-------- Energy Consumption Prediction for Smart Buildings --------

- Predicting energy consumption is crucial for optimizing energy usage and improving efficiency in smart buildings.
- This project aims to develop predictive models to estimate the energy consumption of a house based on various factors, including appliance usage and weather conditions.

In [ ]:

In [1]:
```
import pandas as pd
```

In [2]:
```
energy = pd.read_csv('EnergyConsumption2LakhFinal.csv')
energy
```

Out[2]:

	ID	Time	Gen	Dish Washer	HVAC	Home Office	Fridge	Wine Cellar	Garage Door	Kitchen	...	Temperature	Weather Condition
0	1	1451624400	0.003483	0.000033	0.061917	0.442633	0.124150	0.006983	0.013083	0.000417	...	36.14	clear-night
1	2	1451624401	0.003467	0.000000	0.063817	0.444067	0.124000	0.006983	0.013117	0.000417	...	36.14	clear-night
2	3	1451624402	0.003467	0.000017	0.062317	0.446067	0.123533	0.006983	0.013083	0.000433	...	36.14	clear-night
3	4	1451624403	0.003483	0.000017	0.068517	0.446583	0.123133	0.006983	0.013000	0.000433	...	36.14	clear-night
4	5	1451624404	0.003467	0.000133	0.063983	0.446533	0.122850	0.006850	0.012783	0.000450	...	36.14	clear-night
...	...	...	...	...	...	...	...	...	...	...	...	...	...
199995	199996	1451824395	0.003183	0.000017	0.064117	0.497433	0.004767	0.007000	0.012883	0.000733	...	78.49	partly-cloudy-day
199996	199997	1451824396	0.003200	0.000017	0.063683	0.489967	0.004750	0.006900	0.012850	0.000733	...	78.49	partly-cloudy-day
199997	199998	1451824397	0.003350	0.000017	0.062667	0.495183	0.004933	0.006850	0.012867	0.000733	...	78.49	partly-cloudy-day
199998	199999	1451824398	0.003333	0.000017	0.063000	0.484400	0.004983	0.006767	0.012983	0.000750	...	78.49	partly-cloudy-day
199999	200000	1451824399	0.003217	0.000017	0.062900	0.483683	0.004783	0.006850	0.013000	0.000733	...	78.49	partly-cloudy-day

200000 rows × 25 columns

In [ ]:

Checking Columns

In [3]:
```
energy.columns
```

Out[3]:
```
Index(['ID', 'Time', 'Gen', 'Dish Washer', 'HVAC', 'Home Office ', 'Fridge',
       'Wine Cellar', 'Garage Door', 'Kitchen', 'Barn', 'Well', 'Microwave',
       'Living Room', 'Solar', 'Temperature', 'Weather Condition',
       'Weather Summary', 'Aapparent Temperature', 'Pressure', 'Wind Speed',
       'Precip Intensity', 'Precip Probability', 'Dew Point',
       'TotalEnergyConsumptionOfTheHouse'],
      dtype='object')
```

In [ ]:

In [4]:
```
# str.replace() method is used to replace spaces with an empty string.
energy.columns = energy.columns.str.replace(' ', '')
```

In [5]:
```
energy.columns
```

Out[5]:
```
Index(['ID', 'Time', 'Gen', 'DishWasher', 'HVAC', 'HomeOffice', 'Fridge',
       'WineCellar', 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
       'LivingRoom', 'Solar', 'Temperature', 'WeatherCondition',
       'WeatherSummary', 'AapparentTemperature', 'Pressure', 'WindSpeed',
       'PrecipIntensity', 'PrecipProbability', 'DewPoint',
       'TotalEnergyConsumptionOfTheHouse'],
      dtype='object')
```

--- Data Description ---

The dataset contains 200,000 rows and 20 columns. Here is a brief description of the columns:

- `ID` : Unique identifier for each record.
- `Time` : Timestamp of the record.
- `Gen` : Energy generated.
- `Dish Washer` , `HVAC` , `Home Office` , etc.: Energy consumption of various appliances.
- `Temperature` : Outdoor temperature.
- `Weather Condition` : Description of the weather (e.g., clear-night).
- `WeatherSummary` , `WindSpeed` , `Pressure` , etc.: Energy consumption based on environmental conditions ..
- `TotalEnergyConsumptionOfTheHouse` : Total energy consumption of the house.

In [ ]:

In [ ]:

--- Visualization (EDA) ---

In [ ]:

In [6]:
```
#Displaying top 5 records
energy.head(5)
```

Out[6]:

	ID	Time	Gen	DishWasher	HVAC	HomeOffice	Fridge	WineCellar	GarageDoor	Kitchen	...	Temperature	WeatherC
0	1	1451624400	0.003483	0.000033	0.061917	0.442633	0.124150	0.006983	0.013083	0.000417	...	36.14	cl
1	2	1451624401	0.003467	0.000000	0.063817	0.444067	0.124000	0.006983	0.013117	0.000417	...	36.14	cl
2	3	1451624402	0.003467	0.000017	0.062317	0.446067	0.123533	0.006983	0.013083	0.000433	...	36.14	cl
3	4	1451624403	0.003483	0.000017	0.068517	0.446583	0.123133	0.006983	0.013000	0.000433	...	36.14	cl
4	5	1451624404	0.003467	0.000133	0.063983	0.446533	0.122850	0.006850	0.012783	0.000450	...	36.14	cl

5 rows × 25 columns

In [7]:
```
#Displaying last 5 records
energy.tail(5)
```

Out[7]:

	ID	Time	Gen	DishWasher	HVAC	HomeOffice	Fridge	WineCellar	GarageDoor	Kitchen	...	Temperature
199995	199996	1451824395	0.003183	0.000017	0.064117	0.497433	0.004767	0.007000	0.012883	0.000733	...	78.49
199996	199997	1451824396	0.003200	0.000017	0.063683	0.489967	0.004750	0.006900	0.012850	0.000733	...	78.49
199997	199998	1451824397	0.003350	0.000017	0.062667	0.495183	0.004933	0.006850	0.012867	0.000733	...	78.49
199998	199999	1451824398	0.003333	0.000017	0.063000	0.484400	0.004983	0.006767	0.012983	0.000750	...	78.49
199999	200000	1451824399	0.003217	0.000017	0.062900	0.483683	0.004783	0.006850	0.013000	0.000733	...	78.49

5 rows × 25 columns

In [ ]:
```
```

In [8]:
```
#checking shape
energy.shape
```

Out[8]:
```
(200000, 25)
```

It has 2Lakh records and 25 columns

In [ ]:
```
```

In [9]:
```
#displaying columns name
energy.columns
```

Out[9]:
```
Index(['ID', 'Time', 'Gen', 'DishWasher', 'HVAC', 'HomeOffice', 'Fridge',
       'WineCellar', 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
       'LivingRoom', 'Solar', 'Temperature', 'WeatherCondition',
       'WeatherSummary', 'AapparentTemperature', 'Pressure', 'WindSpeed',
       'PrecipIntensity', 'PrecipProbability', 'DewPoint',
       'TotalEnergyConsumptionOfTheHouse'],
      dtype='object')
```

In [10]:
```
#dropping ID Columns bec we dont need them

energy = energy.drop(['ID'], axis=1)
energy.columns
```

Out[10]:
```
Index(['Time', 'Gen', 'DishWasher', 'HVAC', 'HomeOffice', 'Fridge',
       'WineCellar', 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
       'LivingRoom', 'Solar', 'Temperature', 'WeatherCondition',
       'WeatherSummary', 'AapparentTemperature', 'Pressure', 'WindSpeed',
       'PrecipIntensity', 'PrecipProbability', 'DewPoint',
       'TotalEnergyConsumptionOfTheHouse'],
      dtype='object')
```

In [ ]:

In [ ]:

Summary Statistics

In [11]:
```
# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(energy.head())
```

First few rows of the dataset:

	Time	Gen	DishWasher	HVAC	HomeOffice	Fridge	\
0	1451624400	0.003483	0.000033	0.061917	0.442633	0.124150	
1	1451624401	0.003467	0.000000	0.063817	0.444067	0.124000	
2	1451624402	0.003467	0.000017	0.062317	0.446067	0.123533	
3	1451624403	0.003483	0.000017	0.068517	0.446583	0.123133	
4	1451624404	0.003467	0.000133	0.063983	0.446533	0.122850	

	WineCellar	GarageDoor	Kitchen	Barn	...	Temperature	\
0	0.006983	0.013083	0.000417	0.031350	...	36.14	
1	0.006983	0.013117	0.000417	0.031500	...	36.14	
2	0.006983	0.013083	0.000433	0.031517	...	36.14	
3	0.006983	0.013000	0.000433	0.031500	...	36.14	
4	0.006850	0.012783	0.000450	0.031500	...	36.14	

	WeatherCondition	WeatherSummary	AapparentTemperature	Pressure	WindSpeed	\
0	clear-night	Clear	29.26	1016.91	9.18	
1	clear-night	Clear	29.26	1016.91	9.18	
2	clear-night	Clear	29.26	1016.91	9.18	
3	clear-night	Clear	29.26	1016.91	9.18	
4	clear-night	Clear	29.26	1016.91	9.18	

	PrecipIntensity	PrecipProbability	DewPoint	\
0	0.0	0.0	24.4	
1	0.0	0.0	24.4	
2	0.0	0.0	24.4	
3	0.0	0.0	24.4	
4	0.0	0.0	24.4	

	TotalEnergyConsumptionOfTheHouse
0	0.932833
1	0.934333
2	0.931817
3	1.022050
4	1.139400

[5 rows x 24 columns]

Observations:

1. The dataset contains energy consumption data for various appliances and features in a smart building, recorded at regular time intervals.
2. Each row represents a specific time point, with corresponding values for energy generation, energy consumption by appliances (e.g., DishWasher, HVAC, Fridge), and weather-related features.
3. The values in the dataset appear to be normalized or scaled, as they range between 0 and 1 for many features, facilitating comparative analysis.
4. Features such as temperature, weather condition, and wind speed are also included, suggesting the dataset encompasses both energy-related and external environmental factors.
5. The "TotalEnergyConsumptionOfTheHouse" column provides an aggregated measure of the house's total energy consumption at each time point, which could serve as a target variable for prediction tasks.

In [ ]:

In [12]:
```
# Geting the basic information about the dataset
print("\nDataset Information:")
print(energy.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 24 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   Time                               200000 non-null  int64
 1   Gen                                200000 non-null  float64
 2   DishWasher                         200000 non-null  float64
 3   HVAC                               200000 non-null  float64
 4   HomeOffice                         200000 non-null  float64
 5   Fridge                             200000 non-null  float64
 6   WineCellar                         200000 non-null  float64
 7   GarageDoor                         200000 non-null  float64
 8   Kitchen                            200000 non-null  float64
 9   Barn                               200000 non-null  float64
 10  Well                               200000 non-null  float64
 11  Microwave                          200000 non-null  float64
 12  LivingRoom                         200000 non-null  float64
 13  Solar                              200000 non-null  float64
 14  Temperature                        200000 non-null  float64
 15  WeatherCondition                   200000 non-null  object
 16  WeatherSummary                     200000 non-null  object
 17  AapparentTemperature               200000 non-null  float64
 18  Pressure                           200000 non-null  float64
 19  WindSpeed                          200000 non-null  float64
 20  PrecipIntensity                    200000 non-null  float64
 21  PrecipProbability                  200000 non-null  float64
 22  DewPoint                           200000 non-null  float64
 23  TotalEnergyConsumptionOfTheHouse   200000 non-null  float64
dtypes: float64(21), int64(1), object(2)
memory usage: 36.6+ MB
None
```

Observations:

--- Dataset Information: ---

- The dataset consists of **200,000 entries (rows)** and **24 columns**, representing various features related to energy consumption in a smart building.
- Features include energy consumption by different appliances (e.g., DishWasher, HVAC, Fridge) as well as environmental variables such as temperature, weather condition, and wind speed.
- The majority of columns contain **numerical data**, with **21 columns** being of type `float64` and **1 column** of type `int64` .
- Two columns, "**WeatherCondition**" and "**WeatherSummary**," are of **object type**, indicating categorical or string data, likely representing textual descriptions of weather conditions.
- There are **no missing values** in the dataset, as indicated by the "**Non-Null Count**" values for all columns, ensuring the dataset is complete and suitable for analysis.
- The **memory usage** of the dataset is approximately **36.6 MB**, providing an estimate of the computational resources required to manipulate and analyze the data.

In [ ]:

In [13]:
```
# Describe the dataset for basic statistical details
print("\nStatistical Summary:")
print(energy.describe())
```

Statistical Summary:

	Time	Gen	DishWasher	HVAC	\
count	2.000000e+05	200000.000000	200000.000000	200000.000000	
mean	1.451724e+09	0.074753	0.034697	0.178629	
std	5.773517e+04	0.133953	0.200665	0.209808	
min	1.451624e+09	0.000000	0.000000	0.000067	
25%	1.451674e+09	0.003200	0.000000	0.064867	
50%	1.451724e+09	0.003467	0.000033	0.078917	
75%	1.451774e+09	0.069717	0.000233	0.115650	
max	1.451824e+09	0.613883	1.393650	0.791033	

	HomeOffice	Fridge	WineCellar	GarageDoor	\
count	200000.000000	200000.000000	200000.000000	200000.000000	
mean	0.077968	0.055234	0.026007	0.013928	
std	0.096248	0.071927	0.045784	0.010789	
min	0.000083	0.000100	0.000017	0.000050	
25%	0.040283	0.005050	0.006800	0.012650	
50%	0.041883	0.005333	0.007100	0.012900	
75%	0.057483	0.122350	0.009217	0.013067	
max	0.971750	0.851267	1.189667	1.060433	

	Kitchen	Barn	...	LivingRoom	Solar	\
count	200000.000000	200000.000000	...	200000.000000	200000.000000	
mean	0.001590	0.065220	...	0.037346	0.074753	
std	0.019267	0.207231	...	0.099667	0.133953	
min	0.000000	0.000000	...	0.000000	0.000000	
25%	0.000433	0.030500	...	0.001450	0.003200	
50%	0.000650	0.031783	...	0.001583	0.003467	
75%	0.000717	0.033317	...	0.001683	0.069717	
max	0.944967	7.027900	...	0.418400	0.613883	

	Temperature	AapparentTemperature	Pressure	WindSpeed	\
count	200000.000000	200000.000000	200000.000000	200000.000000	
mean	39.490374	35.251548	1015.567284	7.427490	
std	14.837572	17.892835	8.672775	4.268729	
min	-12.640000	-32.080000	986.400000	0.000000	
25%	29.320000	23.220000	1009.920000	4.230000	
50%	39.510000	35.100000	1015.670000	6.720000	
75%	49.440000	47.790000	1021.650000	9.830000	
max	79.410000	79.410000	1036.620000	22.910000	

	PrecipIntensity	PrecipProbability	DewPoint	\
count	200000.00000	200000.000000	200000.000000	
mean	0.00244	0.056611	25.424201	

```
std            0.01071           0.163030      14.625872
min            0.00000           0.000000     -27.240000
25%            0.00000           0.000000      15.070000
50%            0.00000           0.000000      25.960000
75%            0.00000           0.000000      35.500000
max            0.18200           0.840000      59.520000

       TotalEnergyConsumptionOfTheHouse
count                     200000.000000
mean                           0.848338
std                            0.670934
min                            0.000000
25%                            0.410167
50%                            0.681933
75%                            1.109221
max                           11.673700

[8 rows x 22 columns]
```

Statistical Summary:

- The dataset comprises **200,000 entries (rows)** and **24 columns**.
- Below are the basic statistical details for each numerical column:

Feature	Mean	Std Dev	Min	25%	50%	75%	Max
Time							
Gen	0.074753	0.133953	0.000	0.0032	0.0035	0.0697	0.6139
DishWasher	0.034697	0.200665	0.000	0.000	0.00003	0.00023	1.3937
HVAC	0.178629	0.209808	0.0001	0.0649	0.0789	0.1157	0.791
...							
DewPoint	25.424	14.626	-27.24	15.07	25.96	35.5	59.52
TotalEnergyConsumptionOfTheHouse	0.848	0.671	0.000	0.410	0.682	1.109	11.674

- The dataset provides insights into the distribution and variability of each feature, aiding in understanding the underlying patterns and characteristics of the data.

In [ ]:

In [ ]:

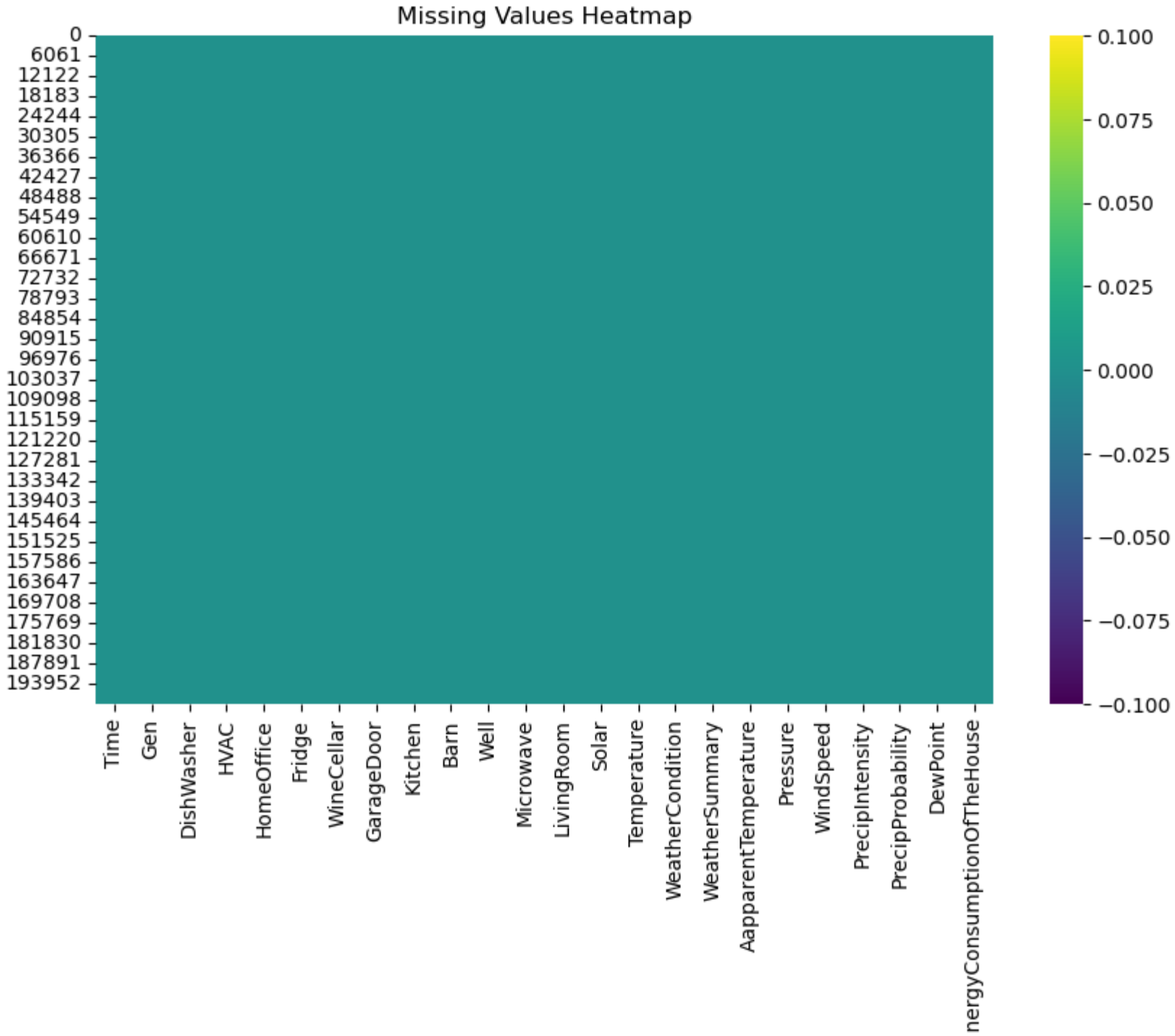
Missing Values Analysis:

- Visualing missing values with a heatmap or bar plot.

In [14]:
```
# Check for missing values
print("\nMissing Values:")
print(energy.isnull().sum())
```

```
Missing Values:
Time                                0
Gen                                 0
DishWasher                          0
HVAC                                0
HomeOffice                          0
Fridge                              0
WineCellar                          0
GarageDoor                          0
Kitchen                             0
Barn                                0
Well                                0
Microwave                           0
LivingRoom                          0
Solar                               0
Temperature                         0
WeatherCondition                    0
WeatherSummary                      0
AapparentTemperature                0
Pressure                            0
WindSpeed                           0
PrecipIntensity                     0
PrecipProbability                   0
DewPoint                            0
TotalEnergyConsumptionOfTheHouse    0
dtype: int64
```

In [15]:
```
# Visualize missing values
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.heatmap(energy.isnull(), cbar=True, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```


Missing Values Heatmap
0
6061
12122
18183
24244
30305
36366
42427
48488
54549
60610
66671
72732
78793
84854
90915
96976
103037
109098
115159
121220
127281
133342
139403
145464
151525
157586
163647
169708
175769
181830
187891
193952
Time
Gen
DishWasher
HVAC
HomeOffice
Fridge
WineCellar
GarageDoor
Kitchen
Barn
Well
Microwave
LivingRoom
Solar
Temperature
WeatherCondition
WeatherSummary
AapparentTemperature
Pressure
WindSpeed
PrecipIntensity
PrecipProbability
DewPoint
nergyConsumptionOfTheHouse
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

TotalE

In [16]:
```
#Here we don't have any nulls in our dataset. So it's showing looking this way
```

In [ ]:
```

```

Correlation Analysis:

- Using a heatmap to visualize correlations between numerical variables.

In [17]:
```
# Correlation matrix
correlation_matrix = energy.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
```

Correlation Matrix:

	Time	Gen	DishWasher	HVAC
Time	1.000000	0.087764	-0.001091	-0.254293
Gen	0.087764	1.000000	0.016304	-0.155586
DishWasher	-0.001091	0.016304	1.000000	-0.028415
HVAC	-0.254293	-0.155586	-0.028415	1.000000
HomeOffice	-0.048642	-0.101542	0.061620	-0.013987
Fridge	0.019483	-0.012865	0.036911	-0.028271
WineCellar	0.119925	0.036478	-0.012867	-0.027340
GarageDoor	0.024136	0.042273	-0.007310	-0.016189
Kitchen	-0.042204	-0.007179	-0.007464	0.008871
Barn	0.010559	0.022468	0.001309	-0.013562
Well	-0.007903	0.018104	0.007495	0.018999
Microwave	0.008734	0.005335	0.005157	-0.000364
LivingRoom	-0.018979	-0.046884	0.006621	0.012813
Solar	0.087764	1.000000	0.016304	-0.155586
Temperature	0.647055	0.007914	-0.006024	-0.226285
AapparentTemperature	0.653757	0.019601	-0.001893	-0.231624
Pressure	-0.012454	0.058179	-0.004048	-0.001497
WindSpeed	-0.103356	-0.110465	-0.012812	0.072355
PrecipIntensity	-0.002256	0.025957	0.008836	0.012537
PrecipProbability	0.039196	0.061280	0.010765	-0.015204
DewPoint	0.534543	0.039454	-0.002338	-0.195973
TotalEnergyConsumptionOfTheHouse	-0.218429	-0.293259	0.302679	0.455568

	HomeOffice	Fridge	WineCellar
Time	-0.048642	0.019483	0.119925
Gen	-0.101542	-0.012865	0.036478
DishWasher	0.061620	0.036911	-0.012867
HVAC	-0.013987	-0.028271	-0.027340
HomeOffice	1.000000	0.040099	-0.013708
Fridge	0.040099	1.000000	0.019844
WineCellar	-0.013708	0.019844	1.000000
GarageDoor	-0.018434	-0.000675	0.014767
Kitchen	-0.005213	-0.011024	-0.001639
Barn	-0.039609	-0.003596	0.022168
Well	-0.009173	0.008297	0.011489
Microwave	-0.008751	0.016850	0.011596
LivingRoom	-0.050203	0.052459	0.066067
Solar	-0.101542	-0.012865	0.036478
Temperature	-0.044146	0.020504	0.070667
AapparentTemperature	-0.041540	0.020546	0.072335
Pressure	0.002451	-0.011287	0.012120
WindSpeed	-0.007979	-0.007958	-0.009924

```
PrecipIntensity                  -0.027389  0.003912   -0.003348
PrecipProbability                -0.036233 -0.006394   -0.000049
DewPoint                         -0.018939  0.021933    0.037992
TotalEnergyConsumptionOfTheHouse  0.151407  0.176047    0.063249

                                 GarageDoor   Kitchen      Barn  ...  \
Time                               0.024136 -0.042204  0.010559  ...
Gen                                0.042273 -0.007179  0.022468  ...
DishWasher                        -0.007310 -0.007464  0.001309  ...
HVAC                              -0.016189  0.008871 -0.013562  ...
HomeOffice                        -0.018434 -0.005213 -0.039609  ...
Fridge                            -0.000675 -0.011024 -0.003596  ...
WineCellar                         0.014767 -0.001639  0.022168  ...
GarageDoor                         1.000000  0.000850  0.020016  ...
Kitchen                            0.000850  1.000000 -0.007503  ...
Barn                               0.020016 -0.007503  1.000000  ...
Well                               0.002491  0.011514  0.002315  ...
Microwave                          0.002404  0.006198  0.006066  ...
LivingRoom                         0.001608  0.061419 -0.000343  ...
Solar                              0.042273 -0.007179  0.022468  ...
Temperature                        0.016176 -0.040578 -0.002189  ...
AapparentTemperature               0.016704 -0.031746  0.002128  ...
Pressure                           0.009045 -0.028459  0.014559  ...
WindSpeed                         -0.011754 -0.030993 -0.036122  ...
PrecipIntensity                   -0.012156 -0.011253 -0.015719  ...
PrecipProbability                 -0.011470 -0.015964 -0.022650  ...
DewPoint                           0.010977 -0.026440  0.016930  ...
TotalEnergyConsumptionOfTheHouse  -0.003353  0.066430  0.330059  ...

                                 LivingRoom     Solar  Temperature  \
Time                              -0.018979  0.087764     0.647055
Gen                               -0.046884  1.000000     0.007914
DishWasher                         0.006621  0.016304    -0.006024
HVAC                               0.012813 -0.155586    -0.226285
HomeOffice                        -0.050203 -0.101542    -0.044146
Fridge                             0.052459 -0.012865     0.020504
WineCellar                         0.066067  0.036478     0.070667
GarageDoor                         0.001608  0.042273     0.016176
Kitchen                            0.061419 -0.007179    -0.040578
Barn                              -0.000343  0.022468    -0.002189
Well                               0.081987  0.018104    -0.004530
Microwave                          0.072375  0.005335     0.004440
LivingRoom                         1.000000 -0.046884    -0.001637
Solar                             -0.046884  1.000000     0.007914
```

```
Temperature                          -0.001637  0.007914     1.000000
AapparentTemperature                 -0.003505  0.019601     0.985480
Pressure                             -0.006485  0.058179    -0.257661
WindSpeed                            -0.018477 -0.110465     0.009909
PrecipIntensity                      -0.002784  0.025957     0.097246
PrecipProbability                    -0.005488  0.061280     0.116419
DewPoint                              0.006029  0.039454     0.791334
TotalEnergyConsumptionOfTheHouse      0.321245 -0.293259    -0.170553

                                   AapparentTemperature  Pressure  WindSpeed  \
Time                                           0.653757 -0.012454  -0.103356
Gen                                            0.019601  0.058179  -0.110465
DishWasher                                    -0.001893 -0.004048  -0.012812
HVAC                                          -0.231624 -0.001497   0.072355
HomeOffice                                    -0.041540  0.002451  -0.007979
Fridge                                         0.020546 -0.011287  -0.007958
WineCellar                                     0.072335  0.012120  -0.009924
GarageDoor                                     0.016704  0.009045  -0.011754
Kitchen                                       -0.031746 -0.028459  -0.030993
Barn                                           0.002128  0.014559  -0.036122
Well                                          -0.004505  0.002726  -0.001920
Microwave                                      0.006656 -0.002416  -0.015908
LivingRoom                                    -0.003505 -0.006485  -0.018477
Solar                                          0.019601  0.058179  -0.110465
Temperature                                    0.985480 -0.257661   0.009909
AapparentTemperature                           1.000000 -0.221654  -0.114969
Pressure                                      -0.221654  1.000000  -0.269121
WindSpeed                                     -0.114969 -0.269121   1.000000
PrecipIntensity                                0.101404 -0.236616   0.030741
PrecipProbability                              0.119527 -0.292104  -0.007200
DewPoint                                       0.807040 -0.434288  -0.156908
TotalEnergyConsumptionOfTheHouse              -0.174445 -0.029344   0.053386

                                   PrecipIntensity  PrecipProbability  \
Time                                     -0.002256           0.039196
Gen                                       0.025957           0.061280
DishWasher                                0.008836           0.010765
HVAC                                      0.012537          -0.015204
HomeOffice                               -0.027389          -0.036233
Fridge                                    0.003912          -0.006394
WineCellar                               -0.003348          -0.000049
GarageDoor                               -0.012156          -0.011470
Kitchen                                  -0.011253          -0.015964
Barn                                     -0.015719          -0.022650
```

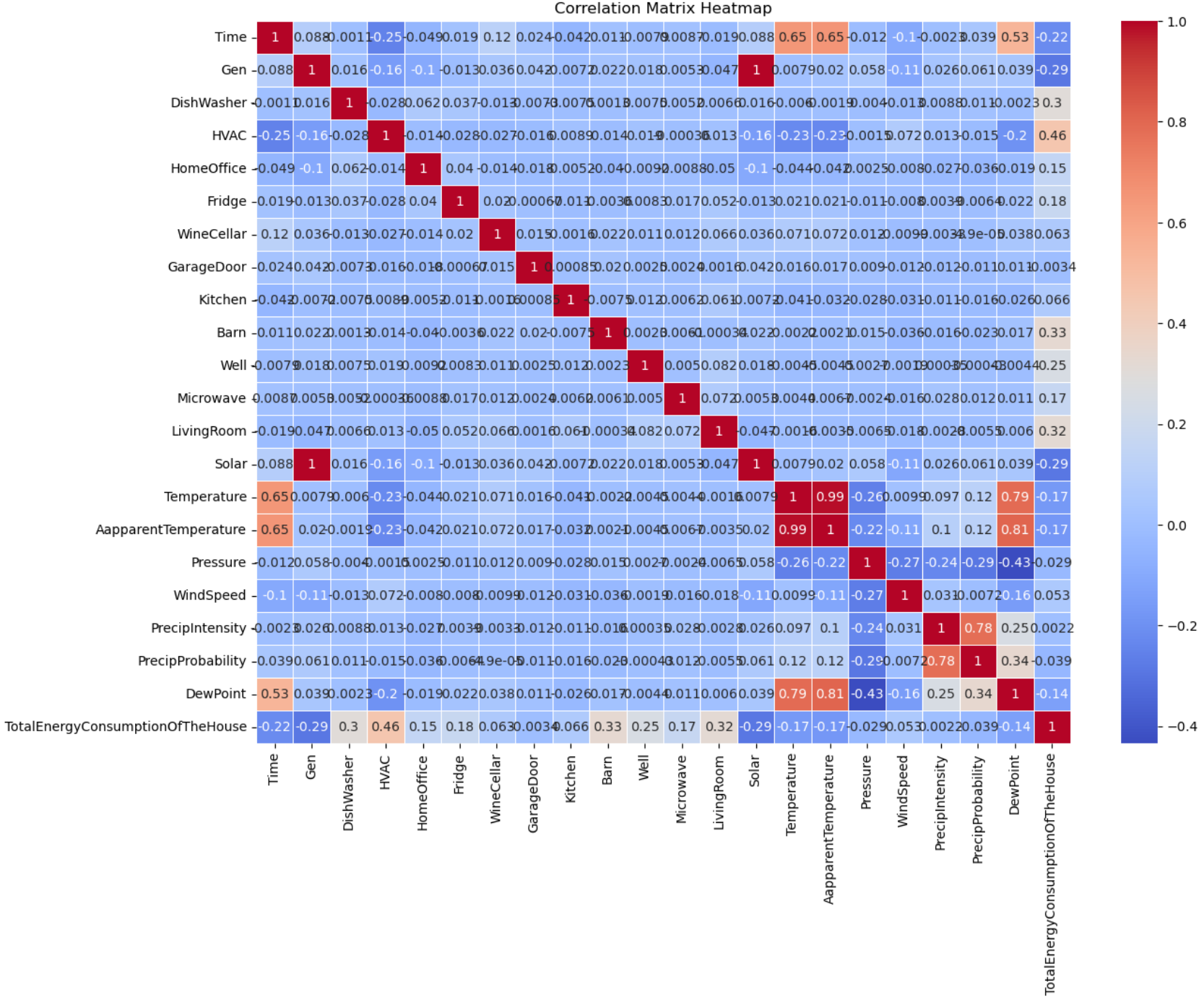
Well	0.000353	-0.000426
Microwave	0.028102	0.012323
LivingRoom	-0.002784	-0.005488
Solar	0.025957	0.061280
Temperature	0.097246	0.116419
AapparentTemperature	0.101404	0.119527
Pressure	-0.236616	-0.292104
WindSpeed	0.030741	-0.007200
PrecipIntensity	1.000000	0.777276
PrecipProbability	0.777276	1.000000
DewPoint	0.253097	0.337386
TotalEnergyConsumptionOfTheHouse	0.002160	-0.039436

	DewPoint	TotalEnergyConsumptionOfTheHouse
Time	0.534543	-0.218429
Gen	0.039454	-0.293259
DishWasher	-0.002338	0.302679
HVAC	-0.195973	0.455568
HomeOffice	-0.018939	0.151407
Fridge	0.021933	0.176047
WineCellar	0.037992	0.063249
GarageDoor	0.010977	-0.003353
Kitchen	-0.026440	0.066430
Barn	0.016930	0.330059
Well	-0.004423	0.251658
Microwave	0.010545	0.165270
LivingRoom	0.006029	0.321245
Solar	0.039454	-0.293259
Temperature	0.791334	-0.170553
AapparentTemperature	0.807040	-0.174445
Pressure	-0.434288	-0.029344
WindSpeed	-0.156908	0.053386
PrecipIntensity	0.253097	0.002160
PrecipProbability	0.337386	-0.039436
DewPoint	1.000000	-0.140192
TotalEnergyConsumptionOfTheHouse	-0.140192	1.000000

[22 rows x 22 columns]

In [18]:

```
# Visualize the correlation matrix
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```


Correlation Matrix Heatmap
Time
Gen
DishWasher
HVAC
HomeOffice
Fridge
WineCellar
GarageDoor
Kitchen
Barn
Well
Microwave
LivingRoom
Solar
Temperature
AapparentTemperature
Pressure
WindSpeed
PrecipIntensity
PrecipProbability
DewPoint
TotalEnergyConsumptionOfTheHouse
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

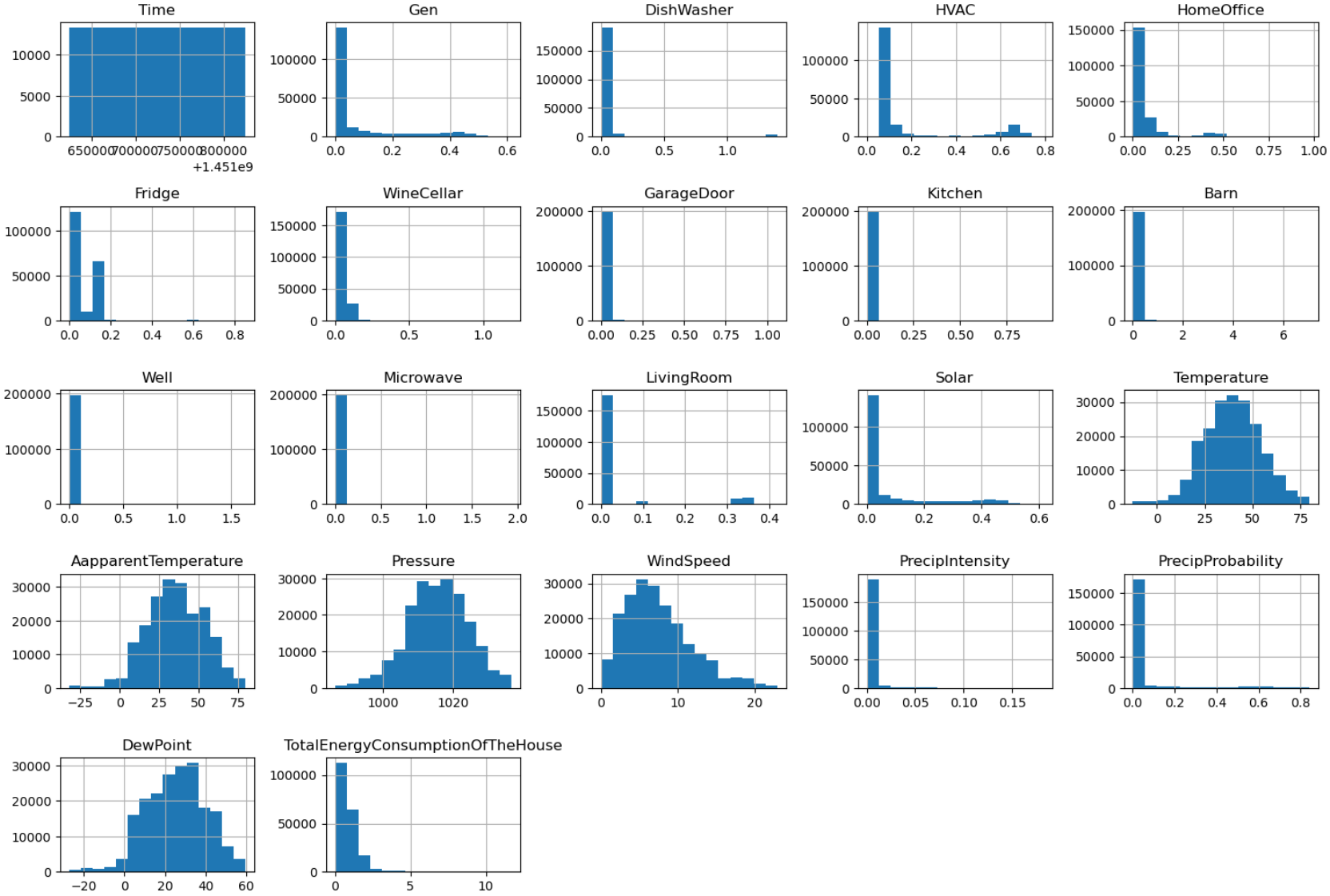
In [ ]:

In [ ]:

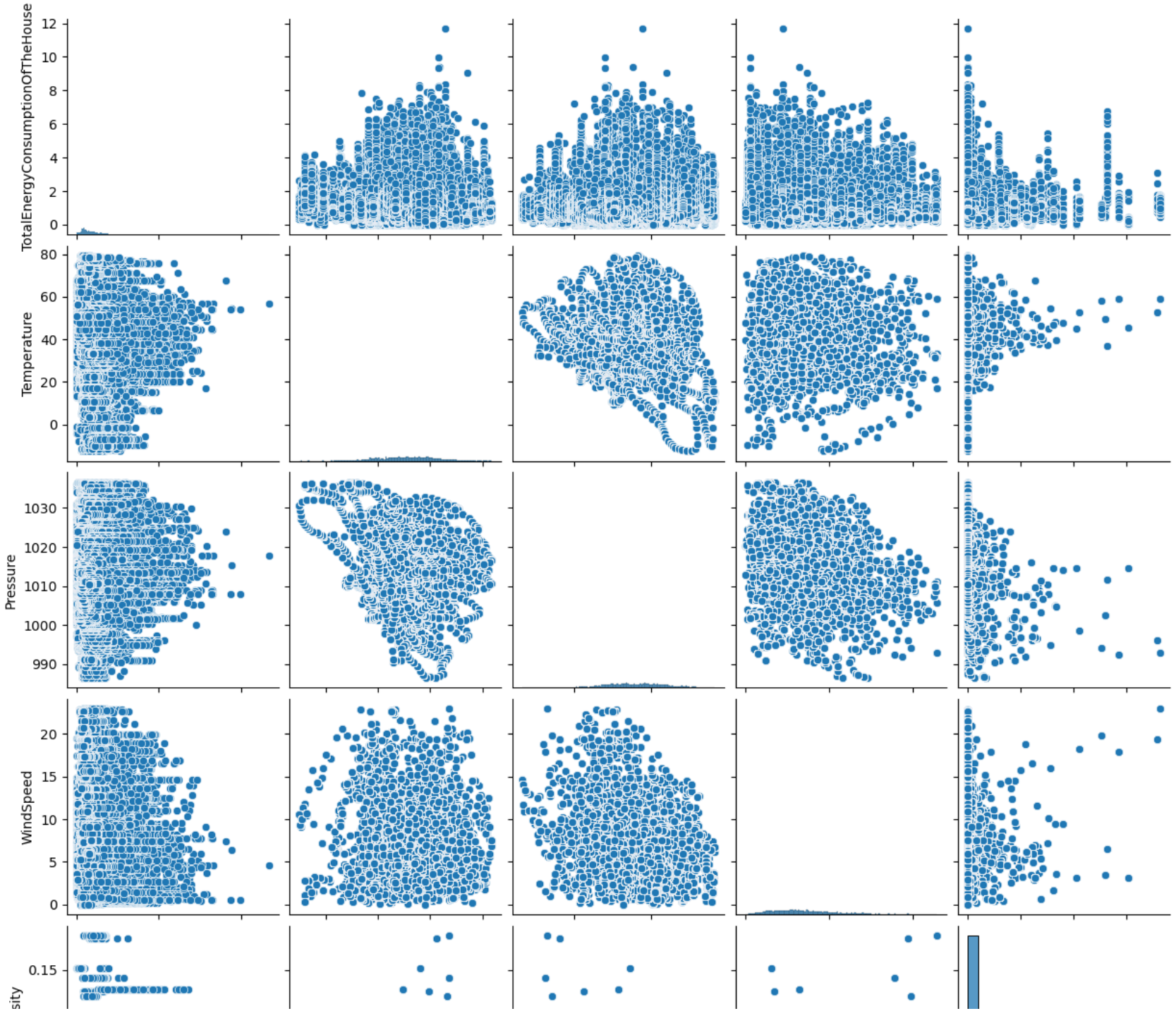
Distribution Analysis:

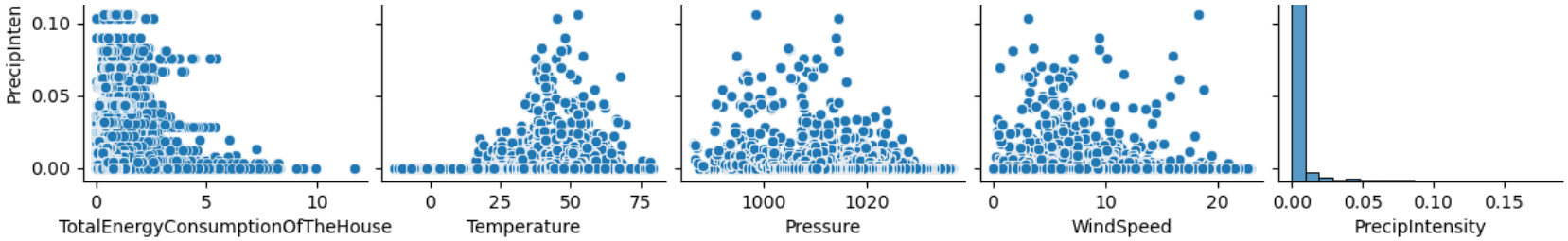
- Ploting histograms and boxplots to understand the distribution of numerical variables and categorical variables.

In [19]:
```
# Plot histograms for numerical columns
numerical_columns = energy.select_dtypes(include=['float64', 'int64']).columns
energy[numerical_columns].hist(bins=15, figsize=(15, 12), layout=(6, 5))
plt.tight_layout()
plt.show()
```



In [20]:
```
# Plot the pairplot for some numerical columns to check pairwise relationships
sns.pairplot(energy[['TotalEnergyConsumptionOfTheHouse', 'Temperature', 'Pressure', 'WindSpeed', 'PrecipIntensity']])
plt.show()
```


TotalEnergyConsumptionOfTheHouse
Temperature
Pressure
WindSpeed
sity
0
2
4
6
8
10
12
0
20
40
60
80
990
1000
1010
1020
1030
0
5
10
15
20
0.15



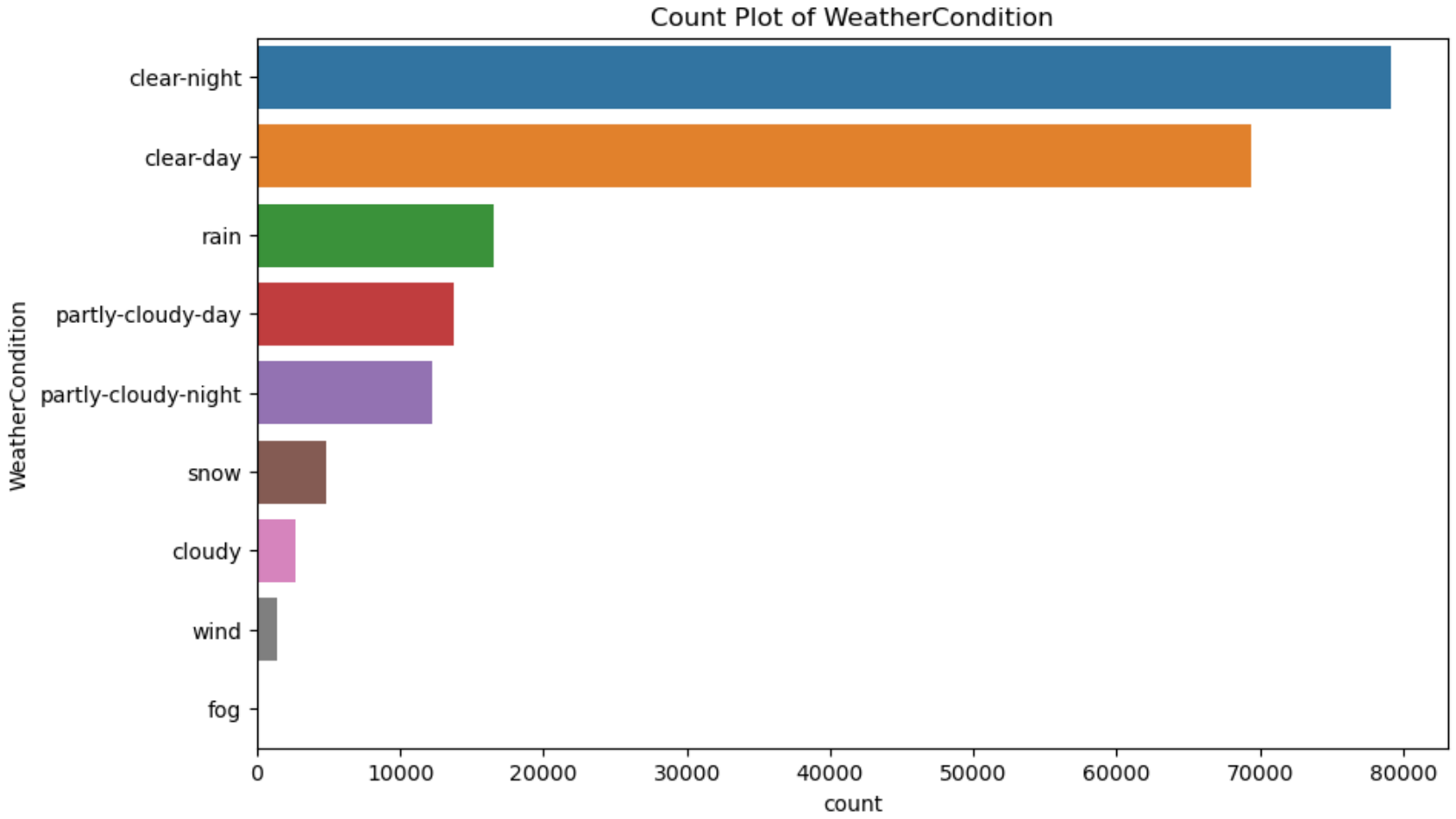
In [ ]:

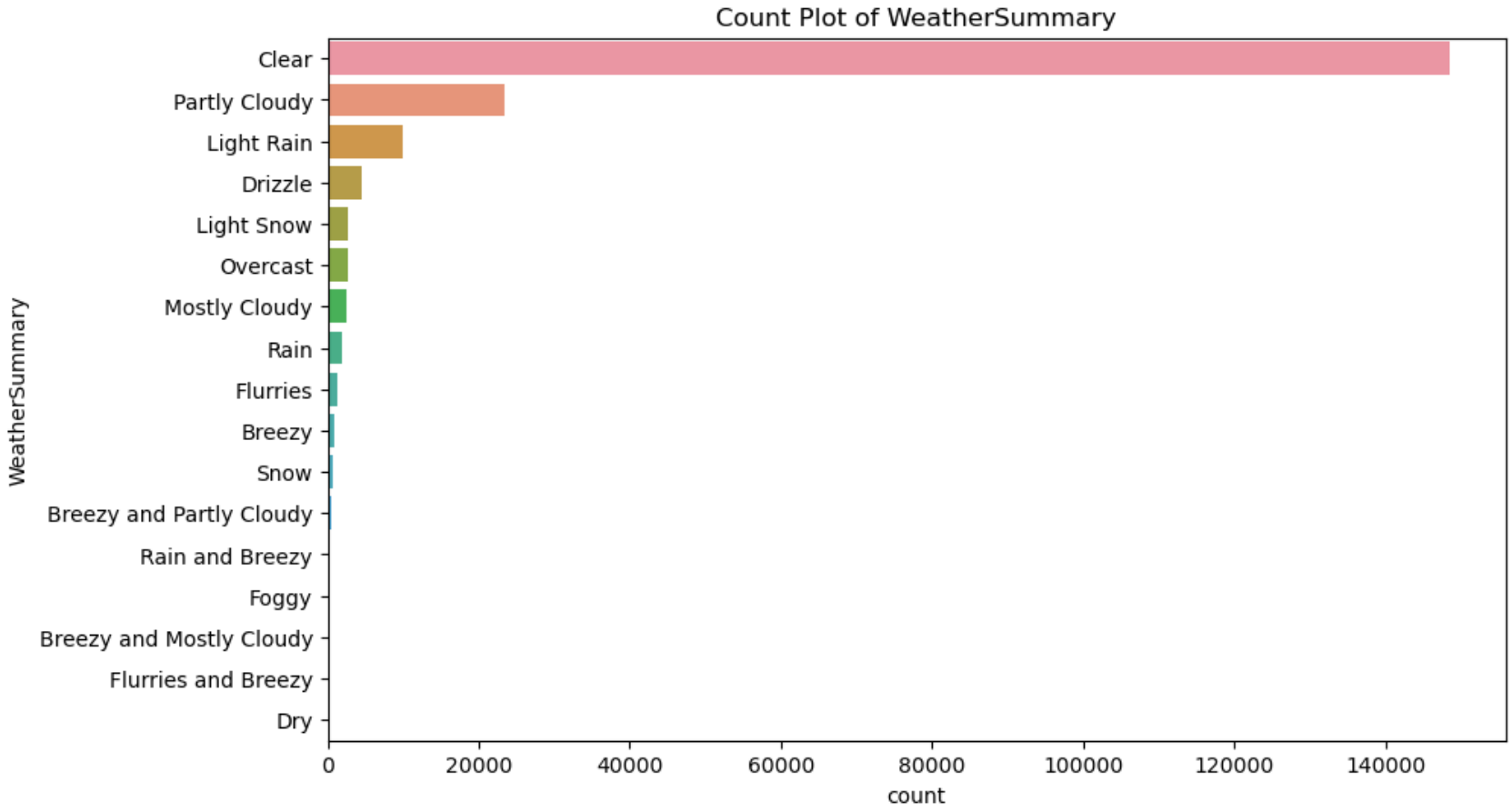
In [21]:
```
# Plot histograms for categorical features
categorical_columns = energy.select_dtypes(include=['object']).columns
print("\nCategorical Features:")
print(categorical_columns)
```

```
Categorical Features:
Index(['WeatherCondition', 'WeatherSummary'], dtype='object')
```

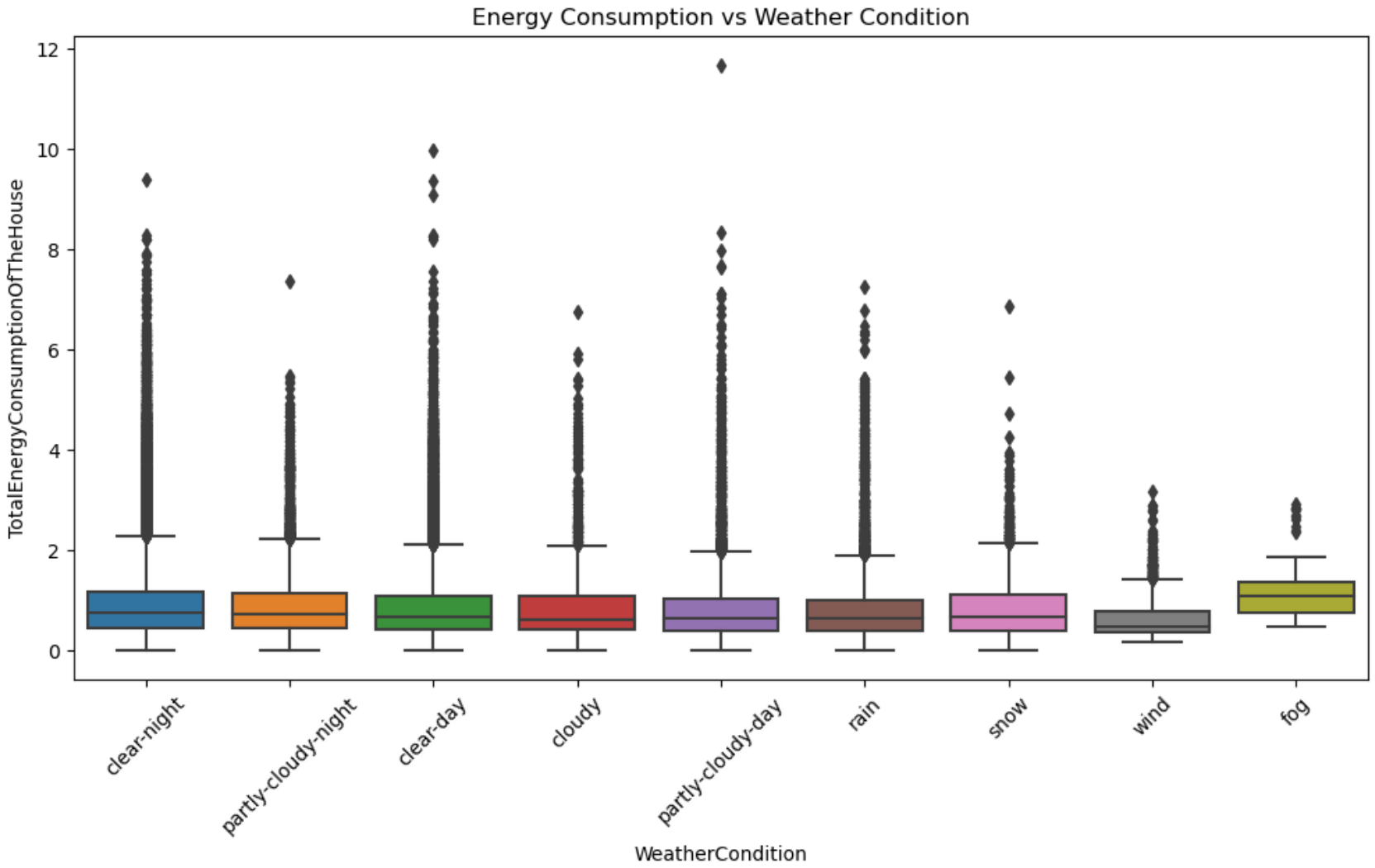
In [22]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Identify categorical columns
categorical_columns = energy.select_dtypes(include=['object']).columns

print("\nCategorical Features:")
print(categorical_columns)

# Count plots for categorical columns
for col in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(y=col, data=energy, order=energy[col].value_counts().index)
    plt.title(f'Count Plot of {col}')
    plt.show()
```

```
Categorical Features:
Index(['WeatherCondition', 'WeatherSummary'], dtype='object')
```


Count Plot of WeatherCondition
clear-night
clear-day
rain
partly-cloudy-day
partly-cloudy-night
snow
cloudy
wind
fog
WeatherCondition
0
10000
20000
30000
40000
50000
60000
70000
80000
count



In [23]:
```
# Analyze the relationship between Weather Conditions and Energy Consumption
plt.figure(figsize=(12, 6))
sns.boxplot(x='WeatherCondition', y='TotalEnergyConsumptionOfTheHouse', data=energy)
plt.xticks(rotation=45)
plt.title('Energy Consumption vs Weather Condition')
plt.show()
```

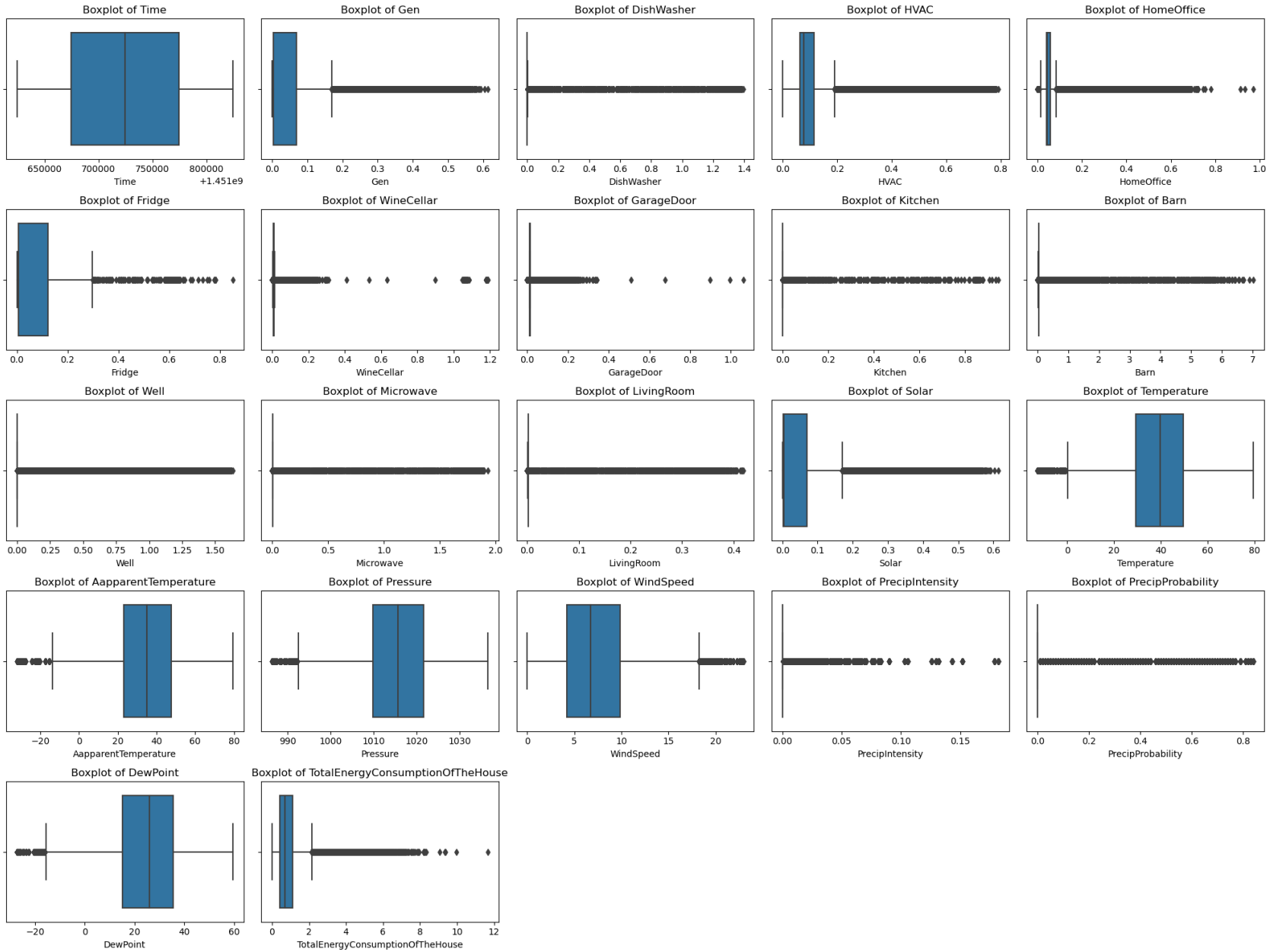

Energy Consumption vs Weather Condition
TotalEnergyConsumptionOfTheHouse
0
2
4
6
8
10
12
clear-night
partly-cloudy-night
clear-day
cloudy
partly-cloudy-day
rain
snow
wind
fog
WeatherCondition

In [ ]:

Outliers

In [24]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20, 15))

for i, col in enumerate(numerical_columns, 1):
    plt.subplot(5, 5, i)
    sns.boxplot(x=energy[col])  # Pass the column as a keyword argument
    plt.title(f'Boxplot of {col}')

plt.tight_layout()
plt.show()
```


Boxplot of Time
650000 700000 750000 800000
Time
+1.451e9
Boxplot of Gen
0.0 0.1 0.2 0.3 0.4 0.5 0.6
Gen
Boxplot of DishWasher
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4
DishWasher
Boxplot of HVAC
0.0 0.2 0.4 0.6 0.8
HVAC
Boxplot of HomeOffice
0.0 0.2 0.4 0.6 0.8 1.0
HomeOffice
Boxplot of Fridge
0.0 0.2 0.4 0.6 0.8
Fridge
Boxplot of WineCellar
0.0 0.2 0.4 0.6 0.8 1.0 1.2
WineCellar
Boxplot of GarageDoor
0.0 0.2 0.4 0.6 0.8 1.0
GarageDoor
Boxplot of Kitchen
0.0 0.2 0.4 0.6 0.8
Kitchen
Boxplot of Barn
0 1 2 3 4 5 6 7
Barn
Boxplot of Well
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Well
Boxplot of Microwave
0.0 0.5 1.0 1.5 2.0
Microwave
Boxplot of LivingRoom
0.0 0.1 0.2 0.3 0.4
LivingRoom
Boxplot of Solar
0.0 0.1 0.2 0.3 0.4 0.5 0.6
Solar
Boxplot of Temperature
0 20 40 60 80
Temperature
Boxplot of AapparentTemperature
−20 0 20 40 60 80
AapparentTemperature
Boxplot of Pressure
990 1000 1010 1020 1030
Pressure
Boxplot of WindSpeed
0 5 10 15 20
WindSpeed
Boxplot of PrecipIntensity
0.00 0.05 0.10 0.15
PrecipIntensity
Boxplot of PrecipProbability
0.0 0.2 0.4 0.6 0.8
PrecipProbability
Boxplot of DewPoint
−20 0 20 40 60
DewPoint
Boxplot of TotalEnergyConsumptionOfTheHouse
0 2 4 6 8 10 12
TotalEnergyConsumptionOfTheHouse

In [ ]:

In [ ]:

In [ ]:

In [ ]:

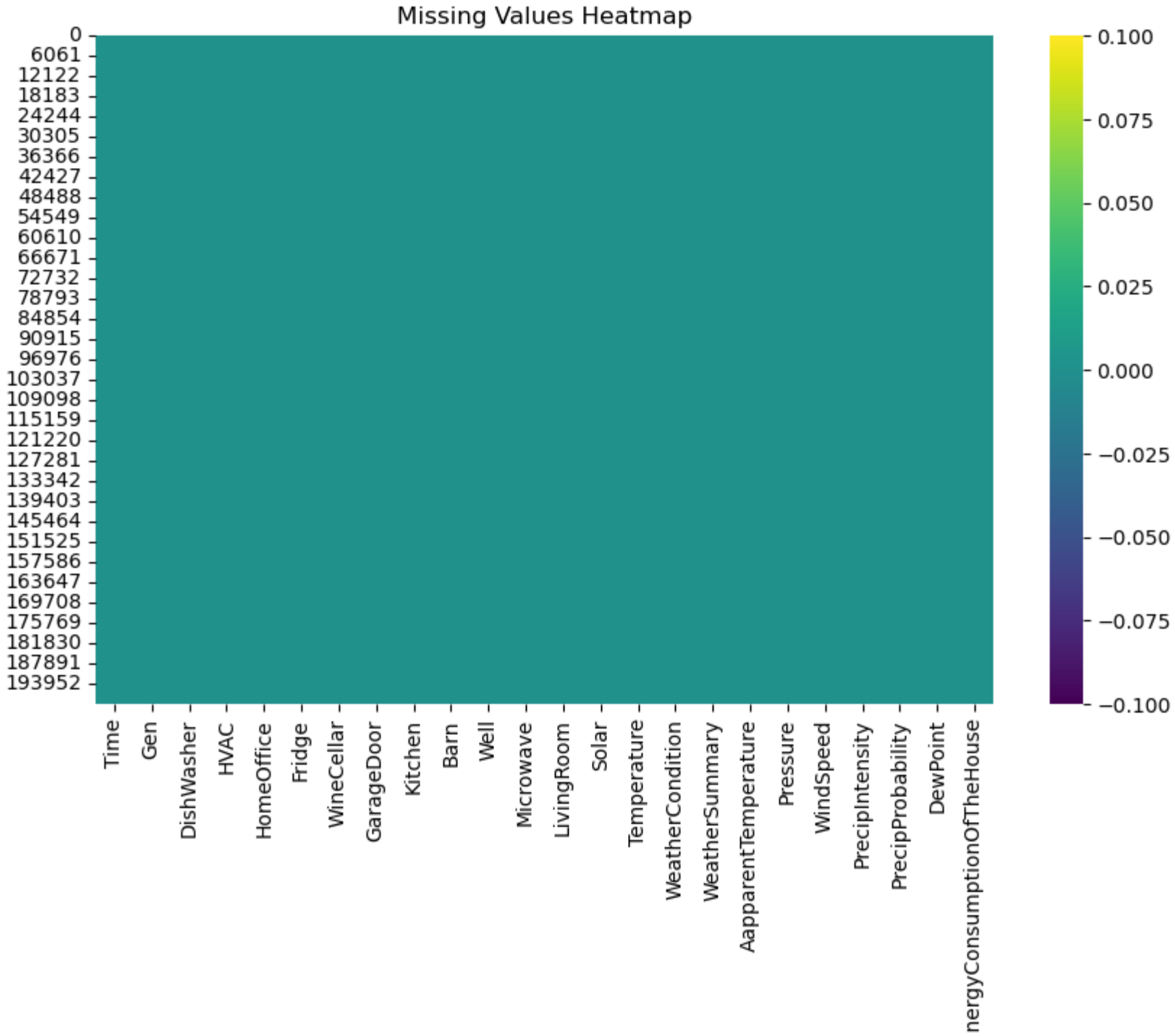
--- Data Cleaning ---

Checking for Null Values

In [25]:
```
energy.isnull().sum()[energy.isnull().sum() > 0]
```

Out[25]:
```
Series([], dtype: int64)
```

In [26]:
```
# Visualize missing values
plt.figure(figsize=(10, 6))
sns.heatmap(energy.isnull(), cbar=True, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```


Missing Values Heatmap
0
6061
12122
18183
24244
30305
36366
42427
48488
54549
60610
66671
72732
78793
84854
90915
96976
103037
109098
115159
121220
127281
133342
139403
145464
151525
157586
163647
169708
175769
181830
187891
193952
Time
Gen
DishWasher
HVAC
HomeOffice
Fridge
WineCellar
GarageDoor
Kitchen
Barn
Well
Microwave
LivingRoom
Solar
Temperature
WeatherCondition
WeatherSummary
AapparentTemperature
Pressure
WindSpeed
PrecipIntensity
PrecipProbability
DewPoint
nergyConsumptionOfTheHouse
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

TotalE

In [27]:
```
#we don't have any null/missing values
```

In [ ]:
```

```

Checking if any object datatype is present or no

In [28]:
```
energy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 24 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Time                              200000 non-null  int64  
 1   Gen                               200000 non-null  float64
 2   DishWasher                        200000 non-null  float64
 3   HVAC                              200000 non-null  float64
 4   HomeOffice                        200000 non-null  float64
 5   Fridge                            200000 non-null  float64
 6   WineCellar                        200000 non-null  float64
 7   GarageDoor                        200000 non-null  float64
 8   Kitchen                           200000 non-null  float64
 9   Barn                              200000 non-null  float64
 10  Well                              200000 non-null  float64
 11  Microwave                         200000 non-null  float64
 12  LivingRoom                        200000 non-null  float64
 13  Solar                             200000 non-null  float64
 14  Temperature                       200000 non-null  float64
 15  WeatherCondition                  200000 non-null  object 
 16  WeatherSummary                    200000 non-null  object 
 17  AapparentTemperature              200000 non-null  float64
 18  Pressure                          200000 non-null  float64
 19  WindSpeed                         200000 non-null  float64
 20  PrecipIntensity                   200000 non-null  float64
 21  PrecipProbability                 200000 non-null  float64
 22  DewPoint                          200000 non-null  float64
 23  TotalEnergyConsumptionOfTheHouse  200000 non-null  float64
dtypes: float64(21), int64(1), object(2)
memory usage: 36.6+ MB
```

In [29]:
```
#we have 2 objects in our dataset.Let's handel them
```

In [ ]:
```

```

Displaying only object columns

In [30]:
```
energy.select_dtypes(include='object').columns
```

Out[30]:
```
Index(['WeatherCondition', 'WeatherSummary'], dtype='object')
```

In [ ]:
```
```

In [31]:
```
energy.WeatherCondition.unique()
```

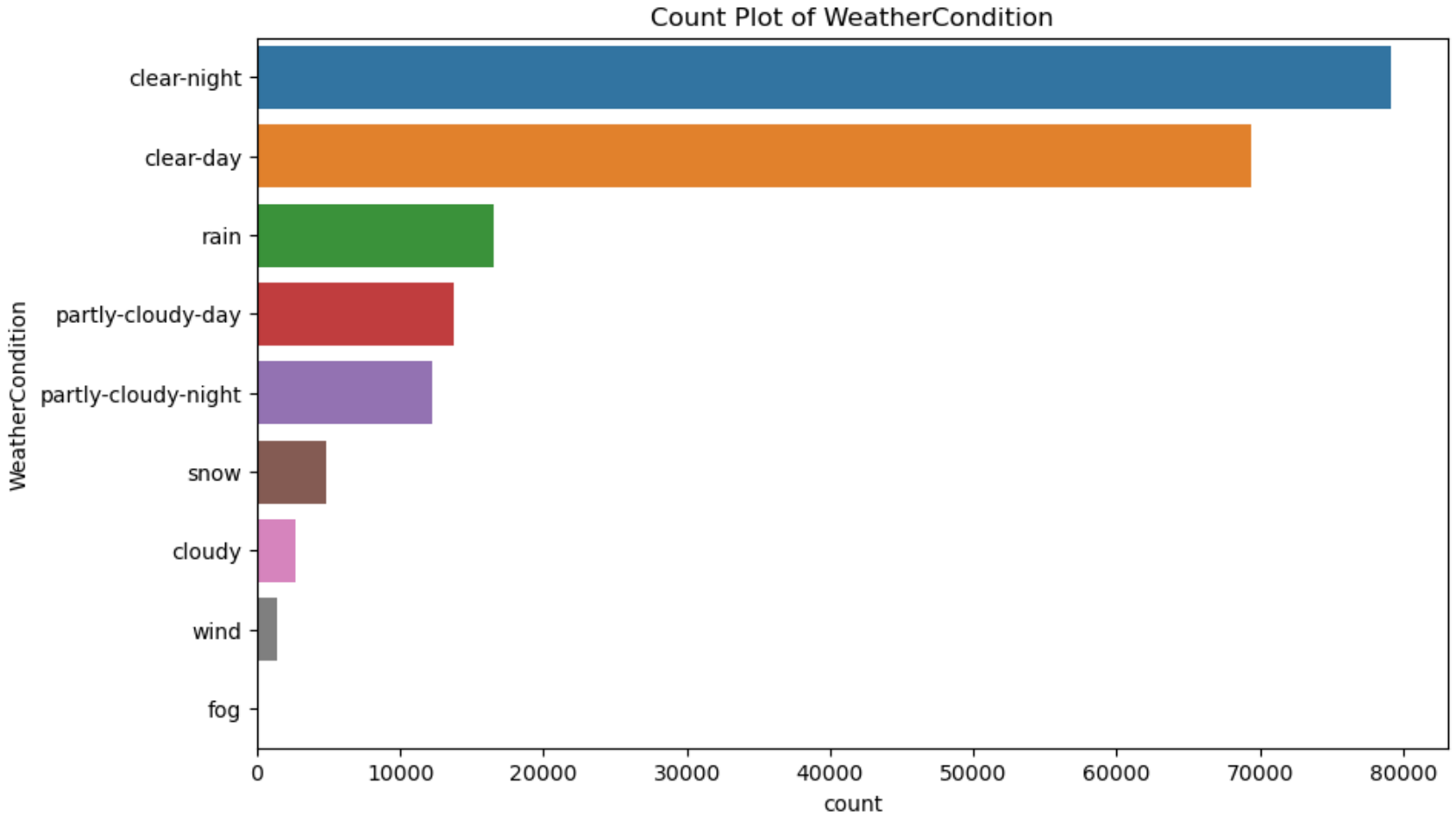
Out[31]:
```
array(['clear-night', 'partly-cloudy-night', 'clear-day', 'cloudy',
       'partly-cloudy-day', 'rain', 'snow', 'wind', 'fog'], dtype=object)
```

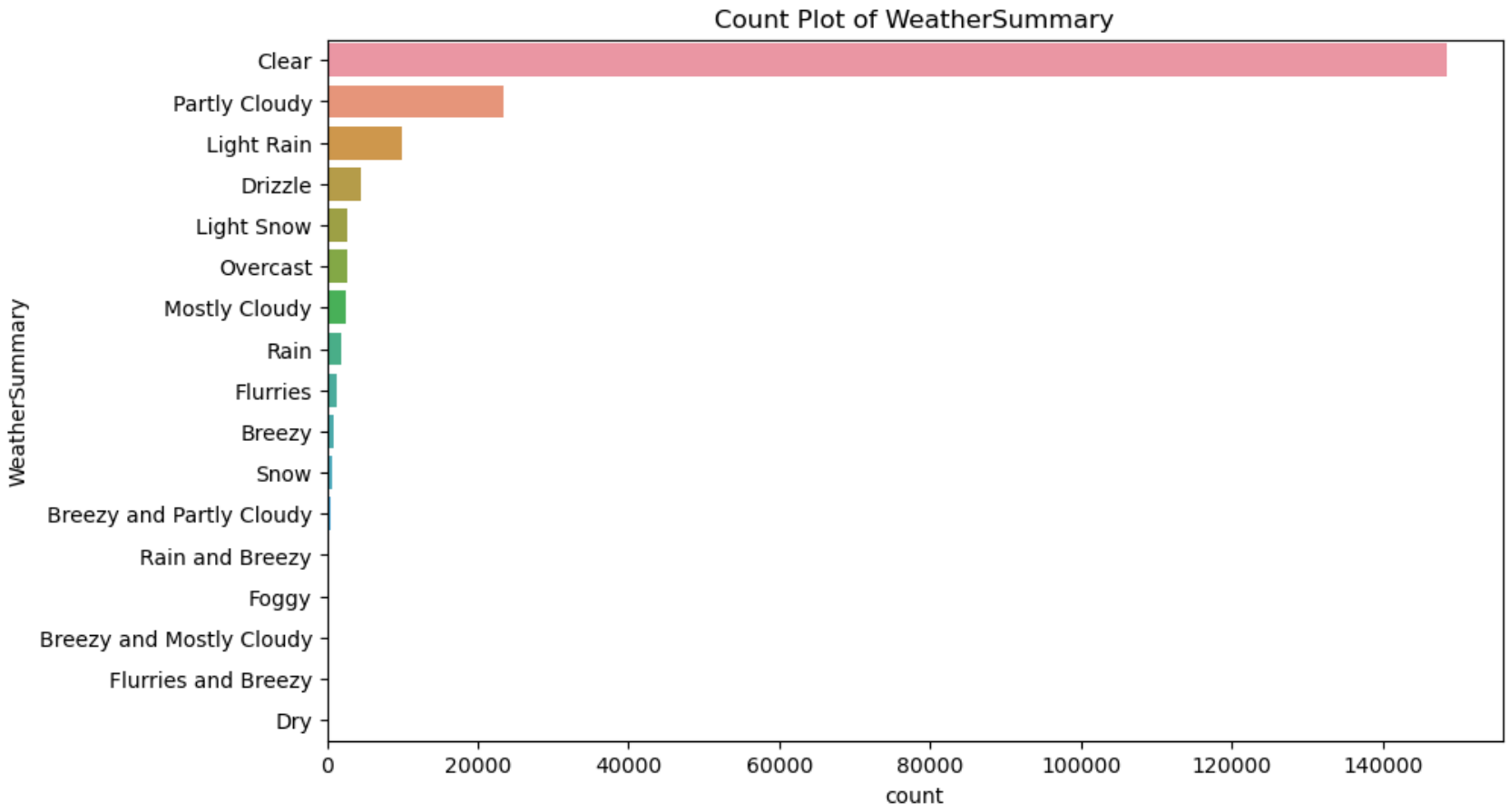
In [32]:
```
energy.WeatherSummary.unique()
```

Out[32]:
```
array(['Clear', 'Mostly Cloudy', 'Overcast', 'Partly Cloudy', 'Drizzle',
       'Light Rain', 'Rain', 'Light Snow', 'Flurries', 'Breezy', 'Snow',
       'Rain and Breezy', 'Foggy', 'Breezy and Mostly Cloudy',
       'Breezy and Partly Cloudy', 'Flurries and Breezy', 'Dry'],
      dtype=object)
```

In [33]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Identify categorical columns
categorical_columns = energy.select_dtypes(include=['object']).columns

print("\nCategorical Features:")
print(categorical_columns)

# Count plots for categorical columns
for col in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(y=col, data=energy, order=energy[col].value_counts().index)
    plt.title(f'Count Plot of {col}')
    plt.show()
```

```
Categorical Features:
Index(['WeatherCondition', 'WeatherSummary'], dtype='object')
```


Count Plot of WeatherCondition
clear-night
clear-day
rain
partly-cloudy-day
partly-cloudy-night
snow
cloudy
wind
fog
WeatherCondition
0
10000
20000
30000
40000
50000
60000
70000
80000
count



In [ ]:

In [34]:
```
#Here we will use LabelEncoder
```

In [35]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [36]:
```
energy[energy.select_dtypes(include = 'object').columns] = energy[energy.select_dtypes(include = 'object').columns]
```

In [37]:
```
energy.select_dtypes(include='object').columns
```

Out[37]:
```
Index([], dtype='object')
```

In [38]:
```
#no object now. Now we are ready to build the model
```

In [ ]:
```

```

--- Model Buiding ---

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
#finding Target Var column index
energy.columns
```

Out[40]:
```
Index(['Time', 'Gen', 'DishWasher', 'HVAC', 'HomeOffice', 'Fridge',
       'WineCellar', 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
       'LivingRoom', 'Solar', 'Temperature', 'WeatherCondition',
       'WeatherSummary', 'AapparentTemperature', 'Pressure', 'WindSpeed',
       'PrecipIntensity', 'PrecipProbability', 'DewPoint',
       'TotalEnergyConsumptionOfTheHouse'],
      dtype='object')
```

In [41]:
```
#Here our Target Variable is at Last Position
```

In [42]:
```
train, test = train_test_split(energy,test_size=.2)

train_x = train.iloc[:, 0:-1]
train_y = train.iloc[:,-1]

test_x = test.iloc[:, 0:-1]
test_y = test.iloc[:,-1]
```

In [ ]:
```

```

--- Linear Regression ---

In [43]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

lr.fit(train_x, train_y)

#predict the target values of Train Data
pred_lr_train = lr.predict(train_x)
#predict the target values of Test Data
pred_lr_test = lr.predict(test_x)
```

In [45]:
```
#EVALUATING THE MODEL USING

#Importing libraries
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Calculate evaluation metrics
lr_mse_train = mean_squared_error(train_y, pred_lr_train)
lr_mse_test  = mean_squared_error(test_y, pred_lr_test)

lr_r2_train = r2_score(train_y, pred_lr_train)
lr_r2_test = r2_score(test_y, pred_lr_test)

# Calculating RMSE
lr_rmse = np.sqrt(lr_mse_test)

# Calculating MAPE
lr_mape = np.mean(np.abs((test_y - pred_lr_test) / test_y)) * 100



# Printing evaluation metrics
print("LinearRegression MSE Train = ", lr_mse_train)
print("LinearRegression MSE Test  = ", lr_mse_test)
print()
print("LinearRegression R²  Train = ", lr_r2_train)
print("LinearRegression R²  Test  = ", lr_r2_test)
print()
print("LinearRegression RMSE      = ", lr_rmse)
print()
print("LinearRegression MAPE      = ", lr_mape)
```

```
LinearRegression MSE Train =  0.13723392228419265
LinearRegression MSE Test  =  0.1361370799850908

LinearRegression R²  Train =  0.6949864121312721
LinearRegression R²  Test  =  0.6981712959130825

LinearRegression RMSE      =  0.3689675866320656

LinearRegression MAPE      =  85.2948858444813
```

In [ ]:

In [49]:
```
lr.coef_      #m(slope)
```

Out[49]:
```
array([-8.56663504e-07, -5.28610401e-01,  1.00508765e+00,  1.30666608e+00,
        9.59088047e-01,  1.41977800e+00,  9.60066811e-01,  6.47339281e-01,
        1.54971291e+00,  1.12261358e+00,  1.05497966e+00,  1.02846468e+00,
        1.79044213e+00, -5.28610401e-01, -1.52860959e-03, -2.19003354e-03,
        1.46203739e-04,  3.26361841e-04, -2.27590352e-03,  1.87845443e-03,
        1.41285887e+00, -9.30308217e-02, -4.36711232e-04])
```

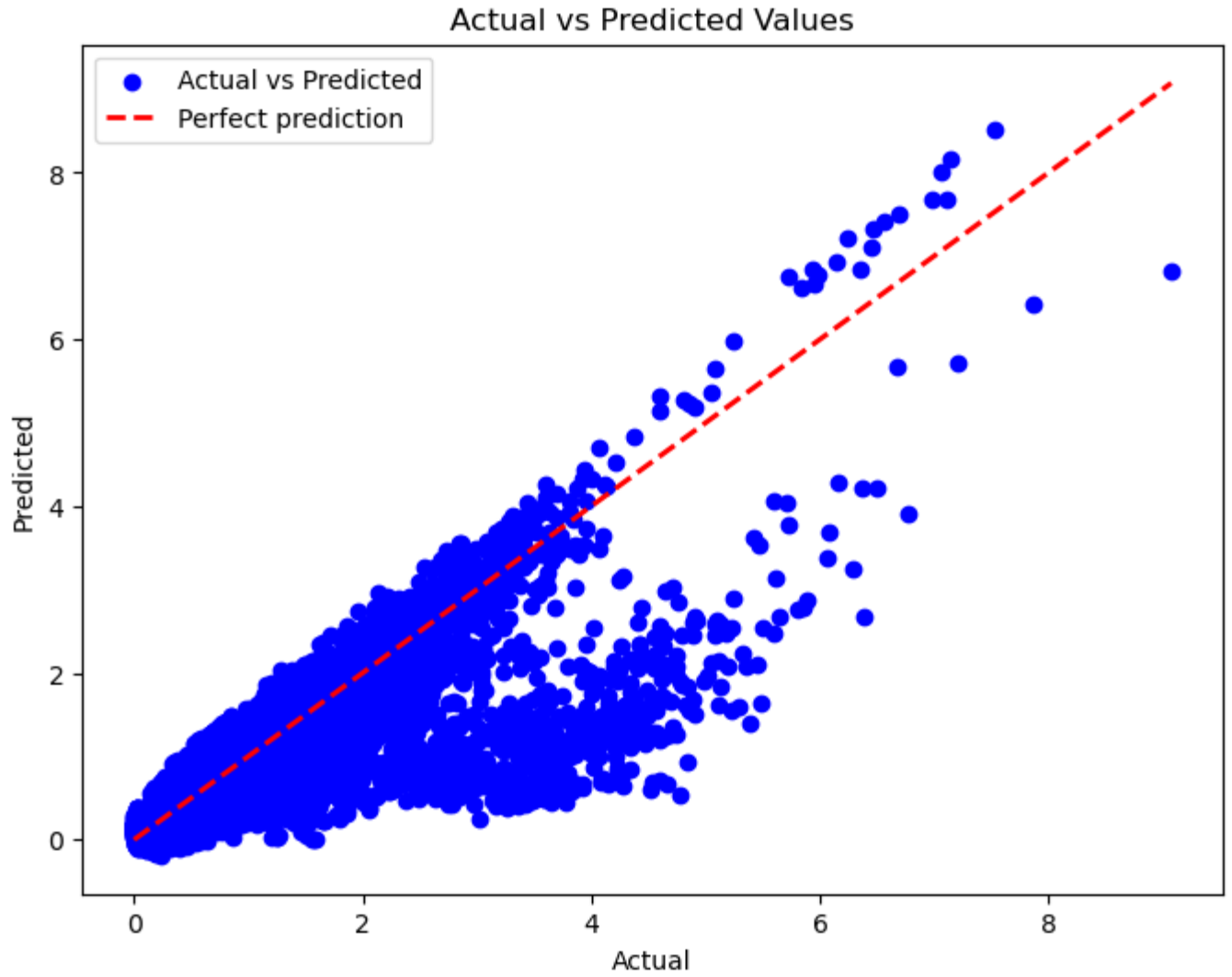
In [51]:
```
lr.intercept_         #intercept(only 1)bo
```

Out[51]:
```
1246.3034681425675
```

In [ ]:
```
'''
In linear regression, the intercept represents the value of the dependent variable (target variable)
when all independent variables (features) are equal to zero. It's the value where the regression line intersects the

The value 1246.3034681425675 means that when all features are zero,
the predicted value of the target variable is approximately 1228.62
'''
```

In [94]:
```
import matplotlib.pyplot as plt

# Plotting actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(test_y, pred_lr_test, color='blue', label='Actual vs Predicted')
plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'k--', lw=2, color='red', label='Perfect predict
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```

```
C:\Users\Aabshaar\AppData\Local\Temp\ipykernel_11120\561969293.py:6: UserWarning: color is redundantly defined by th
e 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'k--', lw=2, color='red', label='Perfect pred
iction')
```



--- outlier detection ---

In [55]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy import stats
import numpy as np



# Example: Create new feature by adding energy consumption of appliances
energy['total_appliances_consumption'] = energy[['DishWasher', 'HVAC', 'HomeOffice', 'Fridge', 'WineCellar',
                                                 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
                                                 'LivingRoom', 'Solar']].sum(axis=1)

# Example: Extract date-time features
energy['Time'] = pd.to_datetime(energy['Time'])  # Convert 'Time' column to datetime-like object
energy['month'] = energy['Time'].dt.month  # Extract month from a date-time column
energy['day_of_week'] = energy['Time'].dt.dayofweek  # Extract day of the week from a date-time column


# Perform outlier detection and handling
z_scores = stats.zscore(energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1))  # Calculate Z-scores for
threshold = 3  # Define a threshold for outlier detection
outliers = (abs(z_scores) > threshold).any(axis=1)  # Identify outliers
energy = energy[~outliers]  # Remove outliers from the dataset

# Split the dataset into features and target variable
X = energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1)  # Features
y = energy['TotalEnergyConsumptionOfTheHouse']  # Target variable

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

# Example: Model Building
# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Example: Model Evaluation
# Predict on the testing set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
lr_new_mse = mean_squared_error(y_test, y_pred)
```

```
lr_new_rmse = np.sqrt(lr_new_mse)
lr_new_mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print("MSE  = ", lr_new_mse)
print("RMSE = ", lr_new_rmse)
print("MAPE = ", lr_new_mape)
```

```
MSE  =  0.023292151346899808
RMSE =  0.15261766394130075
MAPE =  49.45615350956252
```

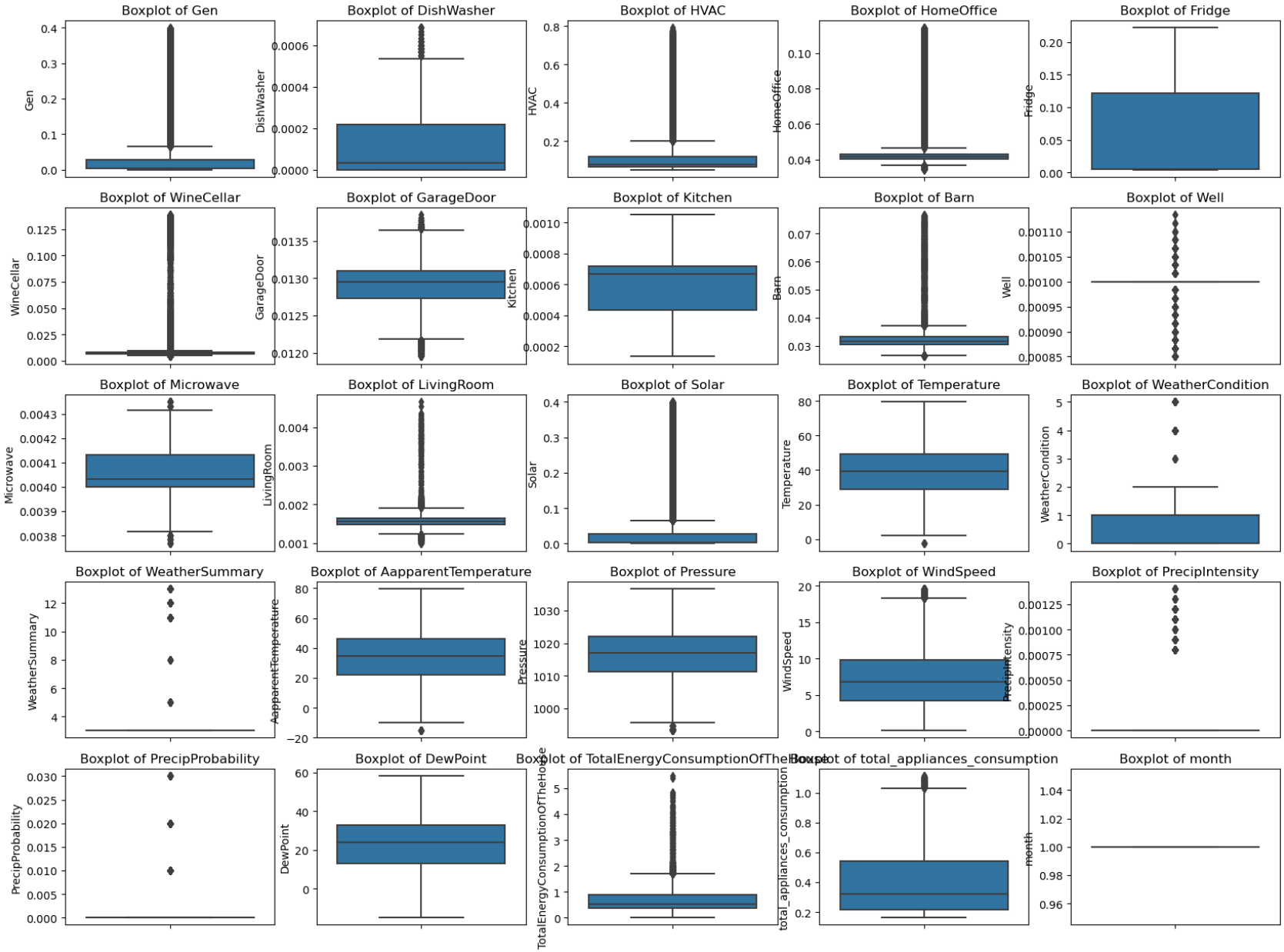
In [56]:
```
lr_new_mse *100
```

Out[56]:
```
2.329215134689981
```

In [60]:

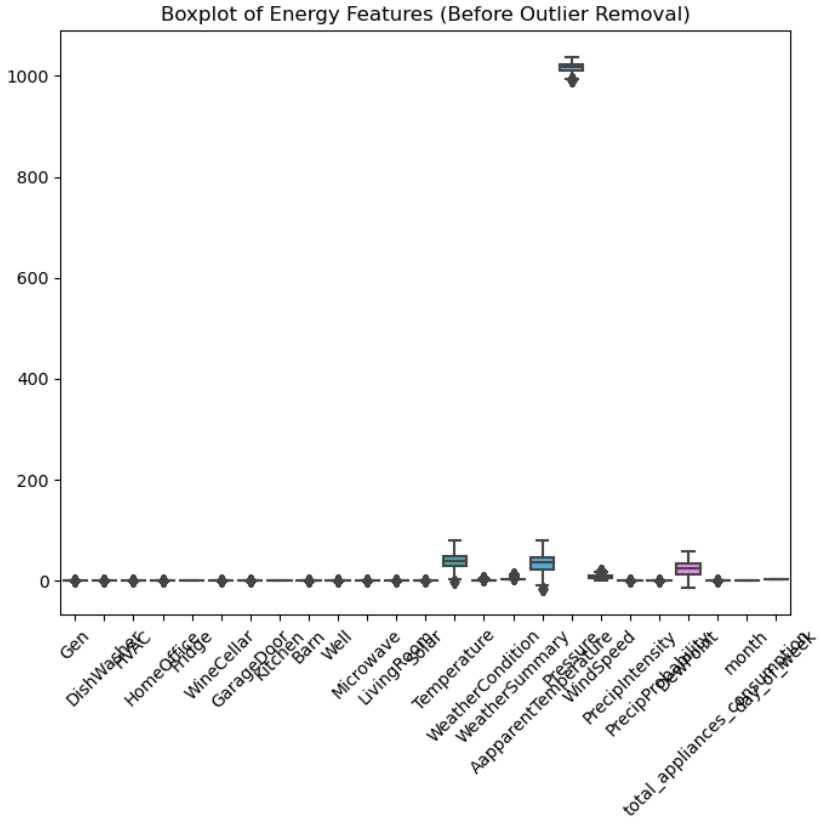
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Example: Assuming 'energy' is your DataFrame
# Create new feature by adding energy consumption of appliances
energy['total_appliances_consumption'] = energy[['DishWasher', 'HVAC', 'HomeOffice', 'Fridge', 'WineCellar',
                                                 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
                                                 'LivingRoom', 'Solar']].sum(axis=1)

# Example: Extract date-time features
energy['Time'] = pd.to_datetime(energy['Time'])  # Convert 'Time' column to datetime-like object
energy['month'] = energy['Time'].dt.month  # Extract month from a date-time column
energy['day_of_week'] = energy['Time'].dt.dayofweek  # Extract day of the week from a date-time column

# Perform outlier detection and handling
z_scores = stats.zscore(energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1))
threshold = 3
outliers = (abs(z_scores) > threshold).any(axis=1)
clean_energy = energy[~outliers]

# List of numerical columns for plotting
numerical_columns = energy.select_dtypes(include=[np.number]).columns

# Plot boxplots after outlier removal
plt.figure(figsize=(20, 15))

for i, col in enumerate(numerical_columns, 1):
    plt.subplot(5, 5, i)
    sns.boxplot(y=clean_energy[col])  # Use clean_energy after outlier removal
    plt.title(f'Boxplot of {col}')

plt.tight_layout()
plt.show()
```

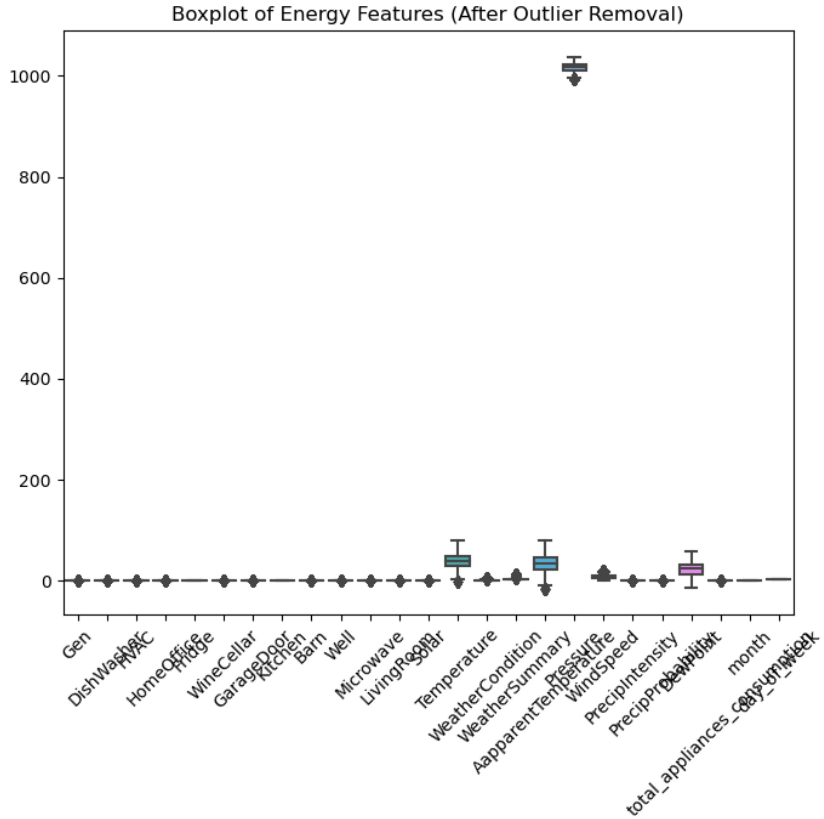
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13844\2853695223.py in <module>
     32
     33 for i, col in enumerate(numerical_columns, 1):
---> 34     plt.subplot(5, 5, i)
     35     sns.boxplot(y=clean_energy[col])  # Use clean_energy after outlier removal
     36     plt.title(f'Boxplot of {col}')

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in subplot(*args, **kwargs)
   1287
   1288     # First, search for an existing subplot with a matching spec.
-> 1289     key = SubplotSpec._from_subplot_args(fig, args)
   1290
   1291     for ax in fig.axes:

~\anaconda3\lib\site-packages\matplotlib\gridspec.py in _from_subplot_args(figure, args)
    606         else:
    607             if not isinstance(num, Integral) or num < 1 or num > rows*cols:
--> 608                 raise ValueError(
    609                     f"num must be 1 <= num <= {rows*cols}, not {num!r}")
    610             i = j = num

ValueError: num must be 1 <= num <= 25, not 26
```


Boxplot of Gen
Gen
Boxplot of DishWasher
DishWasher
Boxplot of HVAC
HVAC
Boxplot of HomeOffice
HomeOffice
Boxplot of Fridge
Fridge
Boxplot of WineCellar
WineCellar
Boxplot of GarageDoor
GarageDoor
Boxplot of Kitchen
Kitchen
Boxplot of Barn
Barn
Boxplot of Well
Well
Boxplot of Microwave
Microwave
Boxplot of LivingRoom
LivingRoom
Boxplot of Solar
Solar
Boxplot of Temperature
Temperature
Boxplot of WeatherCondition
WeatherCondition
Boxplot of WeatherSummary
WeatherSummary
Boxplot of AapparentTemperature
AapparentTemperature
Boxplot of Pressure
Pressure
Boxplot of WindSpeed
WindSpeed
Boxplot of PrecipIntensity
PrecipIntensity
Boxplot of PrecipProbability
PrecipProbability
Boxplot of DewPoint
DewPoint
Boxplot of TotalEnergyConsumptionOfTheHouse
TotalEnergyConsumptionOfTheHouse
Boxplot of total_appliances_consumption
total_appliances_consumption
Boxplot of month
month

In [61]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Example: Assuming 'energy' is your DataFrame
# Create new feature by adding energy consumption of appliances
energy['total_appliances_consumption'] = energy[['DishWasher', 'HVAC', 'HomeOffice', 'Fridge', 'WineCellar',
                                                 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
                                                 'LivingRoom', 'Solar']].sum(axis=1)

# Example: Extract date-time features
energy['Time'] = pd.to_datetime(energy['Time'])  # Convert 'Time' column to datetime-like object
energy['month'] = energy['Time'].dt.month  # Extract month from a date-time column
energy['day_of_week'] = energy['Time'].dt.dayofweek  # Extract day of the week from a date-time column

# Perform outlier detection and handling
z_scores = stats.zscore(energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1))
threshold = 3
outliers = (abs(z_scores) > threshold).any(axis=1)
clean_energy = energy[~outliers]

# Plot boxplots before and after outlier removal
plt.figure(figsize=(14, 7))

# Before outlier removal
plt.subplot(1, 2, 1)
sns.boxplot(data=energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1))
plt.title('Boxplot of Energy Features (Before Outlier Removal)')
plt.xticks(rotation=45)

# After outlier removal
plt.subplot(1, 2, 2)
sns.boxplot(data=clean_energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1))
plt.title('Boxplot of Energy Features (After Outlier Removal)')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

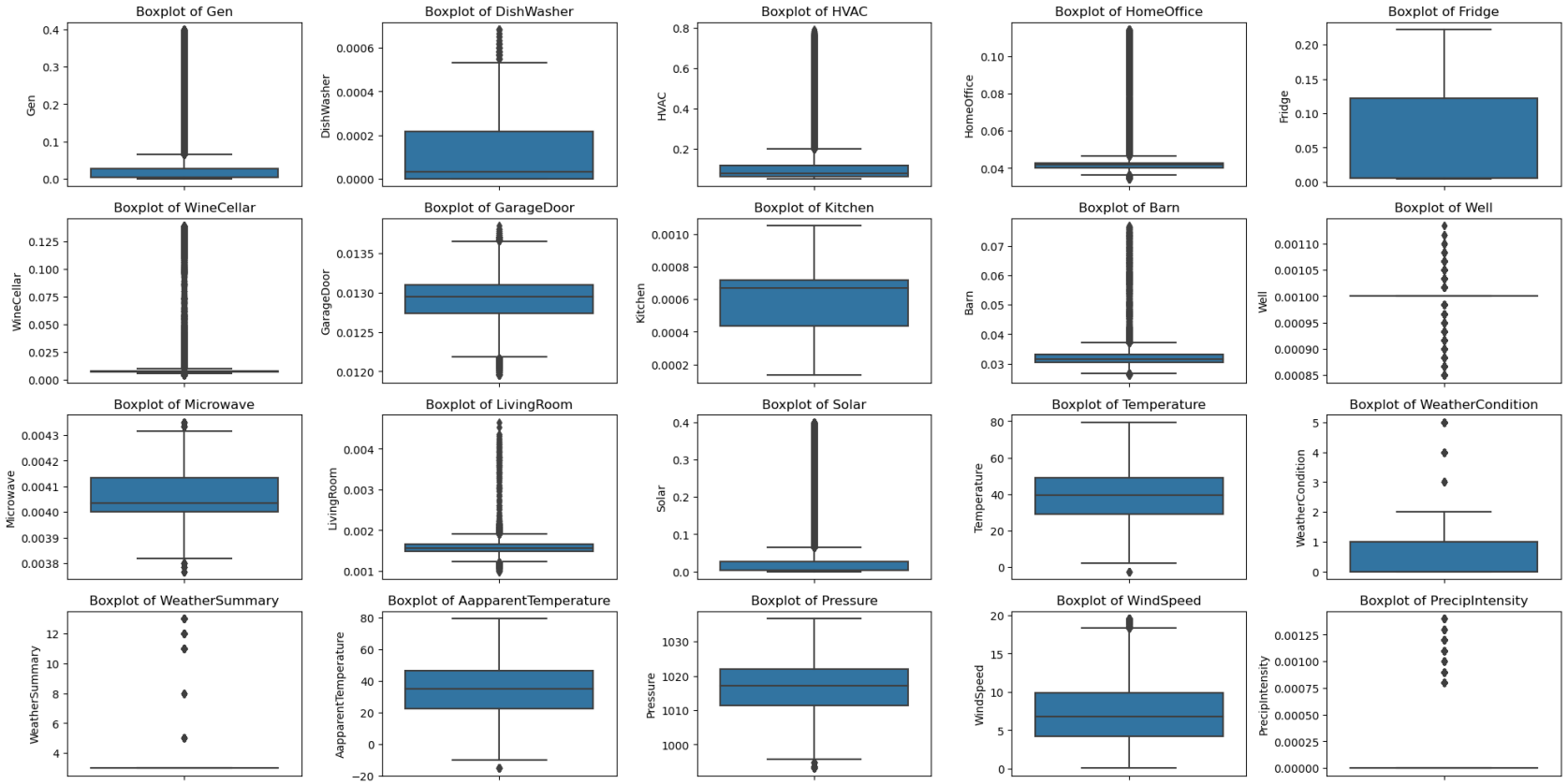

Boxplot of Energy Features (Before Outlier Removal)
1000
800
600
400
200
0
Gen
DishWasher
HVAC
HomeOffice
Fridge
WineCellar
GarageDoor
Kitchen
Barn
Well
Microwave
LivingRoom
Solar
Temperature
WeatherCondition
WeatherSummary
Pressure
AapparentTemperature
WindSpeed
PrecipIntensity
PrecipProbability
DewPoint
month
total_appliances_consumption
day_of_week


Boxplot of Energy Features (After Outlier Removal)
1000
800
600
400
200
0
Gen
DishWasher
HVAC
HomeOffice
Fridge
WineCellar
GarageDoor
Kitchen
Barn
Well
Microwave
LivingRoom
Solar
Temperature
WeatherCondition
WeatherSummary
Pressure
AapparentTemperature
WindSpeed
PrecipIntensity
PrecipProbability
DewPoint
month
total_appliances_consumption
day_of_week

In [62]:

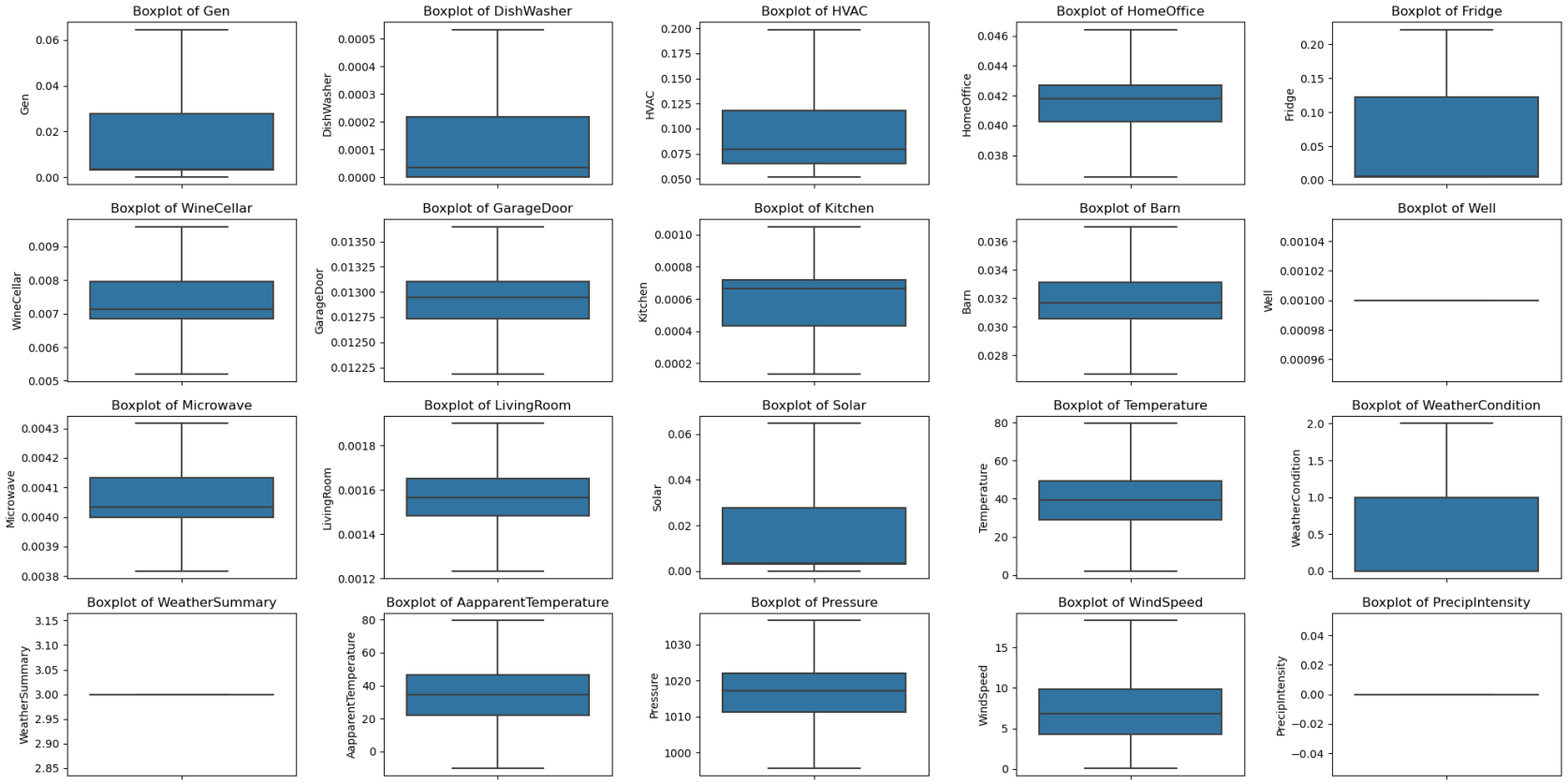
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Example: Assuming 'energy' is your DataFrame
# Create new feature by adding energy consumption of appliances
energy['total_appliances_consumption'] = energy[['DishWasher', 'HVAC', 'HomeOffice', 'Fridge', 'WineCellar',
                                                 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
                                                 'LivingRoom', 'Solar']].sum(axis=1)

# Example: Extract date-time features
energy['Time'] = pd.to_datetime(energy['Time'])  # Convert 'Time' column to datetime-like object
energy['month'] = energy['Time'].dt.month  # Extract month from a date-time column
energy['day_of_week'] = energy['Time'].dt.dayofweek  # Extract day of the week from a date-time column

# Perform outlier detection and handling
z_scores = stats.zscore(energy.drop(['TotalEnergyConsumptionOfTheHouse', 'Time'], axis=1))
threshold = 3
outliers = (abs(z_scores) > threshold).any(axis=1)
clean_energy = energy[~outliers]

# List of numerical columns for plotting
numerical_columns = clean_energy.select_dtypes(include=[np.number]).columns

# Determine number of rows and columns for subplots dynamically
num_cols = len(numerical_columns)
num_rows = (num_cols // 5) + 1  # Adjust rows based on number of columns

# Plot boxplots after outlier removal for a subset of columns
plt.figure(figsize=(20, 15))

# Limiting to first 20 columns for demonstration (adjust as needed)
for i, col in enumerate(numerical_columns[:20], 1):
    plt.subplot(num_rows, 5, i)
    sns.boxplot(y=clean_energy[col])  # Use clean_energy after outlier removal
    plt.title(f'Boxplot of {col}')

plt.tight_layout()
plt.show()
```

```
plt.show()
```


Boxplot of Gen
Gen
Boxplot of DishWasher
DishWasher
Boxplot of HVAC
HVAC
Boxplot of HomeOffice
HomeOffice
Boxplot of Fridge
Fridge
Boxplot of WineCellar
WineCellar
Boxplot of GarageDoor
GarageDoor
Boxplot of Kitchen
Kitchen
Boxplot of Barn
Barn
Boxplot of Well
Well
Boxplot of Microwave
Microwave
Boxplot of LivingRoom
LivingRoom
Boxplot of Solar
Solar
Boxplot of Temperature
Temperature
Boxplot of WeatherCondition
WeatherCondition
Boxplot of WeatherSummary
WeatherSummary
Boxplot of AapparentTemperature
AapparentTemperature
Boxplot of Pressure
Pressure
Boxplot of WindSpeed
WindSpeed
Boxplot of PrecipIntensity
PrecipIntensity

In [63]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'energy' is your DataFrame and you've performed outlier removal

# List of numerical columns for plotting
numerical_columns = clean_energy.select_dtypes(include=[np.number]).columns

# Determine number of rows and columns for subplots dynamically
num_cols = len(numerical_columns)
num_rows = (num_cols // 5) + 1  # Adjust rows based on number of columns

# Plot boxplots after outlier removal for a subset of columns
plt.figure(figsize=(20, 15))

# Limiting to first 20 columns for demonstration (adjust as needed)
for i, col in enumerate(numerical_columns[:20], 1):
    plt.subplot(num_rows, 5, i)
    sns.boxplot(y=clean_energy[col], showfliers=False)  # Use clean_energy after outlier removal, hide outliers
    plt.title(f'Boxplot of {col}')

plt.tight_layout()
plt.show()
```



--- Comparing Both The Values ---

In [56]:
```
 # Before values
print("Before MSE Value ="  , lr_mse_test)
print("Before RMSE Value =" , lr_rmse)
print("Before MAPE Value =" , lr_mape)

print("-------------------------------------------------")

 # After values
print("After MSE Value  ="  , lr_new_mse)
print("After RMSE Value =" , lr_new_rmse)
print("After MAPE Value =" , lr_new_mape)
```

```
Before MSE Value = 0.13697552882087619
Before RMSE Value = 0.37010205190038625
Before MAPE Value = 78.76147134725042
--------------------------------------------------
After MSE Value  = 0.023292151346899808
After RMSE Value = 0.15261766394130075
After MAPE Value = 49.45615350956252
```

Before values

- "MSE" : 0.13
- "RMSE" : 0.37
- "MAPE" : 78

  # After values

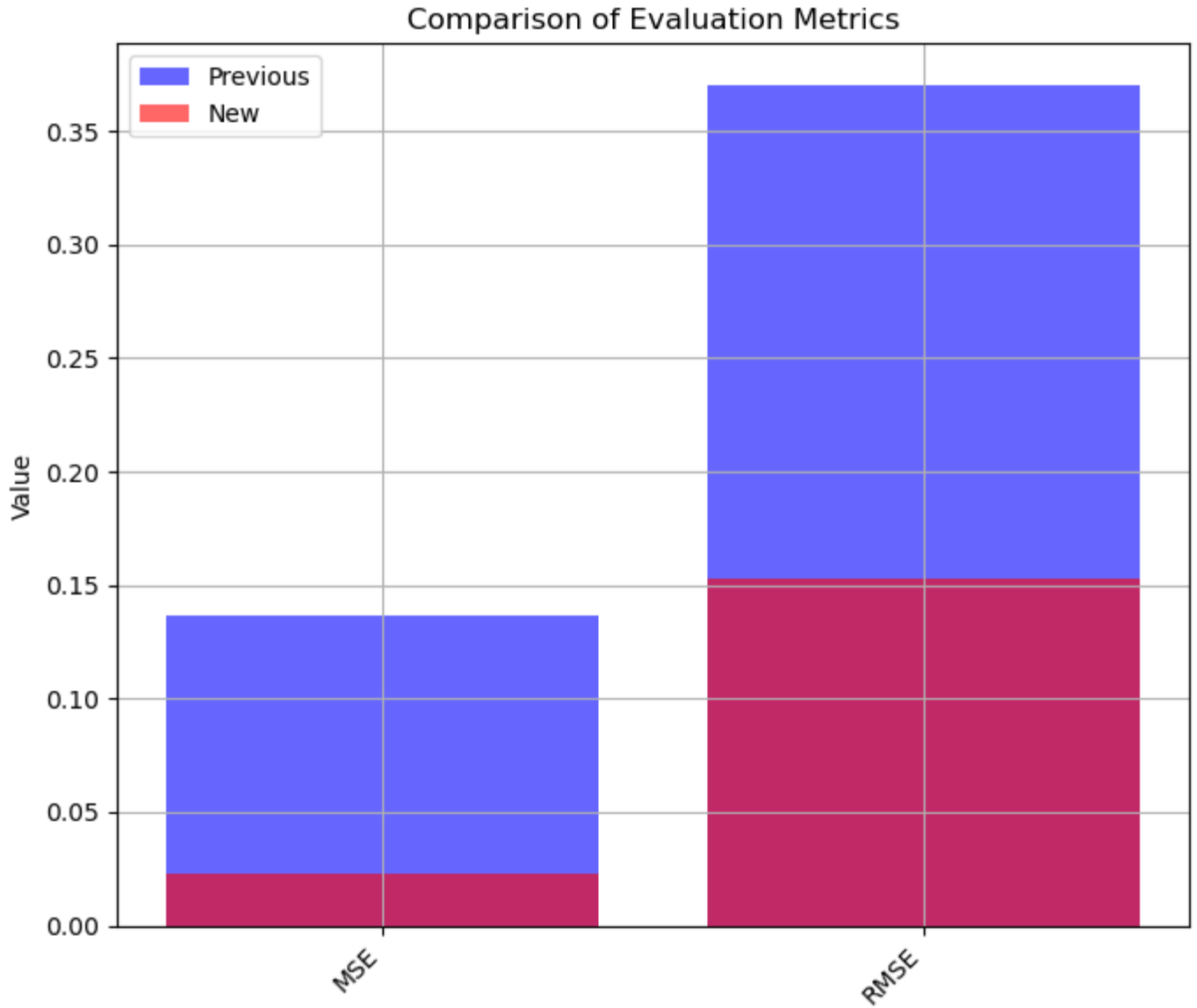
- "MSE" : 0.02
- "RMSE" : 0.15
- "MAPE" : 49

In [ ]:

--- Let's plot them ---

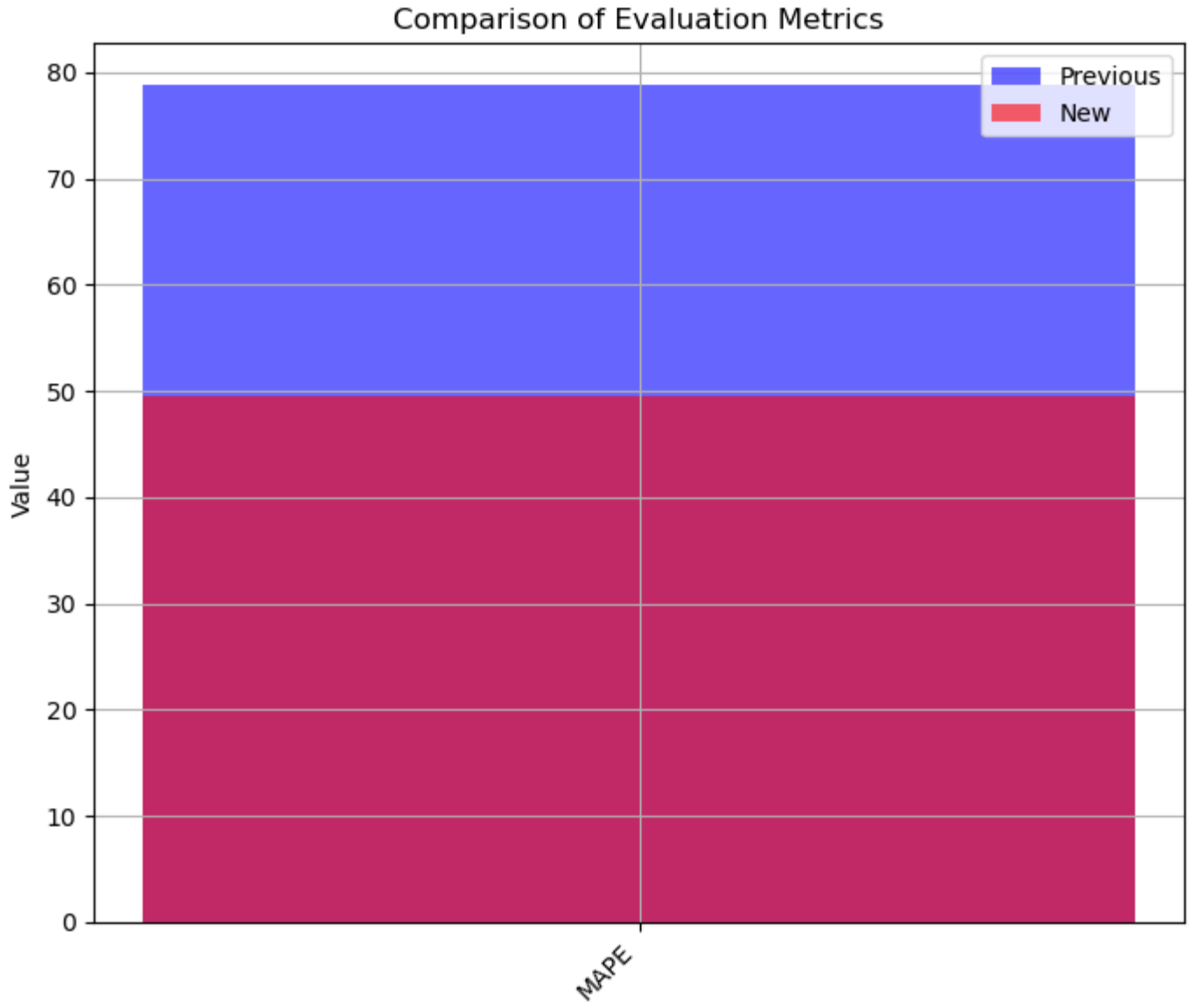
In [57]:

```python
import matplotlib.pyplot as plt

# Previous and new evaluation metrics
prev_metrics = {
    "MSE": lr_mse_train,
    "RMSE": lr_rmse
}

new_metrics = {
    "MSE": lr_new_mse,
    "RMSE": lr_new_rmse
}

# Plotting
plt.figure(figsize=(7, 6))

# Previous metrics
plt.bar(prev_metrics.keys(), prev_metrics.values(), color='b', alpha=0.6, label='Previous')

# New metrics
plt.bar(new_metrics.keys(), new_metrics.values(), color='r', alpha=0.6, label='New')

# Adding labels and title
plt.ylabel('Value')
plt.title('Comparison of Evaluation Metrics')
plt.legend()

plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.grid()
plt.show()
```


Comparison of Evaluation Metrics
Previous
New
Value
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
MSE
RMSE

In [58]:

```python
import matplotlib.pyplot as plt

# Previous and new evaluation metrics
prev_metrics = {"MAPE": lr_mape}
new_metrics = {"MAPE": lr_new_mape}

# Plotting
plt.figure(figsize=(7, 6))

# Previous metrics
plt.bar(prev_metrics.keys(), prev_metrics.values(), color='b', alpha=0.6, label='Previous')

# New metrics
plt.bar(new_metrics.keys(), new_metrics.values(), color='r', alpha=0.6, label='New')

# Adding labels and title
plt.ylabel('Value')
plt.title('Comparison of Evaluation Metrics')
plt.legend()

plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.grid()
plt.show()
```


Comparison of Evaluation Metrics
Previous
New
Value
80
70
60
50
40
30
20
10
0
MAPE

In [ ]:

Regularization:

- Regularization techniques like L1 (Lasso) and L2 (Ridge) regularization to reduce overfitting.

In [59]:
```
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
import numpy as np

# Ridge
ridge = Ridge()
ridge.fit(train_x, train_y)

# Predicting
pred_ridge = ridge.predict(test_x)

# Evaluate the model using metrics
ridge_mse = mean_squared_error(test_y, pred_ridge)
#ridge_mape = np.mean(np.abs((test_y - pred_ridge) / test_y)) * 100

# Printing
print("Ridge MSE = ", ridge_mse)
#print("Ridge MAPE = ", ridge_mape)

# ---------------------------------------------------------------------

# Lasso
lasso = Lasso()
lasso.fit(train_x, train_y)

# Predicting
pred_lasso = lasso.predict(test_x)

# Evaluate the model using metrics
lasso_mse = mean_squared_error(test_y, pred_lasso)
#lasso_mape = np.mean(np.abs((test_y - pred_lasso) / test_y)) * 100

# Printing
print("Lasso MSE = ", lasso_mse)
#print("Lasso MAPE = ", lasso_mape)
```

```
Ridge MSE =  0.13698183926926977
Lasso MSE =  0.4168338319309967
```

In [ ]:

cross validation

- Use techniques like k-fold cross-validation to assess the model's performance on multiple subsets of the data. This helps to ensure that the model's performance is consistent across different partitions of the data and reduces the risk of overfitting to a specific training set.

In [60]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
import numpy as np

# Assuming you have your feature matrix X and target vector y

# Linear Regression cross-validation
lr = LinearRegression()
lr_scores = cross_val_score(lr, X, y, cv=5, scoring='neg_mean_squared_error')
lr_mse_cv = -np.mean(lr_scores)  # Taking the mean of negative MSE scores

# Printing the cross-validated MSE score
print("Linear Regression Cross-Validated MSE:", lr_mse_cv)
```

```
Linear Regression Cross-Validated MSE: 0.028372465707980687
```

In [ ]:

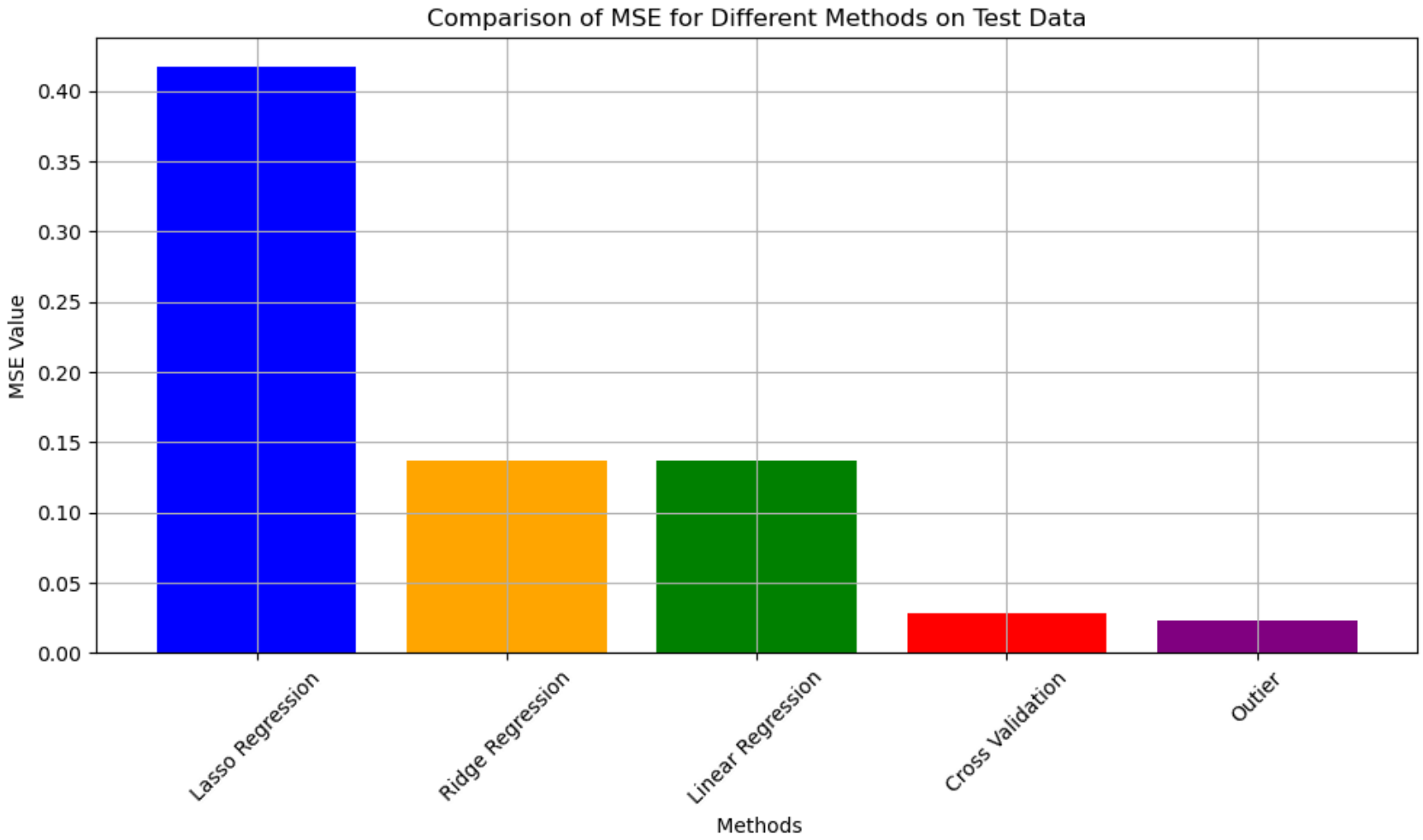
--- Comparing All MSE Together ---

In [62]:
```python
print("Linear Regression MSE = ", lr_mse_test)
print("Outier MSE            = ", lr_new_mse)
print("Ridge Regression MSE  = ", ridge_mse)
print("Lasso Regression MSE  = ", lasso_mse)
print("Cross Validation MSE  = ", lr_mse_cv)
```

```
Linear Regression MSE =  0.13697552882087619
Outier MSE            =  0.023292151346899808
Ridge Regression MSE  =  0.13698183926926977
Lasso Regression MSE  =  0.4168338319309967
Cross Validation MSE  =  0.028372465707980687
```

In [71]:

```python
# MSE values for each method on test data
methods    = ['Linear Regression', 'Outier', 'Ridge Regression', 'Lasso Regression', 'Cross Validation']
mse_values = [lr_mse_test, lr_new_mse, ridge_mse, lasso_mse, lr_mse_cv]

# Creating a DataFrame
df = pd.DataFrame()
df['Methods'] = methods
df['MSE Value'] = mse_values

# Sorting the DataFrame by 'MSE (Test Data)' column in descending order
df_sorted = df.sort_values(by='MSE Value', ascending=False)

# Creating a bar plot
plt.figure(figsize=(10, 6))
plt.bar(df_sorted['Methods'], df_sorted['MSE Value'], color=['blue', 'orange', 'green', 'red', 'purple'])

# Add labels and title
plt.xlabel(' Methods')
plt.ylabel('MSE Value')
plt.title('Comparison of MSE for Different Methods on Test Data')
plt.xticks(rotation=45)
plt.grid()

# Display the plot
plt.tight_layout()
plt.show()
```


Comparison of MSE for Different Methods on Test Data
MSE Value
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Lasso Regression
Ridge Regression
Linear Regression
Cross Validation
Outier
Methods

Observations:

1. **Outlier Detection**: Performs the best, showing the lowest MSE, highlighting the importance of handling outliers for accurate predictions.
2. **Cross-Validation**: Comes second with a competitive MSE, emphasizing its role in preventing overfitting and ensuring model reliability.
3. **Linear Regression**: Shows decent performance, showcasing its simplicity and effectiveness as a baseline model.
4. **Ridge Regression**: Performs well, slightly behind Linear Regression, thanks to its ability to balance model complexity and performance.
5. **Lasso Regression**: Exhibits the highest MSE among the methods, indicating challenges in feature selection and potential sparsity issues.

In essence, these observations underscore the varying effectiveness of regression techniques in predicting outcomes, with each method offering unique advantages and considerations.

In [ ]:

In [ ]:

In [ ]:

--- Decision Tree ---

In [129…

```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()

dt.fit(train_x, train_y)

pred_dt = dt.predict(test_x)
```

In [130…

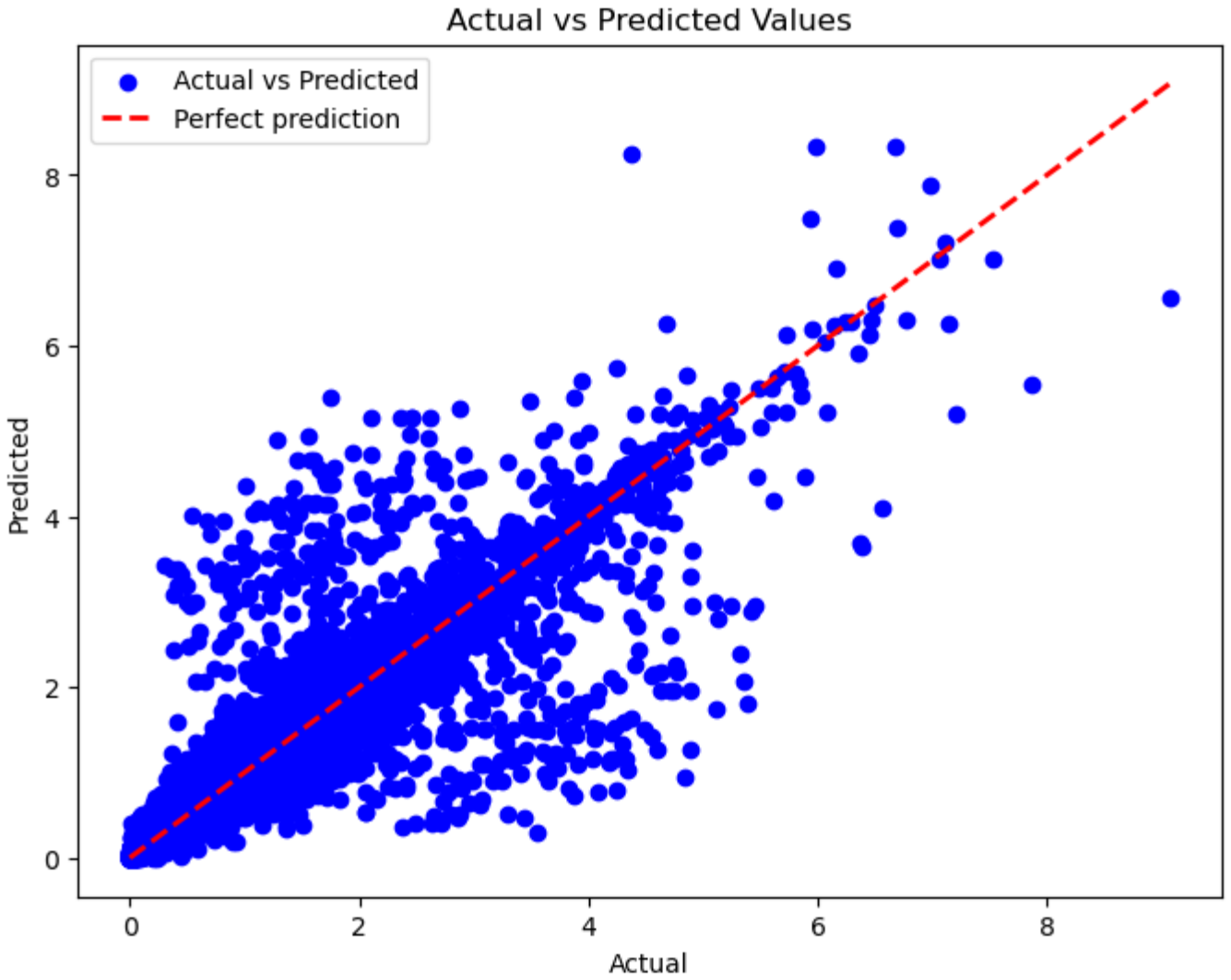
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Calculate evaluation metrics
dt_mse = mean_squared_error(test_y, pred_dt)
dt_rmse = np.sqrt(dt_mse)
dt_r2 = r2_score(test_y, pred_dt)
dt_mape = np.mean(np.abs((test_y - pred_dt) / test_y)) * 100


# Print evaluation metrics
print("MSE  =", dt_mse)
print("RMSE =", dt_rmse)
print("R²   =", dt_r2)
print("MAPE =", dt_mape)
```

```
MSE  = 0.06059623599054048
RMSE = 0.2461630272614888
R²   = 0.861648064580006
MAPE = 14.8554882217281
```

In [131…

```
import matplotlib.pyplot as plt

# Plotting actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(test_y, pred_dt, color='blue', label='Actual vs Predicted')
plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'k--', lw=2, color='red', label='Perfect predict
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```

```
C:\Users\Aabshaar\AppData\Local\Temp\ipykernel_11120\1081709399.py:6: UserWarning: color is redundantly defined by t
he 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'k--', lw=2, color='red', label='Perfect pred
iction')
```



In [ ]:

Hyper Parameter Tuning

In [41]:
```
from sklearn.model_selection import GridSearchCV #to get the best combination of hyper parameter
search_dict = { "criterion" : ["gini", "entropy"],
                "max_depth" : range(3,8),
                "min_samples_split" : range(25,50)} #created a dictionary

from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor()
grid = GridSearchCV(dt, param_grid= search_dict)
```

In [42]:
```
grid.fit(train_x, train_y) #model has been build
```

```
C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:
1250 fits failed out of a total of 1250.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
625 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 680, in _fit_and
_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 1315, in fit
    super().fit(
  File "C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 356, in fit
    criterion = CRITERIA_REG[self.criterion](self.n_outputs_, n_samples)
KeyError: 'gini'

--------------------------------------------------------------------------------
625 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 680, in _fit_and
_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 1315, in fit
    super().fit(
  File "C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 356, in fit
    criterion = CRITERIA_REG[self.criterion](self.n_outputs_, n_samples)
KeyError: 'entropy'

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Aabshaar\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:969: UserWarning: One or more of th
e test scores are non-finite: [nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
```

```
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan]
  warnings.warn(
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_24100\983380623.py in <module>
----> 1 grid.fit(train_x, train_y) #model has been build

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X, y, groups, **fit_params)
    924             refit_start_time = time.time()
    925             if y is not None:
--> 926                 self.best_estimator_.fit(X, y, **fit_params)
    927             else:
    928                 self.best_estimator_.fit(X, **fit_params)

~\anaconda3\lib\site-packages\sklearn\tree\_classes.py in fit(self, X, y, sample_weight, check_input, X_idx_sorted)
   1313         """
   1314
-> 1315         super().fit(
   1316             X,
   1317             y,

~\anaconda3\lib\site-packages\sklearn\tree\_classes.py in fit(self, X, y, sample_weight, check_input, X_idx_sorted)
    354                 )
    355             else:
--> 356                 criterion = CRITERIA_REG[self.criterion](self.n_outputs_, n_samples)
    357             # TODO: Remove in v1.2
    358             if self.criterion == "mse":

KeyError: 'gini'
```

In [ ]:

```
'''

DecisionTreeRegressor is used for regression tasks, so it doesn't support classification criteria like 'gini' or 'ent
To fix this issue, you need to use regression criteria like 'mse' (Mean Squared Error) or 'mae' (Mean Absolute Error)
for the criterion parameter in your parameter grid.

'''
```

In [55]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

# Define the parameter grid
search_dict = {
    "criterion": ["squared_error"],  # Use 'squared_error' instead of 'mse'
    "max_depth": range(3, 30),  # Start with a narrower range, adjust based on performance
    "min_samples_split": range(2, 20)  # Try values from 2 to some fraction of the total number of samples
}


# Initialize DecisionTreeRegressor
dt = DecisionTreeRegressor()

# Initialize GridSearchCV
grid = GridSearchCV(dt, param_grid=search_dict)

# Fit the model
grid.fit(train_x, train_y)

# Get the best parameters
best_params = grid.best_params_
print("Best Hyperparameters are:", best_params)
```

```
Best Hyperparameters are: {'criterion': 'squared_error', 'max_depth': 28, 'min_samples_split': 12}
```

In [ ]:
```
'''

The Best Hyperparameters Are:

Criterion: 'squared_error' (equivalent to 'mse')
Max Depth: 28
Min Samples Split: 12

'''
```

In [ ]:
```
```

Let's use them to train a final DecisionTreeRegressor model

In [132…

```python
#DecisionTreeRegressor with the best hyperparameters
best_dt_model = DecisionTreeRegressor(criterion='squared_error', max_depth=28, min_samples_split=12)

# Train the model on the entire training dataset
best_dt_model.fit(train_x, train_y)

# Make predictions on the testing dataset
predictions = best_dt_model.predict(test_x)

# Evaluate the model using mean squared error
dt_hpt_mse = mean_squared_error(test_y, predictions)
print("Mean Squared Error:", dt_hpt_mse)
```

```
Mean Squared Error: 0.057270828864317426
```

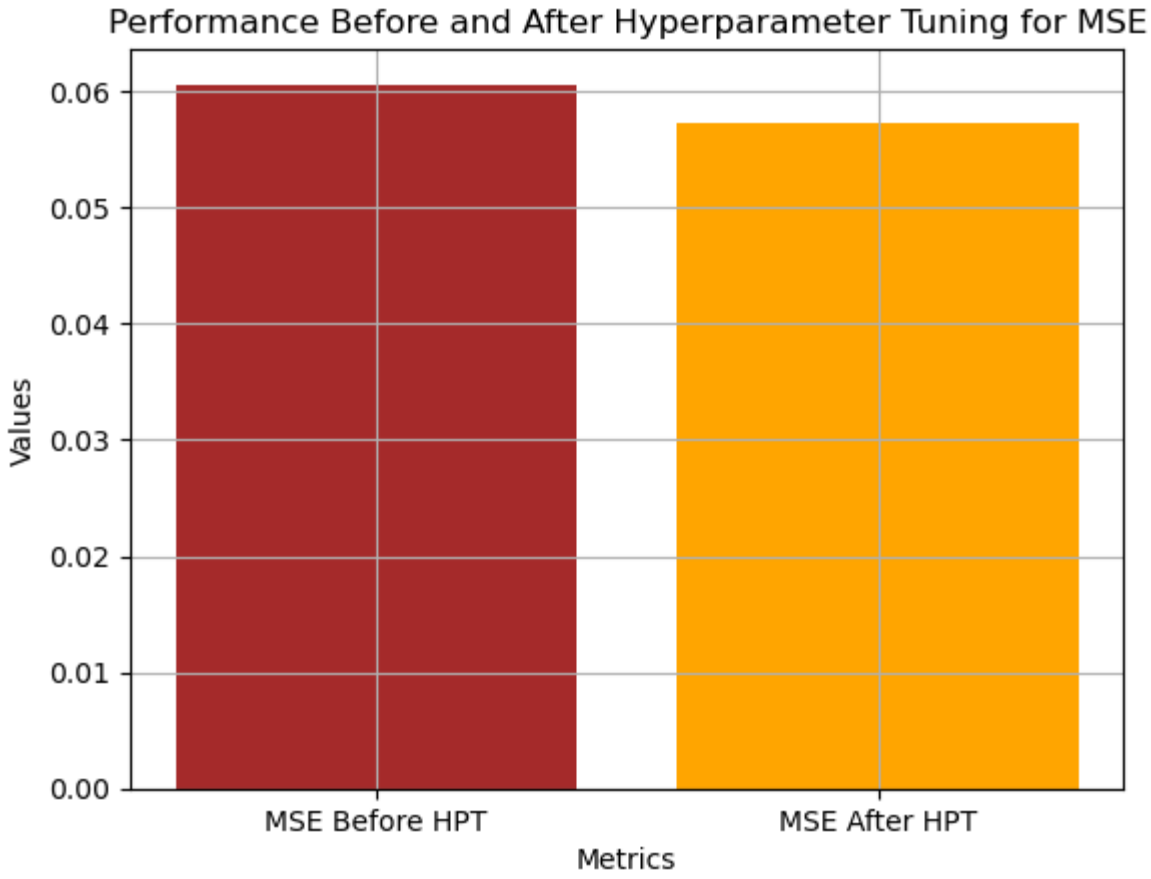
In [188…

```python
#Combining MSE for both

print("--Decision Tree--")
print()
print("MSE Before HPT:", dt_mse)
print("MSE After  HPT:", dt_hpt_mse)

print("-----------------------------------------------------\n")
```

```
--Decision Tree--

MSE Before HPT: 0.06059623599054048
MSE After  HPT: 0.057270828864317426
-----------------------------------------------------
```

In [189…

```python
import matplotlib.pyplot as plt

# Labels for x-axis
labels = ['MSE Before HPT', 'MSE After HPT']

# Values for y-axis
values = [dt_mse, dt_hpt_mse]

# Plotting the graph
plt.bar(labels, values, color=['brown', 'orange'])

# Adding titles and labels
plt.title('Performance Before and After Hyperparameter Tuning for MSE')
plt.xlabel('Metrics')
plt.ylabel('Values')

# Displaying the plot
plt.grid()
plt.show()
```


Performance Before and After Hyperparameter Tuning for MSE
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Values
MSE Before HPT
MSE After HPT
Metrics

MSE Before HPT: 0.06059623599054048

MSE After HPT : 0.057270828864317426

-The MSE has decreased after performing Hyperparameter Tuning.

- This could mean that the hyperparameters selected during tuning have been optimal for the dataset.

In [ ]:

In [ ]:

In [ ]:

--- Random Forest ---

In [134…

```
# Experimenting with different algorithms, ensemble methods
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()

rf.fit(train_x, train_y) #model is build

# Evaluate the Random Forest model
pred_rf = rf.predict(test_x)
```

In [135…

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Calculate evaluation metrics
rf_mse = mean_squared_error(test_y, pred_rf)
rf_rmse = np.sqrt(rf_mse)
rf_r2 = r2_score(test_y, pred_rf)
rf_mape = np.mean(np.abs((test_y - pred_rf) / test_y)) * 100


# Print evaluation metrics
print("MSE  =", rf_mse)
print("RMSE =", rf_rmse)
print("R²   =", rf_r2)
print("MAPE =", rf_mape)
```

```
MSE  = 0.030282506130809513
RMSE = 0.1740186947738935
R²   = 0.9308596769406706
MAPE = 15.7313432832414
```

In [136…

```
#print("Linear Regression MSE = ", lr_mse)
print("Decision Tree MSE     = ", dt_mse)
print("Random Forest MSE     = ", rf_mse)
```

```
Decision Tree MSE     =  0.06059623599054048
Random Forest MSE     =  0.030282506130809513
```

Decision Tree MSE : 0.06059623599054048

Random Forest MSE : 0.030282506130809513

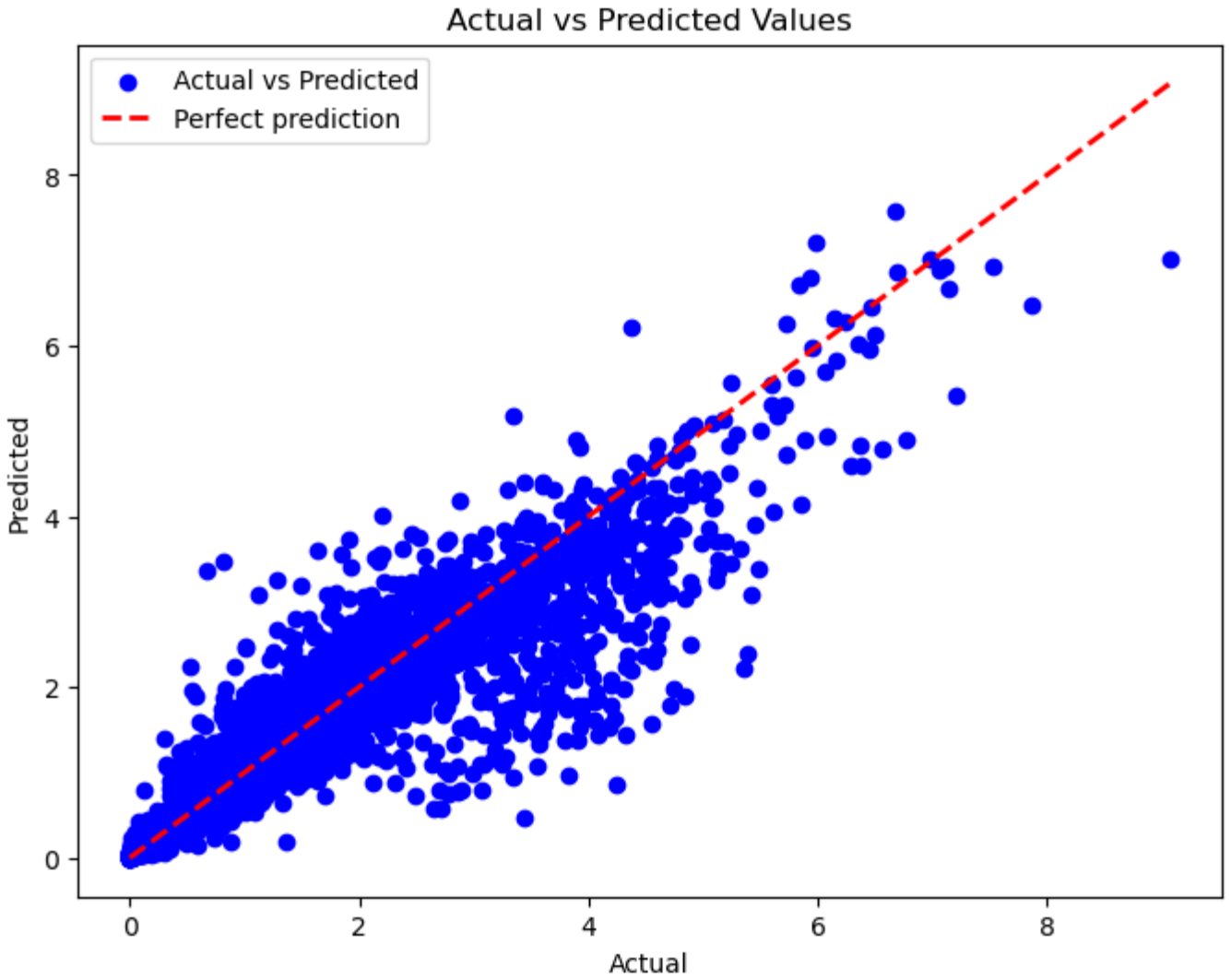
- The Random Forest model outperform the Decision Tree model, as it has a lower MSE.

In [ ]:

In [138…

```
import matplotlib.pyplot as plt

# Plotting actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(test_y, pred_rf, color='blue', label='Actual vs Predicted')
plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'k--', lw=2, color='red', label='Perfect predict
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```

```
C:\Users\Aabshaar\AppData\Local\Temp\ipykernel_11120\4029551332.py:6: UserWarning: color is redundantly defined by t
he 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'k--', lw=2, color='red', label='Perfect pred
iction')
```



In [ ]:

Hyper Parameter Tuning

In [206…

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define the parameter space for Random Forest
search_rf = {
    "n_estimators": [200, 300, 400],    # Number of trees in the forest
    "max_depth": range(40, 60),               # Maximum depth of the trees (up to but not including 21)
    "min_samples_split": range(3, 20)        # Minimum number of samples required to be at a leaf node
}

# Initialize Random Forest model
rf = RandomForestRegressor()

# Initialize RandomizedSearchCV with parameter distributions
random_search_rf = RandomizedSearchCV(rf, param_distributions=search_rf, random_state=42, n_jobs=-1)

# Fit RandomizedSearchCV to the training data
random_search_rf.fit(train_x, train_y)

# Get the best hyperparameters
best_params_rf = random_search_rf.best_params_
print("Best Hyperparameters for Random Forest:", best_params_rf)
```

```
Best Hyperparameters for Random Forest: {'n_estimators': 400, 'min_samples_split': 3, 'max_depth': 52}
```

In [ ]:

```
'''

The Best Hyperparameters Are:

n_estimators     : 400
Max Depth        : 52
Min Samples Split: 3

'''
```

Let's use them to train a final RandomFoerestRegressor model

In [139…

```
#RandomForestRegressor with the best hyperparameters
best_rf_model = RandomForestRegressor(n_estimators=400 , max_depth=52, min_samples_split=3)

# Train the model on the entire training dataset
best_rf_model.fit(train_x, train_y)

# Make predictions on the testing dataset
pred = best_rf_model.predict(test_x)

# Evaluate the model using mean squared error
rf_hpt_mse  = mean_squared_error(test_y, pred)
#best_rf_rmse = np.sqrt(best_rf_mse)
print("Mean Squared Error:", rf_hpt_mse)
```

```
Mean Squared Error: 0.0299646236326495
```

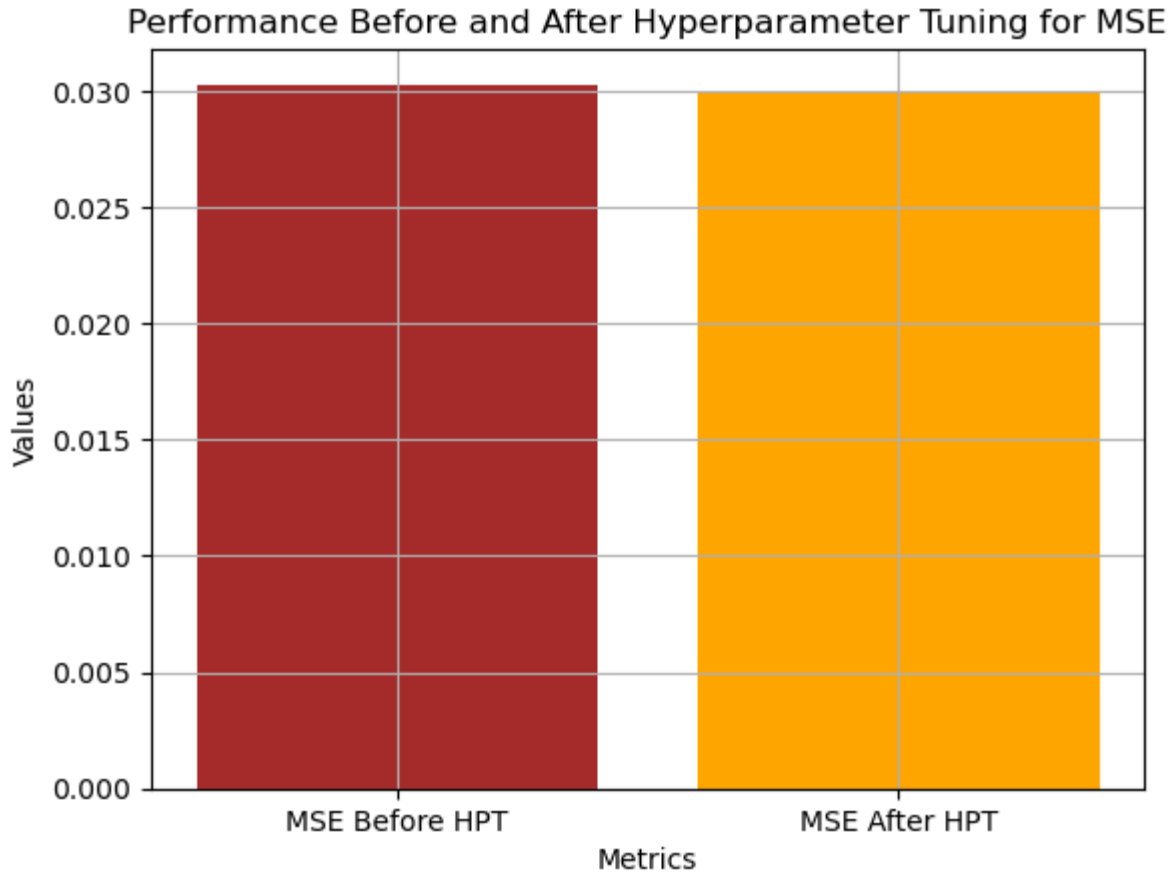
In [140…

```
#Combining MSE for both

print("--Random Forest--")
print()
print("MSE Before HPT:", rf_mse)
print("MSE After  HPT:", rf_hpt_mse)

print("-----------------------------------------------------\n")
```

```
--Random Forest--

MSE Before HPT: 0.030282506130809513
MSE After  HPT: 0.0299646236326495
-----------------------------------------------------
```

In [187…

```
import matplotlib.pyplot as plt

# Labels for x-axis
labels = ['MSE Before HPT', 'MSE After HPT']

# Values for y-axis
values = [rf_mse, rf_hpt_mse]

# Plotting the graph
plt.bar(labels, values, color=['brown', 'orange'])

# Adding titles and labels
plt.title('Performance Before and After Hyperparameter Tuning for MSE')
plt.xlabel('Metrics')
plt.ylabel('Values')

# Displaying the plot
plt.grid()
plt.show()
```


Performance Before and After Hyperparameter Tuning for MSE
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Values
MSE Before HPT
MSE After HPT
Metrics

MSE Before HPT: 0.030282506130809513

MSE After HPT : 0.029964623632649 5

-The MSE has decreased little bit after performing Hyperparameter Tuning.

- This could mean that the hyperparameters selected during tuning have been optimal for the dataset.

In [ ]:

In [ ]:

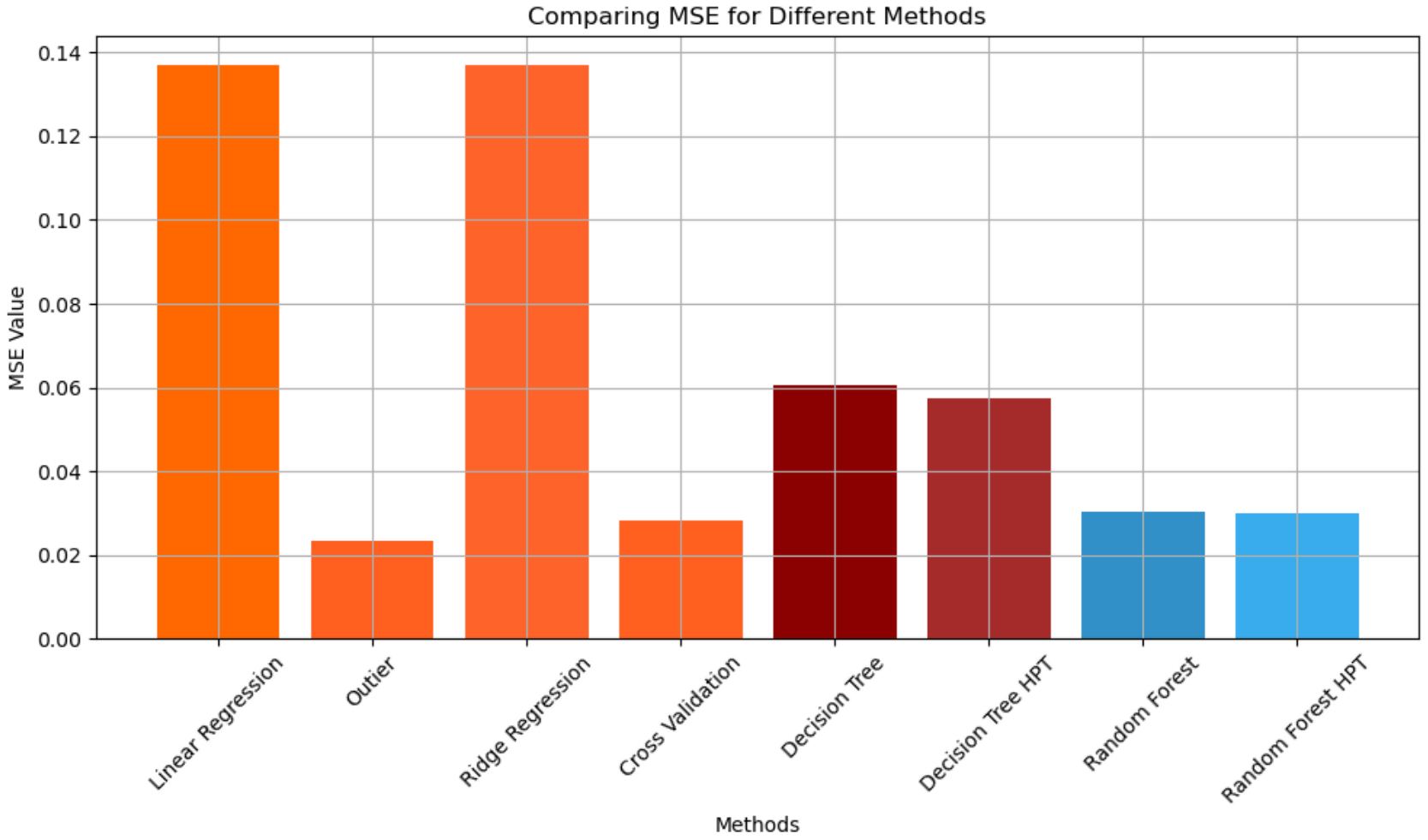
In [ ]:

In [ ]:

Comparing All MSE

In [143…

```
# MSE values for each method on test data
methods     = ['Linear Regression', 'Outier', 'Ridge Regression', 'Cross Validation', 'Decision Tree', 'Decision Tree
mse_values = [lr_mse_test, lr_new_mse, ridge_mse, lr_mse_cv, dt_mse, dt_hpt_mse, rf_mse, rf_hpt_mse]

# Creating a bar plot
plt.figure(figsize=(10, 6))
plt.bar(methods, mse_values, color=['#FF6700', '#FF5F1F', '#FE632A', '#FF5F1F', 'darkred', 'brown', '#3090C7', '#38A0

# Adding labels and title
plt.xlabel('Methods')
plt.ylabel('MSE Value')
plt.title('Comparing MSE for Different Methods')
plt.xticks(rotation=45)
plt.grid()

# Display the plot
plt.tight_layout()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

--- Feature Importance ---

Decision Tree

In [144…

```
df1 = pd.DataFrame()
df1['Feature'] = train_x.columns
df1['Importance'] = best_dt_model.feature_importances_

df1
```

Out[144]:

	Feature	Importance
0	Time	0.011141
1	Gen	0.043949
2	DishWasher	0.014020
3	HVAC	0.216127
4	HomeOffice	0.042277
5	Fridge	0.037054
6	WineCellar	0.010021
7	GarageDoor	0.242792
8	Kitchen	0.037526
9	Barn	0.062523
10	Well	0.107229
11	Microwave	0.015976
12	LivingRoom	0.096476
13	Solar	0.019968
14	Temperature	0.005479
15	WeatherCondition	0.002058
16	WeatherSummary	0.000783
17	AapparentTemperature	0.004847
18	Pressure	0.007183
19	WindSpeed	0.011924
20	PrecipIntensity	0.001286
21	PrecipProbability	0.002049
22	DewPoint	0.007312

In [ ]:

Let's break down the above code..

In [145…
```
best_dt_model.feature_importances_
```

Out[145]:
```
array([0.01114066, 0.0439485 , 0.01402043, 0.21612744, 0.04227749,
       0.03705367, 0.01002141, 0.24279187, 0.03752633, 0.06252319,
       0.10722888, 0.01597625, 0.09647557, 0.01996832, 0.00547872,
       0.00205794, 0.00078309, 0.00484673, 0.00718256, 0.01192394,
       0.00128627, 0.00204857, 0.00731215])
```

In [146…
```
train_x.columns
```

Out[146]:
```
Index(['Time', 'Gen', 'DishWasher', 'HVAC', 'HomeOffice', 'Fridge',
       'WineCellar', 'GarageDoor', 'Kitchen', 'Barn', 'Well', 'Microwave',
       'LivingRoom', 'Solar', 'Temperature', 'WeatherCondition',
       'WeatherSummary', 'AapparentTemperature', 'Pressure', 'WindSpeed',
       'PrecipIntensity', 'PrecipProbability', 'DewPoint'],
      dtype='object')
```

In [ ]:

In [ ]:
```
#combining both these we created one dataframe named as df1
```

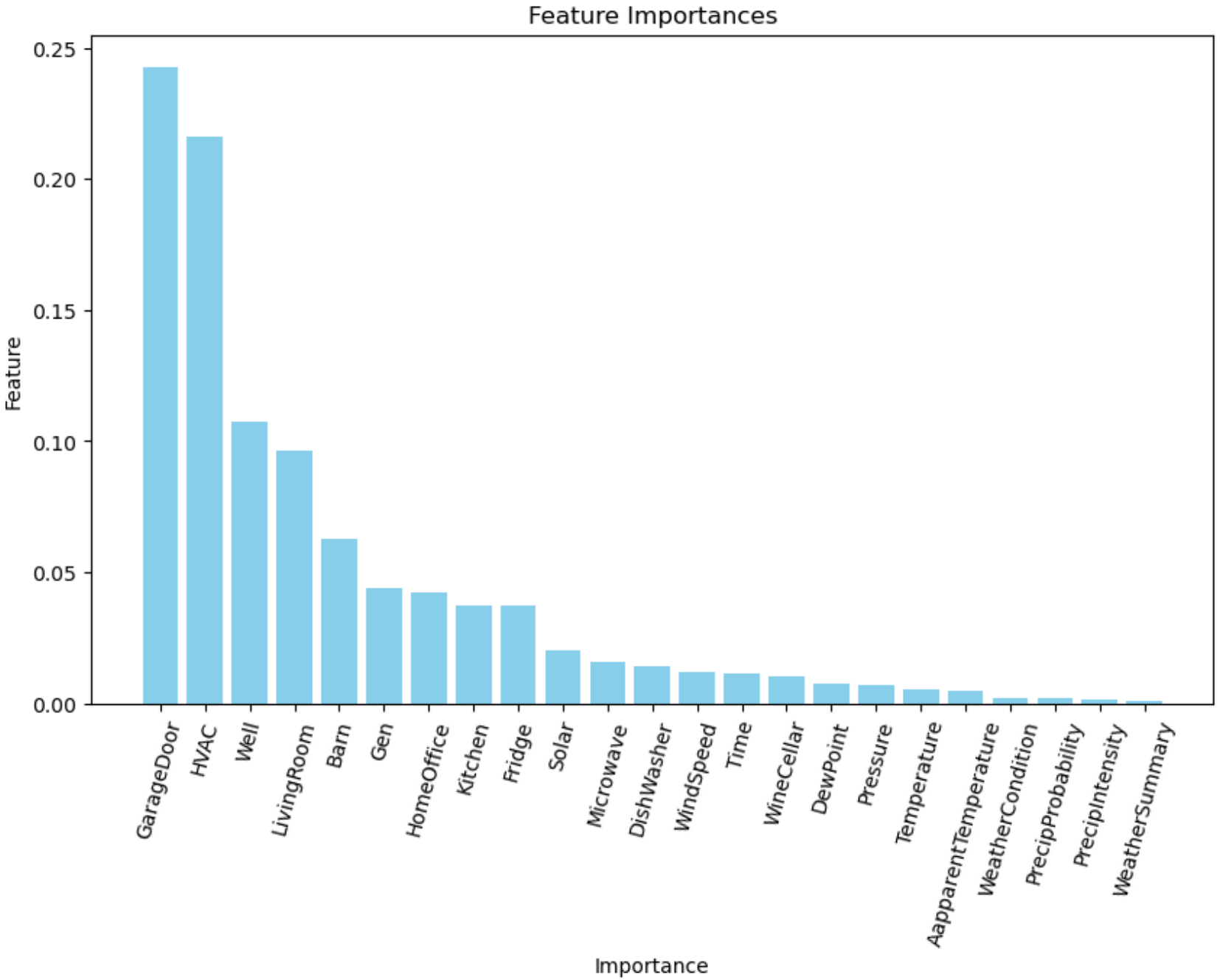
In [147…
```
#let's sort it
df1 = df1.sort_values('Importance', ascending=False)
df1.head(5)
```

Out[147]:

	Feature	Importance
7	GarageDoor	0.242792
3	HVAC	0.216127
10	Well	0.107229
12	LivingRoom	0.096476
9	Barn	0.062523

In [148…

```
# Ploting feature importances
plt.figure(figsize=(10, 6))
plt.bar(df1['Feature'], df1['Importance'], color='skyblue')
plt.xticks(rotation = 75)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances')
plt.show()
```


Feature Importances
Feature
0.25
0.20
0.15
0.10
0.05
0.00
GarageDoor
HVAC
Well
LivingRoom
Barn
Gen
HomeOffice
Kitchen
Fridge
Solar
Microwave
DishWasher
WindSpeed
Time
WineCellar
DewPoint
Pressure
Temperature
AapparentTemperature
WeatherCondition
PrecipProbability
PrecipIntensity
WeatherSummary
Importance

In [ ]:

Using them to build the model again

In [150…
```
energy = energy.loc[: ,['TotalEnergyConsumptionOfTheHouse', 'GarageDoor' , 'HVAC' , 'Well' , 'LivingRoom', 'Barn'] ]
energy.head(2)
```

Out[150]:

	TotalEnergyConsumptionOfTheHouse	GarageDoor	HVAC	Well	LivingRoom	Barn
0	0.932833	0.013083	0.061917	0.001017	0.001517	0.03135
1	0.934333	0.013117	0.063817	0.001017	0.001650	0.03150

In [151…
```
dt_x = energy.iloc[: , 1::]
dt_y = energy.TotalEnergyConsumptionOfTheHouse
```

In [152…
```
from sklearn.tree import DecisionTreeRegressor
ec_dt = DecisionTreeRegressor()

ec_dt.fit(dt_x, dt_y)

pred_ec_dt = ec_dt.predict(dt_x)
```

In [154…
```
dt_new_mse = mean_squared_error(dt_y, pred_ec_dt)
print(dt_new_mse)
```

```
2.840927879854139e-06
```

In [177…
```
print("Decision Tree Old MSE = ", dt_mse)
print("Decision Tree HPT MSE = ", dt_hpt_mse)
print("Decision Tree New MSE = ", dt_new_mse)
```

```
Decision Tree Old MSE =  0.06059623599054048
Decision Tree HPT MSE =  0.057270828864317426
Decision Tree New MSE =  2.840927879854139e-06
```

Decision Tree Old MSE = 0.06059623599054048

Decision Tree HPT MSE = 0.057270828864317426

Decision Tree New MSE = 2.840927879854139e-06

- The decision tree model's performance has significantly improved through feature engineering and model optimization.
- New MSE: 2.8409e-06 These results show that the new model, built using the top features, achieves a dramatically lower MSE, indicating a substantial enhancement in predictive accuracy and overall model performance.

The number "2.8409e-06" is in scientific notation. It represents the value 2.8409 multiplied by 10^-6

- In other words, it is equivalent to 0.0000028409
    - In the context of Mean Squared Error (MSE), this very small value indicates that the model's predictions are extremely close to the actual values, suggesting excellent performance.

In [ ]:

Random Forest

In [158…

```
df2 = pd.DataFrame()
df2['Feature'] = train_x.columns
df2['Importance'] = best_rf_model.feature_importances_

df2
```

Out[158]:

	Feature	Importance
0	Time	0.011307
1	Gen	0.032008
2	DishWasher	0.015343
3	HVAC	0.214737
4	HomeOffice	0.039553
5	Fridge	0.038390
6	WineCellar	0.012079
7	GarageDoor	0.241861
8	Kitchen	0.038452
9	Barn	0.064785
10	Well	0.104799
11	Microwave	0.016973
12	LivingRoom	0.094286
13	Solar	0.032296
14	Temperature	0.004951
15	WeatherCondition	0.001632
16	WeatherSummary	0.000968
17	AapparentTemperature	0.005422
18	Pressure	0.008431
19	WindSpeed	0.010603
20	PrecipIntensity	0.001688
21	PrecipProbability	0.001526
22	DewPoint	0.007909

In [159…

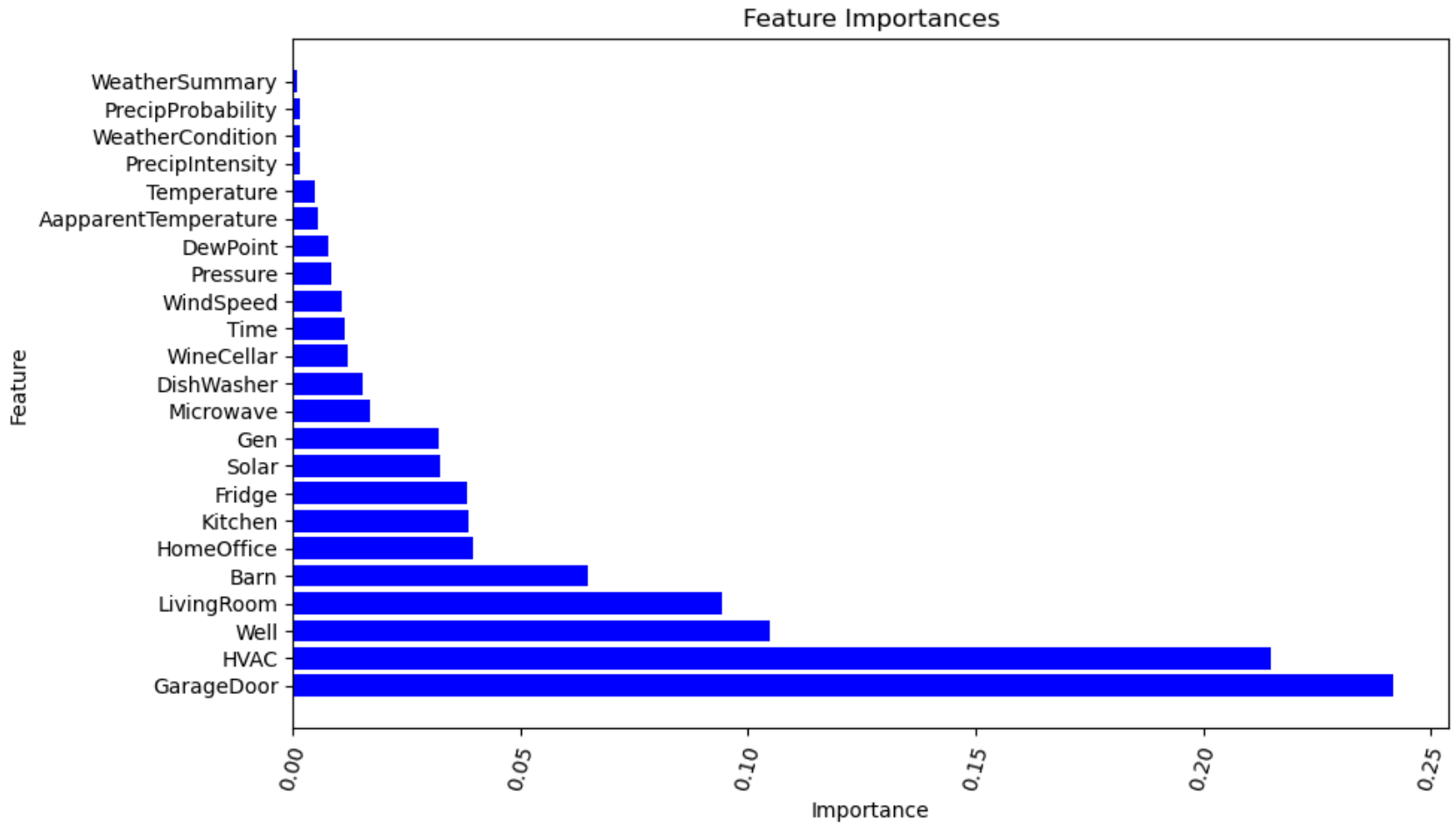
```
df2 = df2.sort_values('Importance', ascending=False)
df2.head(5)
```

Out[159]:

	Feature	Importance
7	GarageDoor	0.241861
3	HVAC	0.214737
10	Well	0.104799
12	LivingRoom	0.094286
9	Barn	0.064785

In [160…

```
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(df2['Feature'], df2['Importance'], color='blue')
plt.xticks(rotation = 75)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances')
plt.show()
```



In [ ]:

Using them to build the model again

In [161…

```
energy = energy.loc[: ,['TotalEnergyConsumptionOfTheHouse', 'GarageDoor' , 'HVAC' , 'Well' , 'LivingRoom', 'Barn'] ]
energy.head(2)
```

Out[161]:

	TotalEnergyConsumptionOfTheHouse	GarageDoor	HVAC	Well	LivingRoom	Barn
0	0.932833	0.013083	0.061917	0.001017	0.001517	0.03135
1	0.934333	0.013117	0.063817	0.001017	0.001650	0.03150

In [163…
```
rf_x = energy.iloc[: , 1::]
rf_y = energy.TotalEnergyConsumptionOfTheHouse
```

In [164…
```
from sklearn.ensemble import RandomForestRegressor
ec_rf = RandomForestRegressor()

ec_rf.fit(rf_x, rf_y)

pred_ec_rf = ec_rf.predict(rf_x)
```

In [165…
```
rf_new_mse = mean_squared_error(rf_y, pred_ec_rf)
print(rf_new_mse)
```

```
0.011584535157549414
```

In [176…
```
print("Random Forest Old MSE = ", rf_mse)
print("Random Forest HPT MSE = ", rf_hpt_mse)
print("Random Forest New MSE = ", rf_new_mse)
```

```
Random Forest Old MSE =  0.030282506130809513
Random Forest HPT MSE =  0.0299646236326495
Random Forest New MSE =  0.011584535157549414
```

Random Forest Old MSE = 0.030282506130809513

Random Forest HPT MSE = 0.029964623632649 5

Random Forest New MSE = 0.011584535157549414

- We can conclude that the model built using the top 5 records, achieves a lower MSE,
- reflecting better predictive accuracy and overall model performance.

In [ ]:

Step 1: Combine feature importances into a DataFrame for easier analysis

In [167…

```
combine_feature_imp = pd.DataFrame()
combine_feature_imp['Features'] = train_x.columns
combine_feature_imp['DT_IMP'] = df1['Importance']
combine_feature_imp['RF_IMP'] = df2['Importance']

combine_feature_imp
```

Out[167]:

	Features	DT_IMP	RF_IMP
0	Time	0.011141	0.011307
1	Gen	0.043949	0.032008
2	DishWasher	0.014020	0.015343
3	HVAC	0.216127	0.214737
4	HomeOffice	0.042277	0.039553
5	Fridge	0.037054	0.038390
6	WineCellar	0.010021	0.012079
7	GarageDoor	0.242792	0.241861
8	Kitchen	0.037526	0.038452
9	Barn	0.062523	0.064785
10	Well	0.107229	0.104799
11	Microwave	0.015976	0.016973
12	LivingRoom	0.096476	0.094286
13	Solar	0.019968	0.032296
14	Temperature	0.005479	0.004951
15	WeatherCondition	0.002058	0.001632
16	WeatherSummary	0.000783	0.000968
17	AapparentTemperature	0.004847	0.005422
18	Pressure	0.007183	0.008431
19	WindSpeed	0.011924	0.010603
20	PrecipIntensity	0.001286	0.001688
21	PrecipProbability	0.002049	0.001526
22	DewPoint	0.007312	0.007909

In [168…

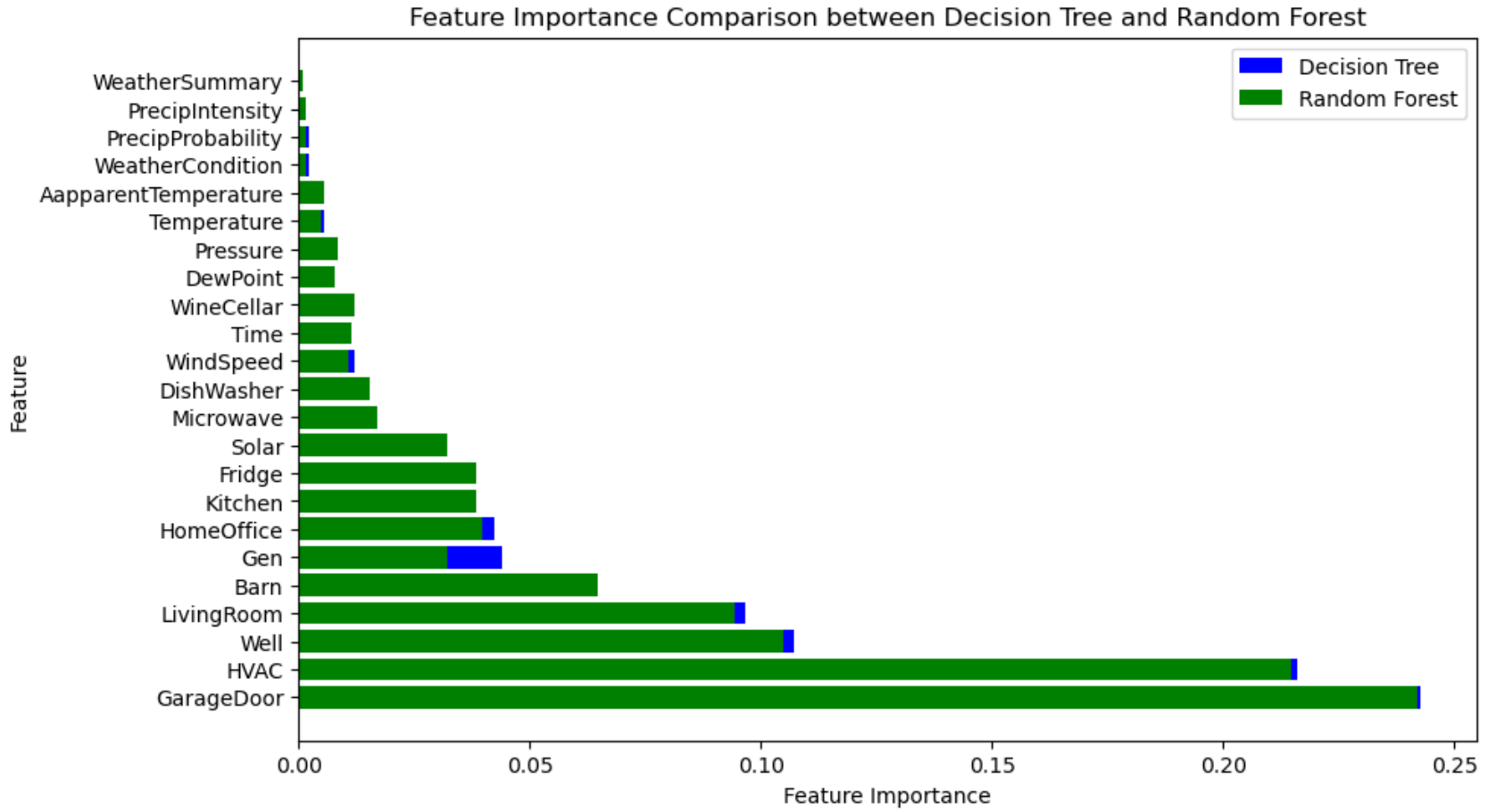
```
#sorting
combine_feature_imp_sorted = combine_feature_imp.sort_values(by=['DT_IMP', 'RF_IMP'], ascending=[False, False])
combine_feature_imp_sorted
```

Out[168]:

	Features	DT_IMP	RF_IMP
7	GarageDoor	0.242792	0.241861
3	HVAC	0.216127	0.214737
10	Well	0.107229	0.104799
12	LivingRoom	0.096476	0.094286
9	Barn	0.062523	0.064785
1	Gen	0.043949	0.032008
4	HomeOffice	0.042277	0.039553
8	Kitchen	0.037526	0.038452
5	Fridge	0.037054	0.038390
13	Solar	0.019968	0.032296
11	Microwave	0.015976	0.016973
2	DishWasher	0.014020	0.015343
19	WindSpeed	0.011924	0.010603
0	Time	0.011141	0.011307
6	WineCellar	0.010021	0.012079
22	DewPoint	0.007312	0.007909
18	Pressure	0.007183	0.008431
14	Temperature	0.005479	0.004951
17	AapparentTemperature	0.004847	0.005422
15	WeatherCondition	0.002058	0.001632
21	PrecipProbability	0.002049	0.001526
20	PrecipIntensity	0.001286	0.001688
16	WeatherSummary	0.000783	0.000968

In [169…

```
# Ploting feature importances
plt.figure(figsize=(10, 6))
plt.barh(combine_feature_imp_sorted['Features'], combine_feature_imp_sorted['DT_IMP'], color='blue', label='Decision
plt.barh(combine_feature_imp_sorted['Features'], combine_feature_imp_sorted['RF_IMP'], color='green', label='Random F
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance Comparison between Decision Tree and Random Forest')
plt.legend()
plt.show()
```



In [ ]:

Step 2 : Selects features based on importance for further analysis

In [170…

```
# Prioritize features based on importance for further analysis or model building
selected_features = combine_feature_imp.loc[(combine_feature_imp['RF_IMP'] > 0.05) & (combine_feature_imp['DT_IMP'] >

print("Selected Features are:")
print(selected_features)
```

```
Selected Features are:
3           HVAC
7     GarageDoor
9           Barn
10          Well
12    LivingRoom
Name: Features, dtype: object
```

- HVAC

- GarageDoor

- Barn

- Well

- LivingRoom

These features have importance values greater than 0.05 for both the Decision Tree (DT) and Random Forest (RF) models.

- indicating that they are considered significant predictors of energy consumption in smart buildings by both models.

Based on the selected features with importance values greater than 0.05 for both the Decision Tree and Random Forest models, we can conclude that 'HVAC', 'GarageDoor', 'Barn', 'Well' and 'LivingRoom' are the top contributors to predicting energy consumption in smart homes.

These features have been identified as significant predictors by both models, indicating that they have a substantial impact on energy consumption.

In [ ]:

In [ ]:

In [ ]:

Boosting Algorithm

In [ ]:

Gradient Boosting

In [171…

```
from sklearn.ensemble import GradientBoostingRegressor

# Initialize the model
gradient_boosting = GradientBoostingRegressor()

# Train the model
gradient_boosting.fit(train_x, train_y)

# Predict using the model
pred_gb = gradient_boosting.predict(test_x)

# Evaluate the model
gb_mse  = mean_squared_error(test_y, pred_gb)
gb_rmse = np.sqrt(mean_squared_error(test_y, pred_gb))
gb_r2   = r2_score(test_y, pred_gb)
print(f"Gradient Boosting Regressor - MSE: {gb_mse}, RMSE: {gb_rmse}, R2: {gb_r2}")
```

```
Gradient Boosting Regressor - MSE: 0.07716752831009109, RMSE: 0.2777904395584756, R2: 0.8238128702425522
```

In [ ]:

XGBOOST

In [172…

```
import xgboost as xgb

# Initialize the model
xgboost_model = xgb.XGBRegressor()

# Train the model
xgboost_model.fit(train_x, train_y)

# Predict using the model
pred_xgb = xgboost_model.predict(test_x)

# Evaluate the model
xgb_mse  = mean_squared_error(test_y, pred_xgb)
xgb_rmse = np.sqrt(mean_squared_error(test_y, pred_xgb))
xgb_r2   = r2_score(test_y, pred_xgb)
print(f"Gradient Boosting Regressor - MSE: {xgb_mse}, RMSE: {xgb_rmse}, R2: {xgb_r2}")
```

```
Gradient Boosting Regressor - MSE: 0.034454620376792595, RMSE: 0.18561955817421988, R2: 0.9213340014380669
```

In [ ]:

Random Forest + Gradient Boosting

In [173…

```python
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import RandomForestRegressor

# Combine models into an ensemble
ensemble_model = VotingRegressor([('rf', rf2), ('gbr', gradient_boosting)])

# Train ensemble model
ensemble_model.fit(train_x, train_y)

# Predict with ensemble model
y_pred_ensemble = ensemble_model.predict(test_x)

# Evaluate ensemble model
mse_ensemble_gb = mean_squared_error(test_y, y_pred_ensemble)
rmse_ensemble_gb = np.sqrt(mse_ensemble_gb)
r2_ensemble_gb = r2_score(test_y, y_pred_ensemble)

print(f"Ensemble Model - MSE: {mse_ensemble_gb}, RMSE: {rmse_ensemble_gb}, R2: {r2_ensemble_gb}")
```

```
Ensemble Model - MSE: 0.04349895264605658, RMSE: 0.2085640252921308, R2: 0.9006841895548747
```

In [ ]:

Random Forest + XGBoost

In [174…

```
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import RandomForestRegressor

# Combine models into an ensemble
ensemble_model = VotingRegressor([('rf', rf2), ('gbx', xgboost_model)])

# Train ensemble model
ensemble_model.fit(train_x, train_y)

# Predict with ensemble model
y_pred_ensemble = ensemble_model.predict(test_x)

# Evaluate ensemble model
mse_ensemble_xg = mean_squared_error(test_y, y_pred_ensemble)
rmse_ensemble_xg = np.sqrt(mse_ensemble_xg)
r2_ensemble_xg = r2_score(test_y, y_pred_ensemble)

print(f"Ensemble Model - MSE: {mse_ensemble_xg}, RMSE: {rmse_ensemble_xg}, R2: {r2_ensemble_xg}")
```

```
Ensemble Model - MSE: 0.027468510561087716, RMSE: 0.16573626809207367, R2: 0.9372845270484409
```

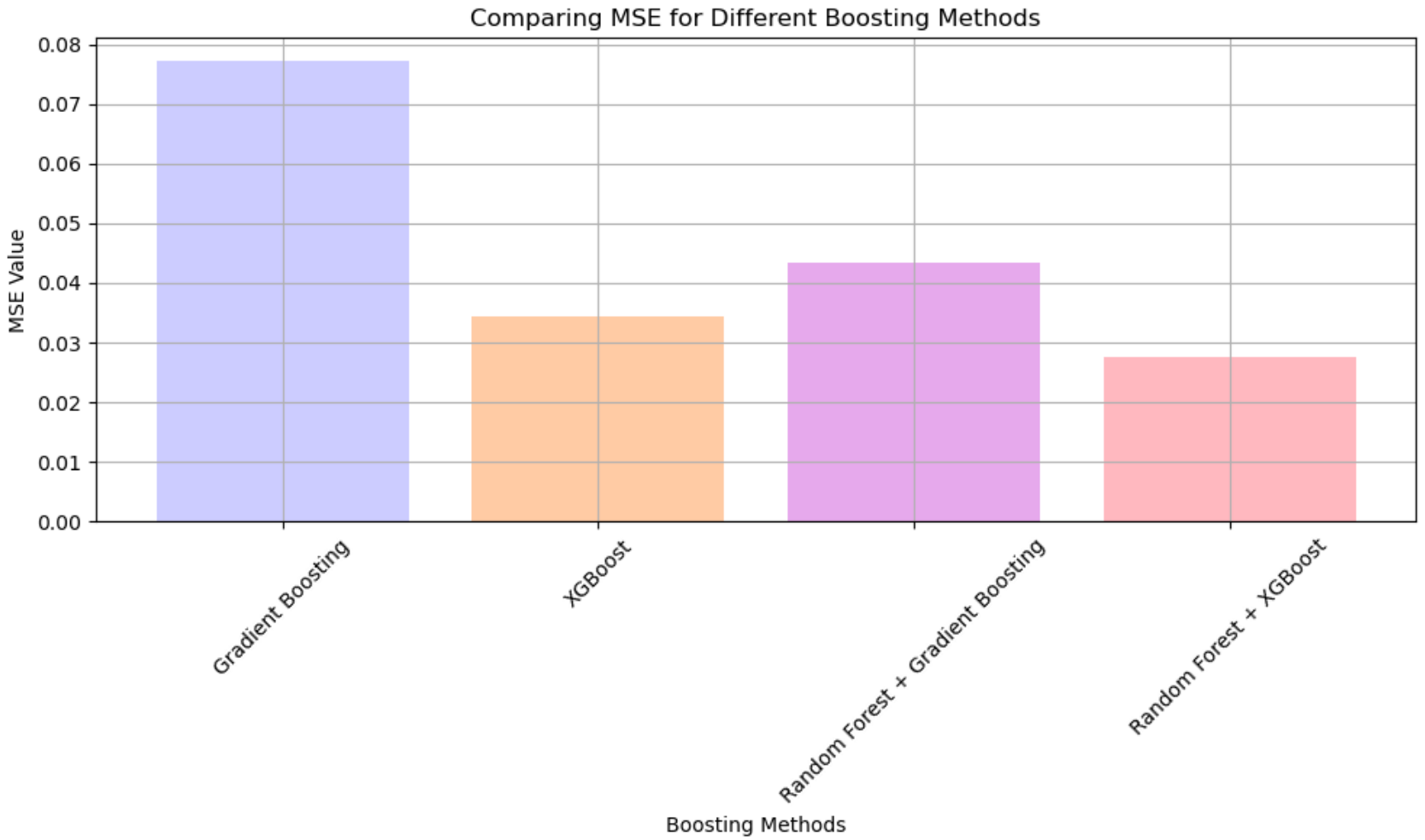
In [ ]:

In [ ]:

Comparing All BOOSTING Algorithm MSE

In [175…

```
# MSE values for each boosting method
methods     = ['Gradient Boosting', 'XGBoost', 'Random Forest + Gradient Boosting', 'Random Forest + XGBoost']
mse_values = [gb_mse, xgb_mse, mse_ensemble_gb, mse_ensemble_xg]

# Creating a bar plot
plt.figure(figsize=(10, 6))
plt.bar(methods, mse_values, color=['#CCCCFF', '#FFCBA4', '#E6A9EC', '#FFB8BF'])

# Adding labels and title
plt.xlabel('Boosting Methods')
plt.ylabel('MSE Value')
plt.title('Comparing MSE for Different Boosting Methods')
plt.xticks(rotation=45)
plt.grid()

# Display the plot
plt.tight_layout()
plt.show()
```



In [179…
```
# From the above graph, we can conclude that Random Forest + XGBoost is giving the lowest MSE among the others
```

In [ ]:

In [ ]:

Model Selection and Evaluation

1. Decision Tree Regressor (Baseline)

- **MSE** : 0.06059623599054048
- **RMSE** : 0.2461630272614888
- **R2** : 0.861648064580006
- **MAPE** : 14.8554882217281

2. Decision Tree Regressor (Hyperparameter Tuning)

- **MSE**: 0.057270828864317426

3. Decision Tree Regressor (Feature Importance)

- **MSE**: 2.840927879854139e-06

Selected Model: Decision Tree Regressor with Feature Importance

- **Reason**: Superior performance with the lowest MSE value.

---

1. Random Forest Regressor (Baseline)

- **MSE** : 0.030282506130809513
- **RMSE** : 0.1740186947738935
- **R2** : 0.9308596769406706
- **MAPE** : 15.7313432832414

2. Random Forest Regressor (Hyperparameter Tuning)

- **MSE**: 0.0299646236326495

3. Random Forest Regressor (Feature Importance)

- **MSE**: 0.011584535157549414

Selected Model: Random Forest Regressor with Feature Importance

- **Reason**: Superior performance with the lowest MSE value.

---

4. Gradient Boosting Regressor

- **MSE** : 0.07716752831009109
- **RMSE**: 0.2777904395584756
- **R2** : 0.8238128702425522

4. XGBoosting Regressor

- **MSE** : 0.034454620376792595
- **RMSE**: 0.18561955817421988
- **R2** : 0.9213340014380669

Reason for Exclusion: Despite using Boosting Algorithm, it did not outperform the Random Forest with Feature Importance.

5. Ensemble Model (Random Forest + Gradient Boosting)

- **MSE** : 0.04349895264605658
- **RMSE**: 0.2085640252921308
- **R2** : 0.9006841895548747

5. Ensemble Model (Random Forest + XGBoosting)

- **MSE** : 0.027468510561087716
- **RMSE**: 0.16573626809207367
- **R2** : 0.9372845270484409

Reason for Exclusion: Although it performed better than the Gradient Boosting Regressor alone, but it did not match the performance of the Random Forest & Decision Tree with Feature Importance.

Conclusion

The Decision Tree Regressor with Feature Importance was selected as the final model due to its superior performance metrics.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

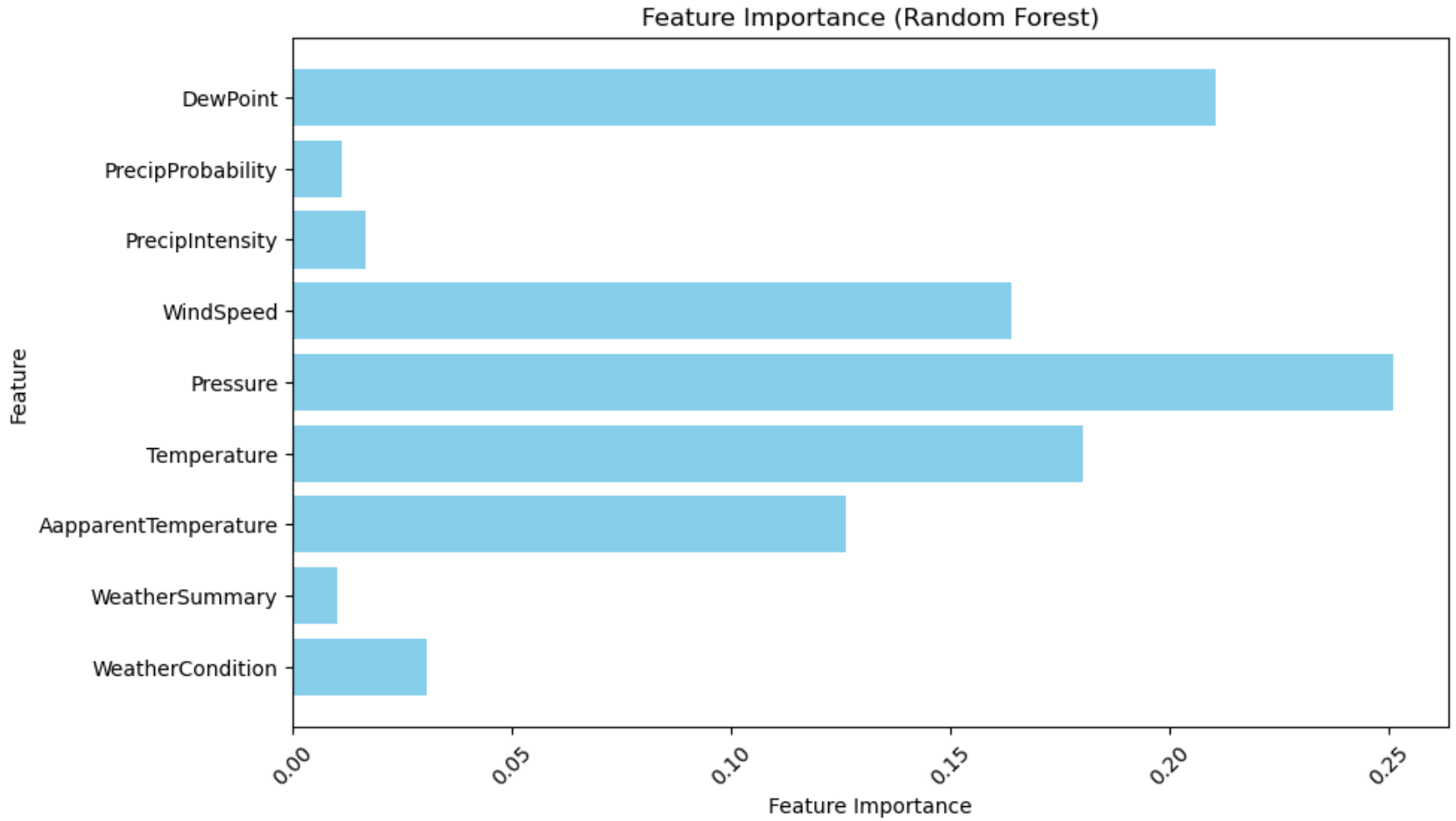
--- Energy Consumption Prediction based on Environmental Conditions ---

In [114…

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load the dataset
energy = pd.read_csv("E:/DA_Project/EnergyConsumption2LakhFinal.csv")

# Remove spaces from column names
energy.columns = energy.columns.str.replace(' ', '')

# Select relevant columns including weather-related variables
X = energy[['WeatherCondition', 'WeatherSummary','AapparentTemperature','Temperature',
            'Pressure', 'WindSpeed', 'PrecipIntensity', 'PrecipProbability', 'DewPoint']]

y = energy['TotalEnergyConsumptionOfTheHouse']

# Encode categorical variables using label encoding
le = LabelEncoder()
X['WeatherCondition'] = le.fit_transform(X['WeatherCondition'])
X['WeatherSummary'] = le.fit_transform(X['WeatherSummary'])

# Split the 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=42)

# Initialize and train a Random Forest regressor model
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Plot feature importances
feature_importances = rf_model.feature_importances_

plt.figure(figsize=(10, 6))
plt.barh(X.columns, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
```

```
plt.ylabel('Feature')
plt.title('Feature Importance (Random Forest)')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()

# Get feature importances and create a DataFrame
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort the DataFrame by feature importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Display the top N most important features
N = 5  # Number of top features to display
top_features = feature_importance_df.head(N)
print("Top", N, "most important features:")
print(top_features)
```

```
C:\Users\Aabshaar\AppData\Local\Temp\ipykernel_11120\3153471918.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returnin
g-a-view-versus-a-copy
  X['WeatherCondition'] = le.fit_transform(X['WeatherCondition'])
C:\Users\Aabshaar\AppData\Local\Temp\ipykernel_11120\3153471918.py:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returnin
g-a-view-versus-a-copy
  X['WeatherSummary'] = le.fit_transform(X['WeatherSummary'])
Mean Squared Error: 0.20830483291831334
```



Top 5 most important features:

	Feature	Importance
4	Pressure	0.250999
8	DewPoint	0.210530
3	Temperature	0.180208
5	WindSpeed	0.163888
2	AapparentTemperature	0.126205

1. Pressure: Importance = 0.250999

2. DewPoint: Importance = 0.210530

3. Temperature: Importance = 0.180208

4. WindSpeed: Importance = 0.163888

5. AapparentTemperature: Importance = 0.126205

- These results indicate that 'Pressure' is considered the most important feature, followed by 'WindSpeed', 'Temperature', 'ApparentTemperature', and 'WeatherCondition'.
- These features play significant roles in predicting the total energy consumption of the house according to the Random Forest model.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: