TITLE: IRIS FLOWER PREDICTION APP DEPLOYMENT

Model Training Section:

Dataset Overview:

The dataset used for training the model is the **Iris dataset**, which contains 150 rows and 5 columns (4 features and 1 class). The features are:

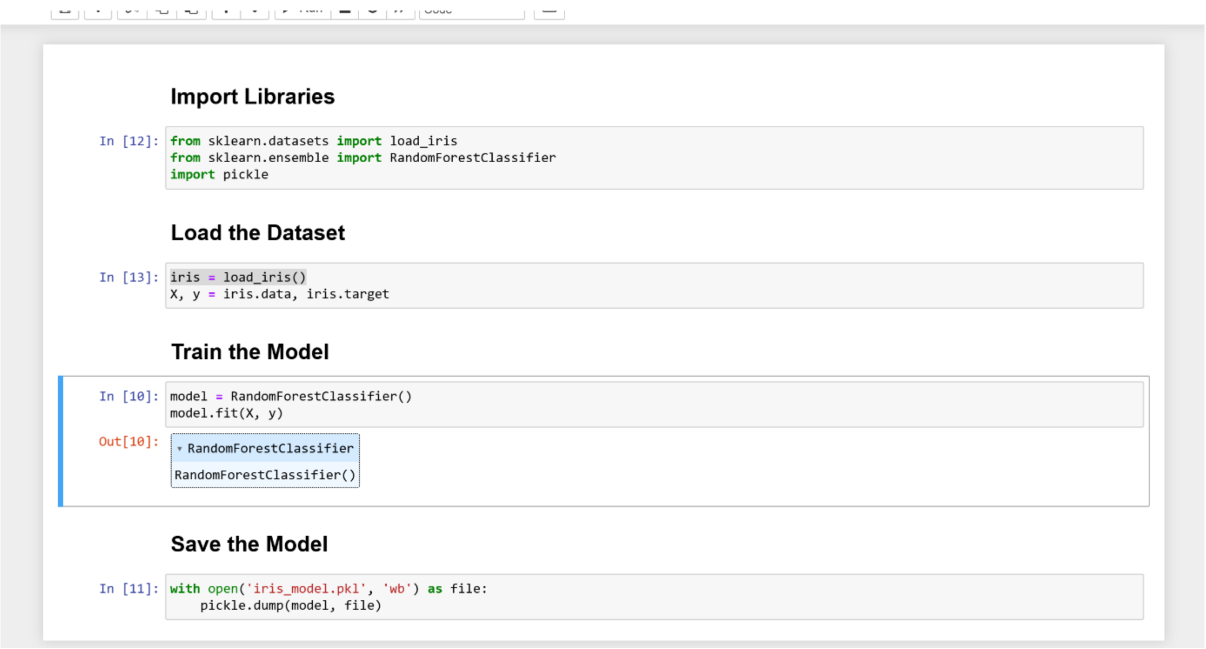
- Sepal length (cm)
- Sepal width (cm)
- Petal length (cm)
- Petal width (cm)

The target variable, “species” consists of three types of Iris flowers:

- Setosa
- Versicolor
- Virginica

Model Overview:

The model used for prediction is Random Forest Classifier, an ensemble machine learning algorithm that builds multiple decision trees and combines their output for better accuracy and reduced overfitting. The model was trained using scikit-learn and saved using the pickle module for deployment in a Flask web app to allow easy loading during the deployment process.



