Elements of AIML
Assignment– 2

Name - Hardik Lal

SAP ID - 500120516

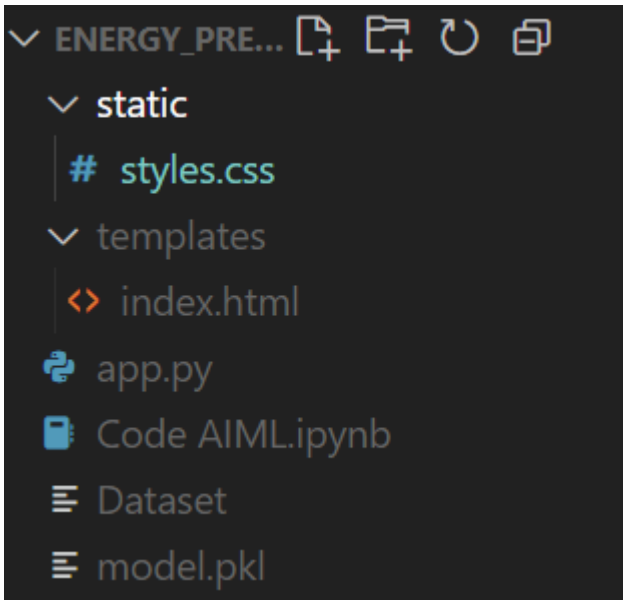
Batch - 12

Roll No. - R2142230372

Step-by-Step Guide

1. Set Up the Project Structure

Create a folder for the project, and inside it, set up the following structure:



2. Train and Save the Model

Using the previous model setup, train your model on the UCI dataset and save it with pickle so it can be loaded in the Flask app.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Step 1: Data Acquisition
# Load the Energy Efficiency dataset from UCI
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00242/ENB2012_data.xlsx"
df = pd.read_excel(url)

# Column names based on UCI dataset description
df.columns = [
    'Relative_Compactness', 'Surface_Area', 'Wall_Area', 'Roof_Area', 'Overall_Height',
    'Orientation', 'Glazing_Area', 'Glazing_Area_Distribution', 'Heating_Load', 'Cooling_Load'
]

# We'll predict 'Heating_Load' as the target variable, so we have an energy prediction task

# Step 2: Define the Methodology and Objectives
# Objective: Predict heating load (Energy Efficiency) for buildings based on input features.
X = df.drop(columns=['Heating_Load', 'Cooling_Load'])  # Features
y = df['Heating_Load']  # Target variable
```

```
    # Step 3: Data Preprocessing
    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    # Scale the data for better ML model performance
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Step 4: Use Multiple ML Methods and Validate with K-Fold Cross Validation
    models = {
        'Linear Regression': LinearRegression(),
        'Random Forest Regressor': RandomForestRegressor(n_estimators=100, random_state=0),
        'XGBoost Regressor': XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=0)
    }

    kf = KFold(n_splits=5, shuffle=True, random_state=0)
    model_scores = {}

    for model_name, model in models.items():
        # Cross-validation
        cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=kf, scoring='neg_mean_squared_error')
        model_scores[model_name] = np.mean(np.abs(cv_scores))  # Using mean of absolute MSE scores
```

```
# Step 5: Comparing Results
# Train the models on the whole training set for final evaluation
results = {}
for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)
    predictions = model.predict(X_test_scaled)
    mse = mean_squared_error(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    results[model_name] = {'MSE': mse, 'MAE': mae, 'R2 Score': r2}

# Display results
print("Model Performance (Mean Absolute Error during Cross-Validation):")
for model_name, score in model_scores.items():
    print(f"{model_name}: {score:.2f}")

print("\nFinal Model Results (Test Set Performance):")
for model_name, metrics in results.items():
    print(f"{model_name}:")
    print(f"  MSE: {metrics['MSE']:.2f}")
    print(f"  MAE: {metrics['MAE']:.2f}")
    print(f"  R2 Score: {metrics['R2 Score']:.2f}\n")
```

3. Create the Flask App (app.py)

This is the main backend file. It loads the model, sets up the prediction function, and defines the route for the web UI.

app.py > index

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

# Initialize __name__: str
app = Flask(__name__)

# Load the saved scaler and model
with open('model.pkl', 'rb') as file:
    scaler, model = pickle.load(file)

# Define the home route
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        # Get form data
        try:
            inputs = [
                float(request.form['Relative_Compactness']),
                float(request.form['Surface_Area']),
                float(request.form['Wall_Area']),
                float(request.form['Roof_Area']),
                float(request.form['Overall_Height']),
                float(request.form['Orientation']),
                float(request.form['Glazing_Area']),
                float(request.form['Glazing_Area_Distribution'])
            ]
```

```python
            # Preprocess and predict
            inputs_scaled = scaler.transform([inputs])
            prediction = model.predict(inputs_scaled)[0]

            return render_template('index.html', prediction=round(prediction, 2))
        except ValueError:
            return render_template('index.html', error="Please enter valid numbers.")

    return render_template('index.html')

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

4. Create the HTML Template (templates/index.html)

This file defines the web interface where users can input features and get a prediction.

templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Energy Prediction</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <h1>Energy Consumption Prediction</h1>
    <form method="POST">
        <label>Relative Compactness: <input type="text" name="Relative_Compactness" required></label><br>
        <label>Surface Area: <input type="text" name="Surface_Area" required></label><br>
        <label>Wall Area: <input type="text" name="Wall_Area" required></label><br>
        <label>Roof Area: <input type="text" name="Roof_Area" required></label><br>
        <label>Overall Height: <input type="text" name="Overall_Height" required></label><br>
        <label>Orientation: <input type="text" name="Orientation" required></label><br>
        <label>Glazing Area: <input type="text" name="Glazing_Area" required></label><br>
        <label>Glazing Area Distribution: <input type="text" name="Glazing_Area_Distribution" required></label><br
        <button type="submit">Predict</button>
    </form>

    {% if prediction %}
        <h2>Predicted Heating Load: {{ prediction }} MWh</h2>
    {% elif error %}
        <h2>{{ error }}</h2>
    {% endif %}
</body>
</html>
```

5. Optional Styling (static/style.css)

This file is optional but can help style the form for a better user experience.

static > # styles.css > ...

```css
body {
    font-family: Arial, sans-serif;
    text-align: center;
}

h1 {
    color: #2c3e50;
}

form {
    display: inline-block;
    text-align: left;
    margin-top: 20px;
}

label {
    display: block;
    margin-bottom: 10px;
}

button {
    padding: 10px 20px;
    background-color: #3498db;
    color: white;
    border: none;
    cursor: pointer;
}
```

6. Run the App

Start the Flask app by running app.py in the terminal:

```
python app.py
```

Visit http://127.0.0.1:5000/ in your browser to access the app.

Instructions for Usage

1. Enter values for each feature (e.g., Relative Compactness, Surface Area).
2. Click "Predict" to get a prediction for the heating load.
3. The result will be displayed below the form, indicating the predicted energy load in MWh.