Week 5: Cloud and API Deployment

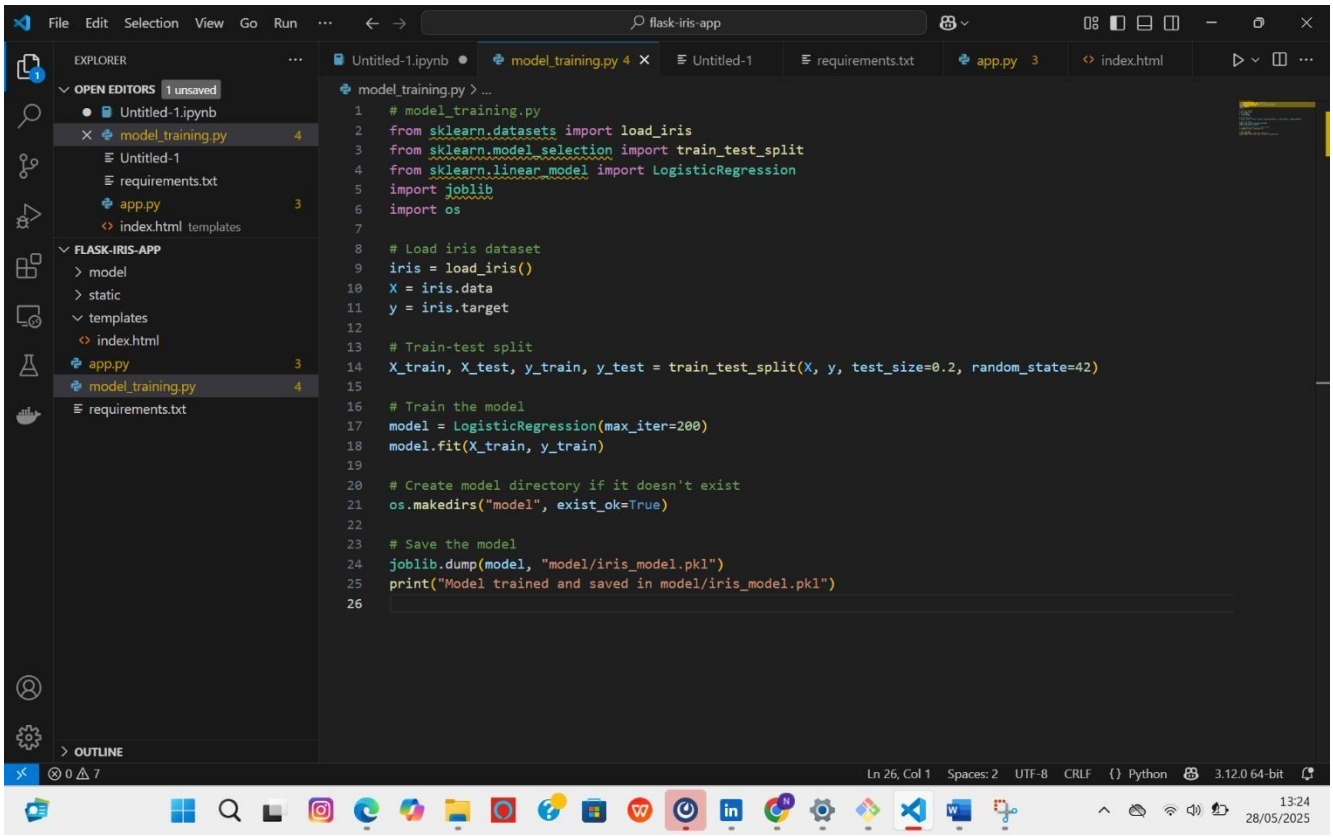
Name: Deployment on Flask
Report date: 05-June-2025
Internship Batch: LISUM45 Version:1.0
Data intake by: Najma Abdi Mohamed
Data intake reviewer : Data Glacier
Data storage location:
https://github.com/najmaabdi99/Data-Glacier-Internship---Flask-App-Deployment.git

Tabular data details:

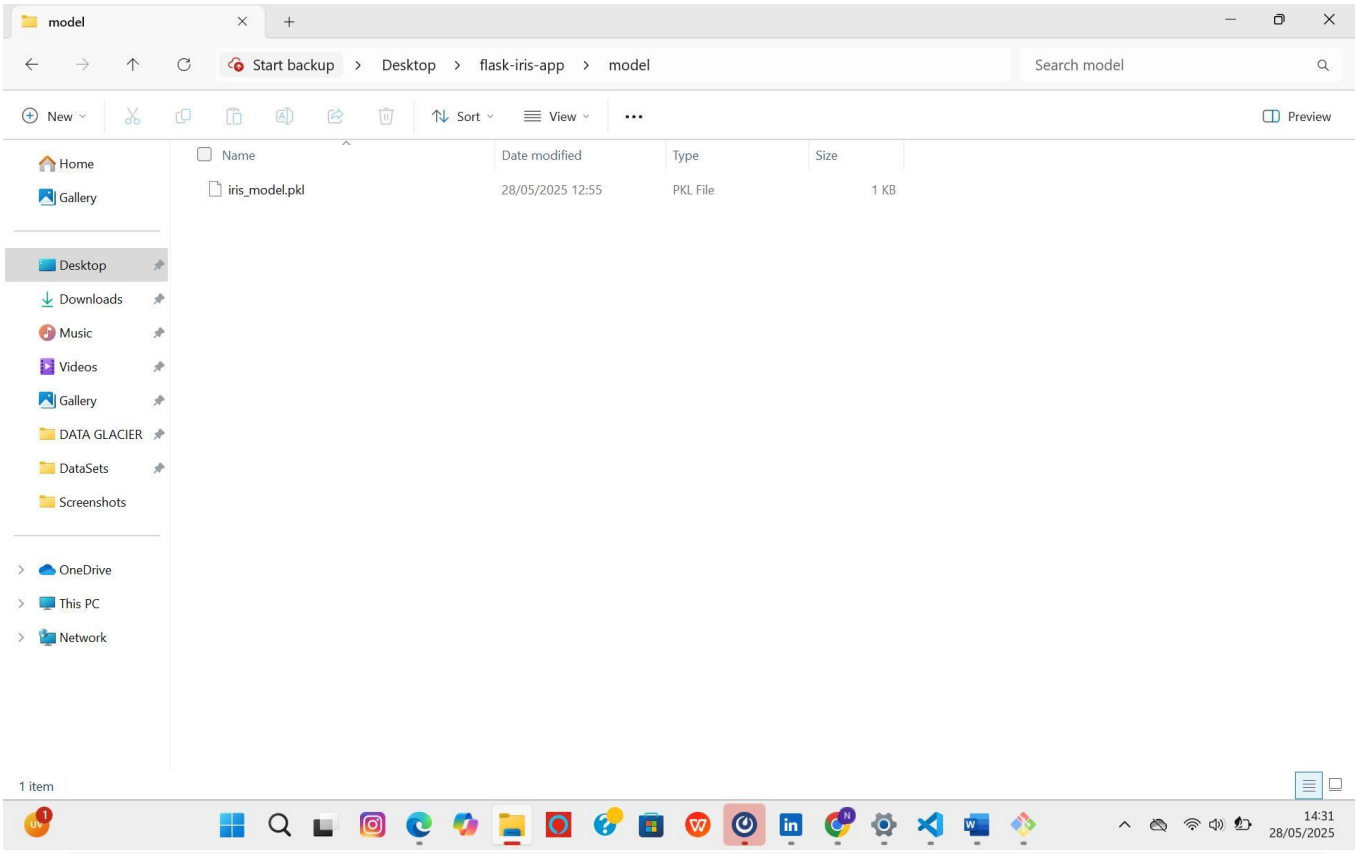
Total number of observations	150 rows (each for one flower sample)
Total number of files	1
Total number of features	4
Base format of the file	.csv
Size of the data	5 KB

1- Building Model Training and Saving.