Import Libraries

In [12]:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
import pickle
```

Load the Dataset

In [13]:

```
iris = load_iris()
X, y = iris.data, iris.target
```

Train the Model

In [10]:

```
model = RandomForestClassifier()
model.fit(X, y)
```

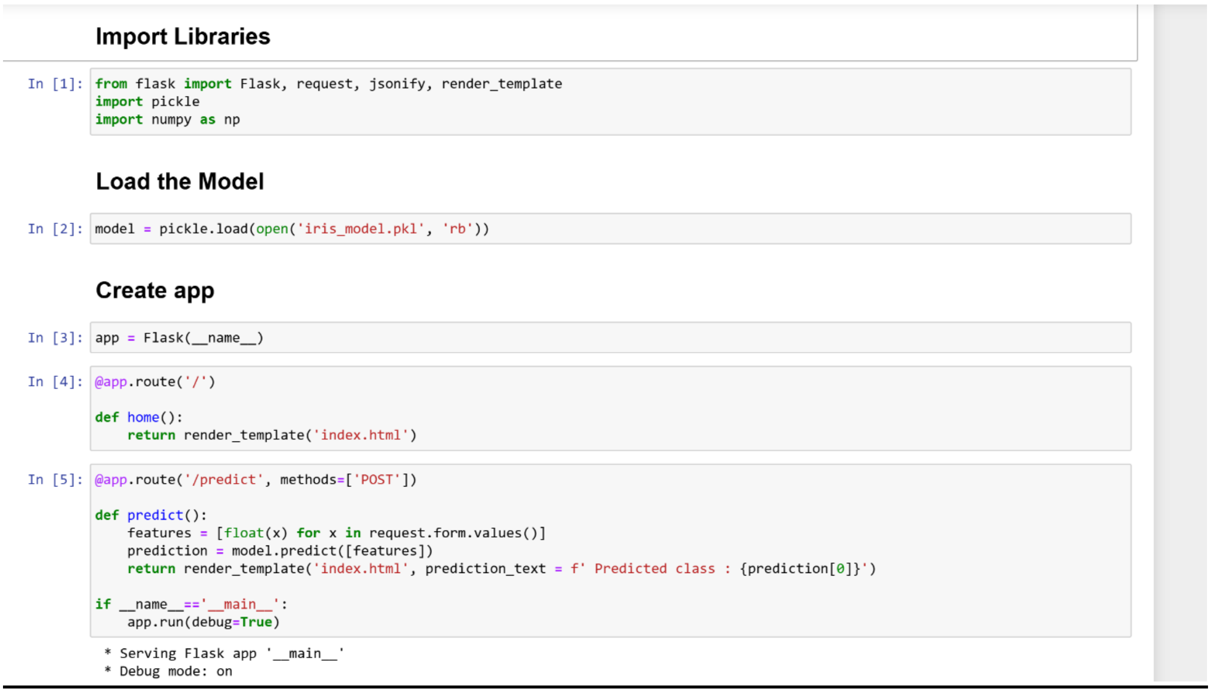
Out[10]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

Save the Model

In [11]:

```
with open('iris_model.pkl', 'wb') as file:
    pickle.dump(model, file)
```



Import Libraries

```
In [1]: from flask import Flask, request, jsonify, render_template
        import pickle
        import numpy as np
```

Load the Model

```
In [2]: model = pickle.load(open('iris_model.pkl', 'rb'))
```

Create app

```
In [3]: app = Flask(__name__)
```

```
In [4]: @app.route('/')

        def home():
            return render_template('index.html')
```

```
In [5]: @app.route('/predict', methods=['POST'])

        def predict():
            features = [float(x) for x in request.form.values()]
            prediction = model.predict([features])
            return render_template('index.html', prediction_text = f' Predicted class : {prediction[0]}')

