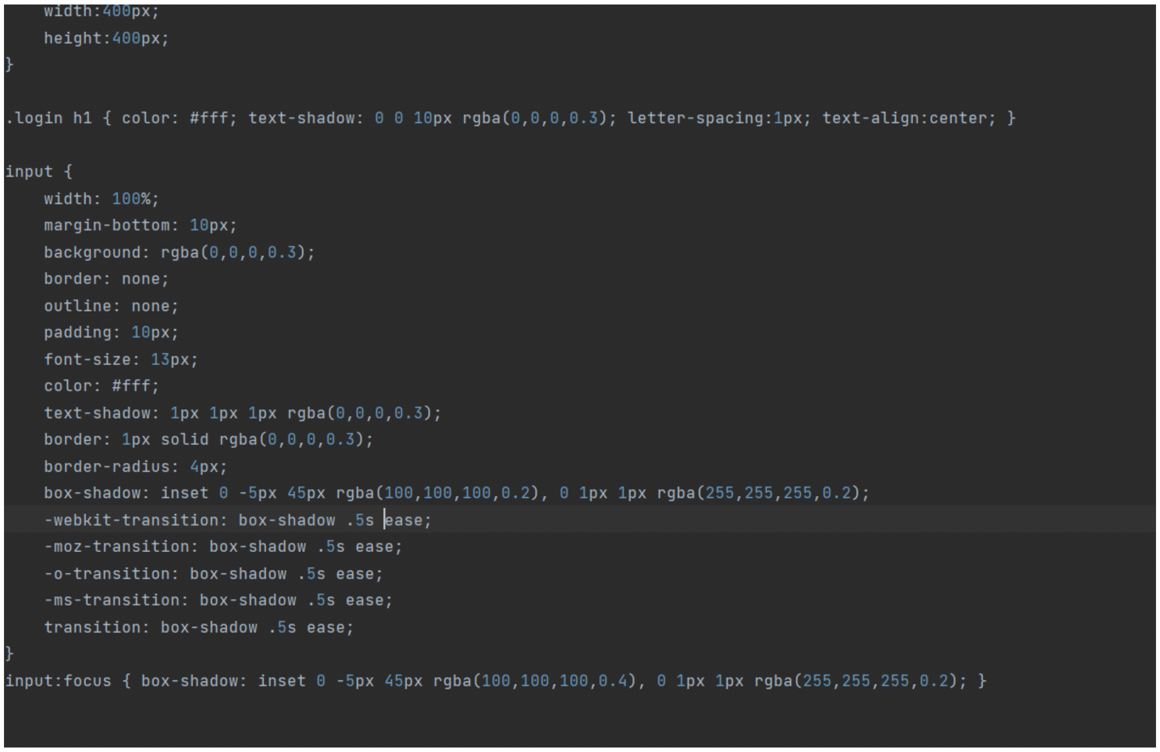


```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>IRIS FLOWER ML DEPLOYMENT USING FLASK</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<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 Iris Class</h1>
     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">

        <input type="text" name="sepal_length" placeholder="Sepal Length (cm)" required="required" />
        <input type="text" name="sepal_width" placeholder="Sepal Width (cm)" required="required" />
        <input type="text" name="petal_length" placeholder="Petal Length (cm)" required="required" />
        <input type="text" name="petal_width" placeholder="Petal Width (cm)" required="required" />

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>
   <br>
   <br>
   {{ prediction_text }}
 </div>
</body>
</html>
```

style.css



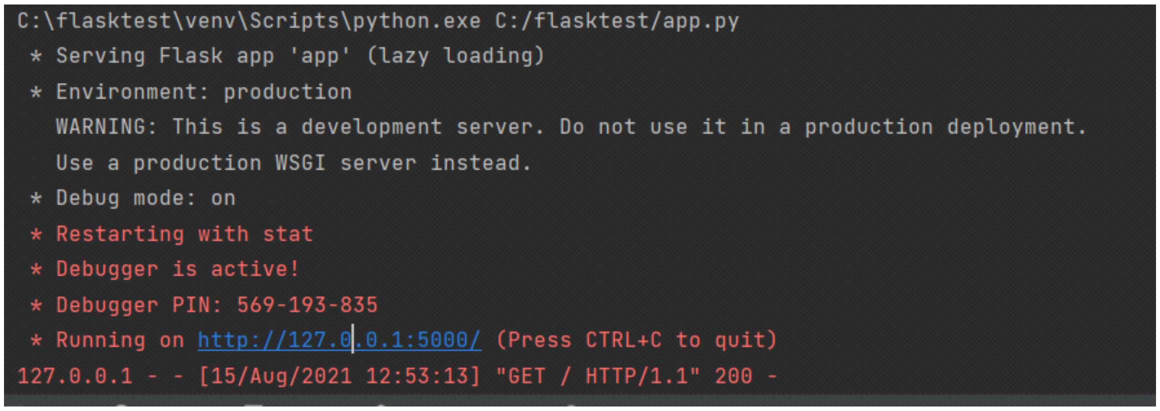
```css
html { width: 100%; height:100%; overflow:hidden; }

body {
    width: 100%;
    height:100%;
    font-family: 'Open Sans', sans-serif;
    background: #092756;
    color: #fff;
    font-size: 18px;
    text-align:center;
    letter-spacing:1.2px;
    background: -moz-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%),-moz-linear-gradient(
    background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -webkit-linear-gr
    background: -o-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -o-linear-gradient(top
    background: -ms-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -ms-linear-gradient(t
    background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), linear-gradient(t
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );

}
.login {
    position: absolute;
    top: 40%;
    left: 50%;
    margin: -150px 0 0 -150px;
    width:400px;
```



```css
    width:400px;
    height:400px;
}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }

input {
    width: 100%;
    margin-bottom: 10px;
    background: rgba(0,0,0,0.3);
    border: none;
    outline: none;
    padding: 10px;
    font-size: 13px;
    color: #fff;
    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); }
```