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



1.2 Saved Model

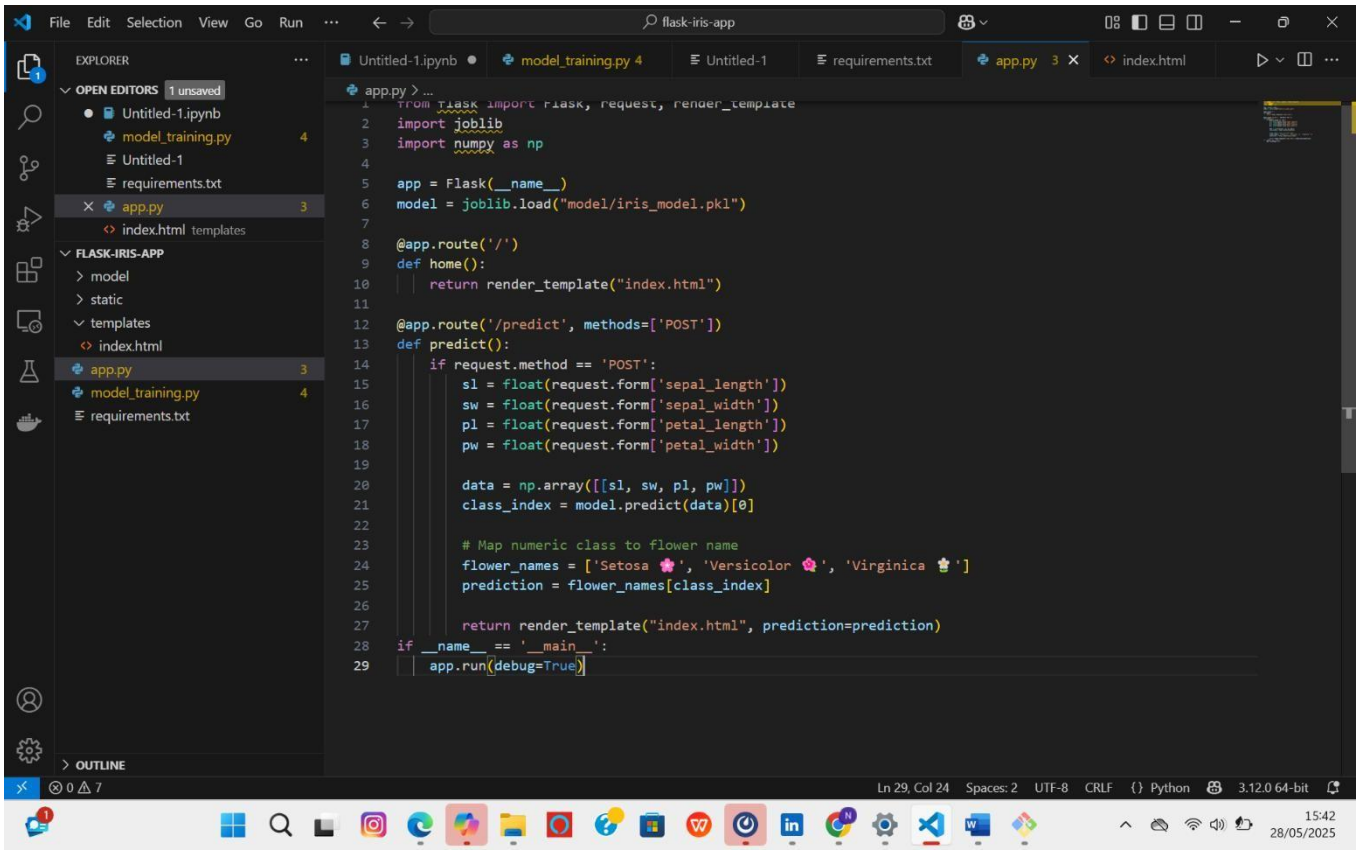


2- Deploying The Model on Flask (Web App)

2.1 app.py

This is the main Flask backend file. It:

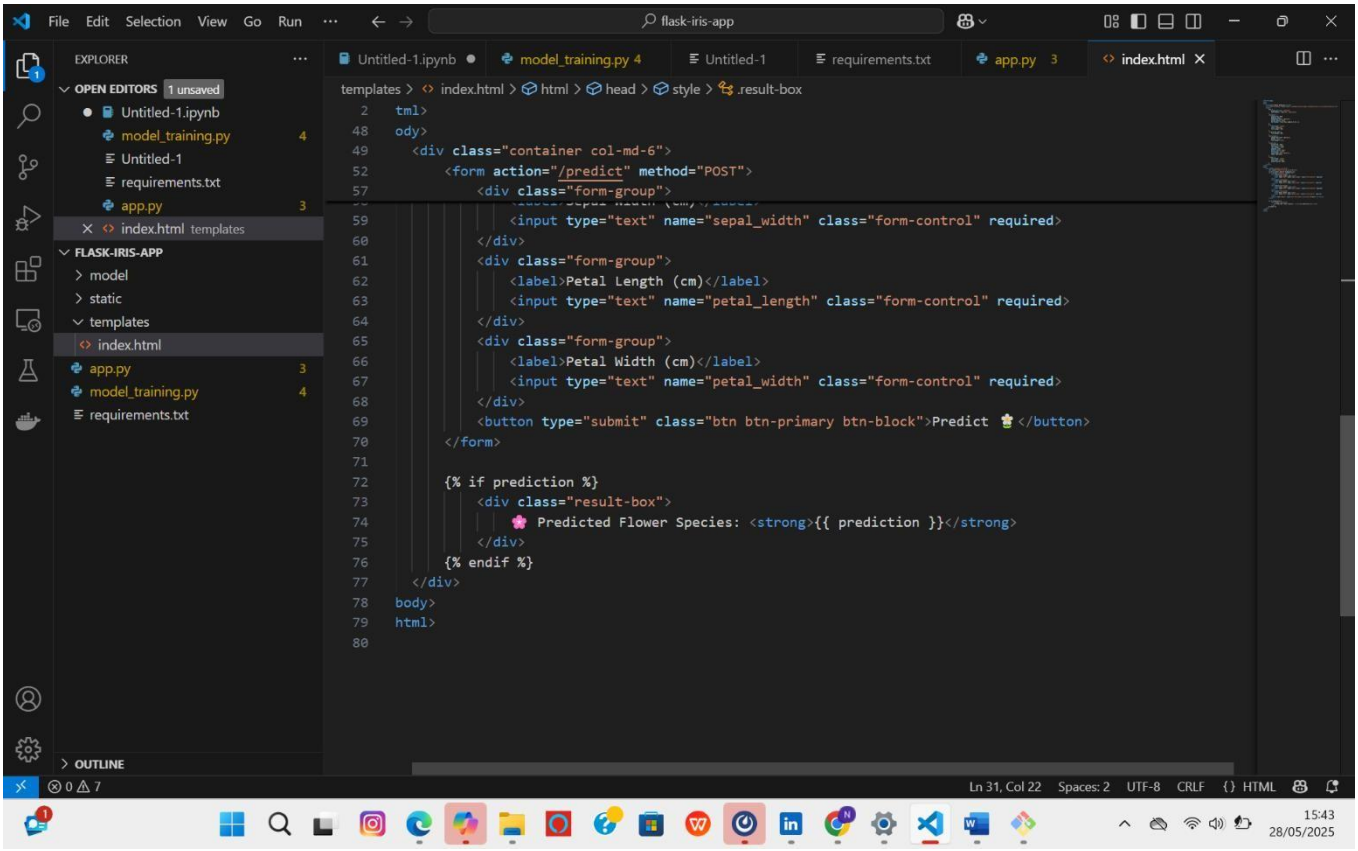
- Loads the saved model (model/iris_model.pkl)
- Sets up routes (/ for form, /predict for predictions)
- Handles user input and displays the prediction result



