FTL Global ML bootcamp 3 | Capstone project | Group 13

Malnutrition Risk Prediction ML Model

Deployment

Group members:

- Lina Ahmed
- Maha Abdalfedil
- Nada Ali
- Linda Adil

1. Overview:

The deployment phase focuses on preparing the trained machine learning model for real-world or production use. This entails saving the model in a format that allows for easy loading, establishing an environment where the model can receive input and generate predictions, and incorporating security and monitoring measures. The objective is to ensure that the model is reliably accessible to end-users or other systems while maintaining both performance and security.

The deployment phase involved creating an accessible machine-learning model after training it in Google Colab. The model was serialized and then served locally through a Flask application, enabling real-time predictions in a production-like environment. This approach allowed for easy testing and integration into applications.

2. Model Serialization:

The trained model was serialized using the **joblib** library, known for efficiently saving large machine-learning models. The model was stored in a **.pkl** (pickle) format, ensuring quick loading times in Google Colab and reducing the overall storage footprint, making it easy to manage.

3. Model Serving:

After serialization, the model was served using a local Flask application. This setup allows external applications to interact with the model for making predictions.

4. User Interface Integration:

The user interface was built using **HTML & CSS**, providing sliders for numerical inputs and buttons for prediction. The sliders allow users to easily adjust feature values, while the "Predict" button triggers the prediction function. The interface is designed to be user-friendly, enabling quick input and output without manually typing values.

- Sliders were used for continuous input features (e.g., numeric features like child stunting, wasting, etc.).

- Dropdowns were added to handle categorical inputs (e.g., country, urban/rural).

4. API Integration;

The Flask application serves as an API for the model, allowing external applications to send requests for predictions.

- **API Endpoint:**
- **POST /predict**: Accepts JSON data containing input features.

5. Security Considerations:

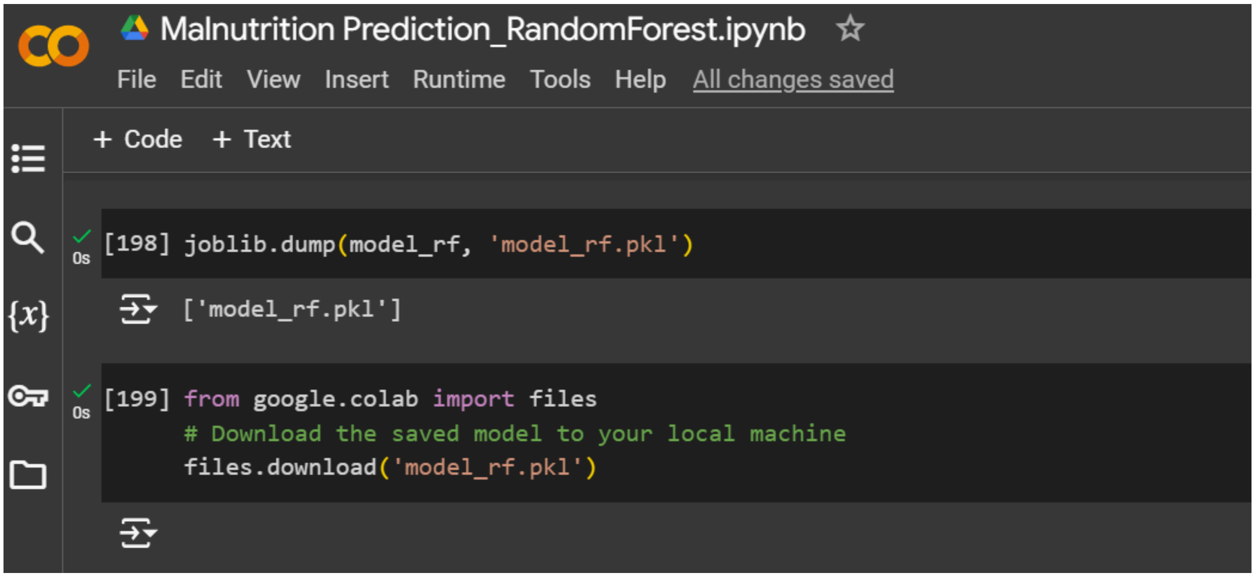
To protect the API, security measures were implemented, including:

- **HTTPS**: Ensuring that the API is served over HTTPS to encrypt data in transit.
- **Authentication**: Implementing API key authentication to restrict access and ensure that only authorized users can make requests.

6. Monitoring and Logging:

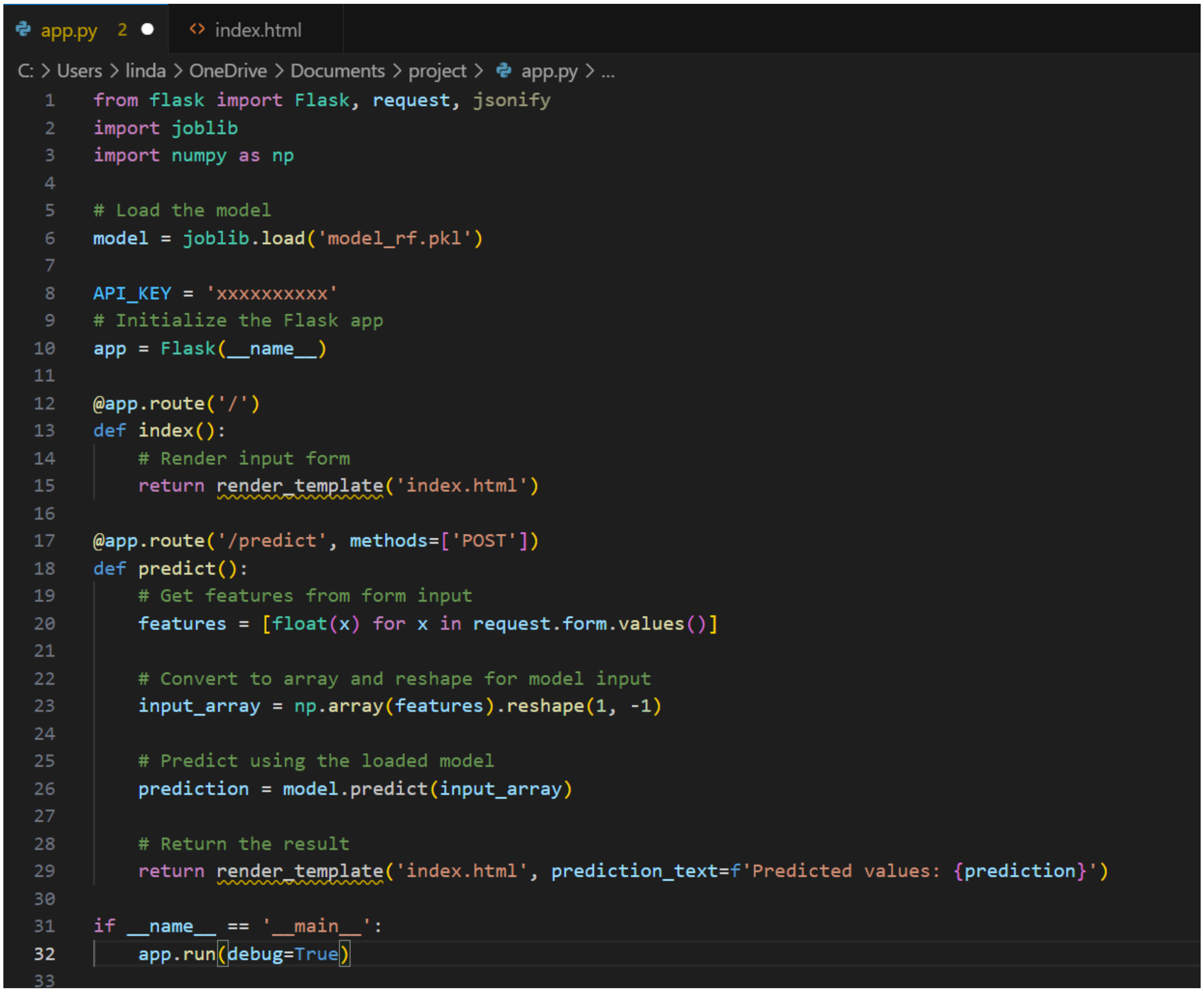
Monitoring ensures the model's performance and the API's functionality.

