Individual Report: Use Flask to Host a Machine Learning Model

In the modern business landscape where data-driven decision-making is paramount, machine learning models have emerged as essential tools. However, they can be complex and challenging to integrate into existing systems or applications. This report outlines the development of a web application using Flask, a Python web framework, to host and interact with a machine learning model.

Machine Learning Model Development

Our first task was to create a machine learning model. We chose the Iris dataset, a popular dataset in machine learning, which contains measurements of 150 iris flowers from three different species. Using Scikit-Learn, a popular Python library for machine learning, we implemented a RandomForestClassifier, an ensemble learning method.

The dataset was split into training and testing sets, with 80% of the data used for training and 20% for testing. After fitting the model on the training data and predicting on the testing data, we found our model to have high accuracy. Once we were satisfied with the model's performance, we saved it into a file using joblib's dump function, facilitating model reuse without retraining.

Building the Flask Web Application

Next, we created a web application to host our model using Flask. Flask is a lightweight but powerful web framework that allows for the rapid development of web applications in Python. It was chosen due to its simplicity and flexibility.

We created two routes within our Flask application. The root route (`"/"`) acted as a home page for our application, while a second route (`"/predict"`) was designed to handle requests for predictions.

To make a prediction, the model needed a set of features (input data). We set up the application to accept these data as a JSON object via a POST request. This JSON object contained a 'features' key, which mapped to a list of values to be used for prediction.

Hosting and Interacting with the Model

With our application complete, we loaded the saved model using joblib's load function. The application was set to listen for POST requests at the `"/predict"` endpoint. Upon receiving a request, the application parsed the JSON object to extract the feature values, used these values to make a prediction with the model, and returned the prediction as a JSON object.

To interact with our application, we needed to send a POST request to the `"/predict"` endpoint. This was achieved using Python's `requests` library, which allowed us to construct and send the HTTP request easily. We sent a JSON object containing a 'features' key and a list of values, and our application returned a prediction.

Challenges and Solutions

One challenge we faced was running the Flask server and sending requests to it at the same time. When started, the Flask server blocks the terminal it is running in until it is stopped. To overcome this, we ran the POST request code in a separate Python script or a different terminal instance.

Conclusion

Through this project, we successfully demonstrated the development and hosting of a machine learning model within a web application, making the model more accessible and interactive. However, it's worth noting that for a production-ready application, additional considerations such as error handling, security, user interface design, and scalability must be addressed. Nevertheless, this project forms an important stepping stone towards the broader application of machine learning in

web-based platforms.

Appendix: Codes

Machine Learning Model Development

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import joblib

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

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a RandomForestClassifier
```

```
clf = RandomForestClassifier(n_estimators=100)

# Train the model on the training set
clf.fit(X_train, y_train)

# Test the model on the test set
y_pred = clf.predict(X_test)
print('Model accuracy: ', accuracy_score(y_test, y_pred))

# Save the model to a file
joblib.dump(clf, 'model.pkl')
```

Flask Web Application Development

```python
from flask import Flask, request, jsonify
import joblib

# Create a Flask app
app = Flask(__name__)
```

```
# Load the model
model = joblib.load('model.pkl')

@app.route('/')
def home():
    return "Welcome to my prediction app!"

@app.route('/predict', methods=['POST'])
def predict():
    # Get the data from the POST request
    data = request.json
    features = data['features']

    # Make a prediction using the model
    prediction = model.predict([features])

    # Return the prediction
    return jsonify({'prediction': prediction.tolist()})

if __name__ == '__main__':
    app.run(port=5003, debug=True)
```

Interacting with the Flask Application

```python
import requests

# The data to send in the POST request
data = {"features": [5.1, 3.5, 1.4, 0.2]}

# Send a POST request to the Flask application
response = requests.post("http://localhost:5003/predict", json=data)

# Print the prediction
print(response.json())
```