Week 4: Deployment on Flask

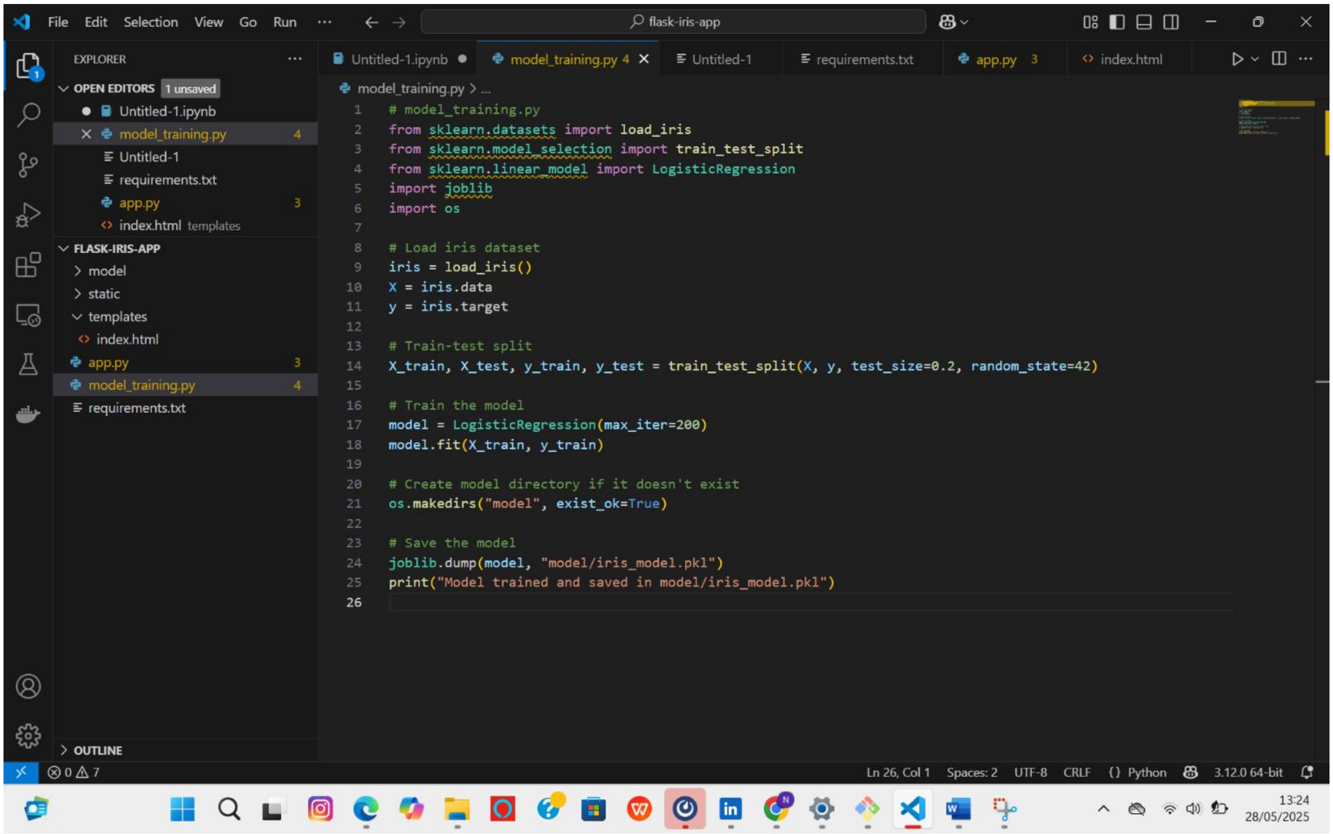
Name: Deployment on Flask
Report date: 28-May-2025
Internship Batch: LISUM45 Version:1.0
Data intake by: Najma Abdi Mohamed
Data intake reviewer : Data Glacier
Data storage location:
https://github.com/najmaabdi99/Data-Glacier-Internship---Flask-App-Deployment.git

Tabular data details:

Total number of observations	150 rows (each for one flower sample)
Total number of files	1
Total number of features	4
Base format of the file	.csv
Size of the data	5 KB

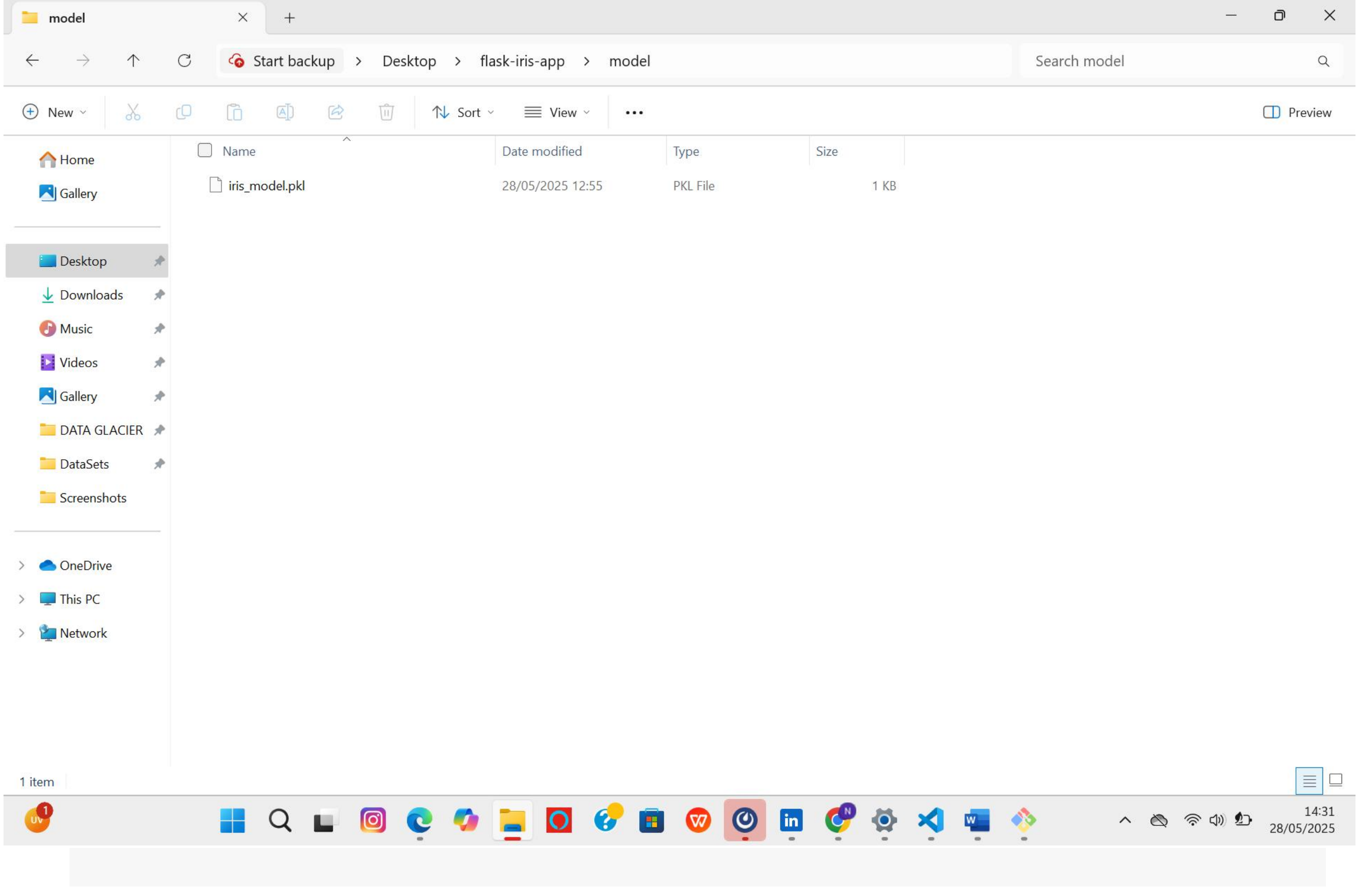
1- Building Model Training and Saving.

Here, iris dataset was loaded, train-test split done, model trained and model saved.



```python
# model_training.py
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import joblib
import os

# Load iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Create model directory if it doesn't exist
os.makedirs("model", exist_ok=True)

# Save the model
joblib.dump(model, "model/iris_model.pkl")
print("Model trained and saved in model/iris_model.pkl")
```

