



**VISHWAKARMA**  
**UNIVERSITY**  
*Maximising Human Potential*

**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:- 17<sup>th</sup> 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.

## Machine Learning

- **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
2) Square Footage	Numerical	1100
3) Number of Occupants	Numerical	1100
4) Appliances Used	Numerical	1100
5) Average Temperature	Numerical	1100
6) Day of Week	Categorical	1100
7) Energy Consumption	Numerical	1100
8) Heat Levels	Categorical	1100

### Dataset Pre-processing:-

#### ❖ Initially:-

Building Type	Square Footage	Number of Occupants	Appliances Used	Average Temperature	Day of Week	Energy Consumption	Heat Levels
Residential	24563	15	4	28.52	Weekday	2865.57	Mild Day
Commercial	27583	56	23	23.07	Weekend	4283.8	Moderate Day
Commercial	45313	4	44	33.56	Weekday	5067.83	Hot Day
Residential	41625	84	17	27.39	Weekend	4624.3	Moderate Day
Residential	36720	58	47	17.08	Weekday	4820.59	Moderate Day
Industrial	31207	47	28	22.82	Weekday	5026.23	Hot Day
Residential	39227	18	44	23.36	Weekend	4404.56	Moderate Day
Residential	7814	21	19	27.27	Weekday	2394.37	Mild Day
Industrial	20482	24	16	23	Weekend	3969.09	Moderate Day
Industrial	21030	90	35	12.96	Weekday	5136.69	Hot Day
Commercial	41401	87	15	15.54	Weekend	5162.33	Hot Day
Industrial	6279	23	41	10.77	Weekend	3810.09	Moderate Day
Industrial	31963	2	20	23.53	Weekend	4400.48	Moderate Day
Commercial	29187	82	39	23.54	Weekday	4991.64	Moderate Day
Industrial	13990	80	27	12.9	Weekend	4474.98	Moderate Day
Residential	8157	10	37	25.96	Weekday	2668.06	Mild Day
Industrial	27165	73	25	30.15	Weekday	4987.52	Moderate Day
Industrial	5819	56	41	26.19	Weekend	4039.98	Moderate Day
Residential	6155	78	43	21.89	Weekday	3388.29	Moderate Day
Commercial	33233	48	4	30.54	Weekend	4068.94	Moderate Day
Industrial	24762	51	12	21.65	Weekend	4379.87	Moderate Day
Commercial	25925	96	40	26.9	Weekend	4921.73	Moderate Day
Industrial	42729	20	17	11.91	Weekend	5116.9	Hot Day
Industrial	18150	87	26	28.24	Weekend	4656.29	Moderate Day
Industrial	42767	40	28	17.94	Weekend	5508.64	Hot Day
Residential	35650	98	1	25.43	Weekday	4205.35	Moderate Day
Residential	40486	27	28	15.73	Weekend	4275.65	Moderate Day
Commercial	44253	30	5	24.14	Weekend	4491.97	Moderate Day
Commercial	27582	24	24	29.99	Weekday	3999.14	Moderate Day
Commercial	21162	10	37	27.83	Weekend	3758.95	Moderate Day
Industrial	27708	47	24	11.26	Weekend	4779.08	Moderate Day
Residential	44539	8	44	26.62	Weekday	4603.83	Moderate Day