- **Logging**: Utilized Python's built-in logging module to track requests and errors, allowing for easier troubleshooting and analysis.
- **Performance Monitoring**: Tracked the application's performance using key metrics such as response time, latency, error rates, and memory usage to ensure optimal operation and user satisfaction.


Malnutrition Prediction_RandomForest.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
0s
0s

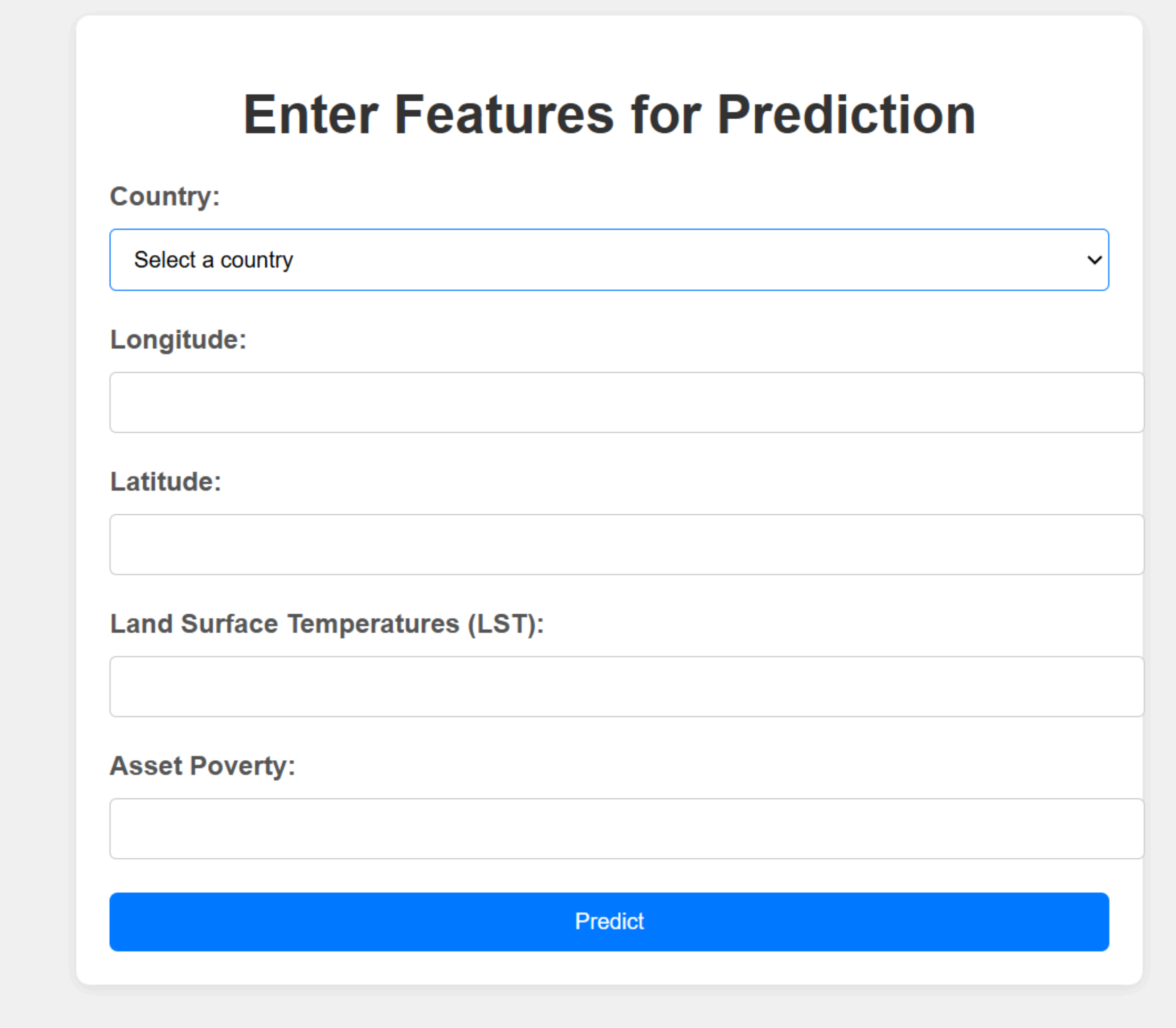
```
[198] joblib.dump(model_rf, 'model_rf.pkl')

      ['model_rf.pkl']

[199] from google.colab import files
      # Download the saved model to your local machine
      files.download('model_rf.pkl')
```