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

         * Serving Flask app '__main__'
         * Debug mode: on
```

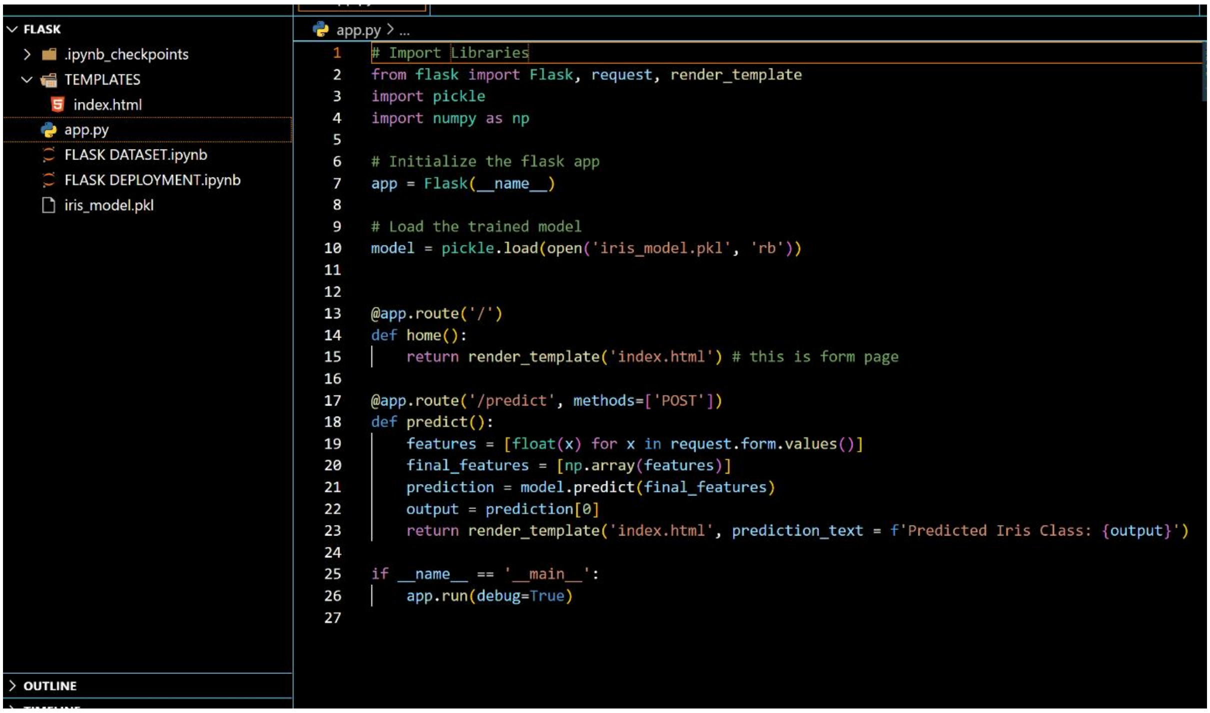
Deployment Section:

Step by step deploying the model with Flask:

1. Install necessary libraries:
   Run the following command to install Flask and scikit-learn if not done already:
   pip install flask scikit-learn numpy

2. Create a Flask app:
   First create a new directory for the project (e.g., iris_predictor)
   Inside this directory, create a python script named app.py

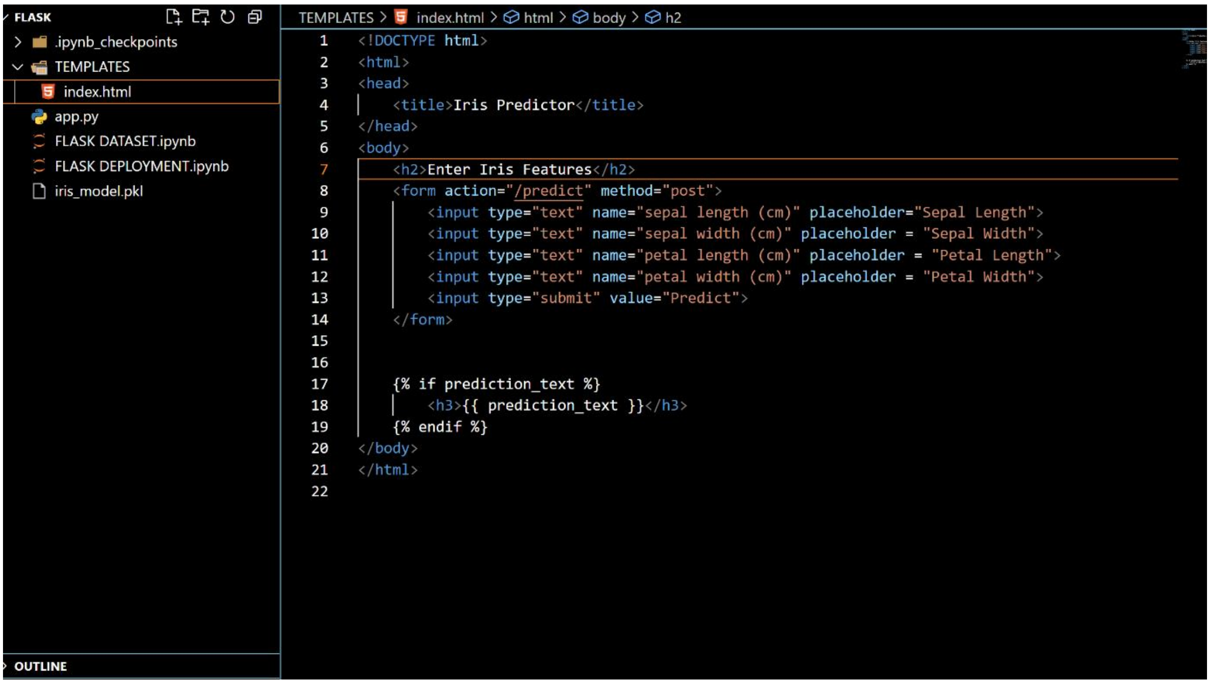
3. Load the Model:
   In your app.py, load the trained model with pickle:
   model = pickle.load(open(‘iris_model.pkl’, ‘rb’))



