Machine Learning Project

Prabal Ghosh

MSc Data Science & Artificial Intelligence

Université Côte d'Azur

Machine Learning

Date- 16/12/2023

Context: Climate Change and Building Energy Efficiency

Climate change stands as an urgent, multifaceted global issue significantly impacting energy policy and infrastructure. Addressing climate change involves both mitigation (reducing greenhouse gas emissions) and adaptation (preparing for inevitable consequences). Mitigation efforts require changes across electricity systems, transportation, buildings, industries, and land use.

According to a report by the International Energy Agency (IEA), the life cycle of buildings, from construction to demolition, accounted for 37% of global CO2 emissions associated with energy and processes in 2020. However, there exists substantial potential to decrease buildings' energy consumption by integrating easily implementable solutions with cutting-edge strategies.

For instance, renovated buildings have shown the capability to reduce heating and cooling energy needs by 50-90%. Moreover, many of these energy efficiency measures yield overall cost savings and additional benefits, such as providing occupants with cleaner air. Achieving this potential is viable while upholding the services offered by the buildings.

The Dataset and Challenge

Accurate predictions of energy consumption for a building based on its characteristics are crucial for policymakers to effectively target renovation efforts and maximize emissions reductions.

The dataset utilized originates from the Lawrence Berkeley National Laboratory (Berkeley Lab). The task at hand involves analyzing variations in building energy efficiency to construct one or more predictive models for estimating the energy consumption of buildings.

The provided data encompasses descriptions of building characteristics along with climatic and meteorological variables specific to the regions where these buildings are situated.

Dataset Description

The dataset comprises around 100k observations gathered over a span of 7 years across various locations, focusing on building energy usage.

It encompasses:

- Building characteristics (e.g., floor area, installation type, etc.).
- Meteorological data specific to each building's location (e.g., mean annual temperature, total annual precipitation, etc.).
- Energy consumption details for each building within a given year.

Each row in the dataset corresponds to a singular building observed in a particular year.

Your objective is to predict the Site Energy Use Intensity (EUI) for each row, leveraging the building characteristics and the meteorological data associated with the building's location.

Evaluation Metrics: Negative Root Mean Square Error

Features

- **id:** Building ID
- **Year_Factor:** Anonymized year when weather and energy usage factors were observed
- **State_Factor:** Anonymized state in which the building is located
- **building_class:** Building classification
- **facility_type:** Building usage type
- **floor_area:** Floor area (in square feet) of the building
- **year_built:** Year in which the building was constructed
- **energy_star_rating:** The Energy Star rating of the building
- **ELEVATION:** Elevation of the building location
- **january_min_temp:** Minimum temperature in January (in Fahrenheit) at the location of the building
- **january_avg_temp:** Average temperature in January (in Fahrenheit) at the location of the building
- **january_max_temp:** Maximum temperature in January (in Fahrenheit) at the location of the building
- **cooling_degree_days:** Cooling degree day for a given day, representing the number of degrees where the daily average temperature exceeds 65 degrees Fahrenheit. The monthly sum produces an annual total at the building's location.
- **heating_degree_days:** Heating degree day for a given day, representing the number of degrees where the daily average temperature falls under 65 degrees Fahrenheit. The monthly sum produces an annual total at the building's location.
- **precipitation_inches:** Annual precipitation in inches at the location of the building
- **snowfall_inches:** Annual snowfall in inches at the location of the building
- **snowdepth_inches:** Annual snow depth in inches at the location of the building
- **avg_temp:** Average temperature over a year at the location of the building
- **days_below_30F:** Total number of days below 30 degrees Fahrenheit at the location of the building
- **days_below_20F:** Total number of days below 20 degrees Fahrenheit at the location of the building

- **days_below_10F:** Total number of days below 10 degrees Fahrenheit at the location of the building
- **days_below_0F:** Total number of days below 0 degrees Fahrenheit at the location of the building
- **days_above_80F:** Total number of days above 80 degrees Fahrenheit at the location of the building
- **days_above_90F:** Total number of days above 90 degrees Fahrenheit at the location of the building
- **days_above_100F:** Total number of days above 100 degrees Fahrenheit at the location of the building
- **days_above_110F:** Total number of days above 110 degrees Fahrenheit at the location of the building
- **direction_max_wind_speed:** Wind direction for maximum wind speed at the location of the building, given in 360-degree compass point directions (e.g., 360 = north, 180 = south, etc.).
- **direction_peak_wind_speed:** Wind direction for peak wind gust speed at the location of the building, given in 360-degree compass point directions (e.g., 360 = north, 180 = south, etc.).
- **max_wind_speed:** Maximum wind speed at the location of the building
- **days_with_fog:** Number of days with fog at the location of the building

Target

- **site_eui:** Site Energy Usage Intensity is the amount of heat and electricity consumed by a building as reflected in utility bills.

Your Job

Your task involves demonstrating your knowledge gained from the machine learning course by constructing an sklearn pipeline to perform the following:

1. **Data Pre-processing:**
    - Correctly pre-process the data based on its category. The specific steps for data pre-processing are not provided here as they are covered in various courses.
2. **Hyper-parameter Tuning:**
    - Choose appropriate hyper-parameters for the selected models. While the specific hyper-parameters are not mentioned here (as they have been covered in the course), your task involves identifying and tuning the right hyper-parameters for the models.

An initial Exploratory Data Analysis (EDA) is crucial to understand the nature of each feature within the dataset.

CAUTION:

- Utilize only the concepts learned in the course. If you include any additional concepts, ensure you understand and can explain them thoroughly.

- Avoid blindly copying code from the internet or generated by ChatGPT without comprehension. It's better to have a concise and fully understood piece of work rather than an extensive application of techniques without understanding their functionality or usefulness.

You'll need to present a written notebook (with comments) that justifies your choices for different stages of the pipeline.

Solution

I referred this book for data preprocessing- Click here to download data preprocessing book

Libraries

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('bmh')
sns.set_style({'axes.grid':False})
%matplotlib inline
```

In [2]:

```
import sklearn
from sklearn.decomposition import PCA

from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_selector as selector
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn import set_config
from sklearn.metrics import mean_squared_error

from imblearn.pipeline import Pipeline as ImbPipeline

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [3]:

```
from sklearn.svm import SVR
from sklearn import linear_model

from sklearn.linear_model import LinearRegression, Ridge, RidgeCV,Lasso,LassoCV,Elas
from sklearn.tree import DecisionTreeRegressor, plot_tree

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
```

Dataset

In [4]:
```
energy_df = pd.read_csv("C:\\Users\\praba\\Desktop\\uca1\\M1\\ML\\final project\\dat
energy_df.tail(5)
```

Out[4]:

	Year_Factor	State_Factor	building_class	facility_type	floor_area	year_built	energy_st
75752	6	State_11	Commercial	Office_Uncategorized	20410.0	1995.0	
75753	6	State_11	Residential	5plus_Unit_Building	40489.0	1910.0	
75754	6	State_11	Commercial	Commercial_Other	28072.0	1917.0	
75755	6	State_11	Commercial	Commercial_Other	53575.0	2012.0	
75756	6	State_11	Residential	2to4_Unit_Building	23888.0	1974.0	

5 rows × 64 columns

In [5]:
```
energy_df.shape
```

Out[5]:
```
(75757, 64)
```

In [6]:
```
'''All id values are unique which are irrelavent for model training'''
energy_df['id'].unique().shape
```

Out[6]:
```
(75757,)
```

In [7]:
```
energy_df['Year_Factor'].unique()
```

Out[7]:
```
array([1, 2, 3, 4, 5, 6], dtype=int64)
```

There are a total of six different values for the 'Year_Factor' feature in this dataset. Therefore, I will use one-hot encoding for these values, considering them as distinct levels or categorical entries.

Checking NAN data

I am checking for missing data because if I have the model can't learn properly, since some of the data would have values that don't represent the reality. Generally we use inputers to replace the missing data with statistical meassures such as the mean/median for numerical columns and the mode for categorical columns, but first we need to check if there is missing data at all.

In [8]:
```
energy_df.describe()
```

Out[8]:

	Year_Factor	floor_area	year_built	energy_star_rating	ELEVATION	january_min_temp
count	75757.000000	7.575700e+04	73920.000000	49048.000000	75757.000000	75757.00000C
mean	4.367755	1.659839e+05	1952.306764	61.048605	39.506323	11.432343
std	1.471441	2.468758e+05	37.053619	28.663683	60.656596	9.381027
min	1.000000	9.430000e+02	0.000000	0.000000	-6.400000	-19.00000C
25%	3.000000	6.237900e+04	1927.000000	40.000000	11.900000	6.00000C
50%	5.000000	9.136700e+04	1951.000000	67.000000	25.000000	11.00000C

	Year_Factor	floor_area	year_built	energy_star_rating	ELEVATION	january_min_temp
75%	6.000000	1.660000e+05	1977.000000	85.000000	42.700000	13.00000C
max	6.000000	6.385382e+06	2015.000000	100.000000	1924.500000	49.00000C

8 rows × 61 columns

In [9]:

```
'''Nan value is present in the dataset'''
energy_df.isnull().any().any()
```

Out[9]:

```
True
```

In [10]:

```
# a = energy_df_id.isna().sum()
# for i in a:
#     if i !=0:
#         print(i)
```

In [11]:

```
energy_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75757 entries, 0 to 75756
Data columns (total 64 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Year_Factor                75757 non-null  int64  
 1   State_Factor               75757 non-null  object 
 2   building_class             75757 non-null  object 
 3   facility_type              75757 non-null  object 
 4   floor_area                 75757 non-null  float64
 5   year_built                 73920 non-null  float64
 6   energy_star_rating         49048 non-null  float64
 7   ELEVATION                  75757 non-null  float64
 8   january_min_temp           75757 non-null  int64  
 9   january_avg_temp           75757 non-null  float64
 10  january_max_temp           75757 non-null  int64  
 11  february_min_temp          75757 non-null  int64  
 12  february_avg_temp          75757 non-null  float64
 13  february_max_temp          75757 non-null  int64  
 14  march_min_temp             75757 non-null  int64  
 15  march_avg_temp             75757 non-null  float64
 16  march_max_temp             75757 non-null  int64  
 17  april_min_temp             75757 non-null  int64  
 18  april_avg_temp             75757 non-null  float64
 19  april_max_temp             75757 non-null  int64  
 20  may_min_temp               75757 non-null  int64  
 21  may_avg_temp               75757 non-null  float64
 22  may_max_temp               75757 non-null  int64  
 23  june_min_temp              75757 non-null  int64  
 24  june_avg_temp              75757 non-null  float64
 25  june_max_temp              75757 non-null  int64  
 26  july_min_temp              75757 non-null  int64  
 27  july_avg_temp              75757 non-null  float64
 28  july_max_temp              75757 non-null  int64  
 29  august_min_temp            75757 non-null  int64  
 30  august_avg_temp            75757 non-null  float64
 31  august_max_temp            75757 non-null  int64  
 32  september_min_temp         75757 non-null  int64  
```

```
 33  september_avg_temp         75757 non-null  float64
 34  september_max_temp         75757 non-null  int64
 35  october_min_temp           75757 non-null  int64
 36  october_avg_temp           75757 non-null  float64
 37  october_max_temp           75757 non-null  int64
 38  november_min_temp          75757 non-null  int64
 39  november_avg_temp          75757 non-null  float64
 40  november_max_temp          75757 non-null  int64
 41  december_min_temp          75757 non-null  int64
 42  december_avg_temp          75757 non-null  float64
 43  december_max_temp          75757 non-null  int64
 44  cooling_degree_days        75757 non-null  int64
 45  heating_degree_days        75757 non-null  int64
 46  precipitation_inches       75757 non-null  float64
 47  snowfall_inches            75757 non-null  float64
 48  snowdepth_inches           75757 non-null  int64
 49  avg_temp                   75757 non-null  float64
 50  days_below_30F             75757 non-null  int64
 51  days_below_20F             75757 non-null  int64
 52  days_below_10F             75757 non-null  int64
 53  days_below_0F              75757 non-null  int64
 54  days_above_80F             75757 non-null  int64
 55  days_above_90F             75757 non-null  int64
 56  days_above_100F            75757 non-null  int64
 57  days_above_110F            75757 non-null  int64
 58  direction_max_wind_speed   34675 non-null  float64
 59  direction_peak_wind_speed  33946 non-null  float64
 60  max_wind_speed             34675 non-null  float64
 61  days_with_fog              29961 non-null  float64
 62  site_eui                   75757 non-null  float64
 63  id                         75757 non-null  int64
dtypes: float64(24), int64(37), object(3)
memory usage: 37.0+ MB
```

In [12]:

```
# energy_df_id.isna().sum()
```

In [13]:

```
'''Get the features with their corresponding total number of missing values.'''
missing_columns = len(energy_df) - energy_df.loc[:, np.sum(energy_df.isnull())>0].co
missing_columns
```