## ❖ Afterwards:-

Square Footage	Number of Occupants	Appliances Used	Average Temperature	Energy Consumption	BuildingType Commercial	BuildingType Industrial	BuildingType Residential	DayOfWeek Weekday	DayOfWeek Weekend	HeatLevel Hot Day	HeatLevel Mild Day	HeatLevel Moderate Day
-0.238376202	-1.20100158	-1.52372118	0.825615431	-0.98747538	0	0	1	1	0	0	1	0
-0.006872632	0.226501352	-0.197477108	0.063378213	0.251643441	1	0	0	0	1	0	0	1
1.35252636	-1.583990172	1.268371603	1.530510032	0.936656682	1	0	0	1	0	1	0	0
1.069542315	1.201381404	-0.616291025	0.667573586	0.549140996	0	0	1	0	1	0	0	1
0.693540655	0.296135642	1.477778562	-0.774383427	0.720641129	0	0	1	1	0	0	0	1
0.270931653	-0.08685295	0.15153449	0.028413203	0.900310431	0	1	0	1	0	1	0	0
0.885719281	-1.096550146	1.268371603	0.103937624	0.357152412	0	0	1	0	1	0	0	1
-1.522301137	-0.992098712	-0.476666386	0.650790381	-1.399166565	0	0	1	1	0	0	1	0
-0.55121265	-0.887647278	-0.686093345	0.05358801	-0.02321187	0	1	0	0	1	0	0	1
-0.509204717	1.410284272	0.640150727	-1.350606791	0.996820211	0	1	0	1	0	1	0	0
1.052371189	1.305832838	-0.755895664	-0.989767888	1.019222083	1	0	0	0	1	1	0	0
-1.639969343	-0.922464423	1.058964644	-1.656902779	-0.162240751	0	1	0	0	1	0	0	1
0.328884203	-1.653624461	-0.406884066	0.127713831	0.353587684	0	1	0	0	1	0	0	1
0.116084894	1.131747114	0.919360005	0.129112432	0.870088873	1	0	0	1	0	0	0	1
-1.04886867	1.062112825	0.081732171	-1.358998394	0.418678926	0	1	0	0	1	0	0	1
-1.496007851	-1.375087304	0.779755366	0.467573728	-1.160041437	0	0	1	1	0	0	1	0
-0.038915179	0.818392812	-0.057872469	1.053587296	0.866489197	0	1	0	1	0	0	0	1
-1.675231476	0.226501352	1.058964644	0.499741538	0.038615969	0	1	0	0	1	0	0	1
-1.649474788	0.992478536	1.198569284	-0.101656634	-0.530770762	0	0	1	1	0	0	0	1
0.426238353	-0.052035805	-1.52372118	1.108132711	0.063918551	1	0	0	0	1	0	0	1
-0.223121497	0.052415629	-0.965302623	-0.135223044	0.335580563	0	1	0	0	1	0	0	1
-0.133969625	1.61918714	0.989162325	0.599042166	0.809007951	1	0	0	1	0	0	0	1
1.154171435	-1.026915857	-0.616291025	-1.497459833	0.979529531	0	1	0	0	1	1	0	0
-0.729976334	1.305832838	0.011929851	0.786454619	0.577090913	0	1	0	0	1	0	0	1
1.157084394	-0.330572963	0.15153449	-0.654103792	1.321795882	0	1	0	1	1	0	0	0
0.611517866	1.688821429	-1.733128138	0.393447907	0.183101052	0	0	1	1	0	0	0	1
0.982230207	-0.783195844	0.15153449	-0.963194481	0.244522721	0	0	1	0	1	0	0	1
1.270996415	-0.67874441	-1.45391886	0.213028456	0.433523224	1	0	0	1	0	0	0	1
-0.006949288	-0.887647278	-0.127674788	1.031209689	0.002933737	1	0	0	1	0	0	0	1
-0.499080018	-1.375087304	0.779755366	0.729112003	-0.206922175	1	0	0	0	1	0	0	1
0.00270947	-0.08685295	-0.127674788	-1.588368859	0.684373512	0	1	0	0	1	0	0	1
1.292920263	-1.444721593	1.268371603	0.559881355	0.531256195	0	0	1	1	0	0	0	1