```python
# Import Libraries
from flask import Flask, request, render_template
import pickle
import numpy as np

# Initialize the flask app
app = Flask(__name__)

# Load the trained model
model = pickle.load(open('iris_model.pkl', 'rb'))


@app.route('/')
def home():
    return render_template('index.html') # this is form page

@app.route('/predict', methods=['POST'])
def predict():
    features = [float(x) for x in request.form.values()]
    final_features = [np.array(features)]
    prediction = model.predict(final_features)
    output = prediction[0]
    return render_template('index.html', prediction_text = f'Predicted Iris Class: {output}')

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

```

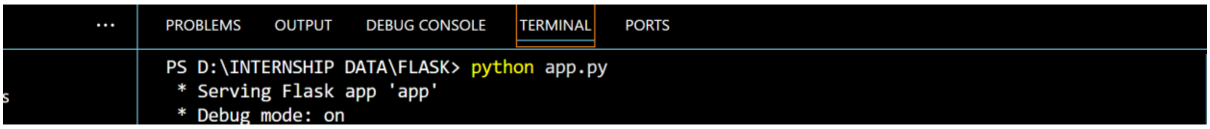
4. Define Routes:
   Define routes in Flask for the home page and prediction page:

```
@app.route(‘/’)
def home():
    return render_template(‘index.html’)