2.2 index.html

This is the frontend HTML page that:

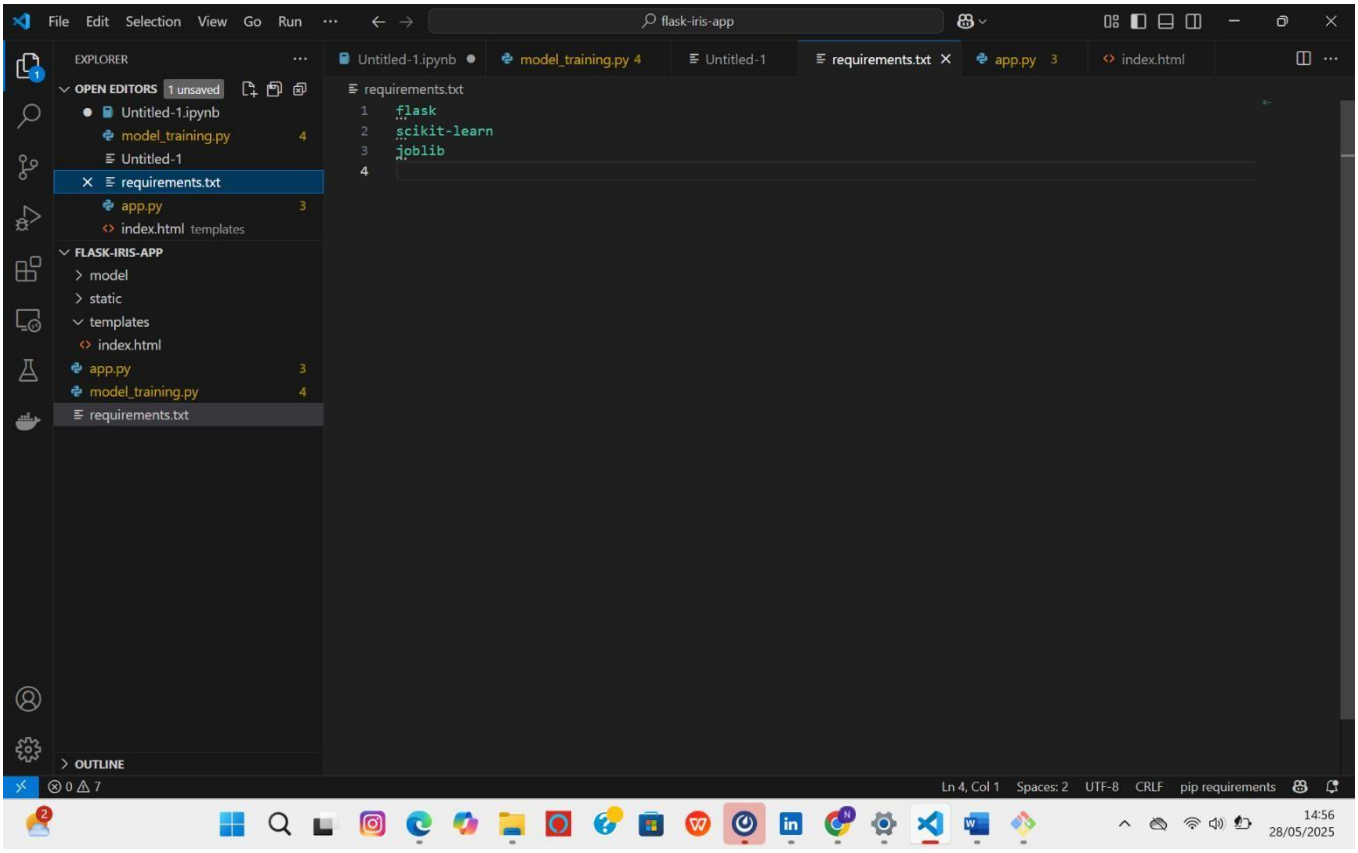
- Collects user input for sepal/petal measurements
- Sends it to Flask via a POST request
- Displays the predicted iris class on the same page



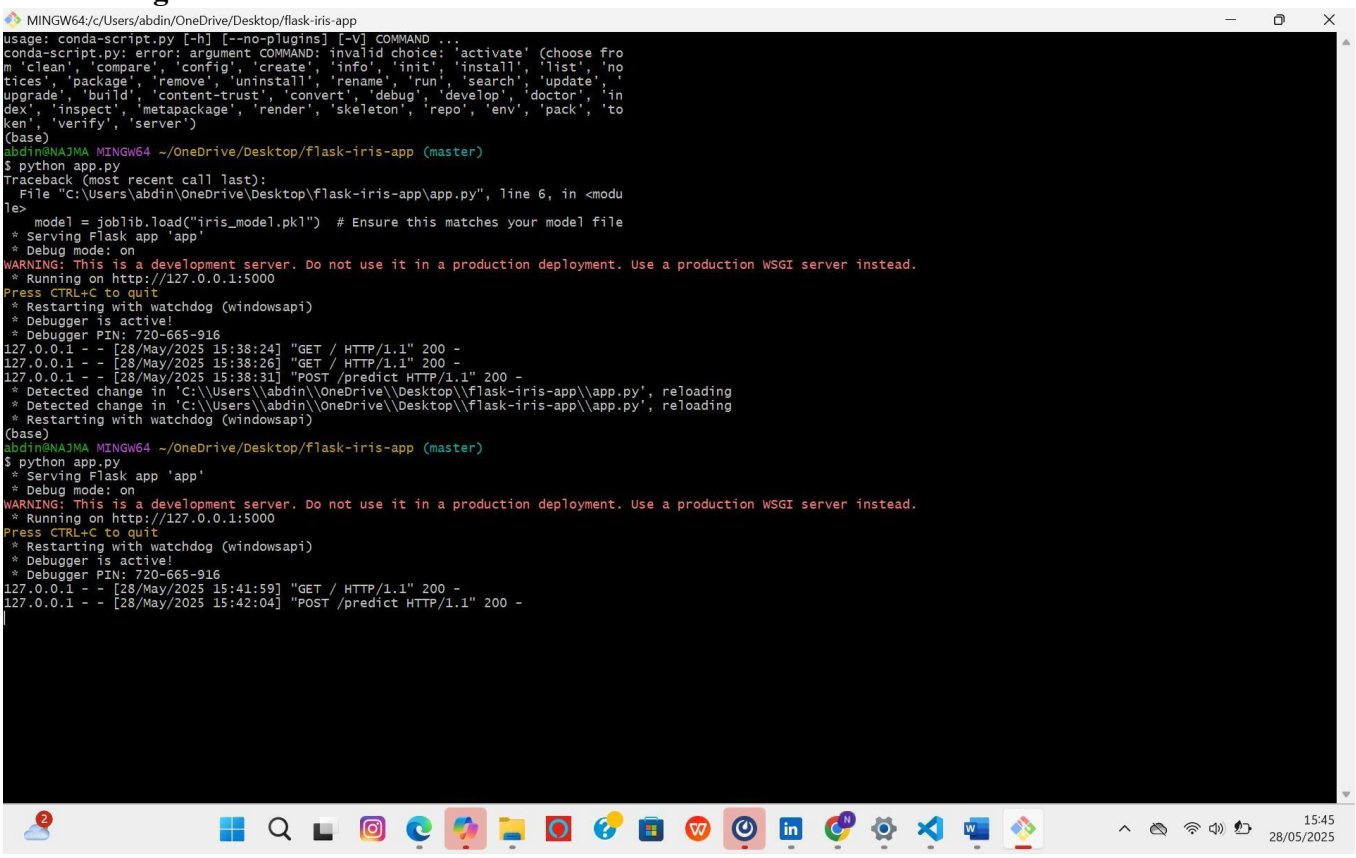
2.3 requirements.txt

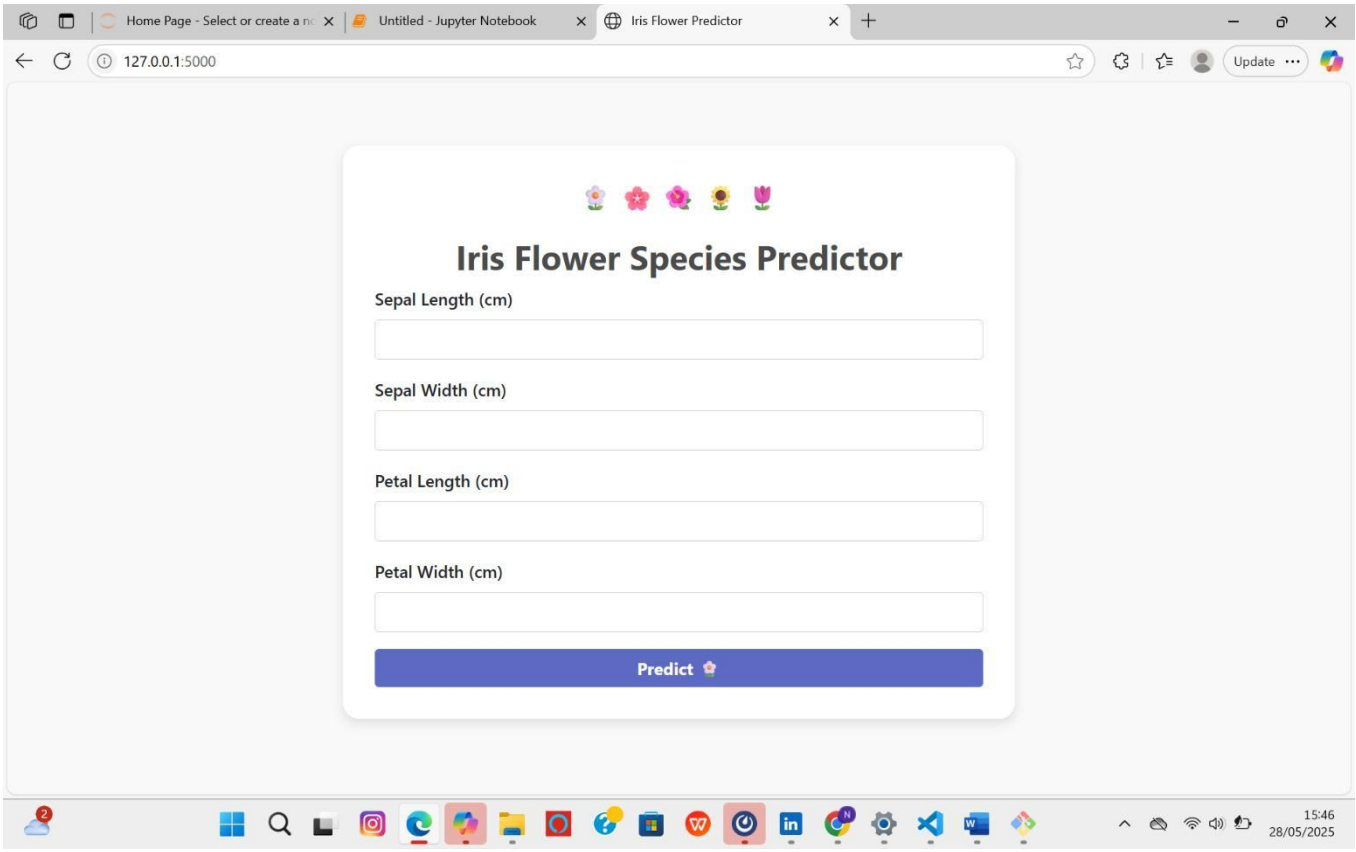
This file lists all the Python packages needed to run the app, such as:

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


File Edit Selection View Go Run
flask-iris-app
EXPLORER
OPEN EDITORS 1 unsaved
Untitled-1.ipynb
model_training.py 4
Untitled-1
requirements.txt
app.py 3
index.html templates
FLASK-IRIS-APP
model
static
templates
index.html
app.py 3
model_training.py 4
requirements.txt
OUTLINE
Untitled-1.ipynb
model_training.py 4
Untitled-1
requirements.txt
app.py 3
index.html
requirements.txt
1 flask
2 scikit-learn
3 joblib
4
0 7
Ln 4, Col 1 Spaces: 2 UTF-8 CRLF pip requirements
14:56
28/05/2025

2.4 Running Flask Code



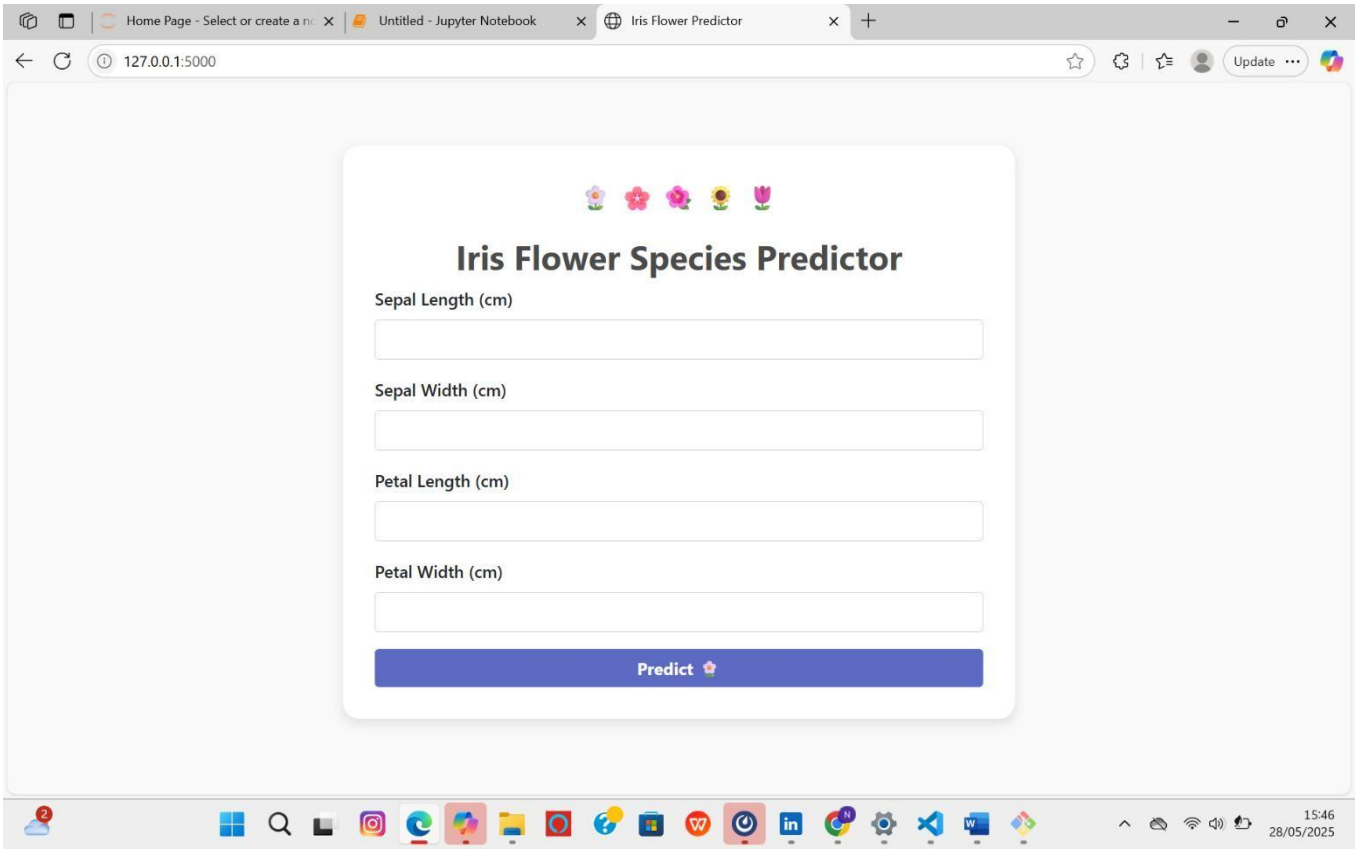


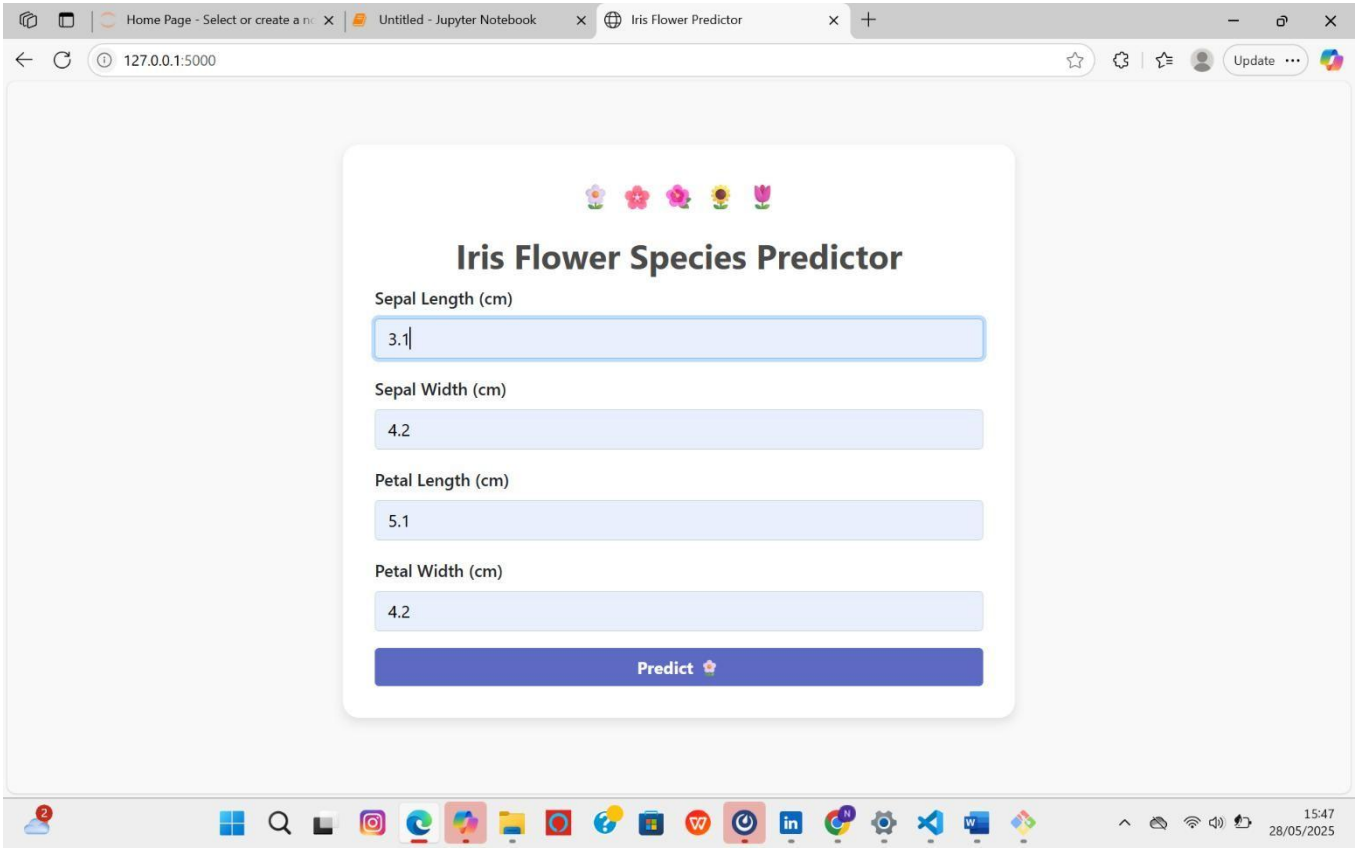
2.5 Prediction Output

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.




Home Page - Select or create a n
Untitled - Jupyter Notebook
Iris Flower Predictor
127.0.0.1:5000
Update
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
15:47
28/05/2025

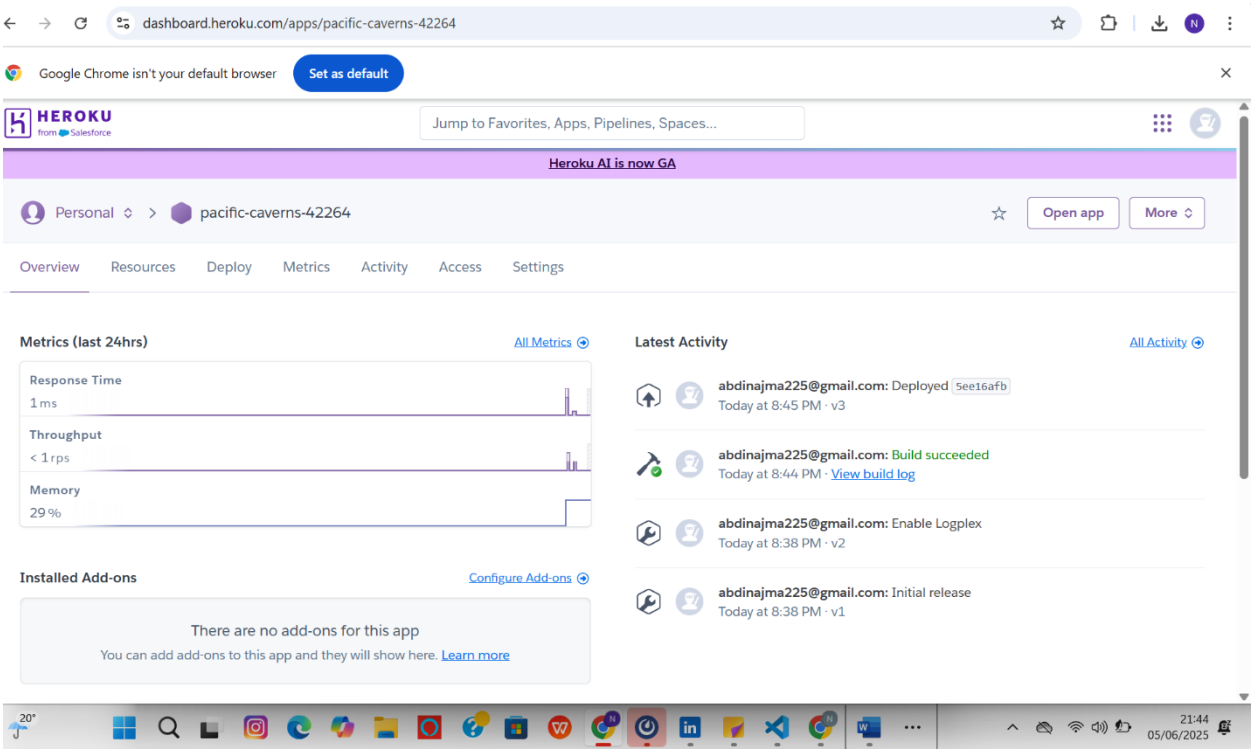
5. Model deployment using Heroku

We're ready to start our Heroku deployment now that our model has been trained, the machine learning pipeline has been set up, and the application has been tested locally. To upload the application source code onto Heroku i went through the following process

- Initialize a Git repo in your project folder
- Add Heroku as a remote.
- Commit code locally.
- Push my code to Heroku's remote repo.

3.1 Cloud Dashboard Screenshot

This screenshot shows the Heroku Dashboard with the app name (pacific-caverns-42264) and status (Deployed). It confirms that the app is deployed on Heroku.



3.2 APP URL SCREENSHOT

This screenshot shows the app's landing page opened in a browser using the URL provided by Heroku. It confirms that the app is live and accessible.