1.2 Saved Model



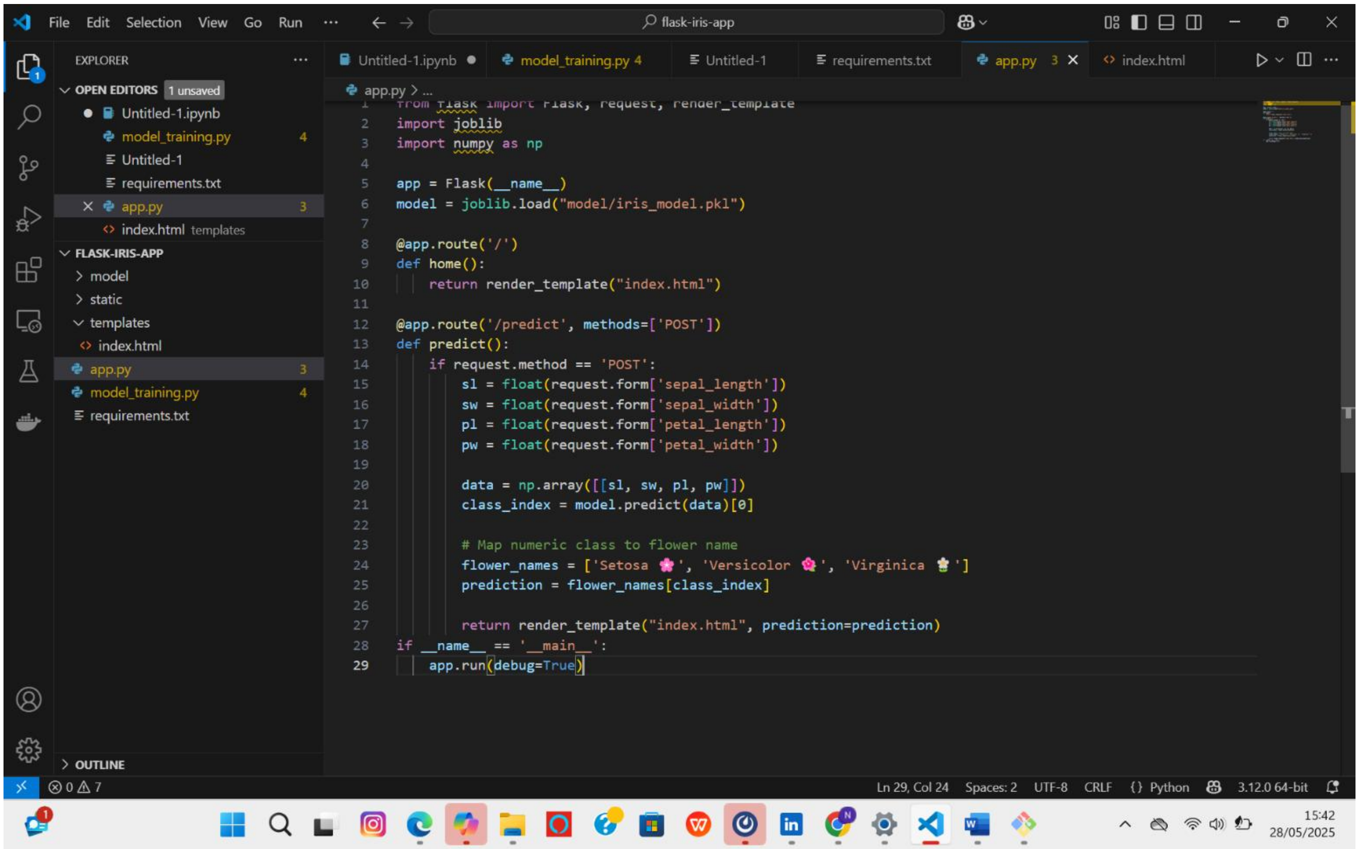
Name	Date modified	Type	Size
iris_model.pkl	28/05/2025 12:55	PKL File	1 KB

2- Deploying The Model on Flask (Web App)

2.1 app.py

This is the main Flask backend file. It:

- Loads the saved model (model/iris_model.pkl)
- Sets up routes (/ for form, /predict for predictions)
- Handles user input and displays the prediction result

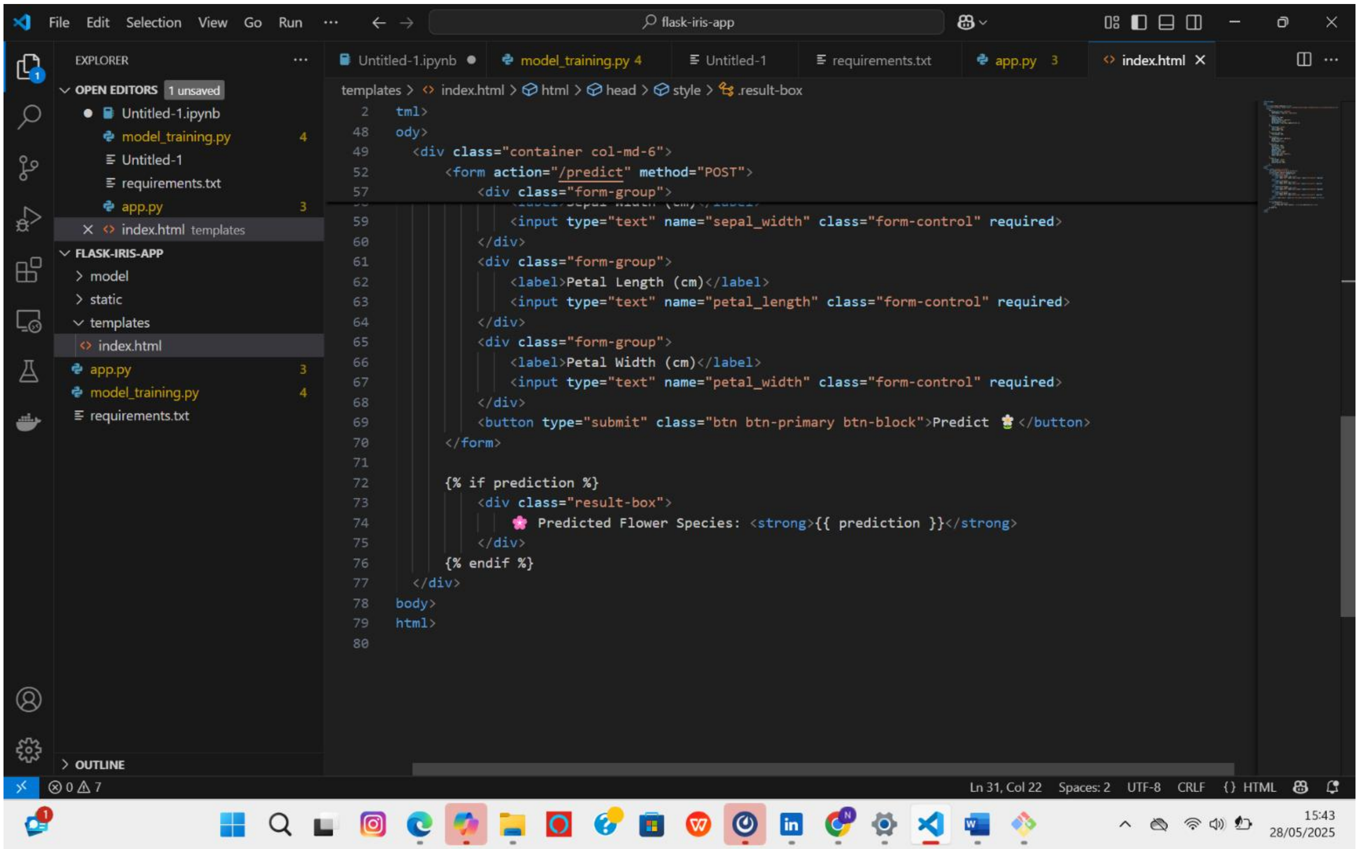


```python
from flask import Flask, request, render_template
import joblib
import numpy as np

app = Flask(__name__)
model = joblib.load("model/iris_model.pkl")

@app.route('/')
def home():
    return render_template("index.html")

@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        sl = float(request.form['sepal_length'])
        sw = float(request.form['sepal_width'])
        pl = float(request.form['petal_length'])
        pw = float(request.form['petal_width'])

        data = np.array([[sl, sw, pl, pw]])
        class_index = model.predict(data)[0]

        # Map numeric class to flower name
        flower_names = ['Setosa 🌸', 'Versicolor 🌺', 'Virginica 🌼']
        prediction = flower_names[class_index]

        return render_template("index.html", prediction=prediction)
if __name__ == '__main__':
    app.run(debug=True)
```