@app.route(‘/predict’, methods = [‘POST’])
def predict():
    features = [float(x) for x in request.form.values()]
    final_features = [np.array(features)]
    prediction = model.predict(final_features)
    output = prediction[0]
    return render_template(‘index.html’, prediction_text = f’Predicted Iris Class: {output}’)
```

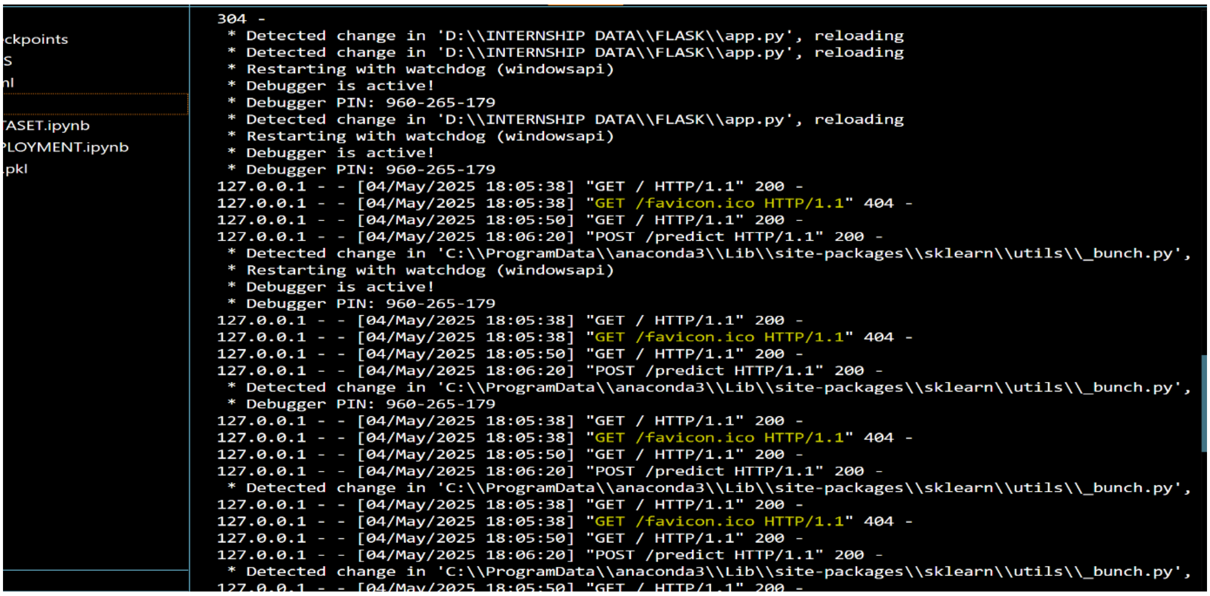


```html
<!DOCTYPE html>
<html>
<head>
    <title>Iris Predictor</title>
</head>
<body>
    <h2>Enter Iris Features</h2>
    <form action="/predict" method="post">
        <input type="text" name="sepal length (cm)" placeholder="Sepal Length">
        <input type="text" name="sepal width (cm)" placeholder = "Sepal Width">
        <input type="text" name="petal length (cm)" placeholder = "Petal Length">
        <input type="text" name="petal width (cm)" placeholder = "Petal Width">
        <input type="submit" value="Predict">
    </form>


    {% if prediction_text %}
        <h3>{{ prediction_text }}</h3>
    {% endif %}
</body>
</html>

```

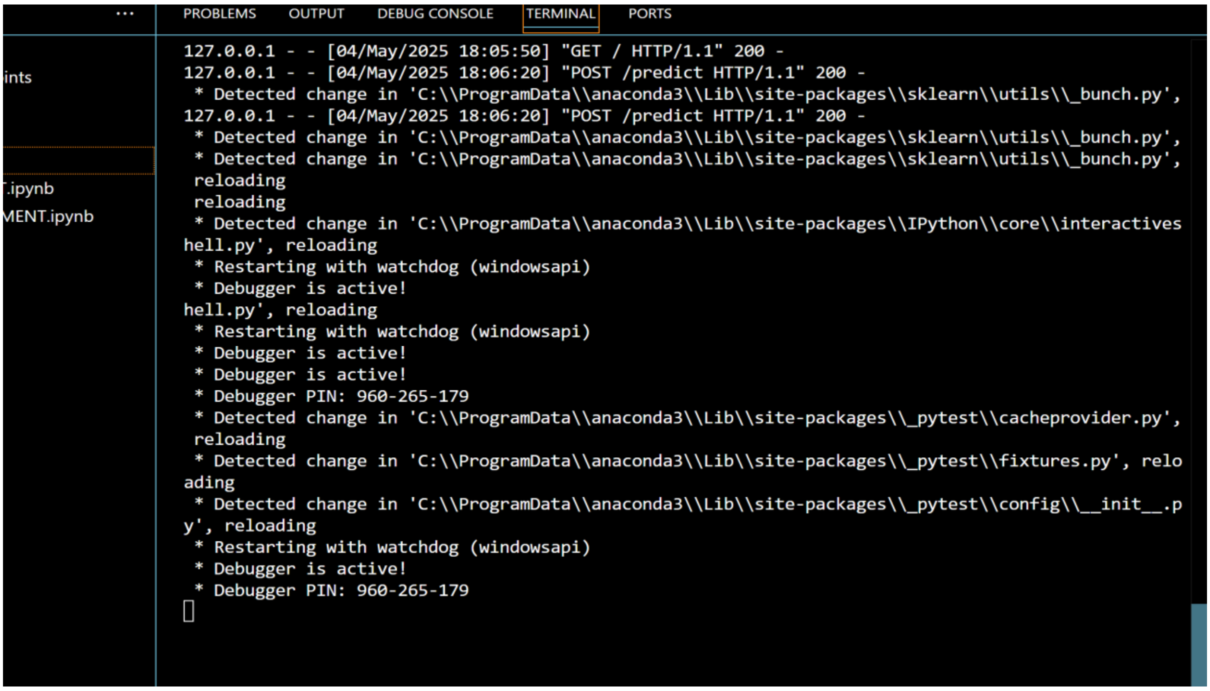
5. Run the Flask App:
   In the terminal, navigate to the folder containing app.py and run:
   python app.py



