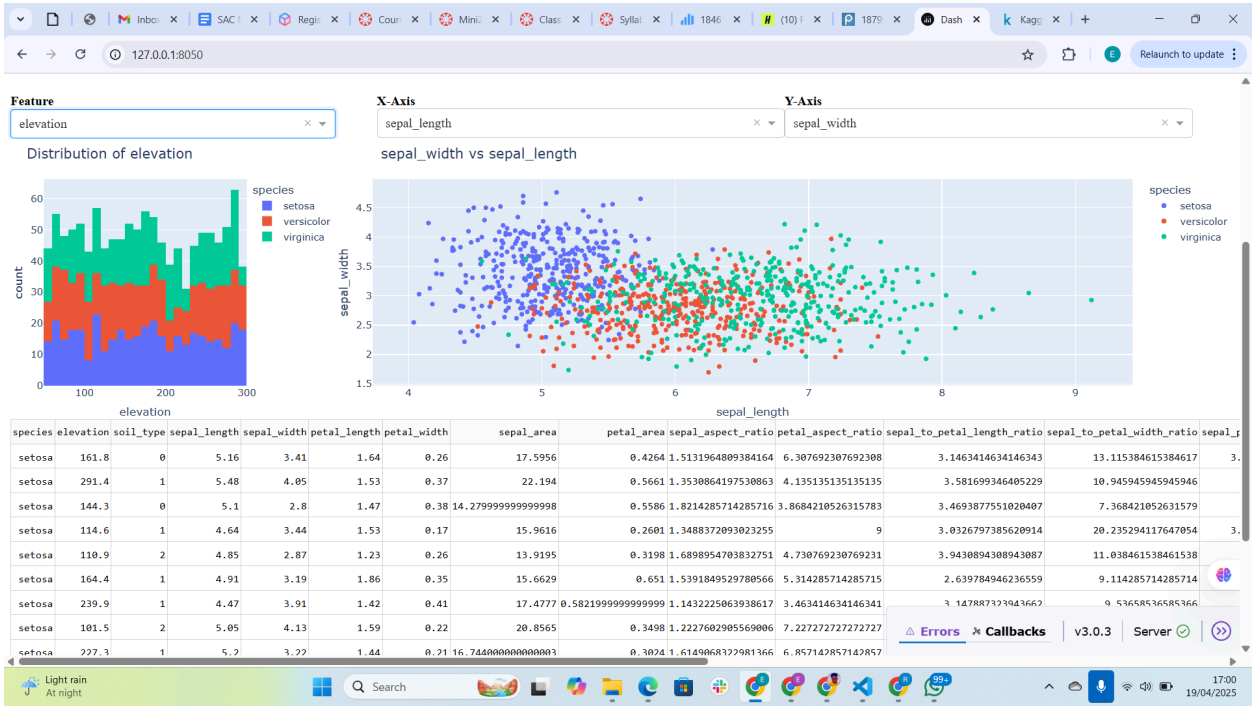
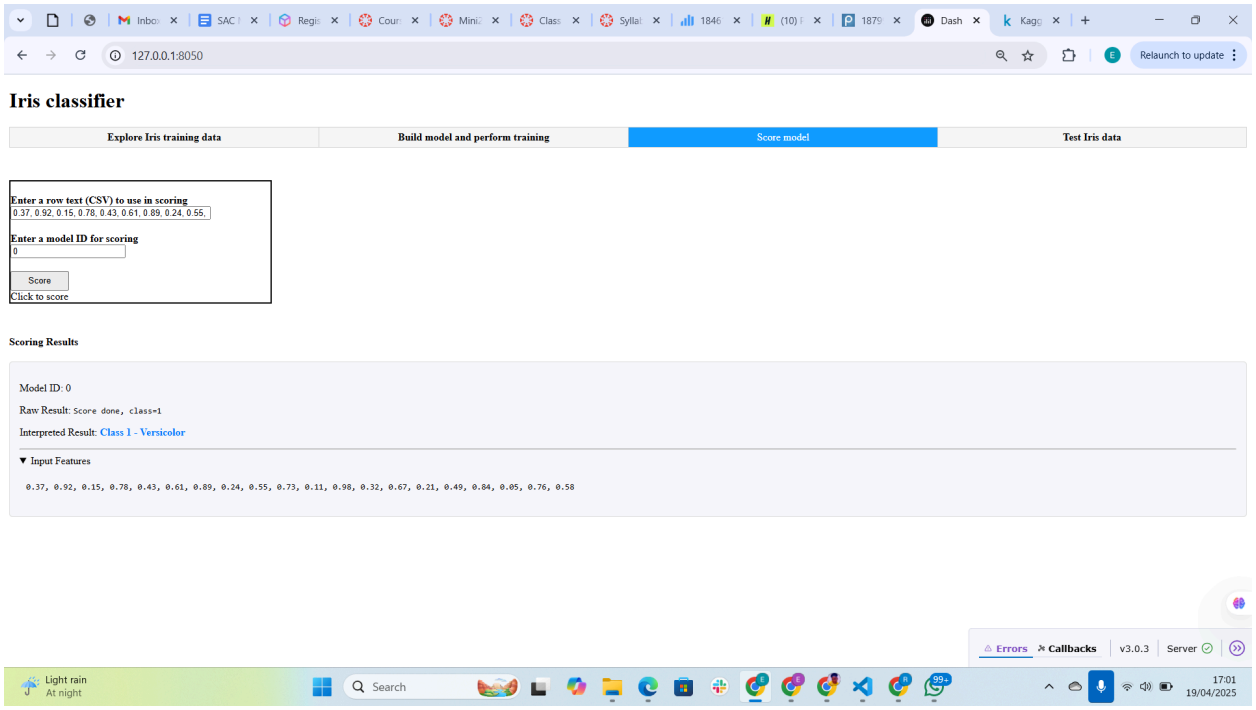
Screenshots

1. Exploring Iris Training Data

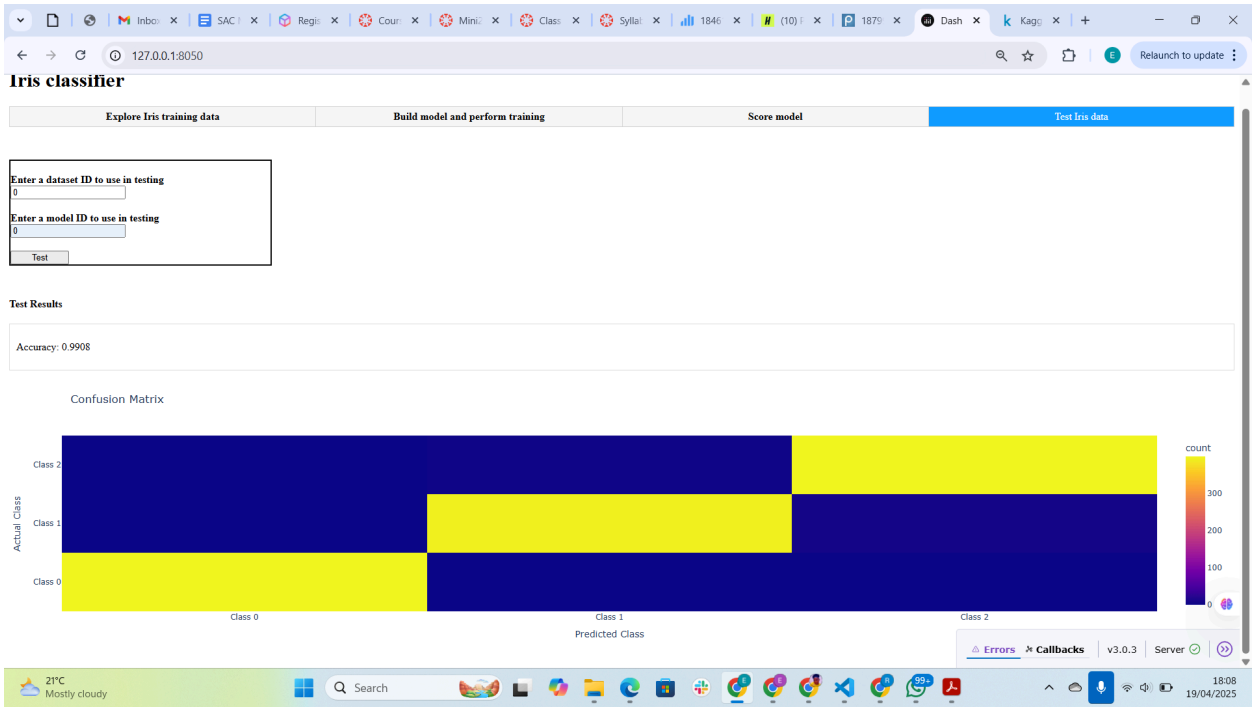


2. Build Model and Perform Training

3. Score Model



4. Test Iris Data



Answers to the Questions

1. What did you find easy to do in Dash?

I found it relatively straightforward to implement callbacks for interactive graph components, both during the Dash tutorials and in the lab. Creating visualizations like histograms and scatter plots using Plotly Express was intuitive. Additionally, using Dash's layout components to organize the tabs and sections made the interface setup more manageable.

2. What was challenging or didn't work as expected?

One of the main challenges was managing the backend communication during model training and dataset uploads. I encountered connection errors when trying to POST data to the backend, which interrupted the workflow. Implementing certain callbacks—especially those involving dynamic updates and annotations—was also tricky and required careful debugging.

3. What additional components or improved dashboard design would better explain the Iris model to a client?

A dedicated Model Metrics tab displaying key performance indicators such as F1 score, accuracy, precision, and recall per class would provide clearer insights into the model's behavior.

A Feature Importance visualization (e.g., a bar chart) would help users understand which features most influence the model's predictions.

Adding a Confusion Matrix visualization would also support model interpretability by showing how often each class is correctly or incorrectly predicted.

4. Can you think of better ways to link the backend Iris model with the frontend Dash interface?

Since this was my first time integrating a backend with a frontend using Dash, I'm still exploring best practices. However, based on my experience and research, some improvements could include:

Using GraphQL APIs instead of REST to allow more efficient and precise data fetching tailored to the frontend's needs.

Incorporating a task queue system (e.g., Celery with Redis) for handling long-running operations like model training, to keep the UI responsive.

Adding feedback components (such as loading indicators or success/failure notifications) to better inform users about the status of backend operations like uploading, training, or scoring.

5. Additional reflections based on the lab

Working on this lab gave me hands-on experience in full-stack ML app development. I implemented endpoints for uploading data, training a model, and scoring inputs through a Dash

frontend. I also visualized prediction results—such as actual vs. predicted values—using Plotly. These components made the dashboard more informative and interactive. Despite the challenges, this integration provided a clearer picture of how frontend tools can support and communicate machine learning results effectively.