Text

Description automatically generated

**Activity based**

**Project Report on**

**Machine Learning**

**Project Phase - I**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Atharva Daga**

**SRN No: 202101245**

**Roll No: 25**

**Div: A**

**Fourth Year Engineering**

**Faculty In charge:- Prof. Gulnaz Thakur**

**Date of Project Phase 1:- 17th March, 2025**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2024-2025 Term-II**

**Machine Learning: Phase I**

**Project Name: Energy Consumption Prediction**

### ****Introduction:****

### Energy usage plays a vital role in contemporary infrastructure, influencing both financial and ecological factors. As energy demands continue to rise, efficient consumption has become imperative for sustainable resource management. This project aims to forecast power consumption using a Multiple Linear Regression (MLR) model, enabling more informed energy management strategies. By utilizing past energy records and critical building attributes, the model seeks to improve energy efficiency and offer meaningful insights into consumption trends.

### Problem Statement:

### Managing energy consumption effectively is a critical challenge for residential, commercial, and industrial buildings. Factors such as building size, number of occupants, appliance usage, and external conditions influence power consumption. However, without a predictive model, energy usage remains unpredictable, leading to inefficiencies, increased costs, and environmental impact.

### This project addresses the need for a data-driven approach to forecast energy consumption, enabling users to plan and optimize power usage based on key influencing factors.

### Objective:

### The objective of this project is to develop a Multiple Linear Regression (MLR) model to predict energy consumption based on key building and environmental factors. By analyzing features such as building type, square footage, number of occupants, appliances used, and heat levels, the model aims to provide accurate forecasts. This will help in optimizing power usage, improving energy efficiency, and reducing unnecessary energy consumption. The outcome is a data-driven approach for better energy management and decision-making.

### ****Motivation:****

* **Rising Energy Demand:** With increasing energy consumption worldwide, efficient power management is essential to prevent resource depletion and reduce costs.
* **Environmental Impact:** Unregulated energy usage contributes to carbon emissions and climate change, making optimization crucial for sustainability.
* **Cost Efficiency:** Predicting energy consumption helps in minimizing unnecessary power usage, leading to significant cost savings for consumers and businesses.
* **Data-Driven Decision Making:** Leveraging machine learning for energy prediction enables more accurate and informed energy management strategies.
* **Smart Infrastructure Development:** Enhancing energy efficiency supports the advancement of smart buildings and grids, promoting sustainable urban development.

### ****Attributes:****

|  |  |  |
| --- | --- | --- |
| Name of Attributes | Categorical/ Numerical | How many fields of rows |
| 1. Building Type | Categorical | 1100 |
| 1. Square Footage | Numerical | 1100 |
| 1. Number of Occupants | Numerical | 1100 |
| 1. Appliances Used | Numerical | 1100 |
| 1. Average Temperature | Numerical | 1100 |
| 1. Day of Week | Categorical | 1100 |
| 1. Energy Consumption | Numerical | 1100 |
| 1. Heat Levels | Categorical | 1100 |

### Dataset Pre-processing:-

### Initially:-

### 

### Afterwards:-

### 

### Codes & it’s explainations:-

### Model building:-

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler,OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score

### The code first imports necessary libraries: pandas for data manipulation and handling tabular data, *numpy* for numerical operations, *seaborn* for statistical data visualization, and *matplotlib.pyplot* for creating plots. From *sklearn.preprocessing*, it imports *StandardScaler* for feature scaling and *OneHotEncoder* for converting categorical data into numerical format. The *train\_test\_split* function from *sklearn.model\_selection* helps split the dataset into training and test sets. The *LinearRegression* model from *sklearn.linear\_model* is used to build a multiple linear regression model. Finally, performance evaluation metrics such as *mean\_absolute\_error*, *mean\_squared\_error*, and *r2\_score* from *sklearn.metrics* help assess the model’s performance.

df = pd.read\_excel("dataset.xlsx")

df.shape

### The dataset is loaded from an Excel file using *pd.read\_excel("dataset.xlsx")*. To understand its dimensions, *df.shape* is used, which returns the number of rows and columns.

df.info()

df.describe()

### *df.info()* provides an overview of the dataset, displaying column names, data types, and the count of non-null values in each column. *df.describe()* offers statistical insights such as mean, standard deviation, minimum, and maximum values for numerical columns.

df.isnull().sum()

### *df.isnull().sum()* checks for any missing values in the dataset by counting the number of null entries in each column.

df.select\_dtypes(include='number').skew()

### Skewness measures the asymmetry of data distribution. *df.select\_dtypes(include='number').skew()* computes the skewness for all numerical columns, helping to identify any highly skewed features that may require transformation.