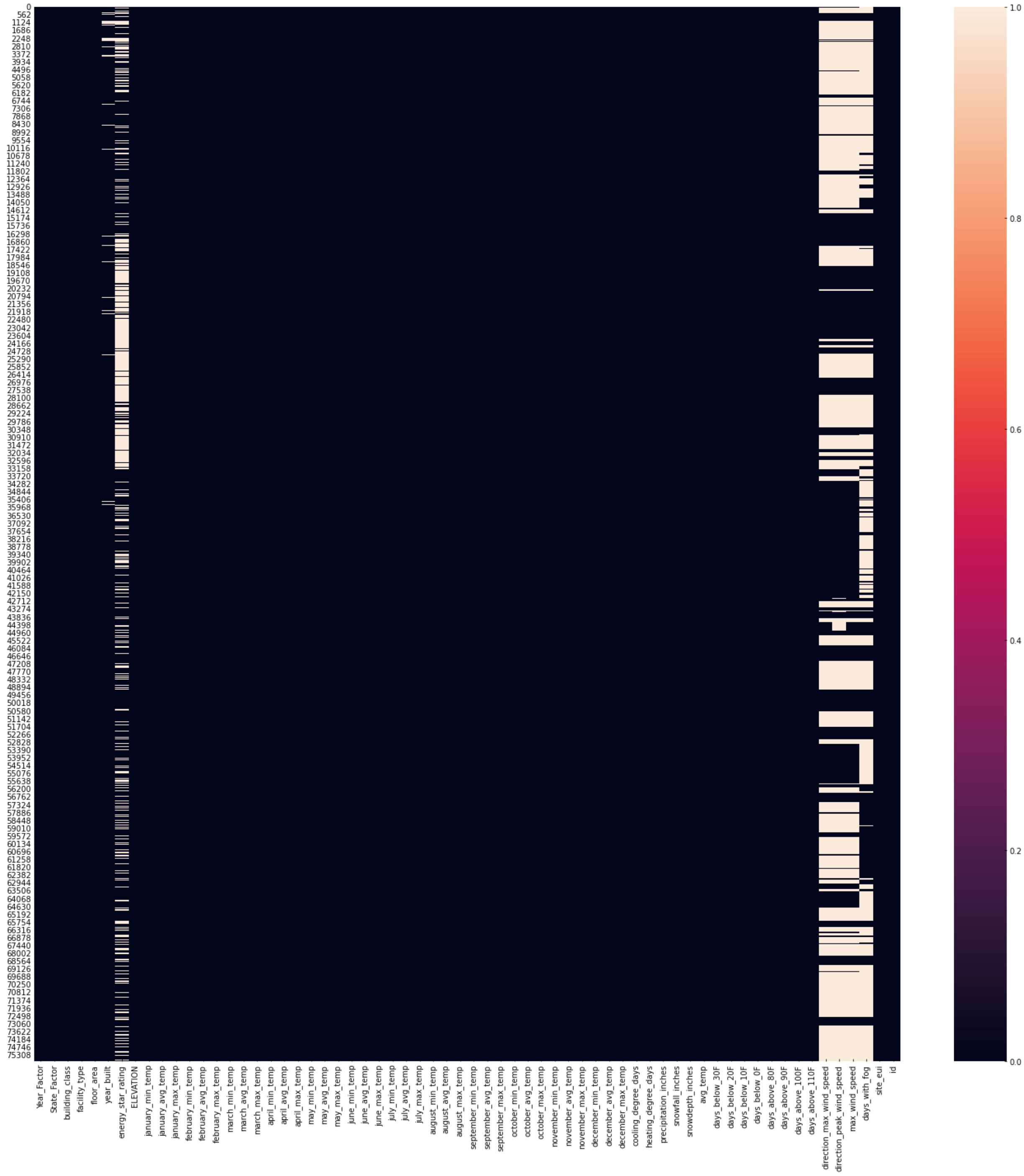
Out[13]:

```
year_built                    1837
energy_star_rating           26709
direction_max_wind_speed     41082
direction_peak_wind_speed    41811
max_wind_speed               41082
days_with_fog                45796
dtype: int64
```

heatmap plot to see the null values in dataset

In [14]:

```
'''heatmap plot to see the null values in dataset'''

plt.figure(figsize=(25,25))
sns.heatmap(energy_df.isnull())
plt.show()
```



Columns with Null Values

The following six columns contain null values:

- year_built
- energy_star_rating
- direction_max_wind_speed
- direction_peak_wind_speed
- max_wind_speed
- days_with_fog

Dropping Columns with Null Values exceeding 40,000 instances

Upon inspecting the dataset, it was found that the following columns: **'direction_max_wind_speed', 'direction_peak_wind_speed', 'max_wind_speed', and**

'days_with_fog' contain a substantial number of null values, each exceeding 40,000 instances, which represents more than 50% of the data in these columns.

Therefore, due to the significant presence of missing values, I have decided to drop these columns from the dataset. This step ensures a more robust and accurate analysis by excluding columns with inadequate or unreliable data.

In [15]:
```
'''All id values are unique which are irrelavent for model training. Thats why I am

energy_df= energy_df.loc[:, energy_df.columns != "id"]
# energy_df.head(3)
```

In [16]:
```
"""'direction_max_wind_speed', 'direction_peak_wind_speed', 'max_wind_speed', and 'd

energy_df.drop(['direction_max_wind_speed','direction_peak_wind_speed','max_wind_spe
```

In [17]:
```
energy_df.shape
```

Out[17]:
```
(75757, 59)
```

Null values in 'year_built' column was just approx 1% of the whole data so i will fill those using mean

In [18]:
```
energy_df["year_built"].isna().sum()
```

Out[18]:
```
1837
```

In [19]:
```
# energy_df['year_built'].mode()
```

In [20]:
```
# np.mean(energy_df['year_built'])
```

In [21]:
```
# energy_df.dropna(subset=['year_built'], inplace=True, axis=0)
# energy_df.shape
```

In [22]:
```
# Fill NaN values in a specific column with the mean
mean_value_year = energy_df['year_built'].mean()
energy_df['year_built'].fillna(mean_value_year, inplace=True)
```

In [23]:
```
'''Get the features (with missing values) and their corresponding missing values.'''
missing_columns = len(energy_df) - energy_df.loc[:, np.sum(energy_df.isnull())>0].co
missing_columns
```

Out[23]:
```
energy_star_rating    26709
dtype: int64
```

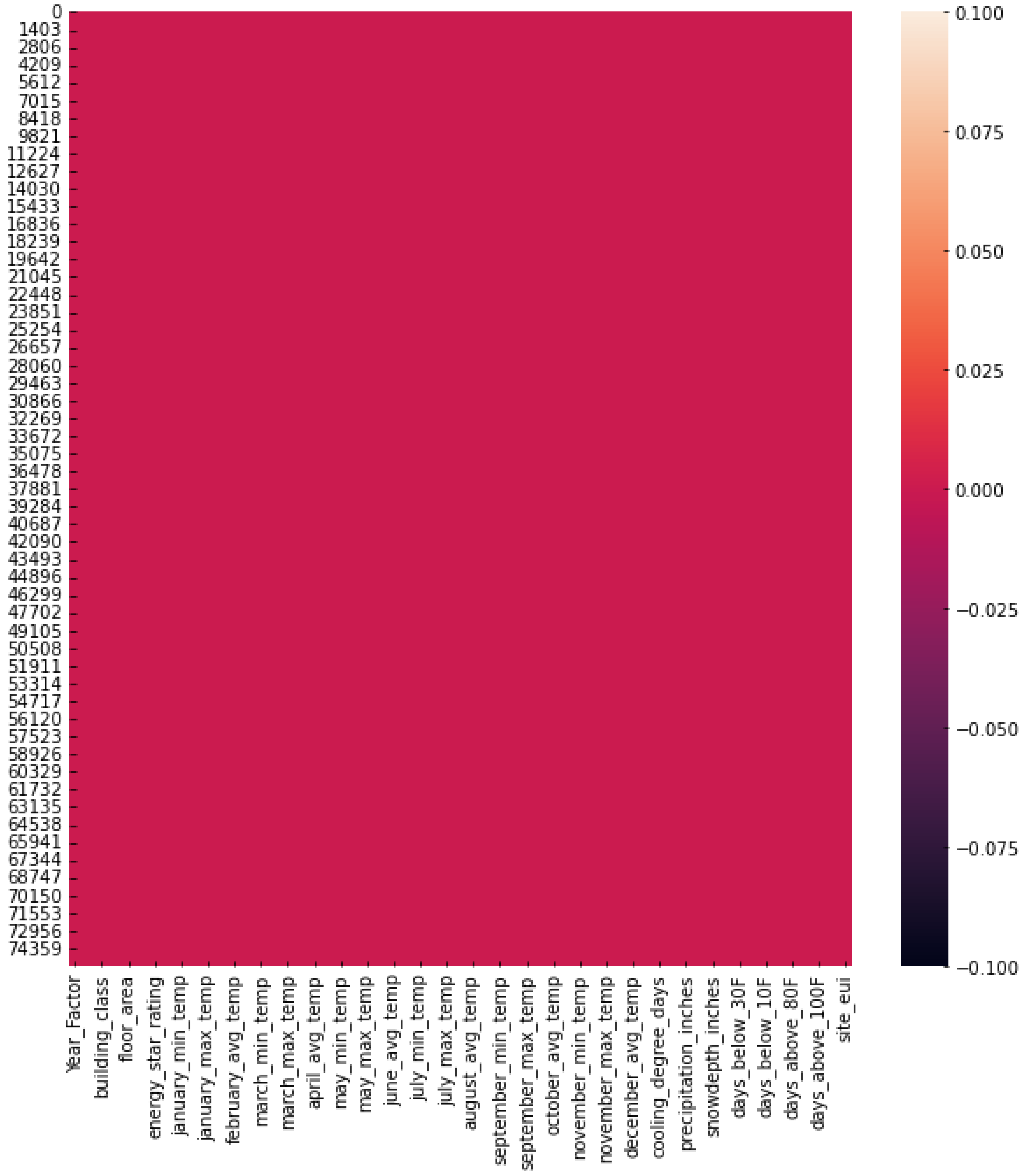
In column "energy_star_rating" 25472 null values present. So I am filling those using the mean value of the column

In [24]:
```
# Fill NaN values in a specific column with the mean
mean_value = energy_df['energy_star_rating'].mean()
```

```
energy_df['energy_star_rating'].fillna(mean_value, inplace=True)
```

In [25]:

```
'''heatmap plot to see the null values in dataset=====>>>> Here no null value presen

plt.figure(figsize=(10,10))
sns.heatmap(energy_df.isnull())
plt.show()
```



In [26]:

```
"""Now there is no nan value present in the dataset."""
energy_df.isnull().sum().sum()
```

Out[26]: 0

In [27]:

```
# features = energy_df.columns.tolist()
# print("The columns in dataset:----=== ",features)
# type(features)
```

There are a total of six different values for the 'Year_Factor' feature in this dataset. Therefore, I will use one-hot encoding for these values, considering them as distinct levels as categorical entries.

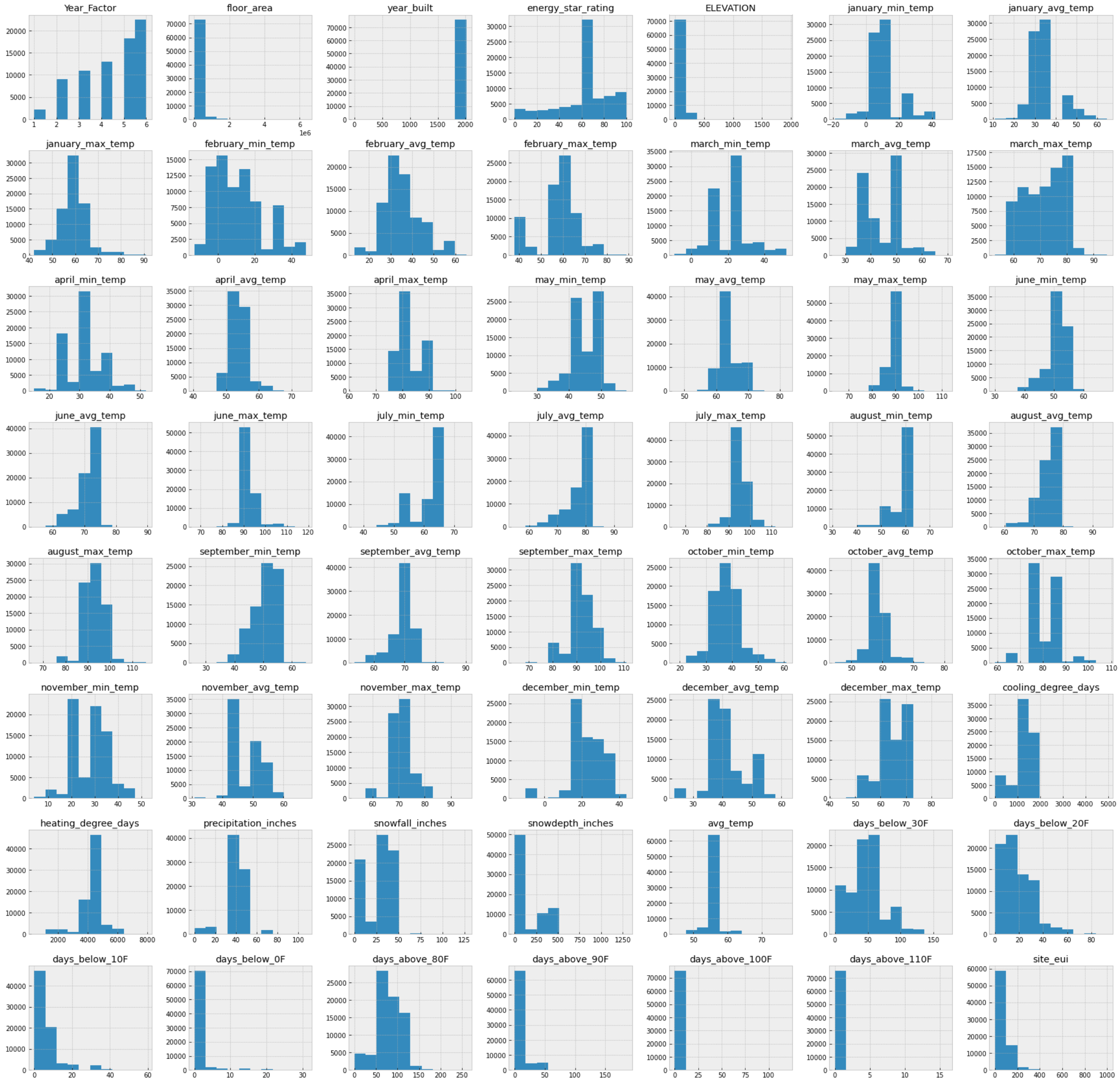
In [28]:
```
'''Numerical and categorical columns are seperated and Year_Factor is choosen as cat
import pandas as pd

features = energy_df.columns.tolist()

Num_features = [feature for feature in features if
                energy_df[feature].dtype != 'object' and
                feature != 'Year_Factor']
Cat_features = [feature for feature in features if
                energy_df[feature].dtype == object or feature == 'Year_Factor' ]


print("Num_features: \n",Num_features)
print("\n")

print("Cat_features: \n",Cat_features)
```

```
Num_features: 
 ['floor_area', 'year_built', 'energy_star_rating', 'ELEVATION', 'january_min_temp',
'january_avg_temp', 'january_max_temp', 'february_min_temp', 'february_avg_temp', 'fe
bruary_max_temp', 'march_min_temp', 'march_avg_temp', 'march_max_temp', 'april_min_te
mp', 'april_avg_temp', 'april_max_temp', 'may_min_temp', 'may_avg_temp', 'may_max_tem
p', 'june_min_temp', 'june_avg_temp', 'june_max_temp', 'july_min_temp', 'july_avg_tem
p', 'july_max_temp', 'august_min_temp', 'august_avg_temp', 'august_max_temp', 'septem
ber_min_temp', 'september_avg_temp', 'september_max_temp', 'october_min_temp', 'octob
er_avg_temp', 'october_max_temp', 'november_min_temp', 'november_avg_temp', 'november
_max_temp', 'december_min_temp', 'december_avg_temp', 'december_max_temp', 'cooling_d
egree_days', 'heating_degree_days', 'precipitation_inches', 'snowfall_inches', 'snowd
epth_inches', 'avg_temp', 'days_below_30F', 'days_below_20F', 'days_below_10F', 'days
_below_0F', 'days_above_80F', 'days_above_90F', 'days_above_100F', 'days_above_110F',
'site_eui']


Cat_features: 
 ['Year_Factor', 'State_Factor', 'building_class', 'facility_type']
```