2.2 index.html

This is the frontend HTML page that:

- Collects user input for sepal/petal measurements
- Sends it to Flask via a POST request
- Displays the predicted iris class on the same page

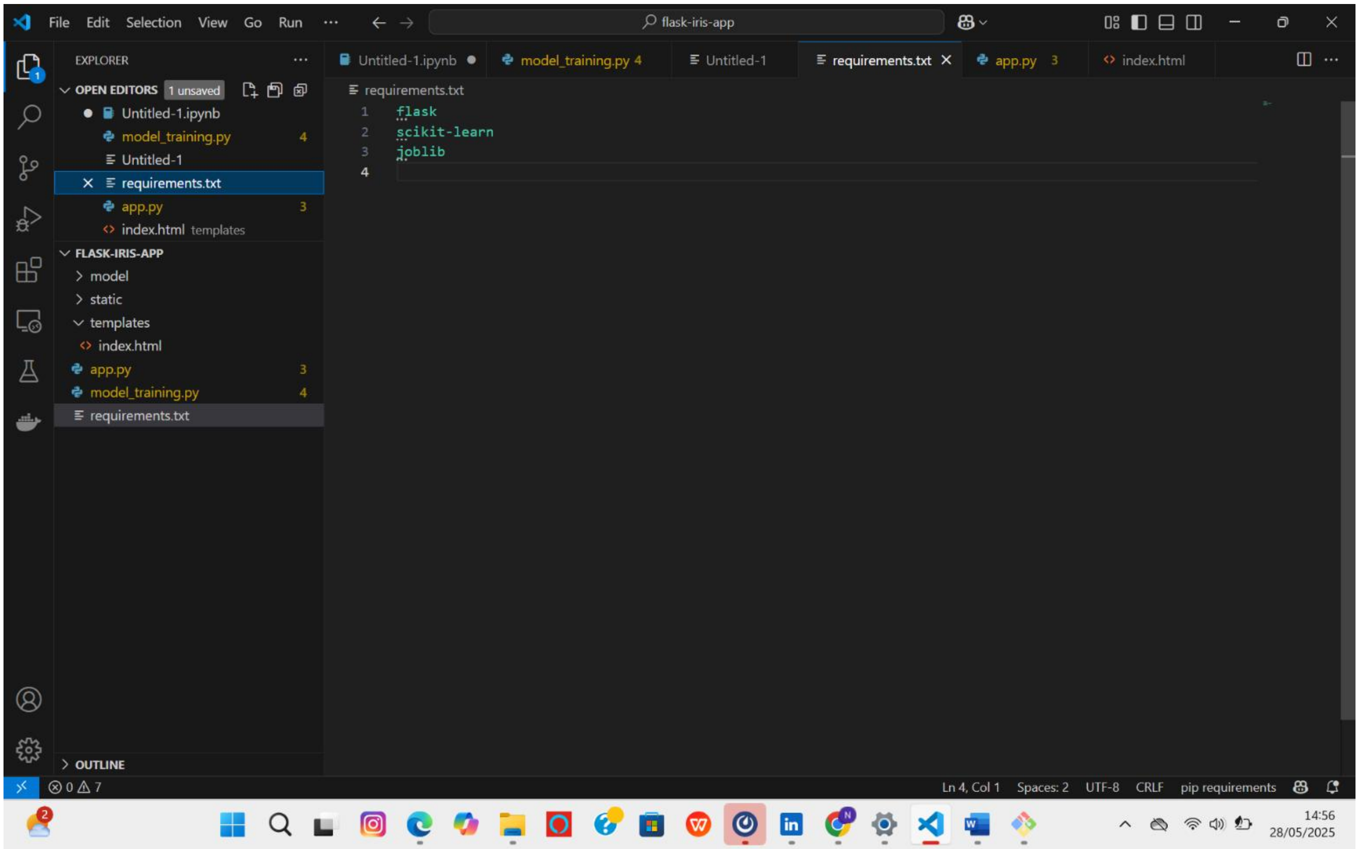


```
tml>
ody>
  <div class="container col-md-6">
      <form action="/predict" method="POST">
          <div class="form-group">
              <input type="text" name="sepal_width" class="form-control" required>
          </div>
          <div class="form-group">
              <label>Petal Length (cm)</label>
              <input type="text" name="petal_length" class="form-control" required>
          </div>
          <div class="form-group">
              <label>Petal Width (cm)</label>
              <input type="text" name="petal_width" class="form-control" required>
          </div>
          <button type="submit" class="btn btn-primary btn-block">Predict 🌼</button>
      </form>

      {% if prediction %}
          <div class="result-box">
              🌸 Predicted Flower Species: <strong>{{ prediction }}</strong>
          </div>
      {% endif %}
  </div>
body>
html>
```