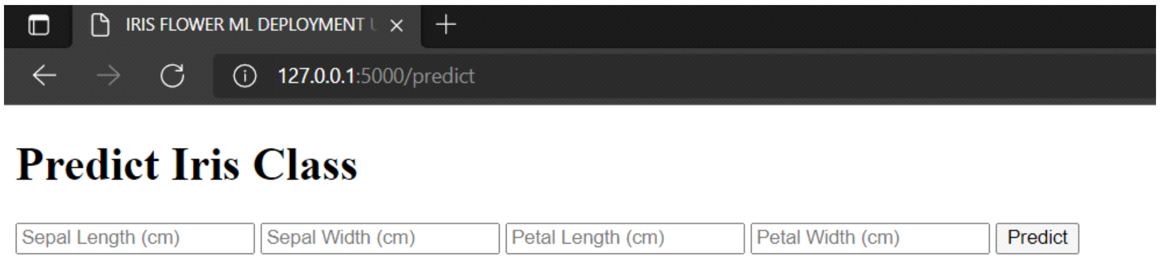
5. Running the app.py file

Now to run the Flask Server, open cmd (command prompt) and type the commond as python app.py.



```
C:\flasktest\venv\Scripts\python.exe C:/flasktest/app.py
 * 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: 569-193-835
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Aug/2021 12:53:13] "GET / HTTP/1.1" 200 -
```

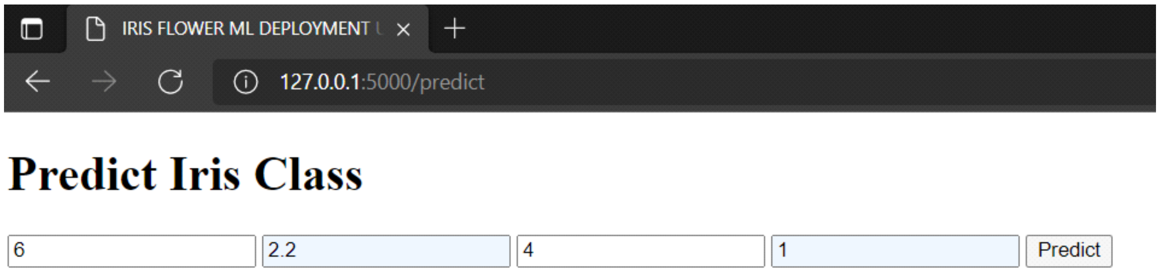
Open http://127.0.0.1:5000/ in your web-browser, and this will appear if everything above mentioned is successfully done.



Predict Iris Class

Sepal Length (cm) | Sepal Width (cm) | Petal Length (cm) | Petal Width (cm)

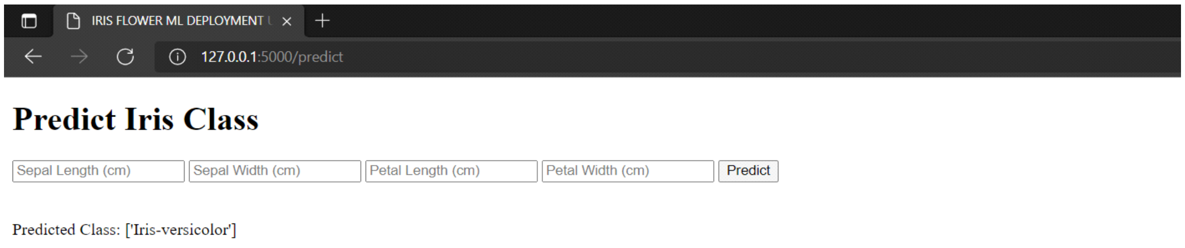
After that insert values in the fields.



Predict Iris Class

6 | 2.2 | 4 | 1

Finally, clicking on the the predit button will display the type of the species as shown below.



Predict Iris Class

Sepal Length (cm) | Sepal Width (cm) | Petal Length (cm) | Petal Width (cm) | Predict

Predicted Class: ['Iris-versicolor']