In [29]:
```
energy_df.hist(figsize=[30,30])
```

Out[29]:
```
array([[<AxesSubplot:title={'center':'Year_Factor'}>,
        <AxesSubplot:title={'center':'floor_area'}>,
        <AxesSubplot:title={'center':'year_built'}>,
        <AxesSubplot:title={'center':'energy_star_rating'}>,
        <AxesSubplot:title={'center':'ELEVATION'}>,
        <AxesSubplot:title={'center':'january_min_temp'}>,
        <AxesSubplot:title={'center':'january_avg_temp'}>],
       [<AxesSubplot:title={'center':'january_max_temp'}>,
        <AxesSubplot:title={'center':'february_min_temp'}>,
        <AxesSubplot:title={'center':'february_avg_temp'}>,
        <AxesSubplot:title={'center':'february_max_temp'}>,
        <AxesSubplot:title={'center':'march_min_temp'}>,
        <AxesSubplot:title={'center':'march_avg_temp'}>,
        <AxesSubplot:title={'center':'march_max_temp'}>],
       [<AxesSubplot:title={'center':'april_min_temp'}>,
        <AxesSubplot:title={'center':'april_avg_temp'}>,
        <AxesSubplot:title={'center':'april_max_temp'}>,
        <AxesSubplot:title={'center':'may_min_temp'}>,
        <AxesSubplot:title={'center':'may_avg_temp'}>,
```

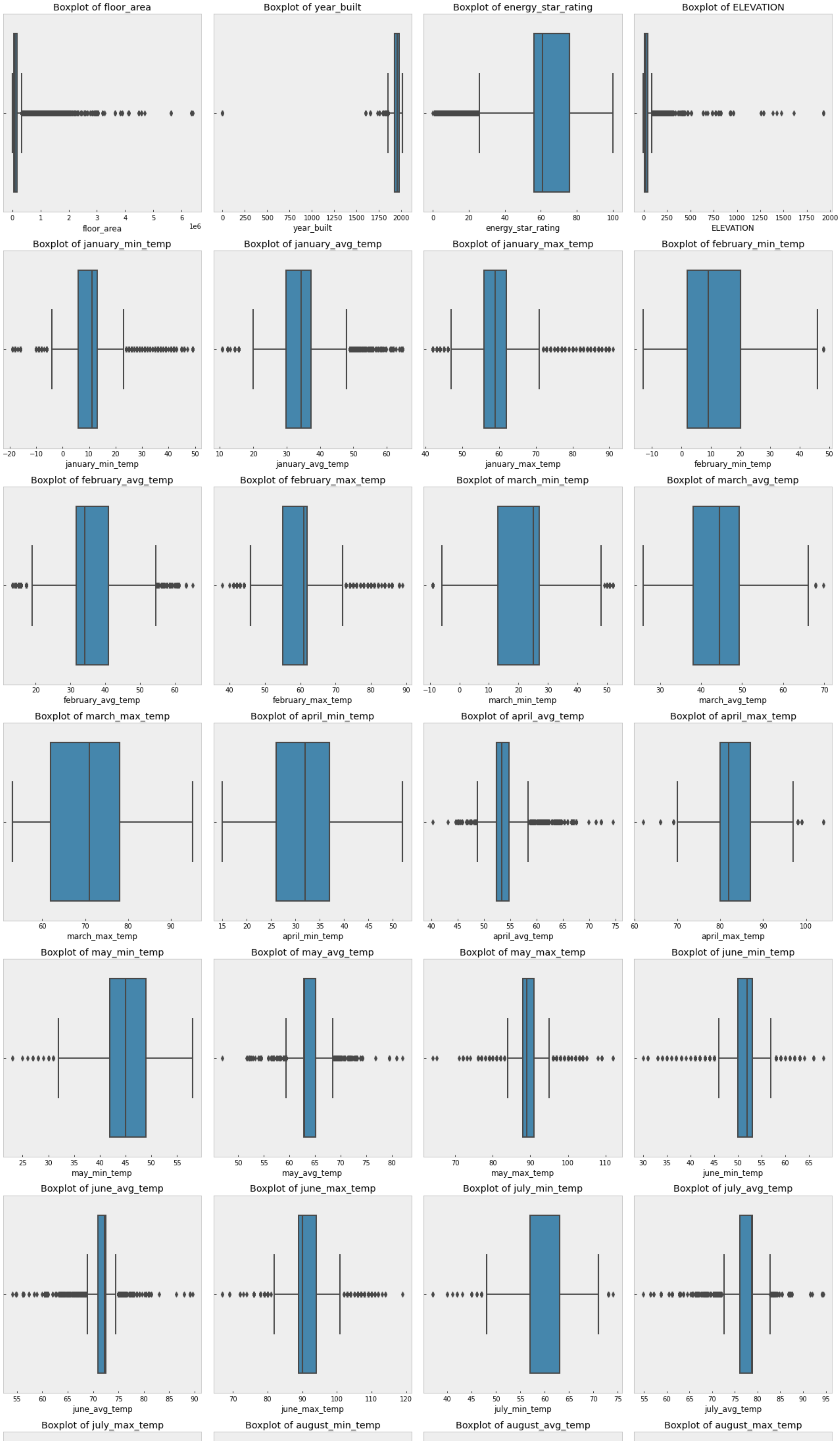
```
 <AxesSubplot:title={'center':'may_max_temp'}>,
 <AxesSubplot:title={'center':'june_min_temp'}>],
[<AxesSubplot:title={'center':'june_avg_temp'}>,
 <AxesSubplot:title={'center':'june_max_temp'}>,
 <AxesSubplot:title={'center':'july_min_temp'}>,
 <AxesSubplot:title={'center':'july_avg_temp'}>,
 <AxesSubplot:title={'center':'july_max_temp'}>,
 <AxesSubplot:title={'center':'august_min_temp'}>,
 <AxesSubplot:title={'center':'august_avg_temp'}>],
[<AxesSubplot:title={'center':'august_max_temp'}>,
 <AxesSubplot:title={'center':'september_min_temp'}>,
 <AxesSubplot:title={'center':'september_avg_temp'}>,
 <AxesSubplot:title={'center':'september_max_temp'}>,
 <AxesSubplot:title={'center':'october_min_temp'}>,
 <AxesSubplot:title={'center':'october_avg_temp'}>,
 <AxesSubplot:title={'center':'october_max_temp'}>],
[<AxesSubplot:title={'center':'november_min_temp'}>,
 <AxesSubplot:title={'center':'november_avg_temp'}>,
 <AxesSubplot:title={'center':'november_max_temp'}>,
 <AxesSubplot:title={'center':'december_min_temp'}>,
 <AxesSubplot:title={'center':'december_avg_temp'}>,
 <AxesSubplot:title={'center':'december_max_temp'}>,
 <AxesSubplot:title={'center':'cooling_degree_days'}>],
[<AxesSubplot:title={'center':'heating_degree_days'}>,
 <AxesSubplot:title={'center':'precipitation_inches'}>,
 <AxesSubplot:title={'center':'snowfall_inches'}>,
 <AxesSubplot:title={'center':'snowdepth_inches'}>,
 <AxesSubplot:title={'center':'avg_temp'}>,
 <AxesSubplot:title={'center':'days_below_30F'}>,
 <AxesSubplot:title={'center':'days_below_20F'}>],
[<AxesSubplot:title={'center':'days_below_10F'}>,
 <AxesSubplot:title={'center':'days_below_0F'}>,
 <AxesSubplot:title={'center':'days_above_80F'}>,
 <AxesSubplot:title={'center':'days_above_90F'}>,
 <AxesSubplot:title={'center':'days_above_100F'}>,
 <AxesSubplot:title={'center':'days_above_110F'}>,
 <AxesSubplot:title={'center':'site_eui'}>]], dtype=object)
```

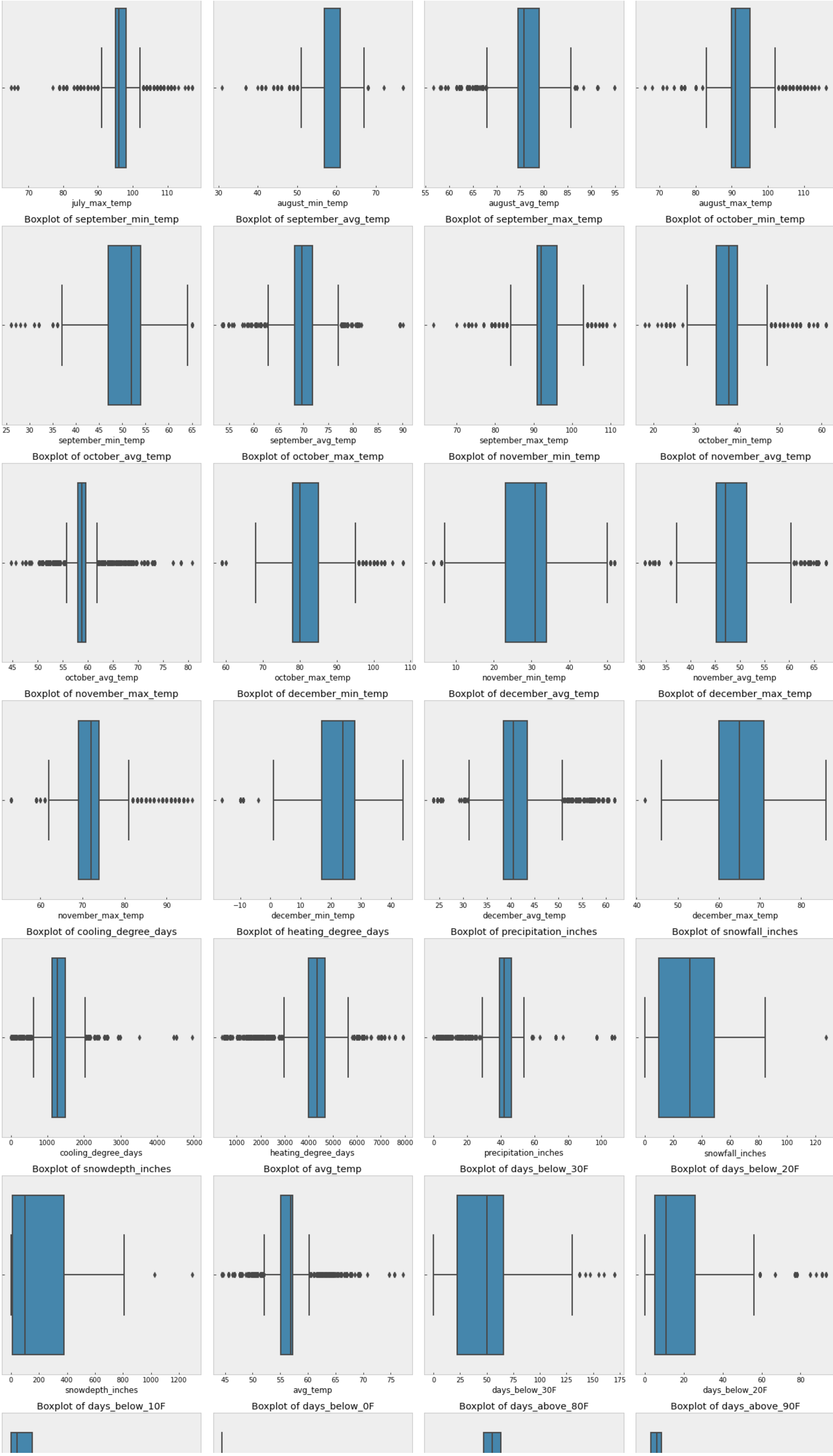


Boxplot to check outlaiers

In [30]:

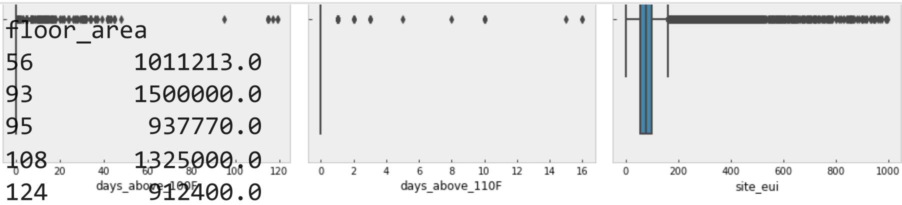
```
import matplotlib.pyplot as plt
import seaborn as sns
import math

num_features_count = len(Num_features)
num_rows = math.ceil(num_features_count / 4)
fig, axes = plt.subplots(nrows=num_rows, ncols=4, figsize=(18, 5*num_rows))

for i, column in enumerate(Num_features):
    row = i // 4
    col = i % 4
    sns.boxplot(x=energy_df[column], ax=axes[row, col])
    axes[row, col].set_title(f'Boxplot of {column}')
    axes[row, col].set_xlabel(column)

for i in range(num_features_count, num_rows * 4):
    row = i // 4
    col = i % 4
    fig.delaxes(axes[row, col])

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


Boxplot of floor_area
floor_area
1e6
Boxplot of year_built
year_built
Boxplot of energy_star_rating
energy_star_rating
Boxplot of ELEVATION
ELEVATION
Boxplot of january_min_temp
january_min_temp
Boxplot of january_avg_temp
january_avg_temp
Boxplot of january_max_temp
january_max_temp
Boxplot of february_min_temp
february_min_temp
Boxplot of february_avg_temp
february_avg_temp
Boxplot of february_max_temp
february_max_temp
Boxplot of march_min_temp
march_min_temp
Boxplot of march_avg_temp
march_avg_temp
Boxplot of march_max_temp
march_max_temp
Boxplot of april_min_temp
april_min_temp
Boxplot of april_avg_temp
april_avg_temp
Boxplot of april_max_temp
april_max_temp
Boxplot of may_min_temp
may_min_temp
Boxplot of may_avg_temp
may_avg_temp
Boxplot of may_max_temp
may_max_temp
Boxplot of june_min_temp
june_min_temp
Boxplot of june_avg_temp
june_avg_temp
Boxplot of june_max_temp
june_max_temp
Boxplot of july_min_temp
july_min_temp
Boxplot of july_avg_temp
july_avg_temp
Boxplot of july_max_temp
Boxplot of august_min_temp
Boxplot of august_avg_temp
Boxplot of august_max_temp


july_max_temp
august_min_temp
august_avg_temp
august_max_temp
Boxplot of september_min_temp
Boxplot of september_avg_temp
Boxplot of september_max_temp
Boxplot of october_min_temp
september_min_temp
september_avg_temp
september_max_temp
october_min_temp
Boxplot of october_avg_temp
Boxplot of october_max_temp
Boxplot of november_min_temp
Boxplot of november_avg_temp
october_avg_temp
october_max_temp
november_min_temp
november_avg_temp
Boxplot of november_max_temp
Boxplot of december_min_temp
Boxplot of december_avg_temp
Boxplot of december_max_temp
november_max_temp
december_min_temp
december_avg_temp
december_max_temp
Boxplot of cooling_degree_days
Boxplot of heating_degree_days
Boxplot of precipitation_inches
Boxplot of snowfall_inches
cooling_degree_days
heating_degree_days
precipitation_inches
snowfall_inches
Boxplot of snowdepth_inches
Boxplot of avg_temp
Boxplot of days_below_30F
Boxplot of days_below_20F
snowdepth_inches
avg_temp
days_below_30F
days_below_20F
Boxplot of days_below_10F
Boxplot of days_below_0F
Boxplot of days_above_80F
Boxplot of days_above_90F

In [31]:

```python
""" checking the number of outlaiers present for each feature using zscore"""
import numpy as np
import scipy.stats


for i in Num_features:
    print(i)
    z = np.abs(scipy.stats.zscore(energy_df[i]))
    outliers = energy_df[z > 3][i]
    print(outliers)
```




```
floor_area
56        1011213.0
93        1500000.0
95         937770.0
108       1325000.0
124        912400.0
            ...
73574     1592914.0
73654      970647.0
73683      962428.0
73729     1765970.0
73775     2200000.0
Name: floor_area, Length: 1516, dtype: float64
year_built
353          0.0
955          0.0
2159         0.0
3415         0.0
4535         0.0
5571         0.0
6931      1789.0
8411      1789.0
9348      1829.0
9907      1789.0
15123     1600.0
16936     1836.0
19948     1600.0
24302     1649.0
26477     1827.0
26790     1836.0
26876     1600.0
27251     1649.0
35003     1827.0
35472     1600.0
35896     1649.0
44412     1827.0
44853     1836.0
44951     1600.0
55459     1827.0
55686     1800.0
56141     1836.0
56260     1600.0
56441     1811.0
56882     1649.0
57581     1833.0
59640     1800.0
61605     1799.0
64737     1841.0
65735     1818.0
65907     1756.0
66037     1800.0
```

```
66550    1818.0
66716    1756.0
66851    1800.0
67375    1818.0
67560    1756.0
67713    1800.0
68337    1732.0
68427    1815.0
68592    1756.0
68796    1800.0
Name: year_built, dtype: float64
energy_star_rating
Series([], Name: energy_star_rating, dtype: float64)
ELEVATION
377       958.6
378       958.6
379       958.6
380       958.6
383      1380.7
          ...
75617     313.0
75618     313.0
75619     313.0
75620     313.0
75621     313.0
Name: ELEVATION, Length: 875, dtype: float64
january_min_temp
358      45
359      45
360      45
361      45
362      45
         ..
10716   -18
10717   -18
10718   -18
10719   -18
10720   -18
Name: january_min_temp, Length: 1016, dtype: int64
january_avg_temp
358      64.274194
359      64.274194
360      64.274194
361      64.274194
362      64.274194
           ...
10716    12.258065
10717    12.258065
10718    12.258065
10719    12.258065
10720    12.258065
Name: january_avg_temp, Length: 229, dtype: float64
january_max_temp
358      89
359      89
360      89
361      89
362      89
         ..
10873    42
10874    42
10875    42
10876    42
10877    42
```

```
Name: january_max_temp, Length: 1450, dtype: int64
february_min_temp
Series([], Name: february_min_temp, dtype: int64)
february_avg_temp
2280    63.339286
2281    63.339286
2293    63.339286
2294    63.339286
2295    63.339286
2301    63.339286
2316    65.107143
Name: february_avg_temp, dtype: float64
february_max_temp
377     86
378     86
381     85
1061    86
1066    86
        ..
2374    84
2375    84
2377    84
2378    84
2388    84
Name: february_max_temp, Length: 100, dtype: int64
march_min_temp
2282     52
2283     52
2284     52
2285     52
2286     52
         ..
12164    -9
12165    -9
12166    -9
12167    -9
12168    -9
Name: march_min_temp, Length: 204, dtype: int64
march_avg_temp
1061    64.548387
1066    64.548387
1067    64.548387
1068    66.096774
1069    64.548387
1070    64.548387
1071    64.548387
1072    64.548387
1073    64.548387
1074    64.548387
1076    64.548387
1077    64.548387
1078    64.548387
1079    64.548387
1087    64.548387
1088    64.548387
1089    64.548387
1090    64.548387
1091    64.548387
1092    64.548387
1093    64.548387
1094    64.548387
1095    64.548387
1110    64.548387
1111    64.548387
```

```
1112    64.548387
1113    64.548387
1114    64.548387
1115    64.548387
1116    64.548387
1117    64.548387
1121    64.548387
1124    64.548387
1127    64.548387
2280    67.854839
2281    67.854839
2293    67.854839
2294    67.854839
2295    67.854839
2301    67.854839
2316    69.758065
Name: march_avg_temp, dtype: float64
march_max_temp
381     95
1068    95
2280    94
2281    94
2293    94
2294    94
2295    94
2301    94
Name: march_max_temp, dtype: int64
april_min_temp
367     49
368     49
369     49
370     49
371     49
        ..
2289    52
2290    52
2291    52
2292    52
2404    15
Name: april_min_temp, Length: 70, dtype: int64
april_avg_temp
377      71.316667
378      71.316667
381      65.316667
382      62.379310
384      62.379310
           ...
71585    45.083333
71837    45.083333
72532    45.083333
72533    45.083333
72534    45.083333
Name: april_avg_temp, Length: 682, dtype: float64
april_max_temp
377      104
378      104
382       99
384       99
385       99
        ...
2434      69
2435      69
2436      69
3540      69
```

```
42199     70
Name: april_max_temp, Length: 110, dtype: int64
may_min_temp
398      28
1021     32
1038     58
1039     58
1040     58
         ..
10704    32
10705    32
10706    32
10707    32
10712    32
Name: may_min_temp, Length: 80, dtype: int64
may_avg_temp
377      80.903226
378      80.903226
397      72.322581
398      51.661290
399      72.322581
           ...
74808    52.145161
75061    53.887097
75754    52.145161
75755    52.145161
75756    53.887097
Name: may_avg_temp, Length: 104, dtype: float64
may_max_temp
358      76
359      76
360      76
361      76
362      76
         ..
74808    79
75061    80
75754    79
75755    79
75756    80
Name: may_max_temp, Length: 295, dtype: int64
june_min_temp
398      33
999      39
1021     34
1030     36
1031     36
         ..
3503     38
3515     38
3532     31
3539     34
42199    40
Name: june_min_temp, Length: 81, dtype: int64
june_avg_temp
0        60.500000
1        60.500000
2        60.500000
3        60.500000
4        60.500000
           ...
74808    56.233333
75061    58.433333
75754    56.233333
```

```
75755    56.233333
75756    58.433333
Name: june_avg_temp, Length: 475, dtype: float64
june_max_temp
358      76
359      76
360      76
361      76
362      76
       ...
4577    106
4578    106
4579    106
4580    106
4581    106
Name: june_max_temp, Length: 2270, dtype: int64
july_min_temp
398      43
999      46
1012     45
1013     45
1014     45
         ..
74808    48
75061    48
75754    48
75755    48
75756    48
Name: july_min_temp, Length: 776, dtype: int64
july_avg_temp
0        62.725806
1        62.725806
2        62.725806
3        62.725806
4        62.725806
           ...
74808    58.758065
75061    60.532258
75754    58.758065
75755    58.758065
75756    60.532258
Name: july_avg_temp, Length: 1464, dtype: float64
july_max_temp
367      81
368      81
369      81
370      81
371      81
         ..
74808    81
75061    83
75754    81
75755    81
75756    83
Name: july_max_temp, Length: 1210, dtype: int64
august_min_temp
377      77
378      77
398      44
999      45
1021     45
         ..
68167    41
68168    41
```

```
68169    41
68170    41
68171    41
Name: august_min_temp, Length: 1575, dtype: int64
august_avg_temp
0        62.161290
1        62.161290
2        62.161290
3        62.161290
4        62.161290
           ...
74806    61.612903
74807    61.612903
74808    61.612903
75754    61.612903
75755    61.612903
Name: august_avg_temp, Length: 1465, dtype: float64
august_max_temp
377      116
378      116
379      105
380      105
382      108
        ...
5614      76
5615      76
5616      76
5617      76
33196     80
Name: august_max_temp, Length: 2137, dtype: int64
september_min_temp
367      65
368      65
369      65
370      65
371      65
372      65
373      65
374      65
375      65
377      65
378      65
1119     27
1223     31
2244     32
2245     32
2249     32
2250     32
2258     36
2259     36
2266     36
2267     36
2270     29
2272     26
2273     26
2282     64
2283     64
2284     64
2285     64
2286     64
2287     64
2288     64
2289     64
2290     64
```

```
2291     64
2292     64
2316     65
2404     31
2431     35
2434     35
2435     35
2436     35
3532     28
3539     36
65627    37
65628    37
69347    37
69354    37
70058    37
70742    37
70743    37
71274    37
71583    37
71584    37
71585    37
71837    37
72532    37
72533    37
72534    37
Name: september_min_temp, dtype: int64
september_avg_temp
377      89.550000
378      89.550000
381      80.500000
388      80.950000
389      80.950000
           ...
74808    53.783333
75061    55.931034
75754    53.783333
75755    53.783333
75756    55.931034
Name: september_avg_temp, Length: 71, dtype: float64
september_max_temp
377      109
378      109
381      108
1016      64
1068     108
2277      74
2278      74
2279      70
2280     109
2281     109
2293     109
2294     109
2295     109
2301     109
2306     108
2307     108
2316     111
2319     108
2320     108
2365     108
2368     108
2369     108
2370     108
2393      77
```

```
2394     77
2395     77
2399     77
2401     77
2402     77
2403     77
2405     77
2406     77
3540     72
72538    73
72544    73
72545    73
73255    73
73946    73
73947    73
74482    73
74796    73
74797    73
74798    73
74799    73
74800    73
74801    73
74802    73
74803    73
74804    73
74805    73
74806    73
74807    73
74808    73
75061    75
75754    73
75755    73
75756    75
Name: september_max_temp, dtype: int64
october_min_temp
367      54
368      54
369      54
370      54
371      54
         ..
3459     55
3460     55
3461     55
3462     55
52611    21
Name: october_min_temp, Length: 1014, dtype: int64
october_avg_temp
358      69.774194
359      69.774194
360      69.774194
361      69.774194
362      69.774194
           ...
74808    47.661290
75061    48.532258
75754    47.661290
75755    47.661290
75756    48.532258
Name: october_avg_temp, Length: 2196, dtype: float64
october_max_temp
377      108
378      108
381      105
```

```
382      108
384      108
        ...
74808     59
75061     60
75754     59
75755     59
75756     60
Name: october_max_temp, Length: 761, dtype: int64
november_min_temp
1038     51
1039     51
1040     51
1041     51
1042     51
         ..
10772     6
10773     6
10774     6
10775     6
10776     6
Name: november_min_temp, Length: 307, dtype: int64
november_avg_temp
358      63.016667
359      63.016667
360      63.016667
361      63.016667
362      63.016667
           ...
10716    31.716667
10717    31.716667
10718    31.716667
10719    31.716667
10720    31.716667
Name: november_avg_temp, Length: 423, dtype: float64
november_max_temp
358      90
359      90
360      90
361      90
362      90
         ..
71585    53
71837    53
72532    53
72533    53
72534    53
Name: november_max_temp, Length: 222, dtype: int64
december_min_temp
2272    -16
2273    -16
12177   -10
12178   -10
12179   -10
         ..
14784    -9
14785    -9
14786    -9
14787    -9
14788   -10
Name: december_min_temp, Length: 2614, dtype: int64
december_avg_temp
1038     59.387097
1039     59.387097
```

```
1040     59.387097
1041     59.387097
1042     59.387097
           ...
14719    23.790323
14720    23.790323
14721    23.790323
14722    23.790323
14788    23.790323
Name: december_avg_temp, Length: 401, dtype: float64
december_max_temp
1061     85
1066     85
1067     85
1069     85
1070     85
         ..
74808    46
75061    42
75754    46
75755    46
75756    42
Name: december_max_temp, Length: 134, dtype: int64
cooling_degree_days
377     4453
378     4453
388     2579
389     2579
393     2579
        ...
3533       4
3536       4
3537       4
3538       4
3540       1
Name: cooling_degree_days, Length: 62, dtype: int64
heating_degree_days
358      1125
359      1125
360      1125
361      1125
362      1125
         ...
10717    7580
10718    7580
10719    7580
10720    7580
42465    6933
Name: heating_degree_days, Length: 2599, dtype: int64
precipitation_inches
358       10.43
359       10.43
360       10.43
361       10.43
362       10.43
          ...
74808    106.32
75061    107.69
75754    106.32
75755    106.32
75756    107.69
Name: precipitation_inches, Length: 2453, dtype: float64
snowfall_inches
398      84.8
```

```
2404    127.3
Name: snowfall_inches, dtype: float64
snowdepth_inches
2270     1023
2404     1292
52611     807
Name: snowdepth_inches, dtype: int64
avg_temp
358      64.251366
359      64.251366
360      64.251366
361      64.251366
362      64.251366
           ...
74808    47.911202
75061    49.127397
75754    47.911202
75755    47.911202
75756    49.127397
Name: avg_temp, Length: 1858, dtype: float64
days_below_30F
1119     147
2270     155
2272     170
2273     170
2404     160
3532     143
10696    137
10697    137
10704    137
10705    137
10706    137
10707    137
10712    137
Name: days_below_30F, dtype: int64
days_below_20F
2272     85
2273     85
10489    78
10490    78
10491    78
         ..
10718    91
10719    91
10720    91
42465    67
52611    67
Name: days_below_20F, Length: 236, dtype: int64
days_below_10F
2272     29
2273     29
10489    45
10490    45
10491    45
         ..
68167    30
68168    30
68169    30
68170    30
68171    30
Name: days_below_10F, Length: 2536, dtype: int64
days_below_0F
2272     22
2273     22
```