# checking for outliers using boxplot

df.boxplot(rot=60)

plt.show

### A boxplot is used to visualize potential outliers in the dataset. *df.boxplot(rot=60)* creates a boxplot for numerical features, where *rot=60* rotates the labels for better readability. The *plt.show()* function is used to display the plot.

# Perform One-Hot Encoding for multiple categorical columns

categorical\_columns = ['Building Type', 'Day of Week', 'Heat Levels']

df\_encoded = pd.get\_dummies(df, columns=categorical\_columns, prefix=['BuildingType', 'DayOfWeek', 'HeatLevel'])

# Convert only the newly created one-hot encoded columns to integers

for col in df\_encoded.columns:

    if any(prefix in col for prefix in ['BuildingType\_', 'DayOfWeek\_', 'HeatLevel\_']):  # Ensuring only relevant columns

        df\_encoded[col] = df\_encoded[col].astype(int)

# Save to a new Excel file

output\_path = "cleaned.xlsx"

df\_encoded.to\_excel(output\_path, index=False)

### Since machine learning models work with numerical data, categorical columns like *Building Type, Day of Week*, and *Heat Levels* need to be converted into numerical values. *pd.get\_dummies(df, columns=categorical\_columns, prefix=['BuildingType', 'DayOfWeek', 'HeatLevel'])* performs one-hot encoding, creating new binary columns for each unique category. To ensure only relevant columns are converted to integers, a loop iterates over the dataset and updates the respective columns. The processed dataset is then saved as *"cleaned.xlsx"* using *df\_encoded.to\_excel(output\_path, index=False)*.

file\_path = "cleaned.xlsx"  # Ensure this is the correct path

df = pd.read\_excel(file\_path)

# Columns to standardize

columns\_to\_standardize = ["Square Footage", "Number of Occupants", "Appliances Used", "Average Temperature", "Energy Consumption"]

# Print original mean and standard deviation

print("Before Standardization:")

for col in columns\_to\_standardize:

    print(f"{col} - Mean: {df[col].mean():.4f}, Std Dev: {df[col].std():.4f}")

# Apply Z-score normalization

df[columns\_to\_standardize] = df[columns\_to\_standardize].apply(lambda x: (x - x.mean()) / x.std())

# Print new mean and standard deviation

print("/nAfter Standardization:")

for col in columns\_to\_standardize:

    print(f"{col} - Mean: {df[col].mean():.4f}, Std Dev: {df[col].std():.4f}")

# Save the standardized dataset

output\_path = "standardized\_dataset.xlsx"

df.to\_excel(output\_path, index=False)

* Since numerical features have different scales, standardization ensures they are on the same scale. The dataset is reloaded from *"cleaned.xlsx"* using *pd.read\_excel(file\_path)*. The columns *Square Footage, Number of Occupants, Appliances Used, Average Temperature,* and *Energy Consumption* are standardized using Z-score normalization, where each value is transformed as *(value - mean) / standard deviation*. Before standardization, the mean and standard deviation of selected columns are printed using a loop. Followed by printing the updated mean and standard deviation to confirm the transformation. Finally, the dataset is saved as *"standardized\_dataset.xlsx".*

import joblib

import pandas as pd

# Load dataset

file\_path = "F:/PROJECTS/Sem - 8/ML/P1/datasets/standardized\_dataset.xlsx"  # Ensure this is the correct path

df = pd.read\_excel(file\_path)

# Define independent variables (X) and dependent variable (y)

X = df.drop(columns=["Energy Consumption", "Average Temperature"])  # Features

y = df["Energy Consumption"]  # Target variable

# Split data into training (80%) and testing (20%)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predictions on test set

y\_pred = model.predict(X\_test)

# Evaluate Model Performance

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nModel Performance:")

print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

# Save the trained model for future predictions

model\_filename = "mlr\_energy\_consumption\_model.pkl"

joblib.dump(model, model\_filename)

print(f"\nModel saved as '{model\_filename}' for future predictions.")

* The dataset is loaded again from *"standardized\_dataset.xlsx"*. Independent variables (*X*) are defined by dropping *Energy Consumption* and *Average Temperature*, while *y* is set as the target variable *Energy Consumption*. The dataset is split into training (80%) and testing (20%) sets using *train\_test\_split(X, y, test\_size=0.2, random\_state=42)*. The *LinearRegression* model is initialized and trained using *model.fit(X\_train, y\_train)*. After training, predictions on the test set are made using *y\_pred = model.predict(X\_test)*.
* The model’s accuracy is assessed using different metrics. *mean\_absolute\_error(y\_test, y\_pred)* calculates the average absolute difference between actual and predicted values, while *mean\_squared\_error(y\_test, y\_pred)* computes the squared differences. The square root of MSE gives *rmse = np.sqrt(mse)*, measuring how far predictions deviate from actual values. *r2\_score(y\_test, y\_pred)* determines how well the model explains variance in the data, with values closer to 1 indicating a better fit.
* To avoid retraining the model each time, *joblib.dump(model, "mlr\_energy\_consumption\_model.pkl")* saves the trained model as a *.pkl* file.

import pickle

with open("mlr\_energy\_consumption\_model.pkl", "rb") as file:

    model = pickle.load(file)

print(f"Model Loaded Successfully! Type: {type(model)}")

### To ensure the model was saved correctly, it is reloaded using *pickle.load(open("mlr\_energy\_consumption\_model.pkl", "rb")),* and its type is printed.

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Predictions

y\_pred = model.predict(X\_test)

#\*Actual vs. Predicted Plot\*\*

plt.figure(figsize=(8, 6))

sns.scatterplot(x=y\_test, y=y\_pred, color="blue", alpha=0.6, edgecolor="black")

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color="red", linestyle="dashed")  # Perfect Prediction Line

plt.xlabel("Actual Energy Consumption")

plt.ylabel("Predicted Energy Consumption")

plt.title("Actual vs Predicted Energy Consumption")

plt.grid(True)

plt.show()

# Residual Plot\*\*

residuals = y\_test - y\_pred

plt.figure(figsize=(8, 6))

sns.histplot(residuals, bins=20, kde=True, color="purple")

plt.axvline(x=0, color="red", linestyle="dashed")  # Mean Residual Line

plt.xlabel("Residuals (Error)")

plt.ylabel("Frequency")

plt.title("Residuals Distribution")

plt.grid(True)

plt.show()

### Two plots are generated to evaluate the model. First, an *Actual vs. Predicted Plot* is created using *sns.scatterplot(x=y\_test, y=y\_pred, color="blue", alpha=0.6, edgecolor="black")*, where a dashed red line represents the ideal prediction scenario. Second, a *Residuals Distribution Plot* is generated using *sns.histplot(residuals, bins=20, kde=True, color="purple")*, showing the error distribution.

from sklearn.ensemble import RandomForestRegressor

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

rf\_pred = rf\_model.predict(X\_test)

def evaluate\_model(y\_test, y\_pred, model\_name):

    mae = mean\_absolute\_error(y\_test, y\_pred)

    mse = mean\_squared\_error(y\_test, y\_pred)

    rmse = np.sqrt(mse)

    r2 = r2\_score(y\_test, y\_pred)

    print(f"\n{model\_name} Performance:")

    print(f"MAE: {mae:.4f}")

    print(f"MSE: {mse:.4f}")

    print(f"RMSE: {rmse:.4f}")

    print(f"R² Score: {r2:.4f}")

evaluate\_model(y\_test, rf\_pred, "Random Forest")

### A *RandomForestRegressor* model is trained to compare results. The dataset is split in the same way, and *RandomForestRegressor(n\_estimators=100, random\_state=42)* is trained on *X\_train, y\_train*. Predictions are made, and the same evaluation metrics *(MAE, MSE, RMSE, R² Score)* are used to assess its performance.

### Output:

### 

### 

### 

### 

### 

### HTML & CSS:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Energy Consumption Prediction</title>

    <style>

        body {

            background: url("{{ url\_for('static', filename='image.jpg') }}") no-repeat center center fixed;

            background-size: cover;

        }

        .container {

            width: 40%;

            margin: auto;

            position: absolute;

            top: 50%;

            left: 50%;

            transform: translate(-50%, -50%);

            background: rgba(0, 0, 0, 0.7);

            padding: 30px;

            border-radius: 10px;

            color: white;

            text-align: center;  /\* Center all content \*/

        }

        h1 {

            margin-bottom: 20px;

            font-size: 26px;

            color: #fff;

            text-align: center;

        }

        form {

            display: flex;

            flex-direction: column;

            align-items: center;

        }

        label {

            font-weight: bold;

            margin-top: 12px;

            color: #ddd;

        }

        input, select {

            width: 85%;

            padding: 12px;

            margin: 8px;

            border-radius: 8px;

            border: 1px solid #ccc;

            font-size: 16px;

            background: rgba(255, 255, 255, 0.8);

            color: black;

            outline: none;

        }

        button {

            margin-top: 20px;

            padding: 12px 25px;

            background: #28a745;

            color: white;

            border: none;

            cursor: pointer;

            font-size: 18px;

            border-radius: 6px;

            width: 60%;

            transition: 0.3s ease-in-out;

        }

        button:hover {

            background: #218838;

        }

        .result {

            margin-top: 25px;

            font-size: 22px;

            font-weight: bold;

            color: white;

            text-align: center;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Energy Consumption Prediction</h1>

        <form method="POST">

            <label>Building Type:</label>

            <select name="building\_type" required>

                <option value="Residential">Residential</option>

                <option value="Commercial">Commercial</option>

                <option value="Industrial">Industrial</option>

            </select>

            <label>Square Footage:</label>

            <input type="number" name="square\_footage" required placeholder="Enter square footage">

            <label>Number of Occupants:</label>

            <input type="number" name="num\_occupants" required placeholder="Enter number of occupants">

            <label>Appliances Used:</label>

            <input type="number" name="appliances" required placeholder="Enter number of appliances">

            <label>Day of the Week:</label>

            <select name="day\_of\_week" required>

                <option value="Weekday">Weekday</option>

                <option value="Weekend">Weekend</option>

            </select>

            <label>Heat Level:</label>

            <select name="heat\_level" required>

                <option value="Mild Day">Mild Day</option>

                <option value="Moderate Day">Moderate Day</option>

                <option value="Hot Day">Hot Day</option>

            </select>

            <button type="submit">Predict</button>

        </form>

        {% if prediction is not none %}

            <div class="result">Predicted Energy Consumption: {{ prediction }}</div>

        {% endif %}

    </div>

</body>

</html>

### ****Defining the HTML Structure:-**** The file starts with *<!DOCTYPE html>*, which declares it as an HTML5 document. The *<html lang="en">* tag specifies English as the document language. The *<head>* section contains metadata and the title of the web page, *"Energy Consumption Prediction"*.

### Setting Up the Stylesheet (CSS for Styling):- Within the *<style>* block, CSS styles are defined to enhance the visual appearance of the page.

### Background Styling:- The body tag has a background image *(image.jpg)* stored in the static folder, set using *url\_for('static', filename='image.jpg')*. It is displayed as a full-screen background, centered, and fixed *(background-size: cover;* ensures it scales properly).

### Container Styling:- A *.container* div is created to hold the form elements. It is positioned in the center of the screen *(top: 50%, left: 50%, transform: translate(-50%, -50%))*. It has a semi-transparent black background *(rgba(0, 0, 0, 0.7))* with rounded corners *(border-radius: 10px)*, padding, and white-colored text.

### Header Styling (h1):- The *title "Energy Consumption Prediction"* is centered, has a white font color, and a larger font size *(font-size: 26px;).*

### Form Styling (form):- The form is aligned to the center using *display: flex; flex-direction: column; align-items: center;*

### Input Fields (input, select):- All form inputs and dropdowns have a width of 85%, padding for better spacing, a light background with some transparency *(rgba(255, 255, 255, 0.8))*, and a defined border. Their font size is set to *16px* for readability.

### Button Styling (button):- The submit button has a green background *(#28a745)*, white text, and rounded corners *(border-radius: 6px)*. It transitions to a darker green *(#218838)* when hovered over.

### Result Display Styling (.result):- If a prediction is generated, it will be displayed in a bold white font with a larger text size *(font-size: 22px)*.

### ****Structuring the Web Page Body *(******<body>******)*:-**** The *<body>* contains a *<div class="container">*, which acts as the main box for form inputs and the prediction output.

#### **Title of the Web Page (**<h1>**):-** Displays "Energy Consumption Prediction" at the top of the container.

#### **Creating the Form (**<form method="POST">**):-** The form collects user inputs for the multiple linear regression model and sends the data using the POST method.

* **Building Type *(<select> Dropdown)***:- Allows the user to choose among "Residential", "Commercial", or "Industrial".
* **Square Footage *(<input type="number">)***:- A number input field for entering the building's square footage.
* **Number of Occupants *(<input type="number">)***:- A number input field for specifying how many people are in the building.
* **Appliances Used *(<input type="number">)***:- A number input field for entering the number of appliances in the building.
* **Day of the Week *(<select> Dropdown)***:- Allows users to choose between "Weekday" or "Weekend".
* **Heat Level *(<select> Dropdown)***:- Lets users select between "Mild Day", "Moderate Day", or "Hot Day".
* **Submit Button *(<button type="submit">Predict</button>)***:- Once all fields are filled, clicking this button submits the form for prediction.

### ****Displaying the Prediction Result *(******{% if prediction is not none %}******):-***** This section is written in ****Jinja2 templating language**** (used in Flask applications). If a prediction value exists, it will be displayed inside the *<div class="result">* as *****"Predicted Energy Consumption: {{ prediction }}"******.*

### ****Main.py:-****

from flask import Flask, render\_template, request

import joblib

import numpy as np

app = Flask(\_\_name\_\_, template\_folder="templates", static\_folder="static")

# Load the trained model

try:

    model = joblib.load("mlr\_energy\_consumption\_model.pkl")

    print(f" Model Loaded Successfully! Type: {type(model)}")

except Exception as e:

    print(f" Model Load Error: {e}")

    model = None

# Mean & Std for normalization (WITHOUT "Average Temperature")

mean\_values = {

    "Square Footage": 27672.6545,

    "Number of Occupants": 49.4945,

    "Appliances Used": 25.8291,

    "Energy Consumption": 3995.7822  # Mean of Energy Consumption (for inverse transform)

}

std\_values = {

    "Square Footage": 13045.1552,

    "Number of Occupants": 28.7215,

    "Appliances Used": 14.3262,

    "Energy Consumption": 1144.5472  # Standard deviation of Energy Consumption (for inverse transform)

}

# Encoding mappings

building\_types = {"Residential": [0, 0, 1], "Commercial": [1, 0, 0], "Industrial": [0, 1, 0]}

day\_of\_week = {"Weekday": [1, 0], "Weekend": [0, 1]}

heat\_levels = {"Mild Day": [0, 1, 0], "Moderate Day": [0, 0, 1], "Hot Day": [1, 0, 0]}

@app.route("/", methods=["GET", "POST"])

def home():

    prediction = None

    if request.method == "POST":

        try:

            # Get user input

            square\_footage = float(request.form.get("square\_footage", "0"))

            num\_occupants = float(request.form.get("num\_occupants", "0"))

            appliances = float(request.form.get("appliances", "0"))

            building\_type = request.form.get("building\_type", "")

            day = request.form.get("day\_of\_week", "")

            heat = request.form.get("heat\_level", "")

            # Standardize numerical features

            def standardize(value, mean, std):

                return (value - mean) / std

            square\_footage = standardize(square\_footage, mean\_values["Square Footage"], std\_values["Square Footage"])

            num\_occupants = standardize(num\_occupants, mean\_values["Number of Occupants"], std\_values["Number of Occupants"])

            appliances = standardize(appliances, mean\_values["Appliances Used"], std\_values["Appliances Used"])

            # One-hot encode categorical variables

            building\_encoded = building\_types.get(building\_type, [0, 0, 1])

            day\_encoded = day\_of\_week.get(day, [1, 0])

            heat\_encoded = heat\_levels.get(heat, [0, 1, 0])

            #  FINAL FEATURE ARRAY (Without "Average Temperature")

            features = np.array([square\_footage, num\_occupants, appliances] +

                                building\_encoded + day\_encoded + heat\_encoded).reshape(1, -1)

            print(f" Input Features: {features}")

            # Predict using model

            if model:

                standardized\_prediction = float(model.predict(features)[0])  # Get standardized output

                actual\_prediction = (standardized\_prediction \* std\_values["Energy Consumption"]) + mean\_values["Energy Consumption"]  # Inverse transform

                prediction = round(actual\_prediction, 2)

                print(f" Standardized Prediction: {standardized\_prediction}")

                print(f" Final Energy Consumption Prediction: {prediction} kWh")

            else:

                prediction = "Model Not Found"

        except Exception as e:

            print(f" Error in Prediction: {e}")

            prediction = "Error in Prediction"

    return render\_template("energy.html", prediction=prediction)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

### ****Importing Required Libraries:-****

* ***from flask import Flask, render\_template, request*** → Imports Flask, which helps create the web application. *render\_template* is used to display HTML templates, and request handles form submissions.
* ***import joblib*** → Loads the trained multiple linear regression model *(mlr\_energy\_consumption\_model.pkl)*.
* ***import numpy as np*** → Used for numerical operations, particularly handling input features as NumPy arrays.

### ****Initializing the Flask App:-****

* *app = Flask(\_\_name\_\_, template\_folder="templates", static\_folder="static")* 
  + Creates a Flask application instance.
  + ***template\_folder="templates"*** → HTML files are stored in the templates directory.
  + ***static\_folder="static"*** → Static assets (like images, CSS) are stored in the static folder.

### ****Loading the Trained Model:-****

* The model is loaded using *joblib.load("mlr\_energy\_consumption\_model.pkl")*.
* If loading is successful, a message is printed*: "Model Loaded Successfully!".*
* If there's an error (e.g., missing model file), it prints "*Model Load Error: {e}*" and sets *model = None*.

### ****Defining Mean and Standard Deviation for Normalization:-**** Since the model was trained with ****standardized**** inputs, we need to ****normalize**** new user inputs. The mean and standard deviation values are predefined for:

* ***"Square Footage"*** → Mean = 27672.6545, Std = 13045.1552
* ***"Number of Occupants"*** → Mean = 49.4945, Std = 28.7215
* ***"Appliances Used"*** → Mean = 25.8291, Std = 14.3262
* ***"Energy Consumption" (used for inverse transformation)*** → Mean = 3995.7822, Std = 1144.5472

### ****Encoding Categorical Features:-**** Since machine learning models work with numbers, categorical inputs like "Building Type", "Day of the Week", and "Heat Level" are converted into ****one-hot encoding****:

#### **Building Type Encoding:-**

* *"Residential"* → [0, 0, 1]
* *"Commercial"* → [1, 0, 0]
* *"Industrial"* → [0, 1, 0]

#### **Day of the Week Encoding:-**

* *"Weekday"* → [1, 0]
* *"Weekend"* → [0, 1]

#### **Heat Level Encoding:-**

* *"Mild Day"* → [0, 1, 0]
* *"Moderate Day"* → [0, 0, 1]
* *"Hot Day"* → [1, 0, 0]

### ****Defining the Home Route (****/****):-**** The web page is served at *"http://127.0.0.1:5000/"*.

* ***@app.route("/", methods=["GET", "POST"])*** → Handles **both GET and POST requests**.
* ***def home():*** → The main function where form input processing and prediction happen.
* ***prediction = None*** → Initializes the prediction value as None before any user input.

### ****Handling User Input (POST Request):-**** If the form is submitted *(request.method == "POST")*, the input values are extracted:

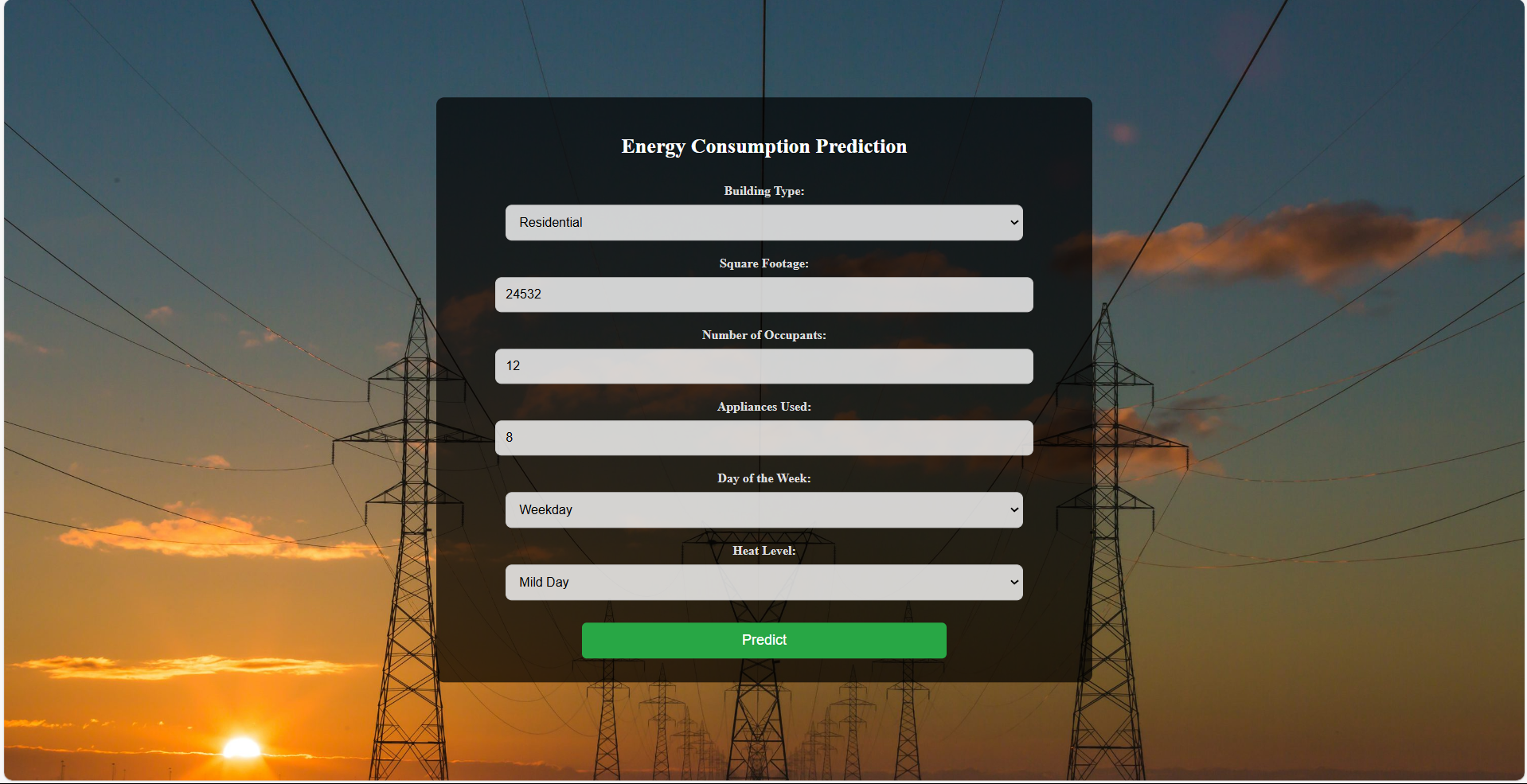
* ***square\_footage = float(request.form.get("square\_footage", "0"))*** → Retrieves and converts it to a float.
* ***num\_occupants = float(request.form.get("num\_occupants", "0"))*** → Retrieves the number of occupants.
* ***appliances = float(request.form.get("appliances", "0"))*** → Retrieves the number of appliances.
* ***building\_type = request.form.get("building\_type", "")*** → Retrieves the building type.
* ***day = request.form.get("day\_of\_week", "")*** → Retrieves the day of the week.
* ***heat = request.form.get("heat\_level", "")*** → Retrieves the selected heat level.
* **Standardizing the Numerical Features & One-Hot Encoding Categorical Variables:-**
* To match the trained model's format, input values are normalized using:
  + standardized value = value − mean / std
* Then categorical values are converted to their respective encoded lists
* **Creating the Final Feature Array:-** All standardized numerical values and encoded categorical values are combined into a *NumPy array*. The *.reshape(1, -1)* ensures the model receives a **2D array**.

### ****Making the Prediction:-**** If the model is loaded (if model:), prediction is performed:

* The model outputs a **standardized** prediction (standardized\_prediction).
* To convert it back to actual energy consumption, we apply the inverse transformation:
* actual value = (predicted standardized value × std) + mean
* The result is rounded to 2 decimal places
  + If the model is missing, *"Model Not Found"* is displayed.
  + If an error occurs, *"Error in Prediction"* is displayed.
* **Rendering the Prediction in HTML:-** After processing, the function returns the *energy.html* template with the prediction value. Then the result is displayed in the frontend.

### ****Running the Flask App:-**** The Flask application runs in ****debug mode**** when executed. This allows easy debugging and live reloading.

### Output:-



### 

### 

### 

**Conclusion:-**

### This project successfully implements a Multiple Linear Regression model for predicting energy consumption based on key building parameters. Using Flask, it provides an interactive web interface where users can input details such as building type, square footage, number of occupants, appliances used, day of the week, and heat level. The application standardizes input values, processes them using a trained model, and returns an accurate energy consumption estimate. By integrating machine learning with web deployment, this project demonstrates a practical approach to energy analysis, aiding in efficient energy management and decision-making.