```
PS D:\INTERNSHIP DATA\FLASK> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
```

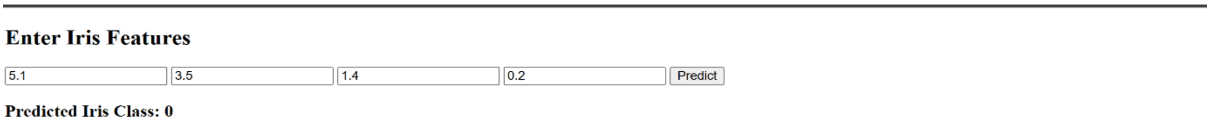


```
304 -
 * Detected change in 'D:\\INTERNSHIP DATA\\FLASK\\app.py', reloading
 * Detected change in 'D:\\INTERNSHIP DATA\\FLASK\\app.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 960-265-179
 * Detected change in 'D:\\INTERNSHIP DATA\\FLASK\\app.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 960-265-179
127.0.0.1 - - [04/May/2025 18:05:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:05:38] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2025 18:05:50] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:06:20] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 960-265-179
127.0.0.1 - - [04/May/2025 18:05:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:05:38] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2025 18:05:50] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:06:20] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
 * Debugger PIN: 960-265-179
127.0.0.1 - - [04/May/2025 18:05:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:05:38] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2025 18:05:50] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:06:20] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
127.0.0.1 - - [04/May/2025 18:05:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:05:38] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2025 18:05:50] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:06:20] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
127.0.0.1 - - [04/May/2025 18:05:50] "GET / HTTP/1.1" 200 -
```



```
127.0.0.1 - - [04/May/2025 18:05:50] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2025 18:06:20] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
127.0.0.1 - - [04/May/2025 18:06:20] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\sklearn\\utils\\_bunch.py',
 reloading
 reloading
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\IPython\\core\\interactives
hell.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
hell.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger is active!
 * Debugger PIN: 960-265-179
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\_pytest\\cacheprovider.py',
 reloading
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\_pytest\\fixtures.py', relo
ading
 * Detected change in 'C:\\ProgramData\\anaconda3\\Lib\\site-packages\\_pytest\\config\\__init__.p
y', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 960-265-179
```

6. Access the Web App:
   Open a web browser and go to http://127.0.0.1:5000. The form should appear, allowing you to enter the Iris flower features and receive predictions.



Enter Iris Features

5.1	3.5	1.4	0.2	Predict

Predicted Iris Class: 0

Conclusion:

This document presented a comprehensive, step-by-step guide to deploying an Iris flower classification model using Flask. It covered model loading, web form creation, route definition, and running the application locally. By following these steps, I developed a fully functional web application that allows users to input Iris flower features and receive real-time predictions. The deployment process not only demonstrates the practical application of machine learning models but also provides hands-on experience in building and serving models through a simple web interface.

Submitted by:

Ankita Roy

Batch Code: LISUM44

Submission Date: 29/04/25

Submitted to: Data Glacier