```
10489    25
10490    25
10491    25
         ..
14720    12
14721    12
14722    12
14788    12
52611    12
Name: days_below_0F, Length: 1980, dtype: int64
days_above_80F
377      246
378      246
379      176
380      176
381      162
        ...
74806      6
74807      6
74808      6
75754      6
75755      6
Name: days_above_80F, Length: 270, dtype: int64
days_above_90F
377      182
378      182
379      113
380      113
381       54
        ...
4581      52
69341     52
69342     52
69343     52
69344     52
Name: days_above_90F, Length: 1285, dtype: int64
days_above_100F
377     119
378     119
379      12
380      12
382      30
       ...
4577     10
4578     10
4579     10
4580     10
4581     10
Name: days_above_100F, Length: 1302, dtype: int64
days_above_110F
377     16
378     16
394      1
395      1
396      1
        ..
2301    10
2316    15
3531     2
3534     2
3535     2
Name: days_above_110F, Length: 61, dtype: int64
site_eui
13       608.839519
```

```
24       287.863448
26       264.068722
113      275.977289
144      285.862933
            ...
75376    268.596672
75424    364.958302
75456    268.380928
75520    275.649545
75755    592.022750
Name: site_eui, Length: 1216, dtype: float64
```

Dropping Columns Based on Box Plot Observations

Upon visual inspection using box plots, it's evident that the columns 'days_above_110F' and 'days_above_100F' and 'days_below_0F' predominantly contain zero values for most data points.

Therefore, considering the lack of variability and information in these columns, I have decided to drop both 'days_above_110F' and 'days_above_100F' and 'days_below_0F' from the dataset.

In [32]:

```
energy_df.drop(['days_above_110F', 'days_above_100F','days_below_0F'], axis=1, inpla
```

In [33]:

```
energy_df.shape
```

Out[33]:

```
(75757, 56)
```

During the analysis, numerous outliers were detected across all features. However, the presence of a large number of outliers does not necessarily indicate their irrelevance.

For this analysis, I have decided not to remove outliers as their presence might hold valuable information or characteristics within the dataset. Retaining outliers can contribute to a more comprehensive understanding of the data and potentially enhance the performance of the models.

In [ ]:

```

```

In [34]:

```
import pandas as pd

# Assuming energy_df_id_remove_2 is your DataFrame and features is a list of column 
features = energy_df.columns.tolist()

Num_features = [feature for feature in features if
                energy_df[feature].dtype != 'object' and
                feature != 'Year_Factor']
Cat_features = [feature for feature in features if
                energy_df[feature].dtype == object or feature == 'Year_Factor' ]


print(Num_features)
print("\n")

print(Cat_features)
```

```
['floor_area', 'year_built', 'energy_star_rating', 'ELEVATION', 'january_min_temp',
'january_avg_temp', 'january_max_temp', 'february_min_temp', 'february_avg_temp', 'fe
bruary_max_temp', 'march_min_temp', 'march_avg_temp', 'march_max_temp', 'april_min_te
```

```
mp', 'april_avg_temp', 'april_max_temp', 'may_min_temp', 'may_avg_temp', 'may_max_tem
p', 'june_min_temp', 'june_avg_temp', 'june_max_temp', 'july_min_temp', 'july_avg_tem
p', 'july_max_temp', 'august_min_temp', 'august_avg_temp', 'august_max_temp', 'septem
ber_min_temp', 'september_avg_temp', 'september_max_temp', 'october_min_temp', 'octob
er_avg_temp', 'october_max_temp', 'november_min_temp', 'november_avg_temp', 'november
_max_temp', 'december_min_temp', 'december_avg_temp', 'december_max_temp', 'cooling_d
egree_days', 'heating_degree_days', 'precipitation_inches', 'snowfall_inches', 'snowd
epth_inches', 'avg_temp', 'days_below_30F', 'days_below_20F', 'days_below_10F', 'days
_above_80F', 'days_above_90F', 'site_eui']


['Year_Factor', 'State_Factor', 'building_class', 'facility_type']
```

In [ ]:

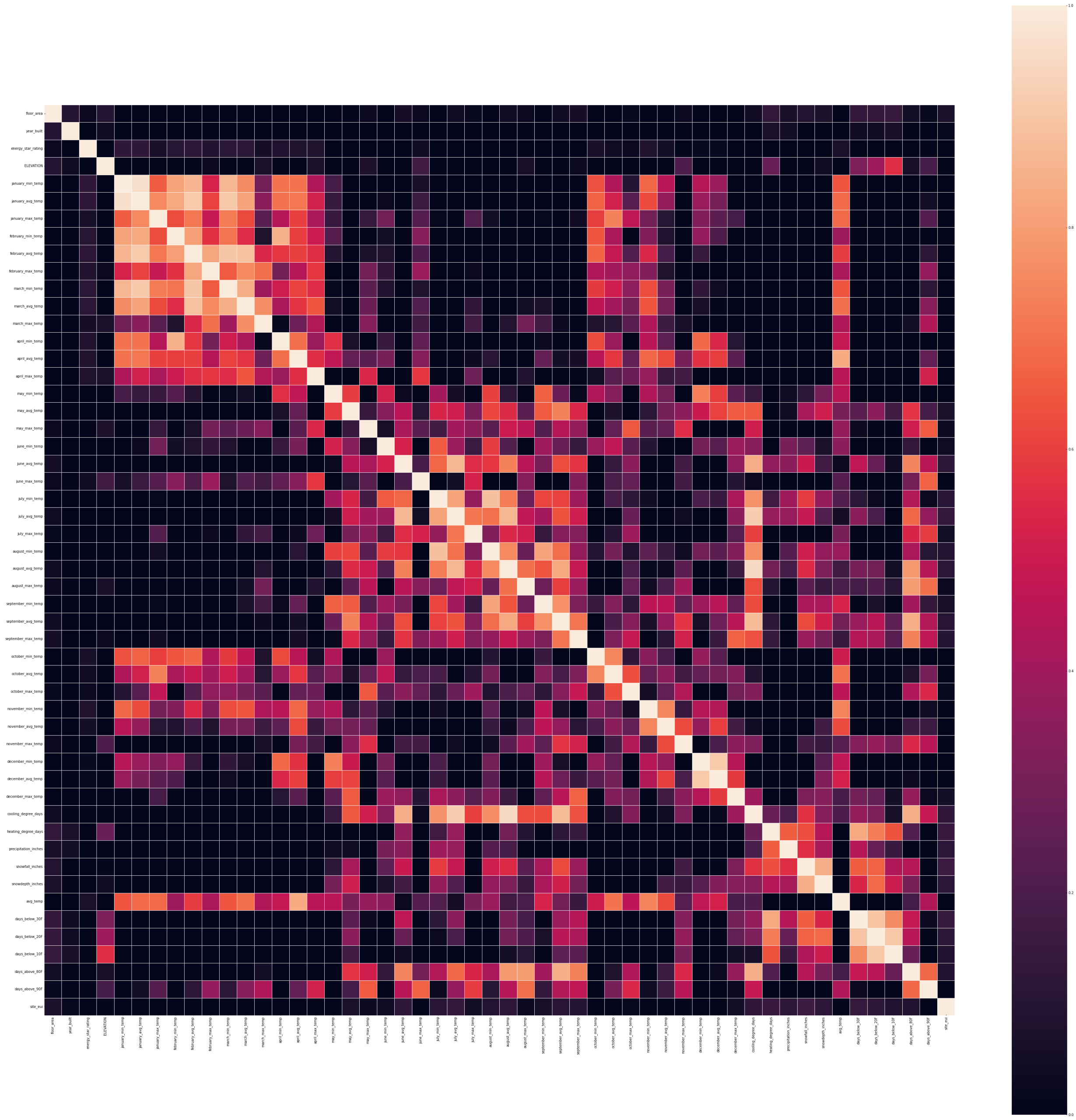
Correlation matrix to check highly correlated features

In [35]:

```
corr = energy_df[Num_features].corr()
plt.subplots(figsize=(60, 60))
sns.heatmap(corr, linewidths=.5, vmin=0, vmax=1, square=True)
```

Out[35]:

```
<AxesSubplot:>
```



I will remove highly_correlated features(threshold = 0.9)

In [36]:

```
# Assuming 'corr' is the correlation matrix calculated from Num_features

threshold = 0.9  # Set your desired threshold for correlation

# Create a mask to focus only on the upper triangle of the correlation matrix (to av
mask = np.triu(np.ones_like(corr, dtype=bool), k=1)

# Find columns with correlation above the threshold
highly_correlated = set()
for i in range(len(corr.columns)):
    for j in range(i+1, len(corr.columns)):
        if mask[i, j] and abs(corr.iloc[i, j]) > threshold:
            col_i = corr.columns[i]
            col_j = corr.columns[j]
            highly_correlated.add(col_i)
#             highly_correlated.add(col_j)

print("Columns highly correlated:", highly_correlated)
```

```
Columns highly correlated: {'january_min_temp', 'february_avg_temp', 'december_min_te
mp', 'days_below_20F', 'january_avg_temp', 'august_avg_temp', 'july_avg_temp'}
```

In [37]:
```
len(highly_correlated)
```

Out[37]: 7

In [38]:
```
"""There are 6 numerical features are highly correlated with other data"""
"""So I am removing those 6 columns """

energy_df.drop(highly_correlated, inplace=True, axis=1)
```

In [39]:
```
energy_df.shape
```

Out[39]: (75757, 49)

Numerical features and Categorical features

In [40]:
```
import pandas as pd

# Assuming energy_df_id_remove_2 is your DataFrame and features is a list of column
features = energy_df.columns.tolist()

Num_features = [feature for feature in features if
                energy_df[feature].dtype != 'object' and
                feature != 'Year_Factor' and feature !='site_eui']
Cat_features = [feature for feature in features if
                energy_df[feature].dtype == object or feature == 'Year_Factor' ]


print(Num_features)
print("\n")

print(Cat_features)
```

```
['floor_area', 'year_built', 'energy_star_rating', 'ELEVATION', 'january_max_temp', 'february_min_temp', 'february_max_temp', 'march_min_temp', 'march_avg_temp', 'march_max_temp', 'april_min_temp', 'april_avg_temp', 'april_max_temp', 'may_min_temp', 'may_avg_temp', 'may_max_temp', 'june_min_temp', 'june_avg_temp', 'june_max_temp', 'july_min_temp', 'july_max_temp', 'august_min_temp', 'august_max_temp', 'september_min_temp', 'september_avg_temp', 'september_max_temp', 'october_min_temp', 'october_avg_temp', 'october_max_temp', 'november_min_temp', 'november_avg_temp', 'november_max_temp', 'december_avg_temp', 'december_max_temp', 'cooling_degree_days', 'heating_degree_days', 'precipitation_inches', 'snowfall_inches', 'snowdepth_inches', 'avg_temp', 'days_below_30F', 'days_below_10F', 'days_above_80F', 'days_above_90F']


['Year_Factor', 'State_Factor', 'building_class', 'facility_type']
```

In [ ]:
```

```

Splitting the Data into Train and Test Data

In [41]:
```
X = energy_df.drop(['site_eui'],axis=1)
y = energy_df['site_eui']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size
```

In [42]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape, X.shape, y.shape
```

Out[42]:
```
((60605, 48), (15152, 48), (60605,), (15152,), (75757, 48), (75757,))
```

Plot the decision tree

Data Preprocessing for Decision Tree regressor

For the decision tree Regressor model:

1. **Normalization:** Decision trees do not require normalization as they are not sensitive to the scale of numerical features. Hence, normalization is not necessary for this model.

2. **One-Hot Encoding:** Categorical features need to be one-hot encoded as decision trees typically require categorical variables to be converted into a numerical format for processing.

Therefore, before fitting the data into the decision tree model:

- Apply one-hot encoding to categorical features.
- No need to perform normalization on numerical features.

In [43]:
```
full_pipeline_1 = ColumnTransformer([
#     ('StandardScale', StandardScaler(), Num_features),
    ('onehot', OneHotEncoder(), Cat_features)
],remainder='passthrough')
```

In [44]:
```
pipe1= Pipeline(steps=[('full_pipeline_1',full_pipeline_1),
                        ])
```