## Codes & it's explanations:-

### 1) 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.

## Machine Learning

```
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)
```



## Machine Learning

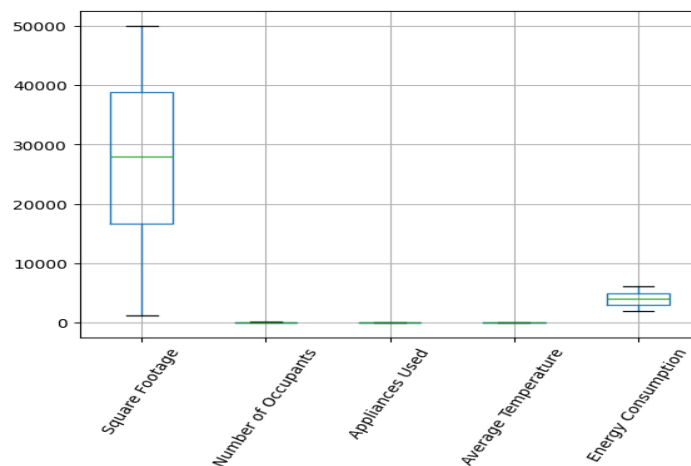
```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
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*, *R<sup>2</sup> Score*) are used to assess its performance.

→ **Output:**



Before Standardization:

Square Footage - Mean: 27672.6545, Std Dev: 13045.1552  
Number of Occupants - Mean: 49.4945, Std Dev: 28.7215  
Appliances Used - Mean: 25.8291, Std Dev: 14.3262  
Average Temperature - Mean: 22.6168, Std Dev: 7.1500  
Energy Consumption - Mean: 3995.7822, Std Dev: 1144.5472

After Standardization:

Square Footage - Mean: 0.0000, Std Dev: 1.0000  
Number of Occupants - Mean: 0.0000, Std Dev: 1.0000  
Appliances Used - Mean: 0.0000, Std Dev: 1.0000  
Average Temperature - Mean: -0.0000, Std Dev: 1.0000  
Energy Consumption - Mean: -0.0000, Std Dev: 1.0000

# Machine Learning

## Model Performance:

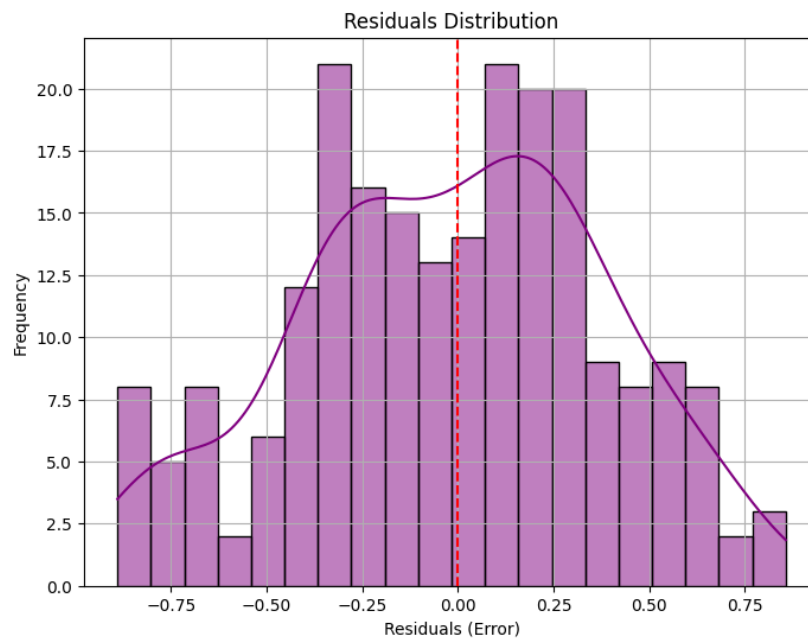