Figure 1: Model Serialization


app.py 2
index.html
C: > Users > linda > OneDrive > Documents > project > app.py > ...

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

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

API_KEY = 'xxxxxxxxxx'
# Initialize the Flask app
app = Flask(__name__)

@app.route('/')
def index():
    # Render input form
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Get features from form input
    features = [float(x) for x in request.form.values()]

    # Convert to array and reshape for model input
    input_array = np.array(features).reshape(1, -1)

    # Predict using the loaded model
    prediction = model.predict(input_array)

    # Return the result
    return render_template('index.html', prediction_text=f'Predicted values: {prediction}')

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

```

Figure 2: Model Serving and API Integration



Enter Features for Prediction

Country: Select a country

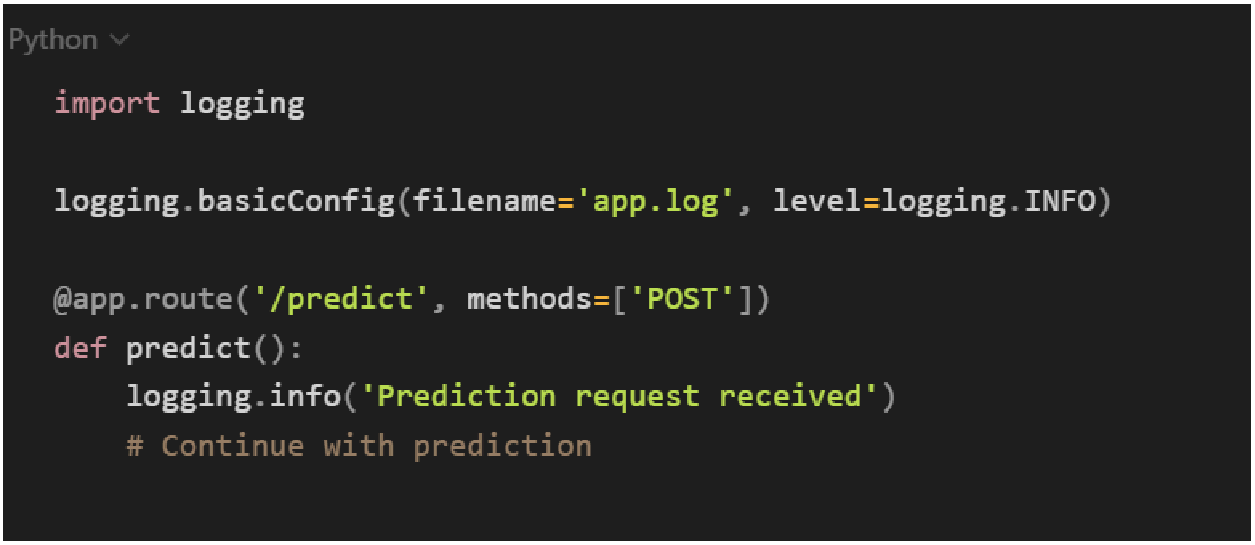
Longitude:

Latitude:

Land Surface Temperatures (LST):

Asset Poverty:

Figure 3: User Interface



```python
import logging

logging.basicConfig(filename='app.log', level=logging.INFO)

@app.route('/predict', methods=['POST'])
def predict():
    logging.info('Prediction request received')
    # Continue with prediction
```

Figure 4: Logging in Flask