2.3 requirements.txt

This file lists all the Python packages needed to run the app, such as:

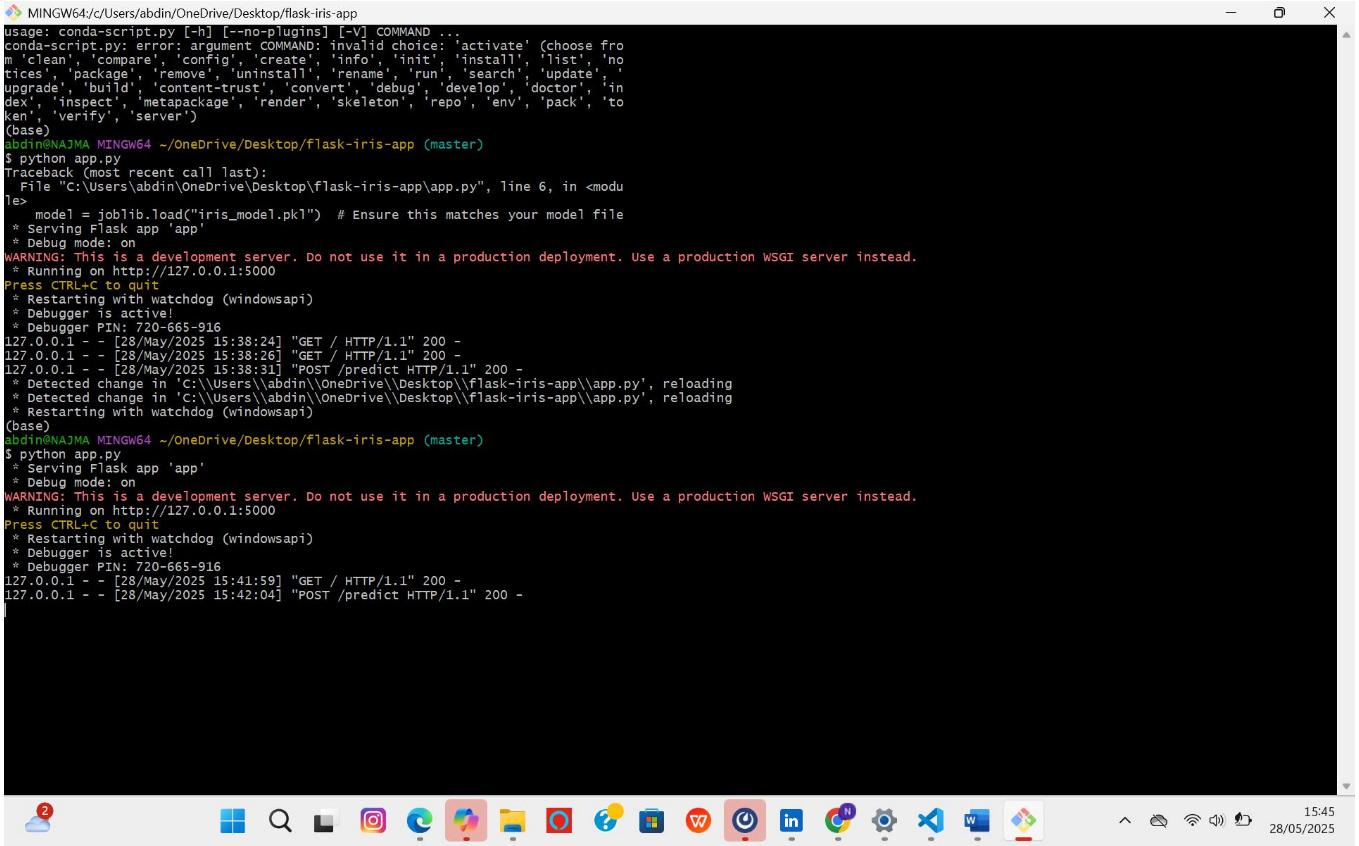
- Flask for the web server
- scikit-learn for the model
- joblib for loading the saved model



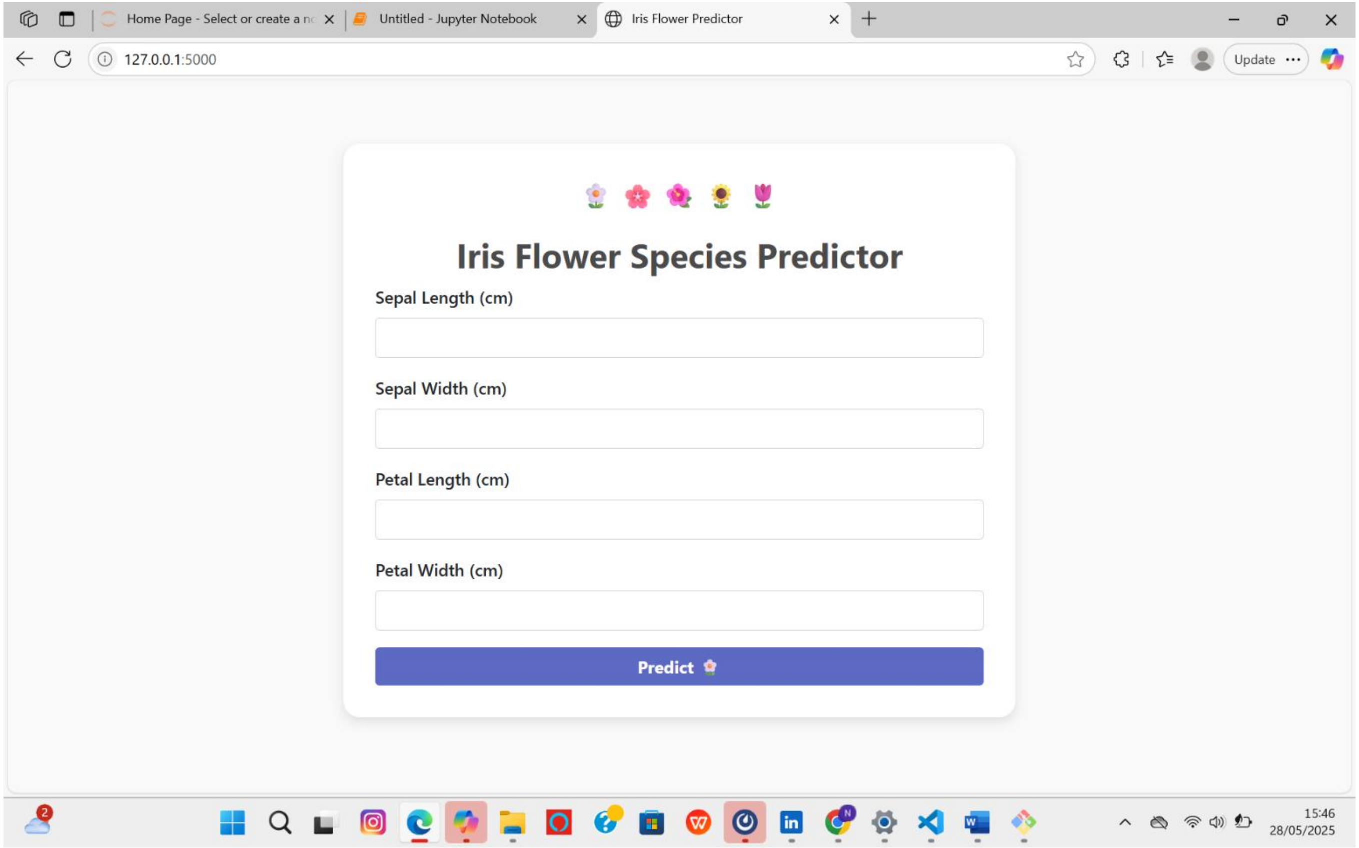
requirements.txt

```
flask
scikit-learn
joblib

```

2.4 Running Flask Code



```
usage: conda-script.py [-h] [--no-plugins] [-V] COMMAND ...
conda-script.py: error: argument COMMAND: invalid choice: 'activate' (choose fro
m 'clean', 'compare', 'config', 'create', 'info', 'init', 'install', 'list', 'no
tices', 'package', 'remove', 'uninstall', 'rename', 'run', 'search', 'update', '
upgrade', 'build', 'content-trust', 'convert', 'debug', 'develop', 'doctor', 'in
dex', 'inspect', 'metapackage', 'render', 'skeleton', 'repo', 'env', 'pack', 'to
ken', 'verify', 'server')
(base)
abdin@NAJMA MINGW64 ~/OneDrive/Desktop/flask-iris-app (master)
$ python app.py
Traceback (most recent call last):
  File "C:\Users\abdin\OneDrive\Desktop\flask-iris-app\app.py", line 6, in <modu
le>
    model = joblib.load("iris_model.pkl")  # Ensure this matches your model file
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 720-665-916
127.0.0.1 - - [28/May/2025 15:38:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 15:38:26] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 15:38:31] "POST /predict HTTP/1.1" 200 -
 * Detected change in 'C:\\Users\\abdin\\OneDrive\\Desktop\\flask-iris-app\\app.py', reloading
 * Detected change in 'C:\\Users\\abdin\\OneDrive\\Desktop\\flask-iris-app\\app.py', reloading
 * Restarting with watchdog (windowsapi)
(base)
abdin@NAJMA MINGW64 ~/OneDrive/Desktop/flask-iris-app (master)
$ python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 720-665-916
127.0.0.1 - - [28/May/2025 15:41:59] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 15:42:04] "POST /predict HTTP/1.1" 200 -
```



🌸 🌸 🌺 🌻 🌷

Iris Flower Species Predictor

Sepal Length (cm)

Sepal Width (cm)

Petal Length (cm)

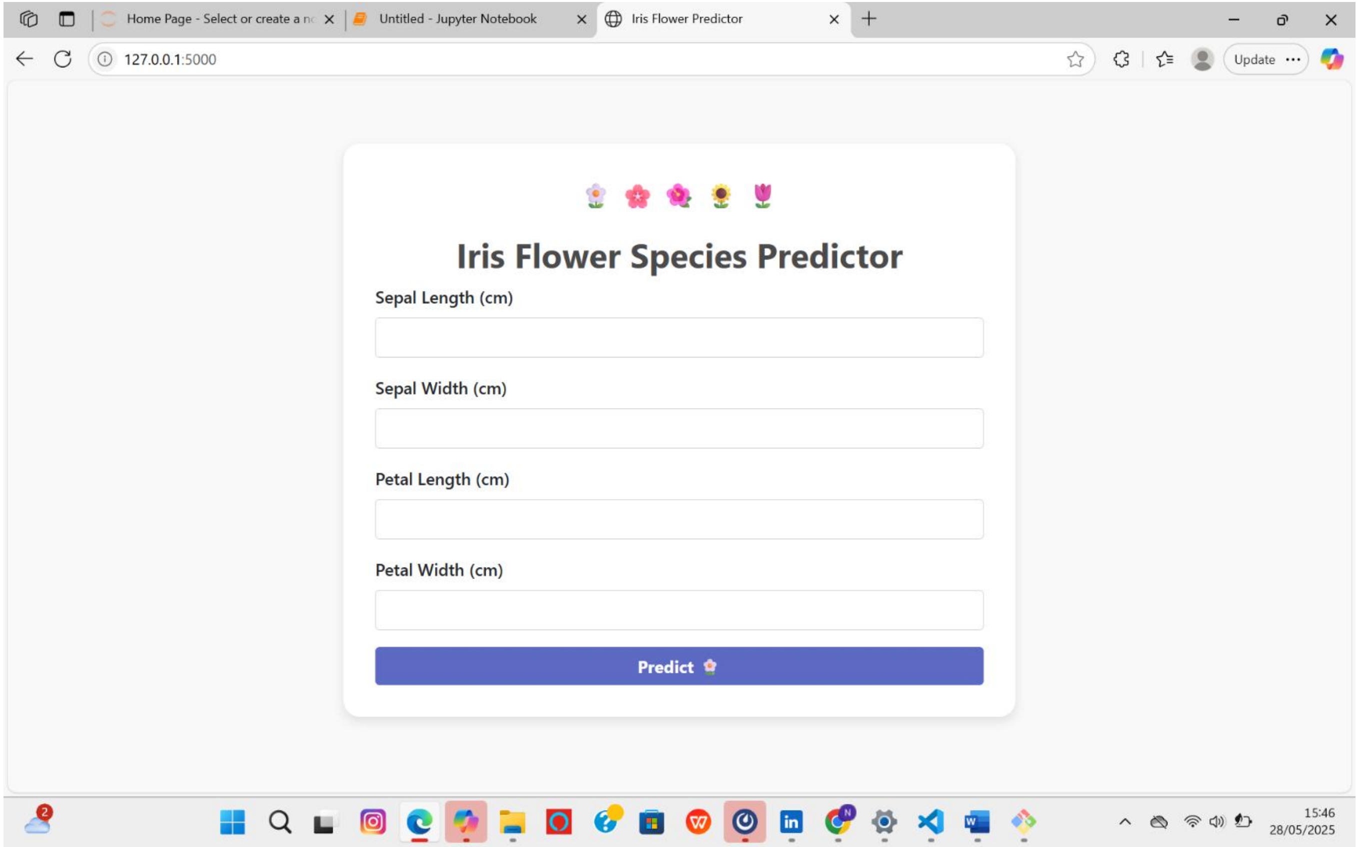
Petal Width (cm)

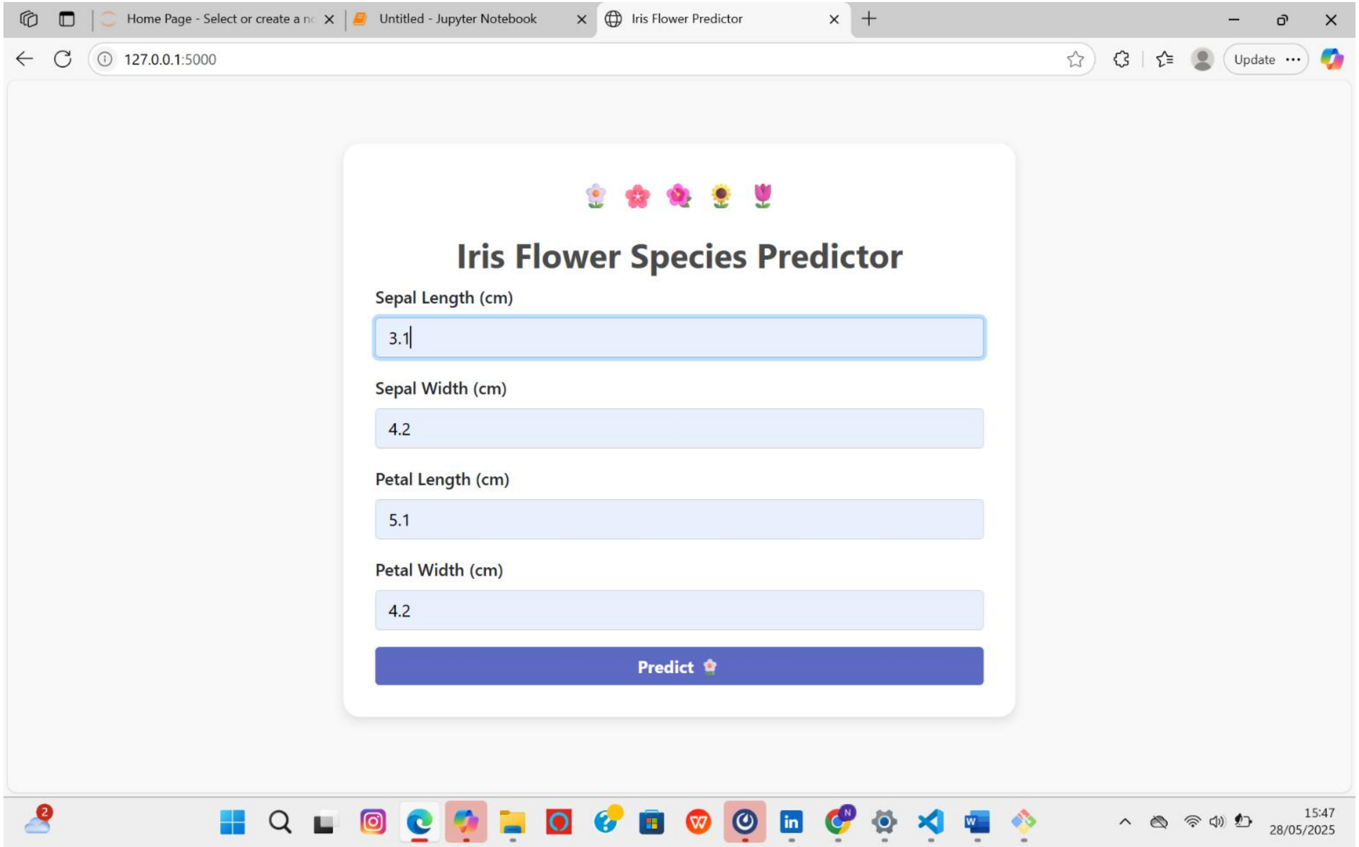
2.5 Prediction Output

This section showcases the final output of the web application after submitting user input through the form.

1. After entering flower feature values (sepal and petal measurements) into the form on the homepage (index.html), the user clicks the "Predict" button.
2. The input is sent to the Flask backend, where the model makes a prediction.
3. The predicted Iris flower species (e.g., Setosa, Versicolor, or Virginica) is displayed dynamically on the same page below the form.

This confirms the model is working as expected and integrated correctly with the web interface.





🌸 🌸 🌺 🌻 🌷

Iris Flower Species Predictor

Sepal Length (cm)

3.1

Sepal Width (cm)

4.2

Petal Length (cm)

5.1

Petal Width (cm)

4.2

Predict 🌸