In [ ]:
```

```

In [45]:
```
X_train_transformed = pipe1.fit_transform(X_train)
```

In [46]:
```
X_test_transformed = pipe1.transform(X_test)
```

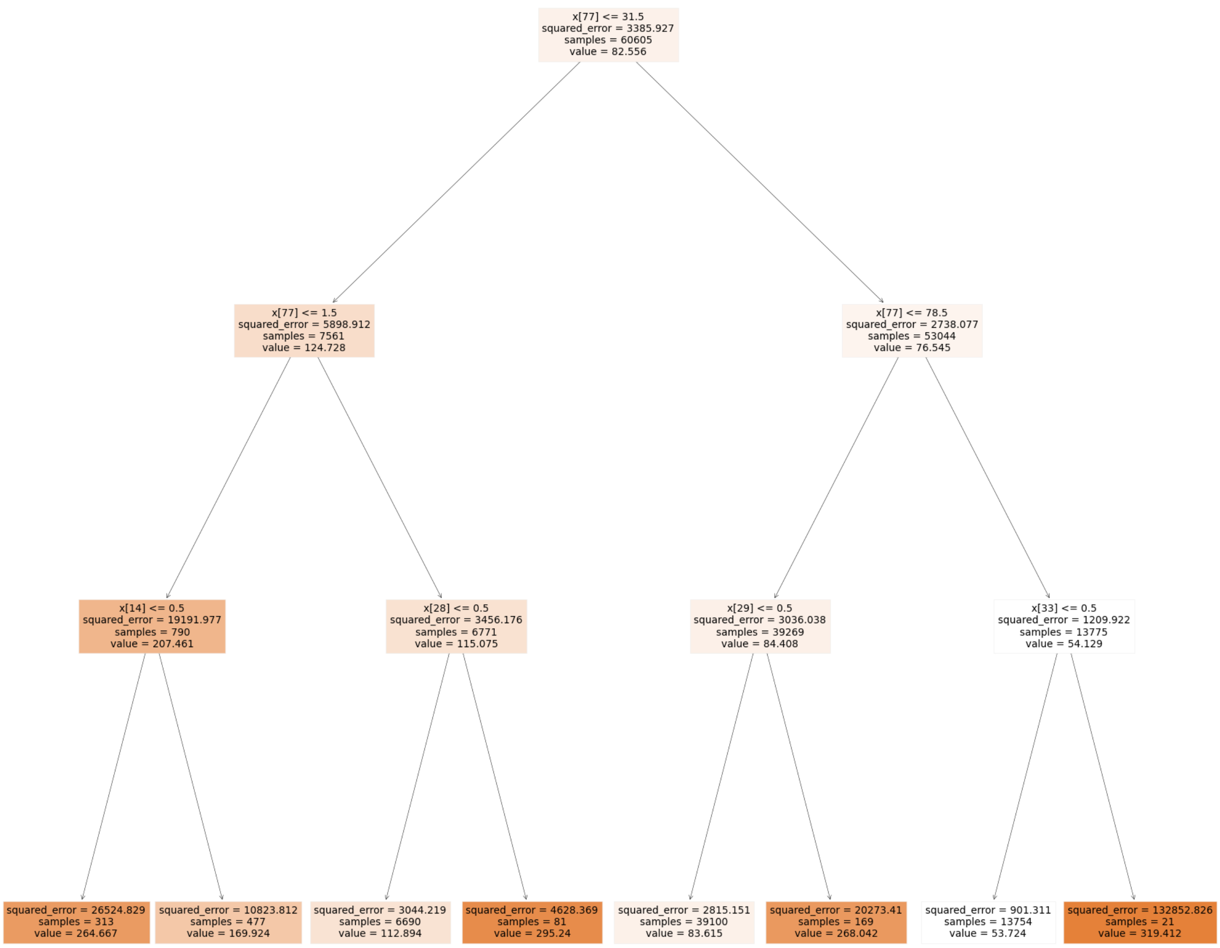
In [47]:
```
# X_train_transformed
```

In [48]:
```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
```

In [ ]:
```

```

In [49]:
```
from sklearn.tree import DecisionTreeRegressor, plot_tree  # Add plot_tree import
import matplotlib.pyplot as plt

def train_using_entropy(X_train, y_train):
```

```
    # Decision tree with entropy
    clf_entropy = DecisionTreeRegressor(max_depth=3, criterion='squared_error')

    # Performing training
    clf_entropy.fit(X_train, y_train)

    plt.figure(figsize=(30, 30))
    plot_tree(clf_entropy, filled=True)  # Use plot_tree from sklearn.tree
    plt.show()

    return clf_entropy


clf_object = train_using_entropy(X_train_transformed, y_train)
```



Here for this Decision tree i have used depth as 3 just to visulize the tree. But for proper data prediction I should choose much higher depth value othervise predcition will be not properly correct

In [50]:

```
def prediction(X_test, clf_object):

    y_pred = clf_object.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print("Mean Squared Error:", mse)
    print("Predicted values:")
```

```
    print(y_pred)
    return y_pred
```

In [51]:
```
y_pred_entropy = prediction(X_test_transformed, clf_object)
```

```
Mean Squared Error: 2764.9598676970622
Predicted values:
[83.6145246  53.72382701 83.6145246  ... 83.6145246  83.6145246
 53.72382701]
```

In [ ]:

Plot the PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique used to reduce the number of features (variables) while retaining the most important information or patterns present in the original dataset.

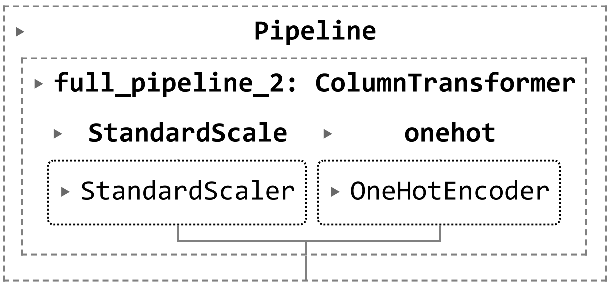
Normalizing the data and handling categorical features are important steps before applying PCA.

In [52]:
```
full_pipeline_2 = ColumnTransformer([
    ('StandardScale', StandardScaler(), Num_features),
    ('onehot', OneHotEncoder(), Cat_features),

])
```

In [53]:
```
pipe2= Pipeline(steps=[('full_pipeline_2',full_pipeline_2),
# ('pca' , PCA(n_components=0.90, svd_solver='full'))

                        ])
```

In [54]:
```
pipe2
```

Out[54]:



In [55]:
```
X_train_transformed2 = pipe2.fit_transform(X_train)
X_test_transformed2  = pipe2.transform(X_test)
```

In [ ]:

In [56]:
```
pca_model = PCA(n_components=0.90, svd_solver='full')
X_train_pca = pca_model.fit_transform(X_train_transformed2)
X_test_pca = pca_model.transform(X_test_transformed2)
```

In [57]:
```
X_train_pca.shape, X_test_pca.shape
```

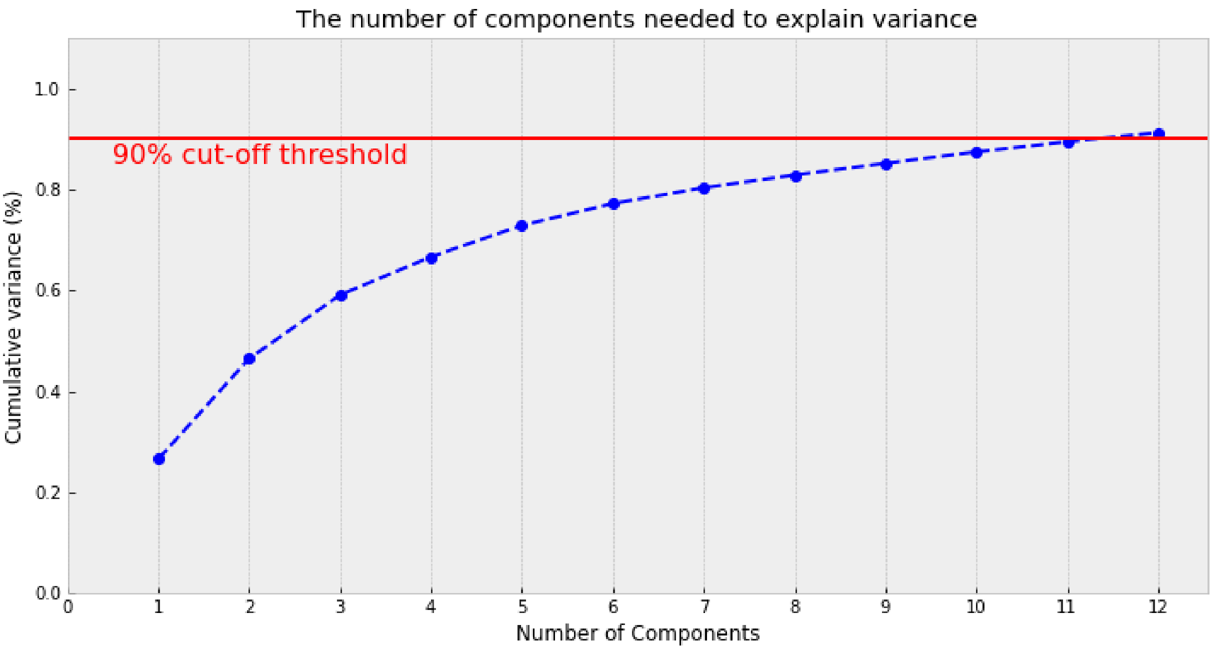
Out[57]:
```
((60605, 12), (15152, 12))
```

In your case, if you've conducted PCA and it resulted in reducing the number of features to 12, it means that these 12 components retain the most relevant information from the original dataset as I am using 90% data relevancy.

In [58]:
```
pca_model.explained_variance_ratio_
```

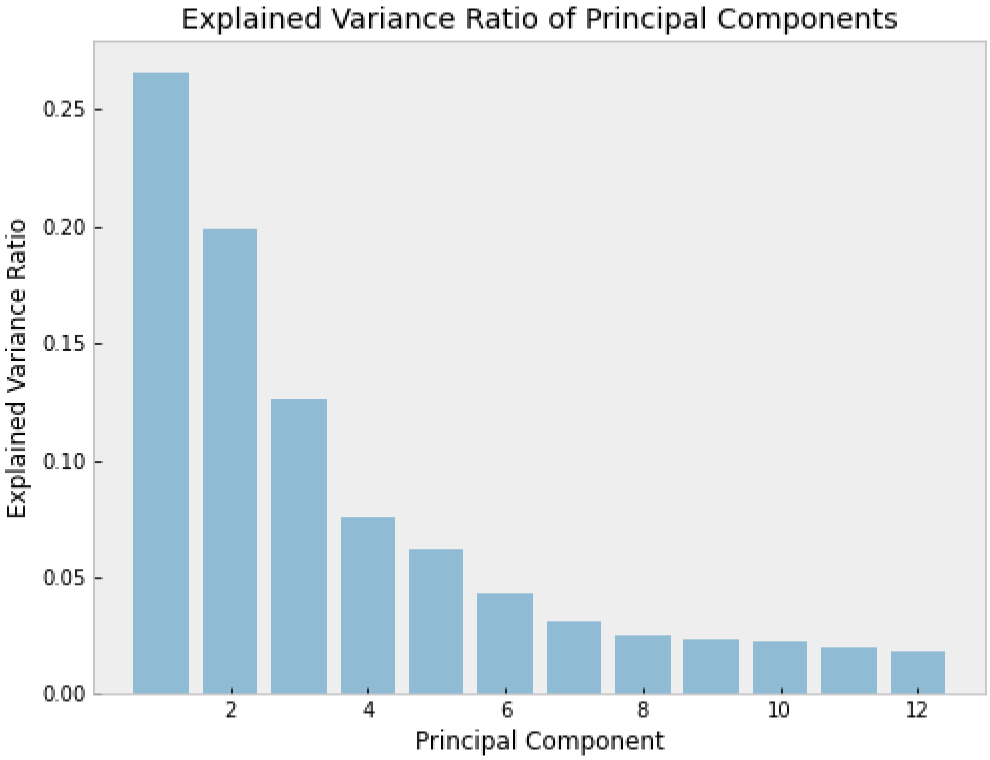
Out[58]:
```
array([0.26600548, 0.19917598, 0.12599467, 0.07552799, 0.06216106,
       0.04343502, 0.03120888, 0.02515717, 0.02320775, 0.02275822,
       0.01984301, 0.01830348])
```

In [59]:
```
# % matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12,6)

fig, ax = plt.subplots()
xi = np.arange(1, 13, step=1)   #  X_train_pca.shape =(60605, 12) thats why  1 am us
y = np.cumsum(pca_model.explained_variance_ratio_)

plt.ylim(0.0,1.1)
plt.plot(xi, y, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 13, step=1)) #change from 0-based array index to 1-based hum
plt.ylabel('Cumulative variance (%)')
plt.title('The number of components needed to explain variance')

plt.axhline(y=0.90, color='r', linestyle='-')
plt.text(0.5, 0.85, '90% cut-off threshold', color = 'red', fontsize=16)

ax.grid(axis='x')
plt.show()
```



In [60]:

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler

# pca_model.plot(figsize=(10,8))
# plt.show()


# Plotting the explained variance ratio
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(pca_model.explained_variance_ratio_) + 1), pca_model.explained_
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio of Principal Components')
plt.show()
```

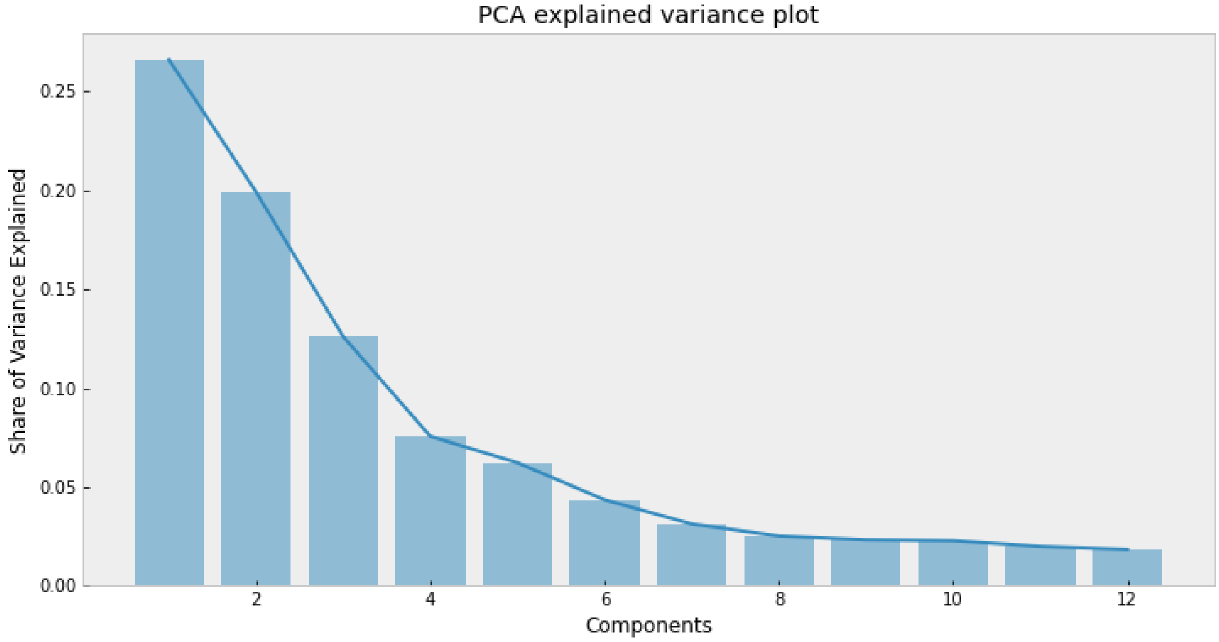


In [61]:

```
explained_variance = pca_model.explained_variance_ratio_
singular_values = pca_model.singular_values_
```

In [62]:

```
x = np.arange(1,len(explained_variance)+1)
plt.bar(range(1, len(pca_model.explained_variance_ratio_) + 1), pca_model.explained_

plt.plot(x, explained_variance)
plt.ylabel('Share of Variance Explained')
plt.title("PCA explained variance plot")
plt.xlabel("Components")
plt.show()
```



In [63]:

```
for i in range(0, 12):
    print(f"Component {i:>2} accounts for {explained_variance[i]*100:>2.2f}% of vari
```