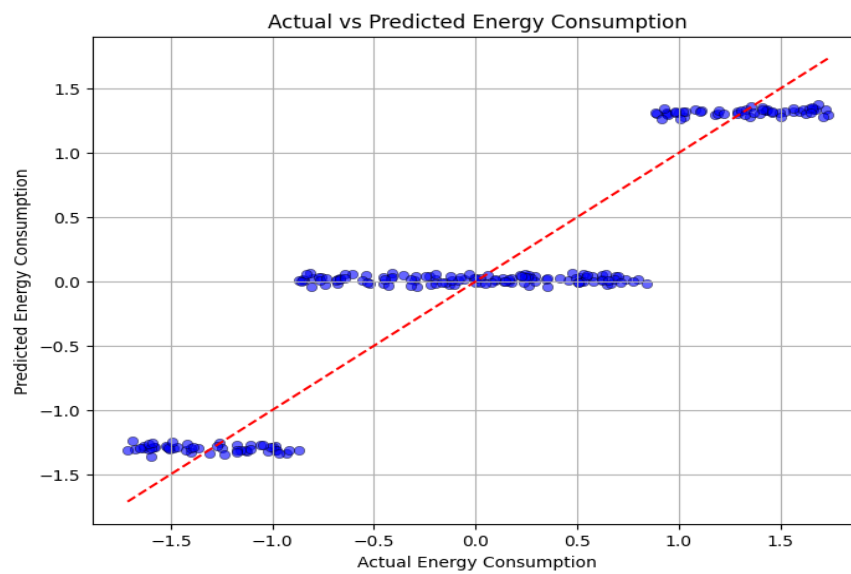
Mean Absolute Error (MAE): 0.3282

Mean Squared Error (MSE): 0.1587

Root Mean Squared Error (RMSE): 0.3984

R-squared ( $R^2$ ): 0.8414

Model saved as 'mlr\_energy\_consumption\_model.pkl' for future predictions.



## 2) 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;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Energy Consumption Prediction</h1>
    <form>
      <label>Enter your energy consumption data</label>
      <input type="text">
    </form>
  </div>
</body>
</html>
```

```
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 }}"**.

### 3) 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.

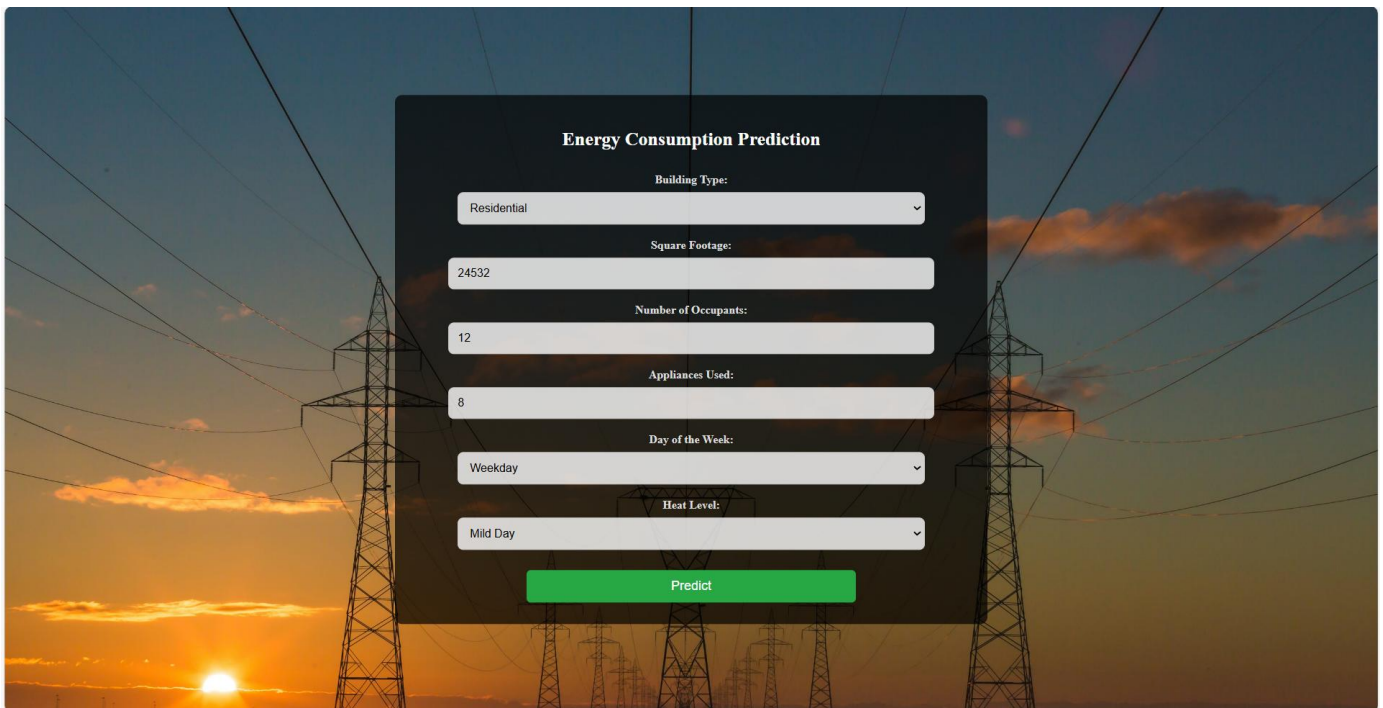
→ **Standardizing the Numerical Features & One-Hot Encoding Categorical Variables:-**

- To match the trained model's format, input values are normalized using:
  - $\text{standardized value} = \text{value} - \text{mean} / \text{std}$
- Then categorical values are converted to their respective encoded lists

## Machine Learning

- **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:
    - $\text{actual value} = (\text{predicted standardized value} \times \text{std}) + \text{mean}$
  - 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:-

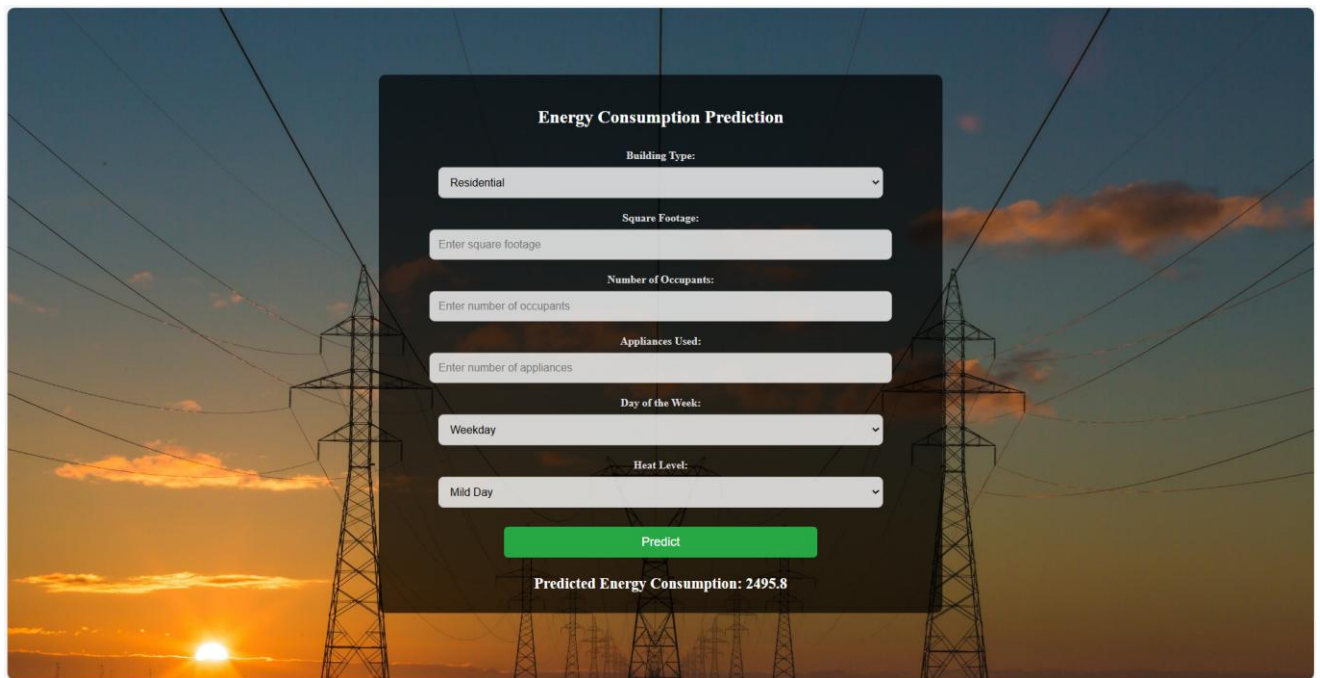


The screenshot shows a web application titled "Energy Consumption Prediction" overlaid on a background image of power lines at sunset. The form contains the following fields and values:

Field	Value
Building Type:	Residential
Square Footage:	24532
Number of Occupants:	12
Appliances Used:	8
Day of the Week:	Weekday
Heat Level:	Mild Day

A green "Predict" button is located at the bottom of the form.

# Machine Learning



The form is titled "Energy Consumption Prediction" and is set against a background of power lines at sunset. It contains several input fields and a "Predict" button. The values entered are: Building Type: Residential, Square Footage: (empty), Number of Occupants: (empty), Appliances Used: (empty), Day of the Week: Weekday, and Heat Level: Mild Day. The predicted energy consumption is 2495.8.

**Energy Consumption Prediction**

Building Type: Residential

Square Footage: Enter square footage

Number of Occupants: Enter number of occupants

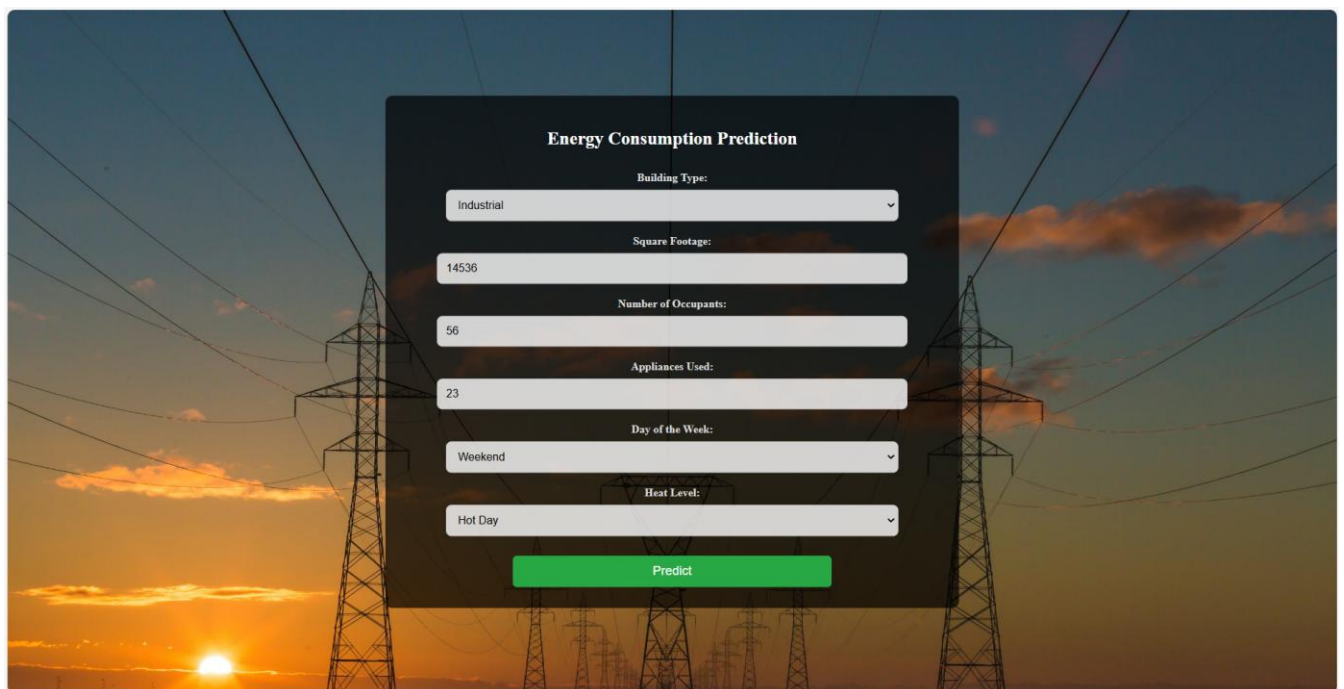
Appliances Used: Enter number of appliances

Day of the Week: Weekday

Heat Level: Mild Day

**Predict**

**Predicted Energy Consumption: 2495.8**



The form is titled "Energy Consumption Prediction" and is set against a background of power lines at sunset. It contains several input fields and a "Predict" button. The values entered are: Building Type: Industrial, Square Footage: 14536, Number of Occupants: 56, Appliances Used: 23, Day of the Week: Weekend, and Heat Level: Hot Day. The predicted energy consumption is 2495.8.

**Energy Consumption Prediction**

Building Type: Industrial

Square Footage: 14536

Number of Occupants: 56

Appliances Used: 23

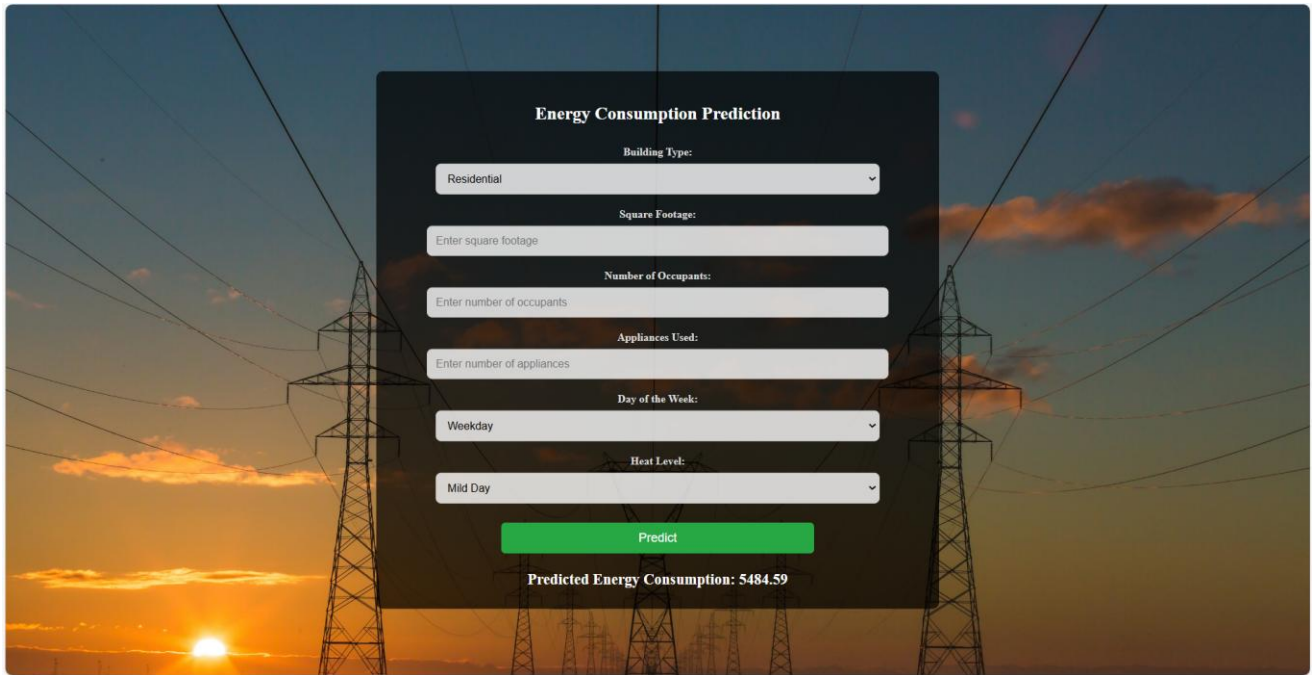
Day of the Week: Weekend

Heat Level: Hot Day

**Predict**

**Predicted Energy Consumption: 2495.8**



A screenshot of a web application titled "Energy Consumption Prediction". The interface is overlaid on a background image of power lines and a sunset. The form contains several input fields: "Building Type" (a dropdown menu with "Residential" selected), "Square Footage" (a text input field with placeholder text "Enter square footage"), "Number of Occupants" (a text input field with placeholder text "Enter number of occupants"), "Appliances Used" (a text input field with placeholder text "Enter number of appliances"), "Day of the Week" (a dropdown menu with "Weekday" selected), and "Heat Level" (a dropdown menu with "Mild Day" selected). Below these fields is a green "Predict" button. At the bottom of the form, it displays "Predicted Energy Consumption: 5484.59".

**Energy Consumption Prediction**

Building Type: Residential

Square Footage: Enter square footage

Number of Occupants: Enter number of occupants

Appliances Used: Enter number of appliances

Day of the Week: Weekday

Heat Level: Mild Day

Predict

Predicted Energy Consumption: 5484.59

---

## 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.