NAME: ASHWINI B N

EMAIL: ASHWINIBN3@ICLOUD.COM

BATCH CODE: LISUM46

SUBMISSION DATE: 27/06/2025

ASSIGNMENT TITLE: DEPLOYMENT OF FLASK

1. Dataset Description

The Linnerud dataset from the scikit-learn library is utilized for this project. It is a toy multi-output regression dataset consisting of measurements from 20 individuals, capturing their physical exercise activity and corresponding physiological responses.

- Input features: Number of chin-ups, sit-ups, and jumps performed
- Target variables: Weight (kg), waist circumference (cm), and pulse rate (bpm)

A preview of the feature matrix (X) and target matrix (y) is shown below.

```
import pandas as pd

X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.DataFrame(data.target, columns=data.target_names)

print("Features (Exercise counts):")
print(X.head())

print("\nTargets (Health stats):")
print(y.head())
```

```
Features (Exercise counts):
   Chins  Situps  Jumps
0    5.0   162.0   60.0
1    2.0   110.0   60.0
2   12.0   101.0  101.0
3   12.0   105.0   37.0
4   13.0   155.0   58.0

Targets (Health stats):
   Weight  Waist  Pulse
0   191.0   36.0   50.0
1   189.0   37.0   52.0
2   193.0   38.0   58.0
3   162.0   35.0   62.0
4   189.0   35.0   46.0
```

Figure 1: Preview of the Linnerud dataset showing the input features (exercise counts) and target variables (health measurements).

2. Model Training and Saving

A multi-output Linear Regression model was trained using the scikit-learn library. The model was fitted using 80% of the dataset for training and the remaining 20% for testing. After training, the model was serialized using the joblib library and saved as model.pkl for deployment.

```python
# import Dependencies
from sklearn.datasets import load_linnerud
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
import joblib

# Load the Linnerud dataset again
data = load_linnerud()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.DataFrame(data.target, columns=data.target_names)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Save model to file
joblib.dump(model, 'model.pkl')
print("Linear Regression model saved as 'model.pkl'")
```

```
Linear Regression model saved as 'model.pkl'
```

Figure 2: Training and saving of the multi-output Linear Regression model using scikit-learn and joblib.

3. Flask Backend

A Flask backend was developed to serve the trained machine learning model. The backend accepts user inputs for exercise counts via a web form and returns the predicted physiological stats. The trained model is loaded from disk and used for real-time predictions.

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

app = Flask(__name__)

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

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

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get user inputs
        chinups = float(request.form['chinups'])
        situps = float(request.form['situps'])
        jumps = float(request.form['jumps'])

        # Format input for prediction
        input_data = np.array([[chinups, situps, jumps]])
        prediction = model.predict(input_data)[0]  # [weight, waist, pulse]

        # Return result to HTML
        return render_template('index.html', result={
            'weight': round(prediction[0], 2),
            'waist': round(prediction[1], 2),
            'pulse': round(prediction[2], 2)
        })

    except:
        return render_template('index.html', error="Invalid input. Please enter numbers.")

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

Figure 3: Flask backend code handling user input, loading the trained model, and returning predictions.

4. HTML Frontend (index.html)

The HTML frontend is implemented using a simple index.html file placed within the templates/ directory. It provides a form interface to collect chin-ups, sit-ups, and jumps from the user. On form submission, the input data is sent to the Flask backend using a POST request. If predictions are returned, the predicted health metrics (weight, waist, and pulse) are displayed; otherwise, an error message is shown.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Fitness Predictor</title>
</head>
<body>
    <h2>Enter Exercise Data</h2>
    <form method="POST" action="/predict">
        Chin-ups: <input type="number" name="chinups" step="1" required><br><br>
        Sit-ups: <input type="number" name="situps" step="1" required><br><br>
        Jumps: <input type="number" name="jumps" step="1" required><br><br>
        <input type="submit" value="Predict">
    </form>

    {% if result %}
        <h3>Predicted Health Stats:</h3>
        <p>Weight: {{ result.weight }} kg</p>
        <p>Waist: {{ result.waist }} cm</p>
        <p>Pulse: {{ result.pulse }} bpm</p>
    {% endif %}

    {% if error %}
        <p style="color:red;">{{ error }}</p>
    {% endif %}
</body>
</html>
```

Figure 4: HTML frontend form created using *index.html*, containing input fields and logic to display prediction results or error messages.

5. Running the Flask Application

The Flask application was executed from the terminal using the command python3 app.py. As shown in the output, the development server successfully started on http://127.0.0.1:5000. The warning in red is a standard message that indicates Flask is running in development mode and not intended for production use.

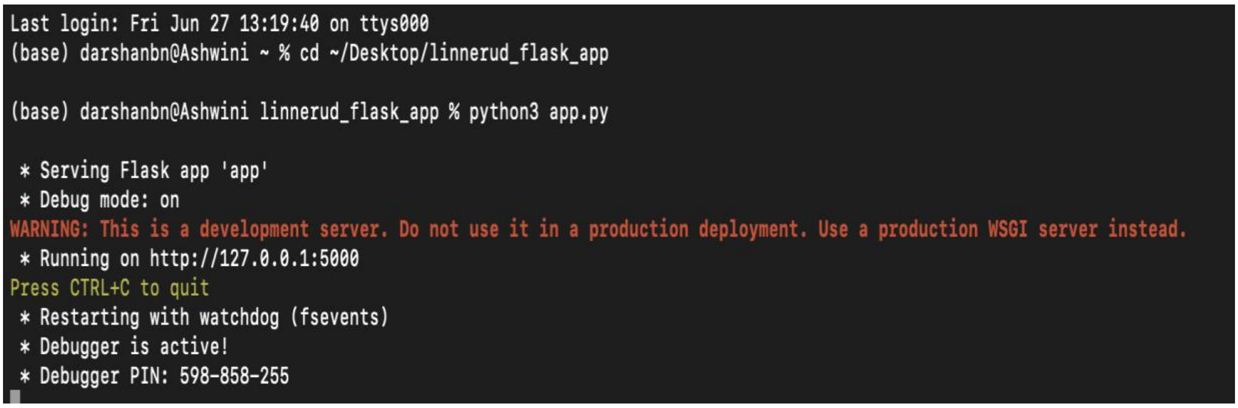

Figure 5: Terminal output showing successful launch of the Flask development server.

6. Web App Output

After launching the application, a test input was provided via the web interface. The form successfully accepted the values for chin-ups, sit-ups, and jumps and returned the corresponding predicted weight, waist, and pulse values using the trained regression model.

Figure 6: Web interface displaying user inputs and the predicted physiological outputs from the deployed machine learning model.

7. Conclusion

This project successfully demonstrates the end-to-end deployment of a multi-output regression model using the Flask framework. The workflow involved loading the Linnerud toy dataset, training a multi-output Linear Regression model, saving the model, and integrating it into a web application for real-time predictions.

The web interface accepts user input for physical activities—chin-ups, sit-ups, and jumps—and returns predicted health metrics including weight, waist circumference, and pulse rate.

It is important to note that the model is trained on a very small toy dataset (only 20 samples). As a result, the predictions, especially for variables like weight, may not always reflect realistic or medically accurate outputs. The model performance can be significantly improved by:

- Using a larger, real-world dataset,
- Incorporating feature scaling,
- Applying more advanced regression techniques,
- Performing model evaluation and tuning.

Despite these limitations, the project serves as a valuable demonstration of the model deployment pipeline and provides practical experience in building a functional ML-powered web application.