```
Component  0 accounts for 26.60% of variance
Component  1 accounts for 19.92% of variance
Component  2 accounts for 12.60% of variance
Component  3 accounts for 7.55% of variance
Component  4 accounts for 6.22% of variance
Component  5 accounts for 4.34% of variance
Component  6 accounts for 3.12% of variance
Component  7 accounts for 2.52% of variance
Component  8 accounts for 2.32% of variance
Component  9 accounts for 2.28% of variance
Component 10 accounts for 1.98% of variance
Component 11 accounts for 1.83% of variance
```

Choosing the Optimal Number of Principal Components

When working with Principal Component Analysis (PCA) for dimensionality reduction, deciding the number of components is crucial. Here are some methods to determine the optimal number of principal components:

1. **Examining the Knee in Explained Variance Plot:**
    - In our dataset, the explained variance plot exhibits a noticeable "knee" around 4-6 principal components. This knee point can be indicative of the optimal number of components.
2. **Keeping Components Explaining Significant Variance:**
    - Another approach involves retaining components that account for more than 1% of the variance in the dataset. For our data, this threshold occurs after 11 components.
3. **Retaining Components with Cumulative Explained Variance:**
    - Considering the cumulative explained variance, it's beneficial to retain principal components that collectively cover a substantial portion of the total variance. For instance, keeping components that contribute to approximately 80% of the explained variance in the dataset. In our case, this would encompass the first 7 components.

These methods assist in striking a balance between computational efficiency and model performance by selecting an appropriate number of principal components for dimensionality reduction.

From this PCA Components plot we can understadant that n_components= 12 will capture 90% of feature knowledges

LinearRegression model is trained after PCA

In [64]:
```
# Creating the final pipeline with  linear regression model
pipeline = Pipeline(steps=[('regressor', LinearRegression())])
pipeline.fit(X_train_pca, y_train)

# Evaluating the model
train_score = pipeline.score(X_train_pca, y_train)
test_score = pipeline.score(X_test_pca, y_test)

print(f"Training R^2 score: {train_score:.4f}")
print(f"Testing R^2 score: {test_score:.4f}")
```

```
Training R^2 score: 0.1669
Testing R^2 score: 0.1703
```

In [65]:
```
from sklearn.metrics import mean_squared_error
import numpy as np

y_pred = pipeline.predict(X_test_pca)

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


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

```
 Mean Squared Error (MSE): 2841.2050
```

So here I got R^2 score: 0.1703 and Mean Squared Error is 2841 which is very poor performance with simple linear regression.

PLOT THE DATA WITH 2 PCA COMPONENTS

In [66]:
```
pca_2 = PCA(n_components=2, whiten=True)
#fit the model to our data and extract the results
X_pca_2 = pca_2.fit_transform(X_train_transformed2)
```

In [67]:
```
df_pca_plot = pd.DataFrame(data = X_pca_2,
                 columns = ["Component 1",
                            "Component 2"])
```

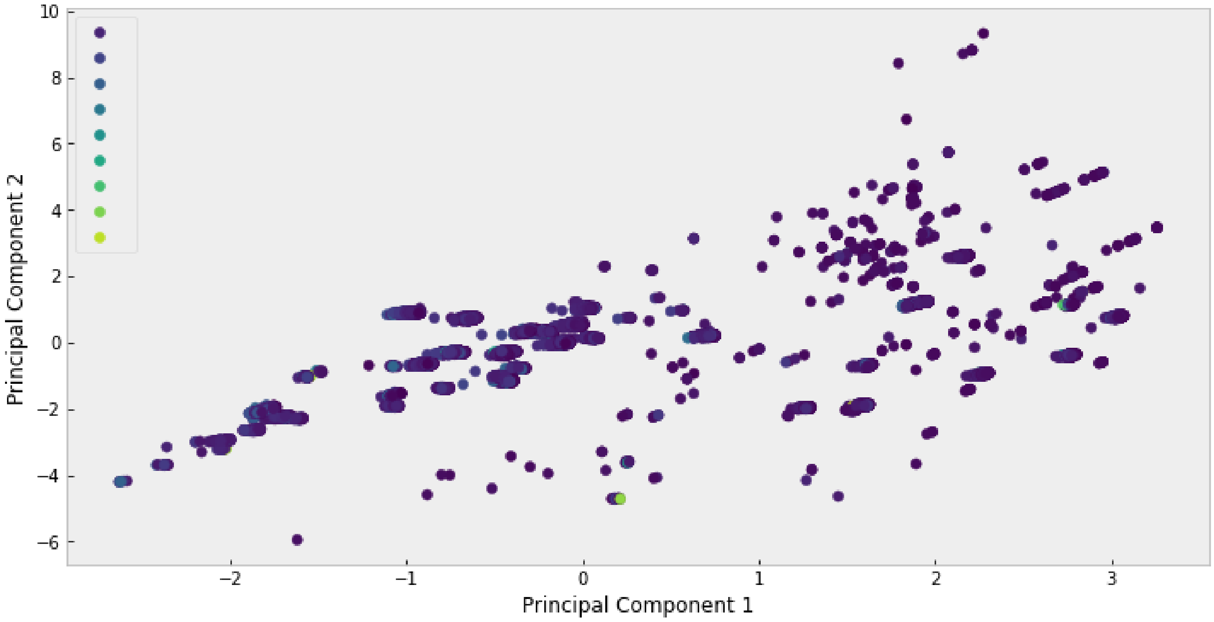
In [68]:
```
df_pca_plot["Component 1"]
```

Out[68]:
```
0       -0.972568
1       -0.884896
2       -0.059776
3       -0.045234
4       -0.697619
```

```
              ...
60600   -0.746157
60601   -0.460480
60602   -0.012080
60603    1.946435
60604   -0.188181
Name: Component 1, Length: 60605, dtype: float64
```

In [69]:

```
#plot the resulting data from two dimensions
plot = plt.scatter(df_pca_plot["Component 1"], df_pca_plot["Component 2"], c=y_train
plt.legend(handles=plot.legend_elements()[0],)
plt.xlabel("Principal Component 1")  # Naming the x-axis
plt.ylabel("Principal Component 2")  # Naming the y-axis
plt.show()
```

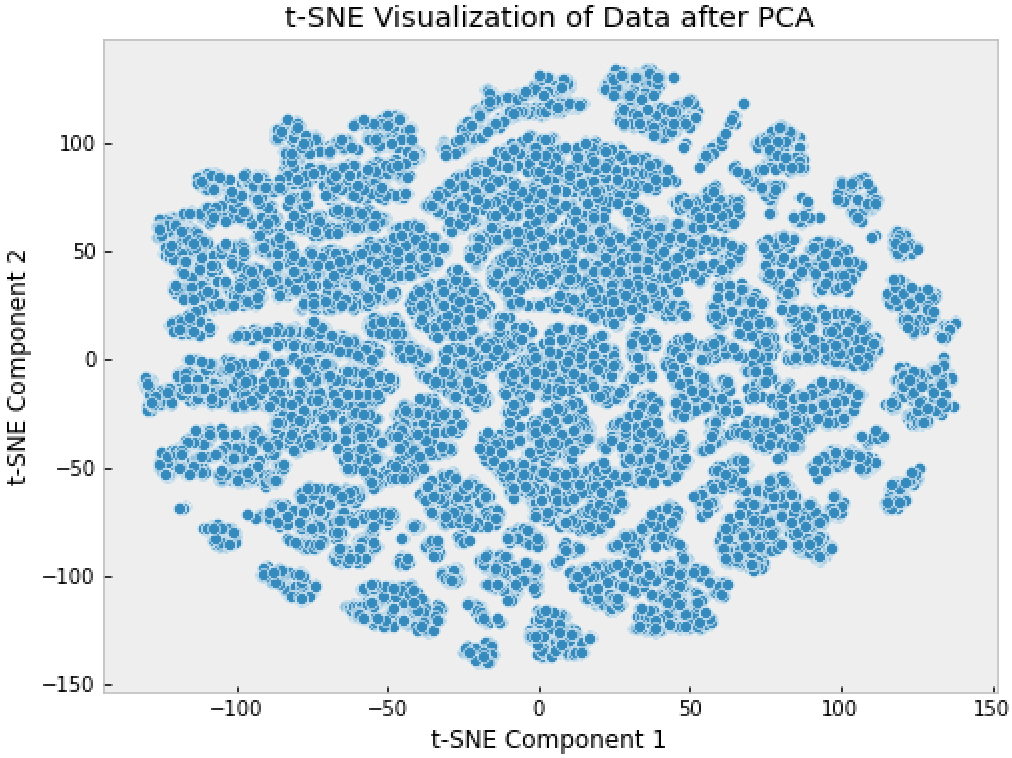


Plot t-SNE on the PCA-transformed data

Dimensionality reduction using PCA (Principal Component Analysis) followed by t-SNE (t-distributed Stochastic Neighbor Embedding) for visualizing the dataset in a 2D space.

In [70]:

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns

# Apply PCA
pca_t = PCA(n_components=2)  # Choose the number of components
X_pca = pca_t.fit_transform(X_train_transformed2)

# Apply t-SNE on the PCA-transformed data
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_pca)

# Plotting the data in 2D
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_tsne[:,0], y=X_tsne[:,1], palette='viridis')
plt.title('t-SNE Visualization of Data after PCA')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



It shows different similar data clusters but as I consider whole training dataset thats why its not able to show that properly. But if I use a smaill part of dataset it will shoe the cluesters properly.

In [ ]:

In [ ]:

Building Pipeline

Using Scikit-Learn's Pipeline for Machine Learning Workflows

Scikit-Learn provides a powerful tool called `Pipeline` that allows you to chain multiple steps together for a machine learning workflow. These steps can include preprocessing, feature selection, and model building.

Benefits of Using Pipelines:

1. **Simplified Workflow:**
    - Pipelines allow you to combine several data processing steps into a single object, making it easier to manage and reproduce the workflow.
2. **Preventing Data Leakage:**
    - Pipelines help in avoiding data leakage by ensuring that preprocessing steps (e.g., scaling, imputation) are applied consistently to training and testing data.
3. **Cross-Validation Handling:**

- It's simpler to perform cross-validation with a pipeline since transformations occur within each fold, preventing data leakage and ensuring a more accurate estimation of model performance.

In [71]:
```
energy_df_pipe = pd.read_csv("C:\\Users\\praba\\Desktop\\uca1\\M1\\ML\\final project
energy_df_pipe.tail(2)
```

Out[71]:

	Year_Factor	State_Factor	building_class	facility_type	floor_area	year_built	energy_star_
75755	6	State_11	Commercial	Commercial_Other	53575.0	2012.0	
75756	6	State_11	Residential	2to4_Unit_Building	23888.0	1974.0	

2 rows × 64 columns

In [72]:
```
X_1 = energy_df_pipe.drop(['site_eui'],axis=1)
y_1 = energy_df_pipe['site_eui']
X_train_pipe, X_test_pipe, y_train_pipe, y_test_pipe = train_test_split(X_1, y_1 , r
```

In [73]:
```
X_train_pipe.shape, y_test_pipe.shape
```

Out[73]:
```
((60605, 63), (15152,))
```

These drop_highly_correlated_columns I found after using coorelation matrix and setting thresold =90

In [74]:
```
drop_highly_correlated_columns =['january_avg_temp', 'december_min_temp', 'august_av
```

In [75]:
```
len(drop_highly_correlated_columns)
```

Out[75]:
```
6
```

In [ ]:
```

```

If a feature contains a vast majority of missing values (for instance, more than 40,000 null values in this case), one common approach is to consider dropping those features from the dataset.
These 4 features contain more tahn 40000 Null Values.

In [76]:
```
drop_high_null_valued_column = ['direction_max_wind_speed','direction_peak_wind_spee
```

all ids are unique so not relavent

In [77]:
```
drop_unique_id =['id']
```

it's evident that the columns 'days_above_110F' and 'days_above_100F' and 'days_below_0F' predominantly contain zero values for most data points. so i am removing

In [78]:
```
drop_ulrelated = ['days_above_110F', 'days_above_100F', 'days_below_0F']
```

Why I am removing those features are explained in data preprocessing step

In [79]:
```
drop_features_1= drop_highly_correlated_columns+drop_high_null_valued_column+drop_un
print(drop_features_1)
print(len(drop_features_1))
```

```
['january_avg_temp', 'december_min_temp', 'august_avg_temp', 'july_avg_temp', 'januar
y_min_temp', 'days_below_20F', 'direction_max_wind_speed', 'direction_peak_wind_spee
d', 'max_wind_speed', 'days_with_fog', 'id', 'days_above_110F', 'days_above_100F', 'd
ays_below_0F']
14
```

In [80]:
```
import pandas as pd

# Assuming energy_df_id_remove_2 is your DataFrame and features is a list of column 
features = X_1.columns.tolist()

Numerical_features_1 = [feature for feature in features if
                X_1[feature].dtype != 'object' and
                feature != 'Year_Factor']
categorical_features_1 = [feature for feature in features if
                X_1[feature].dtype == object or feature == 'Year_Factor' ]


print(len(Numerical_features_1))
# print("\n")

print(len(categorical_features_1))
```

```
59
4
```

In [81]:
```
# Numerical_features_1
```

In [82]:
```
# Numerical_features
```

In [83]:
```
categorical_features_1=['Year_Factor', 'State_Factor', 'building_class', 'facility_t
```

In [84]:
```
common_elements_1 = list(set(Numerical_features_1) & set(drop_features_1))
print(common_elements_1)
```

```
['january_min_temp', 'id', 'days_with_fog', 'july_avg_temp', 'december_min_temp', 'da
ys_below_0F', 'days_above_110F', 'days_below_20F', 'january_avg_temp', 'days_above_10
0F', 'direction_peak_wind_speed', 'direction_max_wind_speed', 'august_avg_temp', 'max
_wind_speed']
```

In [85]:
```
# Remove elements from list1 that are present in list2
Numerical_features_2 = [x for x in Numerical_features_1 if x not in common_elements_

print(len(Numerical_features_2))
```

```
45
```

In [86]:
```
common_elements_2 = list(set(Numerical_features_2) & set(drop_features_1))
print(common_elements_2)
```

```
[]
```

In [87]:
```
X_train_pipe.shape, X_test_pipe.shape, y_train_pipe.shape, y_test_pipe.shape
```

Out[87]:
```
((60605, 63), (15152, 63), (60605,), (15152,))
```

In [88]:
```
common_elements = list(set(Numerical_features_2) & set(drop_features_1))
print(common_elements)
```

```
[]
```

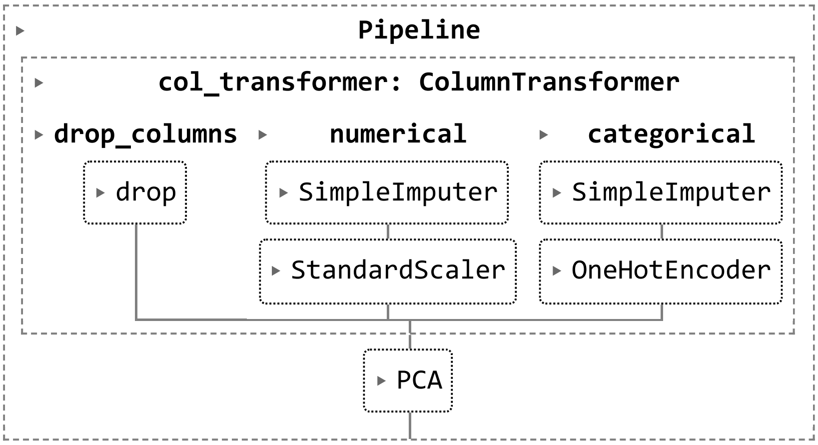
In [89]:
```
"""This is the pipeline To drop specific columns """
drop_transformer = ColumnTransformer(transformers=[('drop_columns','drop',drop_featu
```

In [ ]:
```

```

In [90]:
```
# drop_transformer = ColumnTransformer(transformers=[('drop_columns','drop',drop_fea

# Creating pipelines for numerical and categorical features
numerical_pipeline = Pipeline(steps=[
#     ('outlier_removal', remove_outliers1(remove_outliers)),
    ('imputer', SimpleImputer(strategy='mean')) ,
    ('stdscaler', StandardScaler())
])

categorical_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Filling missing values
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encoding categoric
])

# Creating a ColumnTransformer to apply the appropriate pipeline to each type of fea
col_transformer = ColumnTransformer(transformers=[
   ('drop_columns','drop',drop_features_1),
    ('numerical', numerical_pipeline, Numerical_features_2),
    ('categorical', categorical_pipeline, categorical_features_1),
#     ('scale', StandardScaler())
],remainder='drop')


pipe = Pipeline(steps=[

#     ('drop_transformer',drop_transformer),
                 ('col_transformer',col_transformer),

    ('pca',PCA()),
]
                )
```

In [91]:
```
pipe
```

Out[91]:



Optimizing Model Performance with Hyperparameter Tuning

In [92]:

```
# Initialze the estimators
clf1 = RandomForestRegressor()
clf2 = Lasso()

clf3 = LinearRegression()
clf4 = Ridge()
clf5 = ElasticNet()

clf6 = DecisionTreeRegressor()
```

In [93]:

```
# Initiaze the hyperparameters for each dictionary

#hyperparameters for RandomForestRegressor
param1 = {}
param1['regressor__n_estimators'] = [10,20]
param1['regressor__max_depth'] = [10,15]
param1['regressor'] = [clf1]

#hyperparameters for Lasso

param2 = {}
param2['regressor__alpha'] = [0.1, 1, 10]
param2['regressor'] = [clf2]

#hyperparameters for LinearRegression

param3 = {}
param3['regressor'] = [clf3]

#hyperparameters for Ridge

param4 = {}
param4['regressor__alpha'] = [0.1, 1]
param4['regressor'] = [clf4]

#hyperparameters for ElasticNet

param5 = {}
param5['regressor__alpha'] = [0.1, 1]
param5['regressor__l1_ratio'] = [0.2, 0.5, 0.7]
param5['regressor'] = [clf5]

#hyperparameters for DecisionTreeRegressor
```

```
param6 = {}
param6['regressor__max_depth'] = [3, 6]
param6['regressor'] = [clf6]
```

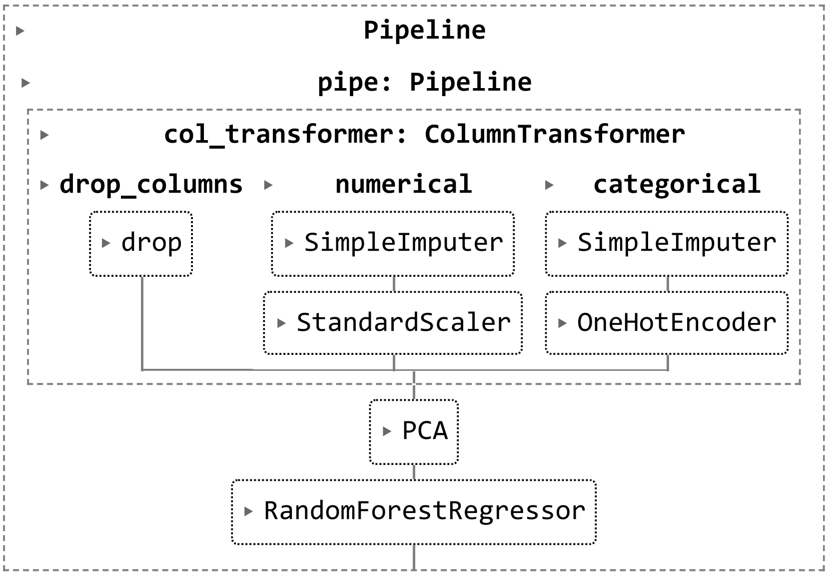
In [94]:

```
pipeline = Pipeline(steps=[('pipe',pipe),
                           ('regressor', clf1)])
# params = [param1, param2, param3, param4, param5,param6]
params = [param1, param2, param3]
```

In [95]:

```
pipeline
```

Out[95]:



In [ ]:

In [96]:

```
# Gridsearchcv is used
# %%time

gs = GridSearchCV(pipeline, params, cv=3, n_jobs=-1, scoring='neg_root_mean_squared_
```

In [97]:

```
# X_train_pipe
```

In [98]:

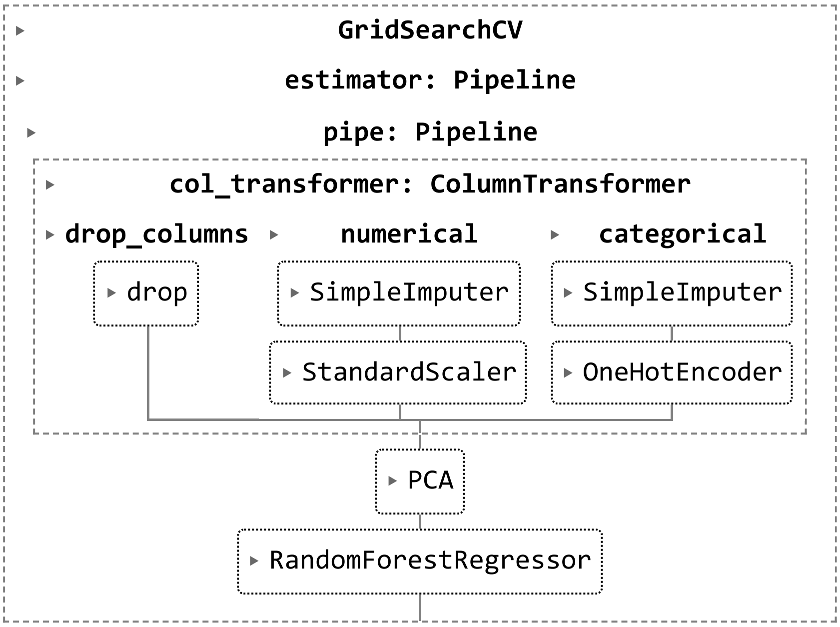
```
%%time

gs_fit = gs.fit(X_train_pipe, y_train_pipe)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Wall time: 11min 53s
```

In [99]:

```
gs
```

Out[99]:



In [100…

```
gs.best_params_
```

Out[100…

```
{'regressor': RandomForestRegressor(),
 'regressor__max_depth': 15,
 'regressor__n_estimators': 20}
```

In [101…

```
gs.score(X_test_pipe,y_test_pipe)
```

Out[101…

```
-45.8964777603188
```

In [102…

```
pred_y = gs.predict(X_test_pipe)
```

In [103…

```
r2_score(y_test_pipe,pred_y)
```

Out[103…

```
0.38487937564878716
```

So After using Gridsearch I found that RandomForestRegressor is the best model with max_depth 15 and n_estimators 20 which shows R2 score 0.3848 and negative mean square error is -45.89.

In [ ]:

```

```

To enhance the RandomForestRegressor model's performance, consider tuning hyperparameters such as `max_depth` and `n_estimators` .

- **`max_depth` :** Determines the maximum depth of each tree in the forest. Higher values can lead to overfitting, so finding an optimal depth is crucial for balancing model complexity.
- **`n_estimators` :** Defines the number of trees in the forest. While increasing this parameter can enhance performance, excessively high values might not significantly improve results and can increase computational load.

So i should use more number max_depth and n_estimators to improve my model performance.But its taking too much time tahts why I have used only 15 as max_debth and

n_estimators as 20.

In [ ]:

In [104…

```
"""This is the pipeline To drop specific columns """
drop_transformer = ColumnTransformer(transformers=[('drop_columns','drop',drop_featu
```

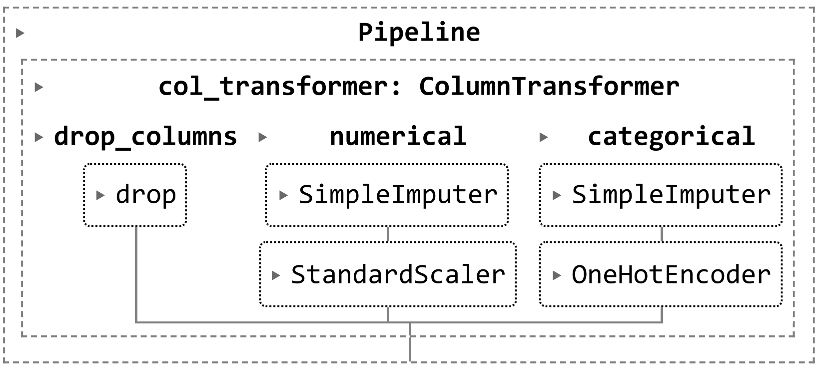
In [105…

```
# drop_transformer = ColumnTransformer(transformers=[('drop_columns','drop',drop_fea

# Creating pipelines for numerical and categorical features
numerical_pipeline = Pipeline(steps=[
#     ('outlier_removal', remove_outliers1(remove_outliers)),
    ('imputer', SimpleImputer(strategy='mean')) ,
    ('stdscaler', StandardScaler())
])

categorical_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Filling missing values
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encoding categoric
])

# Creating a ColumnTransformer to apply the appropriate pipeline to each type of fea
col_transformer = ColumnTransformer(transformers=[
   ('drop_columns','drop',drop_features_1),
    ('numerical', numerical_pipeline, Numerical_features_2),
    ('categorical', categorical_pipeline, categorical_features_1),
#     ('scale', StandardScaler())
],remainder='drop')


pipe_2pp = Pipeline(steps=[

#     ('drop_transformer',drop_transformer),
                 ('col_transformer',col_transformer),

#     ('pca',PCA()),
]
                )
```

In [106…

```
pipe_2pp
```

Out[106…



In [ ]:

In [107…

```
# Initialze the estimators
clf1 = RandomForestRegressor()
clf2 = Lasso()

clf3 = LinearRegression()
clf4 = Ridge()
clf5 = ElasticNet()

clf6 = DecisionTreeRegressor()
```

In [108…

```
# Initiaze the hyperparameters for each dictionary

#hyperparameters for RandomForestRegressor
param1 = {}
param1['regressor__n_estimators'] = [250]
param1['regressor__max_depth'] = [50]
param1['regressor'] = [clf1]

#hyperparameters for Lasso

param2 = {}
param2['regressor__alpha'] = [0.1, 1, 10]
param2['regressor'] = [clf2]

#hyperparameters for LinearRegression

param3 = {}
param3['regressor'] = [clf3]

#hyperparameters for Ridge

param4 = {}
param4['regressor__alpha'] = [0.1, 1]
param4['regressor'] = [clf4]

#hyperparameters for ElasticNet

param5 = {}
param5['regressor__alpha'] = [0.1, 1]
param5['regressor__l1_ratio'] = [0.2, 0.5, 0.7]
param5['regressor'] = [clf5]

#hyperparameters for DecisionTreeRegressor

param6 = {}
param6['regressor__max_depth'] = [3, 6]
param6['regressor'] = [clf6]
```

In [109…

```
pipeline = Pipeline(steps=[('pipe_2pp',pipe_2pp),
                           ('regressor', clf1)])
params = [param1, param2, param3, param4, param5,param6]
# params = [param1]
```

In [110…

```
pipeline
```

Out[110…

```
Pipeline
pipe_2pp: Pipeline
col_transformer: ColumnTransformer
drop_columns    numerical         categorical
drop            SimpleImputer     SimpleImputer
                StandardScaler    OneHotEncoder
RandomForestRegressor
```

In [111…

```python
# Gridsearchcv is used
# %%time

gs = GridSearchCV(pipeline, params, cv=3, n_jobs=-1, scoring='neg_root_mean_squared_
```

In [112…

```python
%%time

gs_fit = gs.fit(X_train_pipe, y_train_pipe)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Wall time: 16min 54s
```

In [113…

```python
gs
```

Out[113…

```
GridSearchCV
estimator: Pipeline
pipe_2pp: Pipeline
col_transformer: ColumnTransformer
drop_columns    numerical         categorical
drop            SimpleImputer     SimpleImputer
                StandardScaler    OneHotEncoder
RandomForestRegressor
```

In [114…

```python
gs.best_params_
```

Out[114…

```
{'regressor': RandomForestRegressor(),
 'regressor__max_depth': 50,
 'regressor__n_estimators': 250}
```

In [115…

```python
gs.score(X_test_pipe,y_test_pipe)
```

Out[115…

```
-39.12102044360718
```

In [116…

```
pred_y = gs.predict(X_test_pipe)
```

In [117…

```
r2_score(y_test_pipe,pred_y)
```

Out[117…

```
0.5530880963854016
```

So After using Gridsearch I found that RandomForestRegressor is the best model with max_depth 50 and n_estimators 250 which shows R2 score 0.5530 and negative mean square error is -39.12 [with out PCA]

As without PCA the RMSE reduced from 45 to 39 and the speed of the model training in improved. So without PCA my model works better.

In [ ]:

In [ ]: