CAPSTONE 2

Supervised Learning

Research Question

What is the prediction on the total consumption of gas and electricity?

- Show 2 Regression models
- Target: is going to be the total KWH & total THERMS
- Variables:
  - TOTAL UNITS, KWH TOTAL SQFT, THERMS TOTAL SQFT,AVERAGE HOUSESIZE
  - Dummy (BUILDING TYPE) there are four: Residential, Commercial, Industrial, Other

Content

1. The Data

CHICAGO ENERGY USAGE

The data is based on Chicago energy usage for 2010. I loaded the data from Kaggle database. It shows the consumption of energy and gas for commercial, residential, industrial.

- https://www.kaggle.com/chicago/chicago-energy-usage-2010 (https://www.kaggle.com/chicago/chicago-energy-usage-2010)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import ensemble
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
# This is the model we'll be using.
from sklearn import tree
# A convenience for displaying visualizations.
from IPython.display import Image
import seaborn as sns

# Packages for rendering our tree.
import pydotplus
import graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn import ensemble
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error
from statsmodels.tools.eval_measures import mse, rmse

# Display preferences.
%matplotlib inline
pd.options.display.float_format = '{:.3f}'.format
```

In [2]:
```
df = pd.read_csv('energy_2010.csv')
```

In [3]:
```
#display all columns in dataframe
from IPython.display import display
pd.options.display.max_columns= None
display(df)
```

	COMMUNITY AREA NAME	CENSUS BLOCK	BUILDING TYPE	BUILDING_SUBTYPE	KWH JANUARY 2010	KWH FEBRUARY 2010	KWH MARCH 2010	KWH APRIL 2010	KWH MAY 2010	J
0	Archer Heights	170315704001006.000	Residential	Multi < 7	nan	nan	nan	nan	nan	
1	Ashburn	170317005014004.000	Residential	Multi 7+	7334.000	7741.000	4214.000	4284.000	2518.000	427
2	Auburn Gresham	170317105001006.000	Commercial	Multi < 7	nan	nan	nan	nan	nan	
3	Austin	170312503003003.000	Commercial	Multi < 7	nan	nan	nan	nan	nan	
4	Austin	170312504002008.000	Commercial	Multi < 7	nan	nan	nan	nan	nan	
...	...	...	...	...	...	...	...	...	...	
67046	Woodlawn	170318439002011.000	Residential	Single Family	2705.000	1318.000	1582.000	1465.000	1494.000	299
67047	Woodlawn	170318439002012.000	Commercial	Multi < 7	1005.000	1760.000	1521.000	1832.000	2272.000	236
67048	Woodlawn	170318439002012.000	Residential	Multi < 7	3567.000	3031.000	2582.000	2295.000	7902.000	498
67049	Woodlawn	170318439002013.000	Residential	Single Family	1208.000	1055.000	1008.000	1109.000	1591.000	136
67050	Woodlawn	170318439002014.000	Residential	Multi < 7	2717.000	3057.000	2695.000	3793.000	4237.000	538

67051 rows × 73 columns

Take a look at all the columns, count, and type

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67051 entries, 0 to 67050
Data columns (total 73 columns):
COMMUNITY AREA NAME                67051 non-null object
CENSUS BLOCK                       66974 non-null float64
BUILDING TYPE                      66974 non-null object
BUILDING_SUBTYPE                   66974 non-null object
KWH JANUARY 2010                   66180 non-null float64
KWH FEBRUARY 2010                  66180 non-null float64
KWH MARCH 2010                     66180 non-null float64
KWH APRIL 2010                     66180 non-null float64
KWH MAY 2010                       66180 non-null float64
KWH JUNE 2010                      66180 non-null float64
KWH JULY 2010                      66180 non-null float64
KWH AUGUST 2010                    66180 non-null float64
KWH SEPTEMBER 2010                 66180 non-null float64
KWH OCTOBER 2010                   66180 non-null float64
KWH NOVEMBER 2010                  66180 non-null float64
KWH DECEMBER 2010                  66180 non-null float64
TOTAL KWH                          66180 non-null float64
ELECTRICITY ACCOUNTS               66180 non-null object
ZERO KWH ACCOUNTS                  67051 non-null int64
THERM JANUARY 2010                 64821 non-null float64
THERM FEBRUARY 2010                62819 non-null float64
THERM MARCH 2010                   65569 non-null float64
TERM APRIL 2010                    65476 non-null float64
THERM MAY 2010                     65194 non-null float64
THERM JUNE 2010                    65284 non-null float64
THERM JULY 2010                    65231 non-null float64
THERM AUGUST 2010                  65143 non-null float64
THERM SEPTEMBER 2010               64769 non-null float64
THERM OCTOBER 2010                 65329 non-null float64
THERM NOVEMBER 2010                65492 non-null float64
THERM DECEMBER 2010                65507 non-null float64
TOTAL THERMS                       65755 non-null float64
GAS ACCOUNTS                       65755 non-null object
KWH TOTAL SQFT                     65901 non-null float64
THERMS TOTAL SQFT                  65378 non-null float64
KWH MEAN 2010                      66180 non-null float64
KWH STANDARD DEVIATION 2010        57095 non-null float64
KWH MINIMUM 2010                   66180 non-null float64
KWH 1ST QUARTILE 2010              66180 non-null float64
KWH 2ND QUARTILE 2010              66180 non-null float64
```

```
KWH 3RD QUARTILE 2010                  66180 non-null float64
KWH MAXIMUM 2010                       66180 non-null float64
KWH SQFT MEAN 2010                     65901 non-null float64
KWH SQFT STANDARD DEVIATION 2010       51666 non-null float64
KWH SQFT MINIMUM 2010                  65901 non-null float64
KWH SQFT 1ST QUARTILE 2010             65901 non-null float64
KWH SQFT 2ND QUARTILE 2010             65901 non-null float64
KWH SQFT 3RD QUARTILE 2010             65901 non-null float64
KWH SQFT MAXIMUM 2010                  65901 non-null float64
THERM MEAN 2010                        65755 non-null float64
THERM STANDARD DEVIATION 2010          56821 non-null float64
THERM MINIMUM 2010                     65755 non-null float64
THERM 1ST QUARTILE 2010                65755 non-null float64
THERM 2ND QUARTILE 2010                65755 non-null float64
THERM 3RD QUARTILE 2010                65755 non-null float64
THERM MAXIMUM 2010                     65755 non-null float64
THERMS SQFT MEAN 2010                  65378 non-null float64
THERMS SQFT STANDARD DEVIATION 2010    51367 non-null float64
THERMS SQFT MINIMUM 2010               65378 non-null float64
THERMS SQFT 1ST QUARTILE 2010          65378 non-null float64
THERMS SQFT 2ND QUARTILE 2010          65378 non-null float64
THERMS SQFT 3RD QUARTILE 2010          65378 non-null float64
THERMS SQFT MAXIMUM 2010               65378 non-null float64
TOTAL POPULATION                       67037 non-null float64
TOTAL UNITS                            67037 non-null float64
AVERAGE STORIES                        67051 non-null float64
AVERAGE BUILDING AGE                   67051 non-null float64
AVERAGE HOUSESIZE                      67037 non-null float64
OCCUPIED UNITS                         67037 non-null float64
OCCUPIED UNITS PERCENTAGE              64606 non-null float64
RENTER-OCCUPIED HOUSING UNITS          67037 non-null float64
RENTER-OCCUPIED HOUSING PERCENTAGE     64433 non-null float64
OCCUPIED HOUSING UNITS                 67037 non-null float64
dtypes: float64(67), int64(1), object(5)
memory usage: 37.3+ MB
```

2. Clean Data

2.1 Missing Values

Find what is the total and percentage of the missing values of the top 20 variable

In [5]:
```
total_missing= df.isnull().sum().sort_values(ascending= False)
percent_missing = (df.isnull().sum()/df.isnull().count()).sort_values(ascending= False)
missing_data = pd.concat ([total_missing, percent_missing], axis=1, keys = ['Total', 'Percent'])
missing_data.head(20)
```

Out[5]:

	Total	Percent
THERMS SQFT STANDARD DEVIATION 2010	15684	0.234
KWH SQFT STANDARD DEVIATION 2010	15385	0.229
THERM STANDARD DEVIATION 2010	10230	0.153
KWH STANDARD DEVIATION 2010	9956	0.148
THERM FEBRUARY 2010	4232	0.063
RENTER-OCCUPIED HOUSING PERCENTAGE	2618	0.039
OCCUPIED UNITS PERCENTAGE	2445	0.036
THERM SEPTEMBER 2010	2282	0.034
THERM JANUARY 2010	2230	0.033
THERM AUGUST 2010	1908	0.028
THERM MAY 2010	1857	0.028
THERM JULY 2010	1820	0.027
THERM JUNE 2010	1767	0.026
THERM OCTOBER 2010	1722	0.026
THERMS SQFT 2ND QUARTILE 2010	1673	0.025
THERMS SQFT MAXIMUM 2010	1673	0.025
THERMS SQFT MINIMUM 2010	1673	0.025
THERMS SQFT 3RD QUARTILE 2010	1673	0.025
THERMS SQFT MEAN 2010	1673	0.025
THERMS TOTAL SQFT	1673	0.025

Non-Numeric Value

In [6]:
```
non_numeric_columns = df.select_dtypes(['object']).columns
print(non_numeric_columns)

print("The number of non-numerical columns is {}".format(len(non_numeric_columns)))
```

```
Index(['COMMUNITY AREA NAME', 'BUILDING TYPE', 'BUILDING_SUBTYPE',
       'ELECTRICITY ACCOUNTS', 'GAS ACCOUNTS'],
      dtype='object')
The number of non-numerical columns is 5
```

Let's check the missing value for the categorical variable:

- Building Type

Building type will be our categorical variable, which will later be converted into a dummie to use in the modeling section.

First, lets see how many null values there is

In [7]:
```
#had to rename a varible, for better use in coding
rename= df.rename ({'BUILDING TYPE': 'BUILDING_TYPE'},axis=1, inplace=True)
```

In [8]:
```
#percentage of missing value
percent_missing_build_type = df['BUILDING_TYPE'].isnull().sum() * 100 / len(df)
total_missing_build_type= df.BUILDING_TYPE.isnull().sum()
print( 'Percentage of missing values: {}'.format(percent_missing_build_type))
print( 'Total of missing values: {}'.format(total_missing_build_type))
```

```
Percentage of missing values: 0.11483795916541141
Total of missing values: 77
```

In [9]:
```
df['BUILDING_TYPE']= df['BUILDING_TYPE'].fillna('Other')
df['BUILDING_TYPE'].unique()
```

Out[9]:
```
array(['Residential', 'Commercial', 'Industrial', 'Other'], dtype=object)
```

In [10]:
```
#drop categorical features that will not be use. EX: account number is an information not needed.
#Also drop CENSUS BLOCK. Is the address (geocoding algorithms)
df= df.drop(['ELECTRICITY ACCOUNTS', 'GAS ACCOUNTS','BUILDING_SUBTYPE','CENSUS BLOCK','COMMUNITY AREA
NAME'], axis=1) #categorical features
```

Numeric Values

Fill in null values

Since some building types either consume gas or electricity, we will fill in the null values for Total Therms and Total KWH variables with 0.

In [11]:
```
#unique values
df['TOTAL THERMS'].unique()
```

Out[11]:
```
array([10917.,    nan,  6057., ..., 18598.,  7080., 11656.])
```

In [12]:
```
#unique values
df['TOTAL KWH'].unique()
```

Out[12]:
```
array([   nan, 82064.,  1994., ..., 29288., 41977., 57736.])
```

In [13]:
```
#fill NAN values with 0, since each building type either uses Gas or Electricity
df= df.fillna(0)
df.shape
```

Out[13]:
```
(67051, 68)
```

In [14]:
```
#double checking that we do not have any null values left
total_missing= df.isnull().sum().sort_values(ascending= False)
percent_missing = (df.isnull().sum()/df.isnull().count()).sort_values(ascending= False)
missing_data = pd.concat ([total_missing, percent_missing], axis=1, keys = ['Total', 'Percent'])
missing_data.head(10)
```

Out[14]:

	Total	Percent
OCCUPIED HOUSING UNITS	0	0.000
THERM OCTOBER 2010	0	0.000
TERM APRIL 2010	0	0.000
THERM MAY 2010	0	0.000
THERM JUNE 2010	0	0.000
THERM JULY 2010	0	0.000
THERM AUGUST 2010	0	0.000
THERM SEPTEMBER 2010	0	0.000
THERM NOVEMBER 2010	0	0.000
RENTER-OCCUPIED HOUSING PERCENTAGE	0	0.000

In [15]:
```
df
```

Out[15]:

	BUILDING_TYPE	KWH JANUARY 2010	KWH FEBRUARY 2010	KWH MARCH 2010	KWH APRIL 2010	KWH MAY 2010	KWH JUNE 2010	KWH JULY 2010	KWH AUGUST 2010	KWH SEPTEMBER 2010	KWH OCTOBER 2010
0	Residential	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	Residential	7334.000	7741.000	4214.000	4284.000	2518.000	4273.000	4566.000	2787.000	3357.000	5540.000
2	Commercial	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	Commercial	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	Commercial	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
...	...	...	...	...	...	...	...	...	...	...	...
67046	Residential	2705.000	1318.000	1582.000	1465.000	1494.000	2990.000	2449.000	2351.000	1213.000	2174.000
67047	Commercial	1005.000	1760.000	1521.000	1832.000	2272.000	2361.000	3018.000	3030.000	2886.000	3833.000
67048	Residential	3567.000	3031.000	2582.000	2295.000	7902.000	4987.000	5773.000	3996.000	3050.000	3103.000
67049	Residential	1208.000	1055.000	1008.000	1109.000	1591.000	1367.000	1569.000	1551.000	1376.000	1236.000
67050	Residential	2717.000	3057.000	2695.000	3793.000	4237.000	5383.000	5544.000	6929.000	5280.000	5971.000

67051 rows × 68 columns

In [16]:
```
#how many only use gas and no electricty
only_use_kwh =df[df['TOTAL KWH']== 0]
print('Buildings that only use KWH:',only_use_kwh.shape)
#how many only use electricty and no gas
only_use_therms =df[df['TOTAL THERMS']== 0]
print('Buildings that only use therms:', only_use_therms.shape)
```

```
Buildings that only use KWH: (871, 68)
Buildings that only use therms: (1296, 68)
```

871 buildings only use gas energy, and 1,296 buildings only use electrical energy. Leaving 64,884 buildings using both types of energy. Since only a total of 2,167 buildings uses either gas or electrical power, we will drop those rows.

In [17]:
```
#drop any bulding that uses only one type of enery
df= df[df['TOTAL THERMS'] != 0]
df= df[df['TOTAL KWH'] != 0]
df
```

Out[17]:

	BUILDING_TYPE	KWH JANUARY 2010	KWH FEBRUARY 2010	KWH MARCH 2010	KWH APRIL 2010	KWH MAY 2010	KWH JUNE 2010	KWH JULY 2010	KWH AUGUST 2010	KWH SEPTEMBER 2010	KWH OCTOBER 2010
97	Residential	1526.000	1665.000	1824.000	1579.000	2916.000	4211.000	4884.000	6874.000	4105.000	3460.000
103	Residential	242.000	136.000	134.000	134.000	144.000	122.000	3427.000	1626.000	2194.000	2218.000
104	Commercial	0.000	0.000	0.000	0.000	0.000	9.000	96.000	406.000	516.000	1891.000
120	Commercial	6171.000	3593.000	3812.000	4376.000	4431.000	5945.000	10220.000	9527.000	4271.000	3009.000
131	Residential	1959.000	2039.000	1647.000	1504.000	1790.000	3340.000	4368.000	4376.000	2426.000	3208.000
...	...	...	...	...	...	...	...	...	...	...	...
67046	Residential	2705.000	1318.000	1582.000	1465.000	1494.000	2990.000	2449.000	2351.000	1213.000	2174.000
67047	Commercial	1005.000	1760.000	1521.000	1832.000	2272.000	2361.000	3018.000	3030.000	2886.000	3833.000
67048	Residential	3567.000	3031.000	2582.000	2295.000	7902.000	4987.000	5773.000	3996.000	3050.000	3103.000
67049	Residential	1208.000	1055.000	1008.000	1109.000	1591.000	1367.000	1569.000	1551.000	1376.000	1236.000
67050	Residential	2717.000	3057.000	2695.000	3793.000	4237.000	5383.000	5544.000	6929.000	5280.000	5971.000

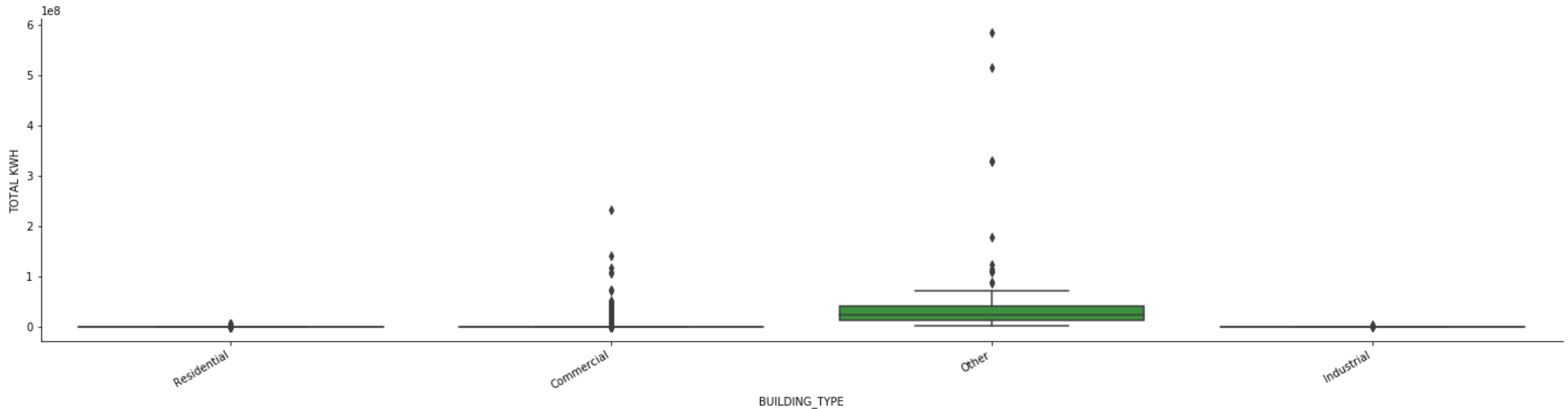
64884 rows × 68 columns

2.2 Outliers

This are just the outliers for the variables we will be using for the target: TOTAL KWH AND TOTAL THERMS

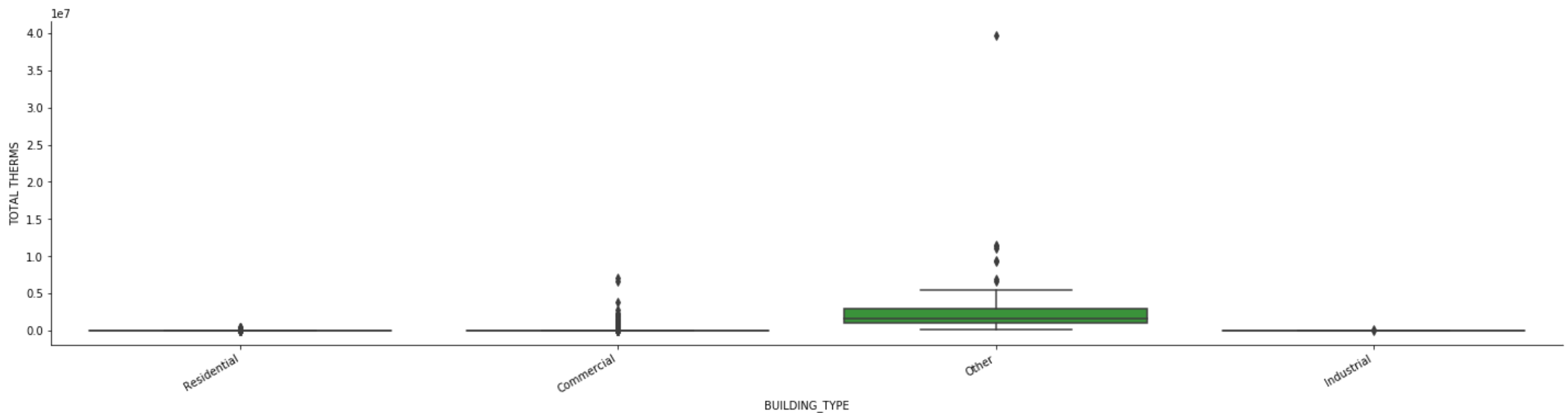
In [18]:
```
chart=sns.catplot(
    data=df,
    y='TOTAL KWH',
    x='BUILDING_TYPE',
    kind='box',
    height=5, # make the plot 5 units high
    aspect=4)
chart.set_xticklabels(rotation=30, horizontalalignment='right')
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x107196a10>



In [19]:
```
chart=sns.catplot(
    data=df,
    y='TOTAL THERMS',
    x='BUILDING_TYPE',
    kind='box',
    height=5, # make the plot 5 units high
    aspect=4)
chart.set_xticklabels(rotation=30, horizontalalignment='right')
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x11b466f10>



There seems to be some outliers just on this two variables alone, and since the data is too big to display it with boxplot. We will just clean the outliers for the whole data.

In [20]:
```
# See the numeric variables to use on winsorize
numeric_columns = df.select_dtypes(['int64', 'float64']).columns
```

In [21]:
```
#zcore to clean outliers
from scipy import stats
df[(np.abs(stats.zscore(df[numeric_columns]))<3).all(axis=1)]
```

Out[21]:

	BUILDING_TYPE	KWH JANUARY 2010	KWH FEBRUARY 2010	KWH MARCH 2010	KWH APRIL 2010	KWH MAY 2010	KWH JUNE 2010	KWH JULY 2010	KWH AUGUST 2010	KWH SEPTEMBER 2010	KW OCTOBE 201
97	Residential	1526.000	1665.000	1824.000	1579.000	2916.000	4211.000	4884.000	6874.000	4105.000	3460.0(
103	Residential	242.000	136.000	134.000	134.000	144.000	122.000	3427.000	1626.000	2194.000	2218.0(
104	Commercial	0.000	0.000	0.000	0.000	0.000	9.000	96.000	406.000	516.000	1891.0(
120	Commercial	6171.000	3593.000	3812.000	4376.000	4431.000	5945.000	10220.000	9527.000	4271.000	3009.0(
131	Residential	1959.000	2039.000	1647.000	1504.000	1790.000	3340.000	4368.000	4376.000	2426.000	3208.0(
...	...	...	...	...	...	...	...	...	...	...	
67045	Residential	9572.000	9104.000	8525.000	7756.000	11256.000	11669.000	12099.000	13200.000	9694.000	8419.0(
67046	Residential	2705.000	1318.000	1582.000	1465.000	1494.000	2990.000	2449.000	2351.000	1213.000	2174.0(
67047	Commercial	1005.000	1760.000	1521.000	1832.000	2272.000	2361.000	3018.000	3030.000	2886.000	3833.0(
67048	Residential	3567.000	3031.000	2582.000	2295.000	7902.000	4987.000	5773.000	3996.000	3050.000	3103.0(
67050	Residential	2717.000	3057.000	2695.000	3793.000	4237.000	5383.000	5544.000	6929.000	5280.000	5971.0(

61892 rows × 68 columns

To clean outliers, drop values with a zscore less than -3 or greater than 3. Leaving the full data at 61,892 rows.

3- The Target and Variables and Correlation (Feature Engineering)

3.1 Target

Target: TOTAL KWH & TOTAL THERMS

3.2 Correlation with target

CORRELATION WITH TOTAL KWH

In [22]:
```
# See the numeric variables
numeric_columns = df.select_dtypes(['int64', 'float64']).columns
#correlation with target
pd.set_option('display.max_row', 100)  #display all rows
np.abs(df[numeric_columns].iloc[:,1:].corr().loc[:,'TOTAL KWH']).sort_values(ascending=False).head
```

Out[22]:
```
<bound method NDFrame.head of TOTAL KWH                                      1.000
KWH MARCH 2010                      0.998
KWH APRIL 2010                      0.998
KWH OCTOBER 2010                    0.998
KWH SEPTEMBER 2010                  0.997
KWH MAY 2010                        0.996
KWH FEBRUARY 2010                   0.996
KWH NOVEMBER 2010                   0.995
KWH JUNE 2010                       0.995
KWH AUGUST 2010                     0.994
KWH JULY 2010                       0.992
KWH DECEMBER 2010                   0.982
KWH TOTAL SQFT                      0.863
THERMS TOTAL SQFT                   0.847
KWH MAXIMUM 2010                    0.776
THERM DECEMBER 2010                 0.744
THERM JANUARY 2010                  0.733
THERM FEBRUARY 2010                 0.726
THERM MARCH 2010                    0.716
THERM NOVEMBER 2010                 0.715
TOTAL THERMS                        0.710
TERM APRIL 2010                     0.684
THERM OCTOBER 2010                  0.670
THERM MAY 2010                      0.660
THERM JUNE 2010                     0.619
ZERO KWH ACCOUNTS                   0.616
THERM SEPTEMBER 2010                0.613
THERM JULY 2010                     0.598
THERM AUGUST 2010                   0.596
THERM MAXIMUM 2010                  0.556
KWH SQFT MAXIMUM 2010               0.541
THERMS SQFT MAXIMUM 2010            0.538
TOTAL UNITS                         0.486
RENTER-OCCUPIED HOUSING UNITS       0.476
OCCUPIED HOUSING UNITS              0.475
OCCUPIED UNITS                      0.475
TOTAL POPULATION                    0.462
KWH STANDARD DEVIATION 2010         0.451
KWH 3RD QUARTILE 2010               0.394
KWH MEAN 2010                       0.385
KWH 2ND QUARTILE 2010               0.328
AVERAGE HOUSESIZE                   0.315
KWH SQFT STANDARD DEVIATION 2010    0.302
```

```
THERMS SQFT STANDARD DEVIATION 2010   0.271
KWH SQFT 3RD QUARTILE 2010            0.243
THERM STANDARD DEVIATION 2010         0.242
KWH SQFT MEAN 2010                    0.239
THERMS SQFT MEAN 2010                 0.232
THERMS SQFT 3RD QUARTILE 2010         0.230
KWH SQFT 2ND QUARTILE 2010            0.210
KWH 1ST QUARTILE 2010                 0.206
THERMS SQFT 2ND QUARTILE 2010         0.205
KWH MINIMUM 2010                      0.204
AVERAGE STORIES                       0.182
THERMS SQFT 1ST QUARTILE 2010         0.167
THERMS SQFT MINIMUM 2010              0.163
KWH SQFT 1ST QUARTILE 2010            0.152
KWH SQFT MINIMUM 2010                 0.147
THERM 3RD QUARTILE 2010               0.137
THERM MEAN 2010                       0.136
THERM 2ND QUARTILE 2010               0.102
THERM 1ST QUARTILE 2010               0.084
THERM MINIMUM 2010                    0.082
OCCUPIED UNITS PERCENTAGE             0.042
AVERAGE BUILDING AGE                  0.032
RENTER-OCCUPIED HOUSING PERCENTAGE    0.010
Name: TOTAL KWH, dtype: float64>
```

In [23]:
```
# See the numeric variables
numeric_columns = df.select_dtypes(['int64', 'float64']).columns
#correlation with target
pd.set_option('display.max_row', 100)  #display all rows
np.abs(df[numeric_columns].iloc[:,1:].corr().loc[:,'TOTAL THERMS']).sort_values(ascending=False)
```

Out[23]:

```
TOTAL THERMS                          1.000
THERM NOVEMBER 2010                   0.996
TERM APRIL 2010                       0.996
THERM MARCH 2010                      0.994
THERM JANUARY 2010                    0.989
THERM FEBRUARY 2010                   0.988
THERM DECEMBER 2010                   0.988
THERM OCTOBER 2010                    0.988
THERM MAY 2010                        0.983
THERM JUNE 2010                       0.955
THERM SEPTEMBER 2010                  0.950
THERM JULY 2010                       0.932
THERM MAXIMUM 2010                    0.929
THERM AUGUST 2010                     0.928
KWH AUGUST 2010                       0.734
KWH JUNE 2010                         0.733
KWH JULY 2010                         0.733
KWH MAY 2010                          0.730
ZERO KWH ACCOUNTS                     0.714
KWH APRIL 2010                        0.712
TOTAL KWH                             0.710
KWH OCTOBER 2010                      0.708
KWH SEPTEMBER 2010                    0.706
KWH MARCH 2010                        0.705
KWH NOVEMBER 2010                     0.689
KWH TOTAL SQFT                        0.684
KWH FEBRUARY 2010                     0.678
KWH DECEMBER 2010                     0.664
THERMS TOTAL SQFT                     0.653
TOTAL POPULATION                      0.591
TOTAL UNITS                           0.568
RENTER-OCCUPIED HOUSING UNITS         0.564
OCCUPIED HOUSING UNITS                0.563
OCCUPIED UNITS                        0.563
AVERAGE HOUSESIZE                     0.463
KWH MAXIMUM 2010                      0.443
THERM STANDARD DEVIATION 2010         0.442
KWH SQFT MAXIMUM 2010                 0.363
THERMS SQFT MAXIMUM 2010              0.349
THERM MEAN 2010                       0.257
THERM 3RD QUARTILE 2010               0.254
THERM 2ND QUARTILE 2010               0.219
KWH SQFT STANDARD DEVIATION 2010      0.163
```

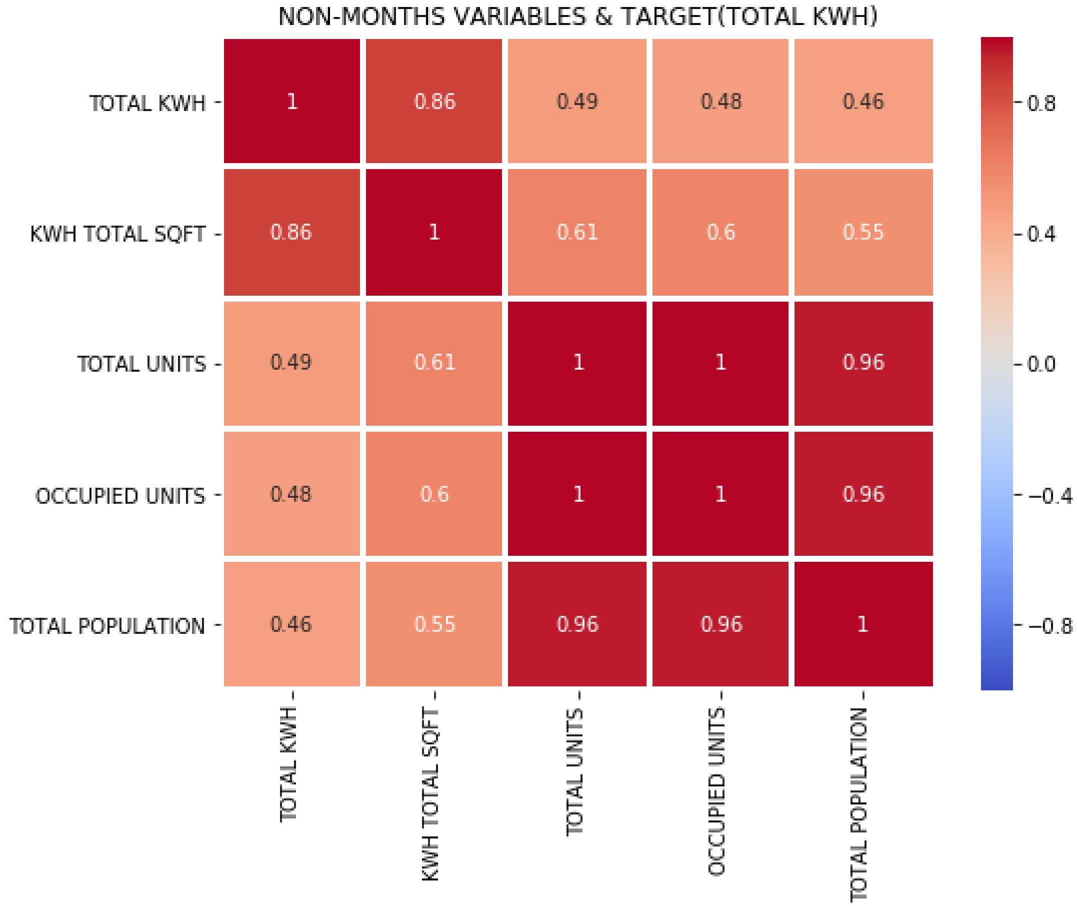
```
THERMS SQFT STANDARD DEVIATION 2010    0.156
THERM 1ST QUARTILE 2010                0.152
THERM MINIMUM 2010                     0.150
KWH STANDARD DEVIATION 2010            0.139
KWH SQFT 3RD QUARTILE 2010             0.101
KWH SQFT MEAN 2010                     0.095
AVERAGE STORIES                        0.094
THERMS SQFT 3RD QUARTILE 2010          0.091
THERMS SQFT MEAN 2010                  0.087
KWH MEAN 2010                          0.078
KWH 3RD QUARTILE 2010                  0.078
KWH SQFT 2ND QUARTILE 2010             0.076
THERMS SQFT 2ND QUARTILE 2010          0.068
KWH 2ND QUARTILE 2010                  0.052
THERMS SQFT 1ST QUARTILE 2010          0.049
KWH SQFT 1ST QUARTILE 2010             0.049
THERMS SQFT MINIMUM 2010               0.047
KWH SQFT MINIMUM 2010                  0.047
KWH 1ST QUARTILE 2010                  0.034
KWH MINIMUM 2010                       0.033
AVERAGE BUILDING AGE                   0.027
OCCUPIED UNITS PERCENTAGE              0.021
RENTER-OCCUPIED HOUSING PERCENTAGE     0.002
Name: TOTAL THERMS, dtype: float64
```

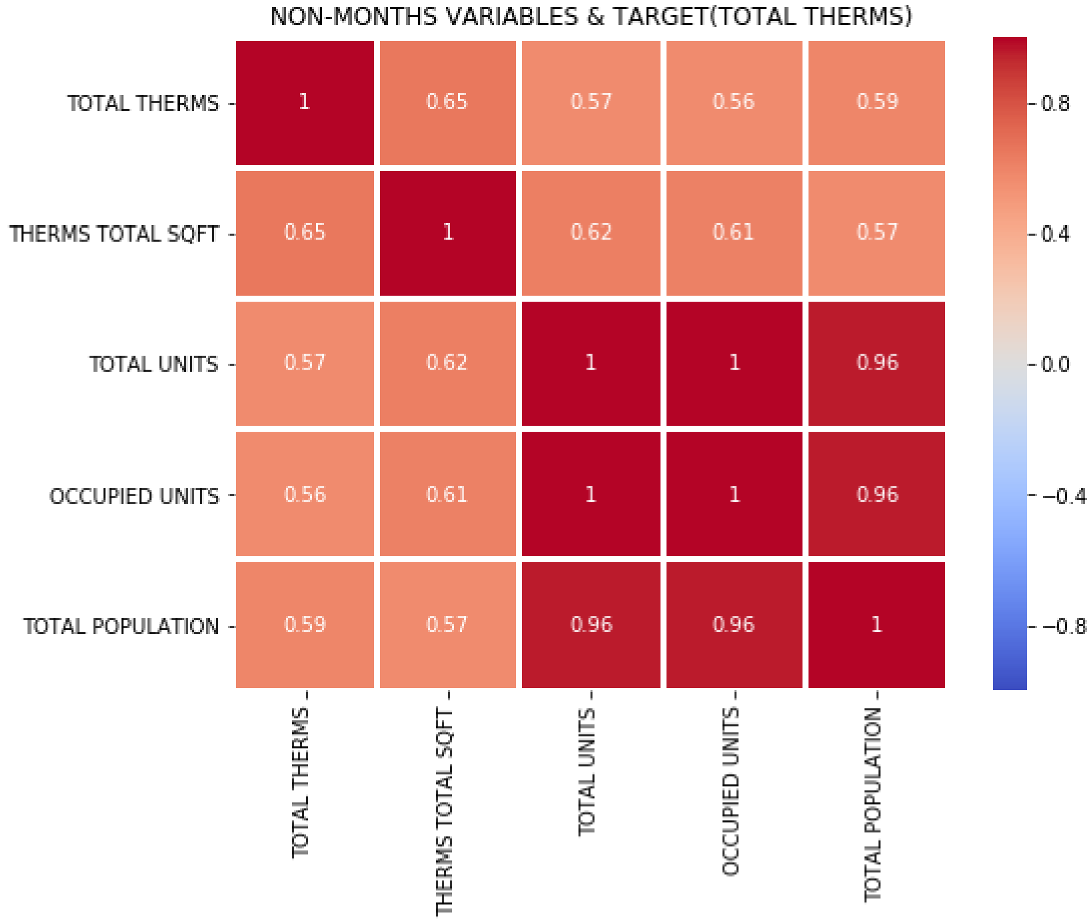
Grab the variables that have an important significances, and have a correlation close as possible to a .5 from the targets. Which are the following four:

- TOTAL UNITS: Total number of housing units
- OCCUPIED UNITS: Number of housing units that are occupied
- TOTAL POPULATION: Total population from Census 2010 report (QT-P6) Race alone or in combination and Hispanic or Latino 2010
- KWH TOTAL SQFT: Total square footage associated with the electric energy
- THERMS TOTAL SQFT: Total square footage associated with the natural gas energy usage for Kilowatt Hours

In [24]:
```
df_corr_kwh= df[['TOTAL KWH','KWH TOTAL SQFT','TOTAL UNITS','OCCUPIED UNITS','TOTAL POPULATION']]
df_corr_therms= df[['TOTAL THERMS', 'THERMS TOTAL SQFT','TOTAL UNITS','OCCUPIED UNITS','TOTAL POPULAT
ION']]
```

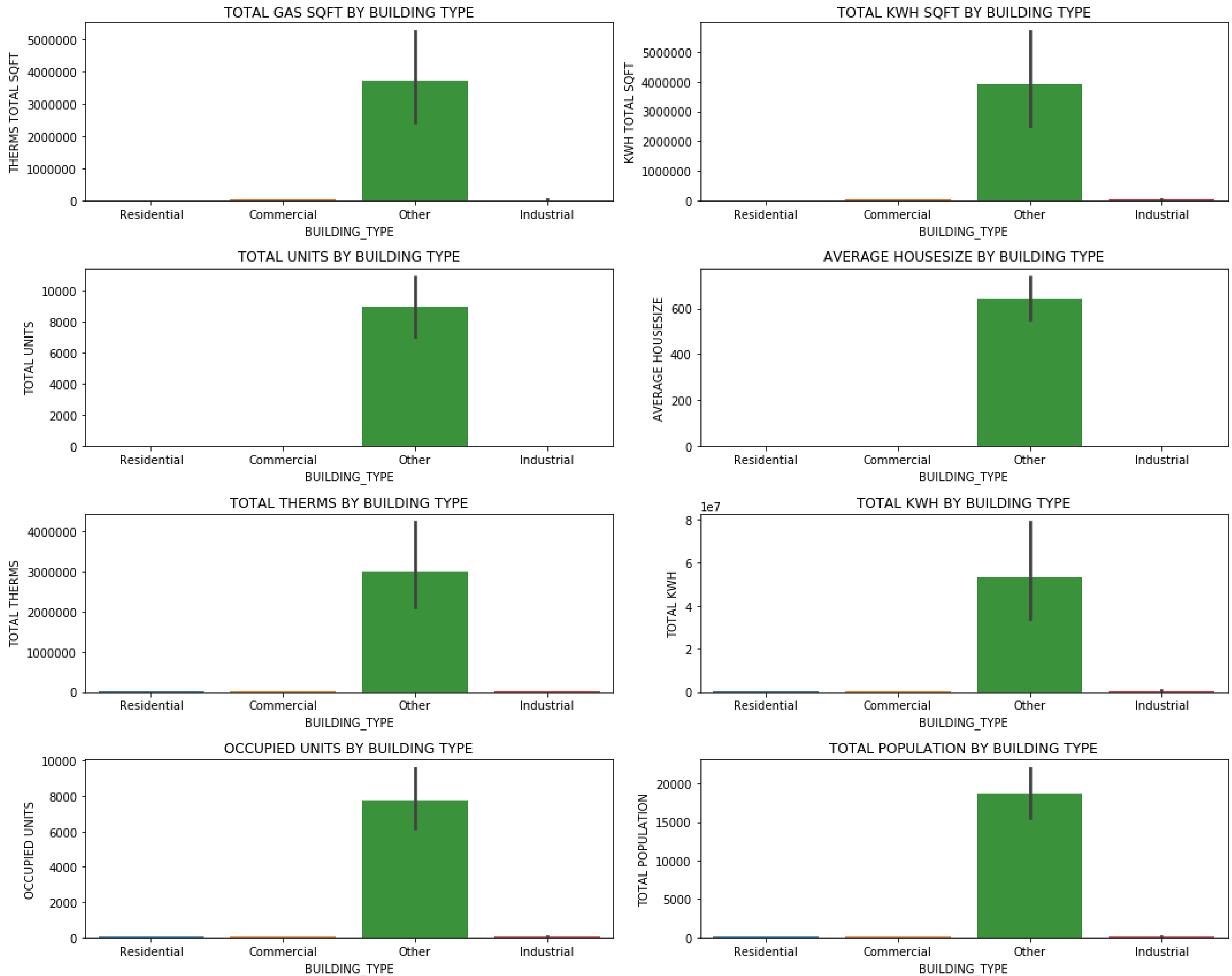
In [25]:
```
fig, ax = plt.subplots(figsize=(8,6))

sns.heatmap(df_corr_kwh.corr(),annot = True,vmin=-1, vmax=1, center= 0, cmap= 'coolwarm',linewidths=2
, linecolor='white')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_title('NON-MONTHS VARIABLES & TARGET(TOTAL KWH)')
plt.show()

fig, ax = plt.subplots(figsize=(8,6))

sns.heatmap(df_corr_therms.corr(),annot = True,vmin=-1, vmax=1, center= 0, cmap= 'coolwarm',linewidth
s=2, linecolor='white')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_title('NON-MONTHS VARIABLES & TARGET(TOTAL THERMS)')
plt.show()
```


NON-MONTHS VARIABLES & TARGET(TOTAL KWH)
TOTAL KWH
KWH TOTAL SQFT
TOTAL UNITS
OCCUPIED UNITS
TOTAL POPULATION
1
0.86
0.49
0.48
0.46
0.86
1
0.61
0.6
0.55
0.49
0.61
1
1
0.96
0.48
0.6
1
1
0.96
0.46
0.55
0.96
0.96
1
0.8
0.4
0.0
−0.4
−0.8



Display bar graphs to show the variables and target with our categorical variable, building type. As well as the target and building type.

In [26]:
```
import seaborn as sns
plt.figure(figsize=(15,12))

#categorical variable: building type
plt.subplot(4,2,1)
sns.barplot(df["BUILDING_TYPE"], df["THERMS TOTAL SQFT"])
plt.title("TOTAL GAS SQFT BY BUILDING TYPE")

plt.subplot(4,2,2)
sns.barplot(df["BUILDING_TYPE"], df["KWH TOTAL SQFT"])
plt.title("TOTAL KWH SQFT BY BUILDING TYPE")

plt.subplot(4,2,3)
sns.barplot(df["BUILDING_TYPE"], df["TOTAL UNITS"])
plt.title("TOTAL UNITS BY BUILDING TYPE")

plt.subplot(4,2,4)
sns.barplot(df["BUILDING_TYPE"], df["AVERAGE HOUSESIZE"])
plt.title("AVERAGE HOUSESIZE BY BUILDING TYPE")

plt.subplot(4,2,5)
sns.barplot(df["BUILDING_TYPE"], df["TOTAL THERMS"])
plt.title("TOTAL THERMS BY BUILDING TYPE")

plt.subplot(4,2,6)
sns.barplot(df["BUILDING_TYPE"], df["TOTAL KWH"])
plt.title("TOTAL KWH BY BUILDING TYPE")

plt.subplot(4,2,7)
sns.barplot(df["BUILDING_TYPE"], df["OCCUPIED UNITS"])
plt.title("OCCUPIED UNITS BY BUILDING TYPE")

plt.subplot(4,2,8)
sns.barplot(df["BUILDING_TYPE"], df["TOTAL POPULATION"])
plt.title("TOTAL POPULATION BY BUILDING TYPE")


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


TOTAL GAS SQFT BY BUILDING TYPE
THERMS TOTAL SQFT
0
1000000
2000000
3000000
4000000
5000000
TOTAL KWH SQFT BY BUILDING TYPE
KWH TOTAL SQFT
0
1000000
2000000
3000000
4000000
5000000
TOTAL UNITS BY BUILDING TYPE
TOTAL UNITS
0
2000
4000
6000
8000
10000
AVERAGE HOUSESIZE BY BUILDING TYPE
AVERAGE HOUSESIZE
0
200
400
600
TOTAL THERMS BY BUILDING TYPE
TOTAL THERMS
0
1000000
2000000
3000000
4000000
TOTAL KWH BY BUILDING TYPE
1e7
TOTAL KWH
0
2
4
6
8
OCCUPIED UNITS BY BUILDING TYPE
OCCUPIED UNITS
0
2000
4000
6000
8000
10000
TOTAL POPULATION BY BUILDING TYPE
TOTAL POPULATION
0
5000
10000
15000
20000
Residential
Commercial
Other
Industrial
BUILDING_TYPE

As shown, most of variance in the data is in building type 'other'.

3.3 Set X and Y

In [27]:
```
#setting Y and X
#GET DUMMIES
df= pd.concat([df, pd.get_dummies(df.BUILDING_TYPE, prefix= 'Type', drop_first=True)], axis=1)
dummy_BUILDING = list(pd.get_dummies(df.BUILDING_TYPE, prefix= 'Type', drop_first=True).columns)


#EVERYTHING WITH 2 IS FOR TARGET : TOTAL THERMS
X = df[['KWH TOTAL SQFT','TOTAL UNITS','OCCUPIED UNITS','TOTAL POPULATION'] + dummy_BUILDING ]
X2 = df[['THERMS TOTAL SQFT','TOTAL UNITS','OCCUPIED UNITS','TOTAL POPULATION'] + dummy_BUILDING]
Y = df['TOTAL KWH']
Y2 = df['TOTAL THERMS']
```

In [28]:
```
Y
```

Out[28]:
```
97       41662.000
103      15665.000
104       5357.000
120      68474.000
131      32429.000
           ...
67046    27654.000
67047    41977.000
67048    48850.000
67049    17707.000
67050    57736.000
Name: TOTAL KWH, Length: 64884, dtype: float64
```

4- Split the data for train and test to be perform on regression model

4.1 Split for each target and its corresponding variables

Split train and test for first target (TOTAL KWH)

In [29]:
```
#20% for test and 80% for train
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 60)
print("The number of observations in training set(Target: Total KWH) is {}".format(X_train.shape[0]))
print("The number of observations in test set(Target: Total KWH) is {}".format(X_test.shape[0]))
```

```
The number of observations in training set(Target: Total KWH) is 51907
The number of observations in test set(Target: Total KWH) is 12977
```

Split train and test for second target (TOTAL THERMS)

In [30]:
```
#20% for test and 80% for train
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, Y2, test_size = 0.2, random_state = 60)
print("The number of observations in training set(Target: Total THERMS) is {}".format(X_train2.shape[
0]))
print("The number of observations in test set(Target: Total THERMS) is {}".format(X_test2.shape[0]))
```

```
The number of observations in training set(Target: Total THERMS) is 51907
The number of observations in test set(Target: Total THERMS) is 12977
```

4.2 Standardize the variables

In [31]:
```
#standarize data
from sklearn.preprocessing import StandardScaler

sc= StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train2 = sc.fit_transform(X_train2)
X_test2 = sc.transform(X_test2)
```

In [32]:
```
#standarize y
#y_train = y_train / np.max(y_train)
#y_test = y_test / np.max(y_test)
#y_train2 = y_train2 / np.max(y_train2)
#y_test2 = y_test2 / np.max(y_test2)
```

5- Regression (OLS) Model

5.1 OLS MODEL: FOR FIRST TARGET (TOTAL KWH)

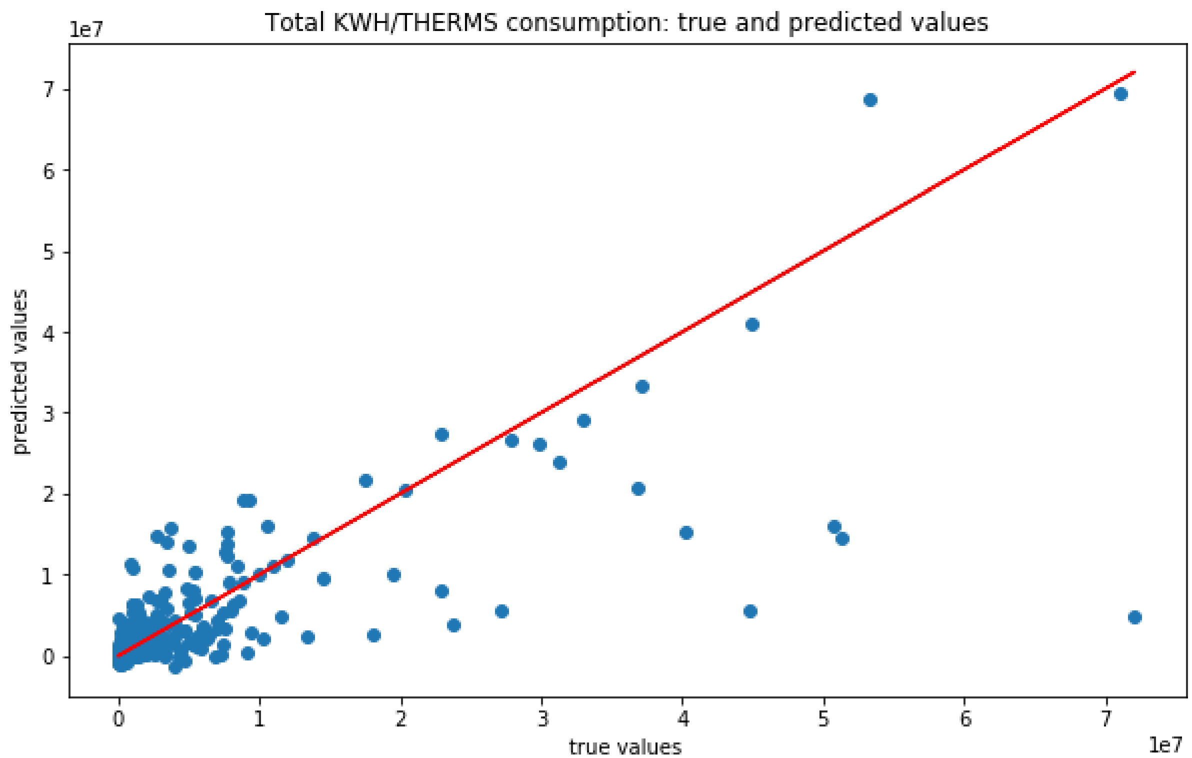
In [33]:
```
#Train with OLS
import statsmodels.api as sm
X_train_sm = sm.add_constant(X_train)
# We fit an OLS model using statsmodels
results_train = sm.OLS(y_train, X_train_sm).fit()

# We print the summary results
print(results_train.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:              TOTAL KWH   R-squared:                       0.770
Model:                            OLS   Adj. R-squared:                  0.770
Method:                 Least Squares   F-statistic:                 2.484e+04
Date:                Tue, 14 Apr 2020   Prob (F-statistic):               0.00
Time:                        22:05:55   Log-Likelihood:            -8.3234e+05
No. Observations:               51907   AIC:                         1.665e+06
Df Residuals:                   51899   BIC:                         1.665e+06
Df Model:                           7                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.369e+05   9776.968     24.230      0.000    2.18e+05    2.56e+05
x1          4.029e+06   1.28e+04    314.638      0.000       4e+06    4.05e+06
x2          3.425e+06   1.91e+05     17.935      0.000    3.05e+06     3.8e+06
x3         -4.344e+06   1.87e+05    -23.272      0.000   -4.71e+06   -3.98e+06
x4          5.546e+05   3.91e+04     14.200      0.000    4.78e+05    6.31e+05
x5          1137.5071   9782.557      0.116      0.907    -1.8e+04    2.03e+04
x6          5.156e+05   1.68e+04     30.703      0.000    4.83e+05    5.49e+05
x7         -3.315e+04   9807.134     -3.381      0.001   -5.24e+04   -1.39e+04
==============================================================================
Omnibus:                   185249.584   Durbin-Watson:                   2.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):     143393419890.645
Skew:                          71.844   Prob(JB):                         0.00
Kurtosis:                    8144.223   Cond. No.                         54.3
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [34]:
```
#RESULTS FOR TEST OLS (total kwh)
X_test_sm = sm.add_constant(X_test)
results_test = sm.OLS(y_test, X_test_sm).fit()
print(results_test.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              TOTAL KWH   R-squared:                       0.630
Model:                            OLS   Adj. R-squared:                  0.630
Method:                 Least Squares   F-statistic:                     3154.
Date:                Tue, 14 Apr 2020   Prob (F-statistic):               0.00
Time:                        22:05:55   Log-Likelihood:            -1.9818e+05
No. Observations:               12977   AIC:                         3.964e+05
Df Residuals:                   12969   BIC:                         3.964e+05
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.188e+05   9121.168     23.992      0.000    2.01e+05    2.37e+05
x1          3.502e+06   3.83e+04     91.507      0.000    3.43e+06    3.58e+06
x2         -9.818e+05   1.52e+05     -6.453      0.000   -1.28e+06   -6.84e+05
x3          4.569e+04   1.25e+05      0.367      0.714   -1.99e+05     2.9e+05
x4          4.216e+05   6.93e+04      6.081      0.000    2.86e+05    5.58e+05
x5         -1608.7726   8890.191     -0.181      0.856    -1.9e+04    1.58e+04
x6          2.097e+05   1.51e+04     13.855      0.000     1.8e+05    2.39e+05
x7         -6.024e+04   9154.834     -6.580      0.000   -7.82e+04   -4.23e+04
==============================================================================
Omnibus:                    35406.808   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):       1764974947.482
Skew:                          33.826   Prob(JB):                         0.00
Kurtosis:                    1808.440   Cond. No.                         51.1
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [35]:
```
#display graph to show predicted values and true values
# We are making predictions here


y_preds = results_test.predict(X_test_sm)
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_preds)
plt.plot(y_test, y_test, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Total KWH/THERMS consumption: true and predicted values")
plt.show()

print("Mean squared error of the prediction is: {}".format(mse(y_test, y_preds)))
```




```
Mean squared error of the prediction is: 1077575562468.0375
```

5.2 OLS MODEL: FOR SECOND TARGET (TOTAL THERMS)

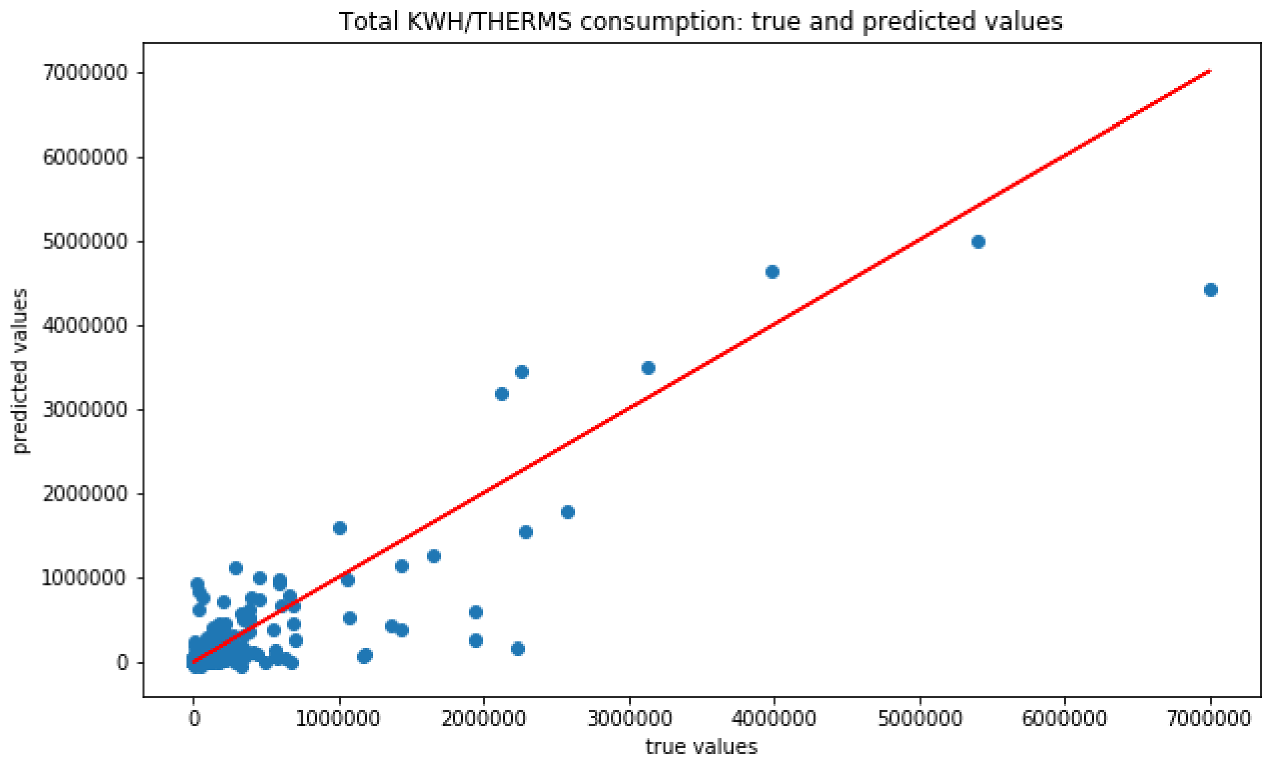
In [36]:
```
#Train with OLS
import statsmodels.api as sm
X_train_sm2 = sm.add_constant(X_train2)
# We fit an OLS model using statsmodels
results_train2 = sm.OLS(y_train2, X_train_sm2).fit()

# We print the summary results
print(results_train2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           TOTAL THERMS   R-squared:                       0.506
Model:                            OLS   Adj. R-squared:                  0.506
Method:                 Least Squares   F-statistic:                     7606.
Date:                Tue, 14 Apr 2020   Prob (F-statistic):               0.00
Time:                        22:05:56   Log-Likelihood:            -6.9478e+05
No. Observations:               51907   AIC:                         1.390e+06
Df Residuals:                   51899   BIC:                         1.390e+06
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.007e+04    690.679     29.052      0.000    1.87e+04    2.14e+04
x1          1.189e+05    926.717    128.264      0.000    1.17e+05    1.21e+05
x2         -2.481e+05    1.36e+04   -18.274      0.000   -2.75e+05   -2.21e+05
x3          1.548e+05    1.32e+04    11.686      0.000    1.29e+05    1.81e+05
x4          1.558e+05   2760.736     56.425      0.000     1.5e+05    1.61e+05
x5           -18.1487    691.074     -0.026      0.979   -1372.661    1336.363
x6          -551.0570   1187.522     -0.464      0.643   -2878.612    1776.499
x7          -479.5142    692.796     -0.692      0.489   -1837.402     878.373
==============================================================================
Omnibus:                   221632.983   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):    1225972352741.483
Skew:                         125.902   Prob(JB):                         0.00
Kurtosis:                   23810.221   Cond. No.                         54.7
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [37]:
```
X_test_sm2 = sm.add_constant(X_test2)
#RESULTS FOR TEST OLS (total therms)
results_test2 = sm.OLS(y_test2, X_test_sm2).fit()
print(results_test2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           TOTAL THERMS   R-squared:                       0.784
Model:                            OLS   Adj. R-squared:                  0.784
Method:                 Least Squares   F-statistic:                     6730.
Date:                Tue, 14 Apr 2020   Prob (F-statistic):               0.00
Time:                        22:05:56   Log-Likelihood:            -1.5935e+05
No. Observations:               12977   AIC:                         3.187e+05
Df Residuals:                   12969   BIC:                         3.188e+05
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        2.03e+04    457.448     44.380      0.000    1.94e+04    2.12e+04
x1          1.248e+05   1810.387     68.910      0.000    1.21e+05    1.28e+05
x2         -5.167e+04   7683.985     -6.724      0.000   -6.67e+04   -3.66e+04
x3         -9.225e+04   6238.160    -14.789      0.000   -1.04e+05       -8e+04
x4          1.963e+05   3488.757     56.252      0.000    1.89e+05    2.03e+05
x5          -206.9153    445.843     -0.464      0.643   -1080.832     667.002
x6          -222.1629    759.915     -0.292      0.770   -1711.708    1267.382
x7         -1308.3213    458.856     -2.851      0.004   -2207.746    -408.896
==============================================================================
Omnibus:                    27879.940   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        437401217.718
Skew:                          18.534   Prob(JB):                         0.00
Kurtosis:                     901.647   Cond. No.                         51.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [38]:
```
#display graph to show predicted values and true values
# We are making predictions here

y_preds2 = results_test2.predict(X_test_sm2)
plt.figure(figsize=(10,6))
plt.scatter(y_test2, y_preds2)
plt.plot(y_test2, y_test2, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Total KWH/THERMS consumption: true and predicted values")
plt.show()

print("Mean squared error of the prediction is: {}".format(mse(y_test2, y_preds2)))
```




```
Mean squared error of the prediction is: 2710135081.5501847
```

6- Random Forest

With Random Forest, there will be 3 random forest model with different parameters to show which one results with the best accuracy score.

6.1 Tests

1st RF

TARGET 1: (TOTAL KWH)

In [39]:
```
#random forest regression model
#number of leaf (levels)= 2
#random state set at 0 to get the same random picking consistent
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(X_train, y_train)
```

Out[39]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=2, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [40]:
```
#accuracy 1st target
regr.score(X_test, y_test)
```

Out[40]:
```
0.4261795586272281
```

TARGET 2: (TOTAL THERMS)

In [41]:
```
#random forest regression model
#number of leaf (levels)= 2
#random state set at 0 to get the same random picking consistent
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(X_train2, y_train2)
```

Out[41]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=2, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [42]:
```
#accuracy TARGET 2
regr.score(X_test2, y_test2)
```

Out[42]:
```
0.6593803824063855
```

2nd RF

TARGET 1: (TOTAL KWH)

In [43]:
```
#random forest regression model
#n_estimator = 5 how many trees
#random state set at 0 to get the same random picking consistent
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=5, random_state=0, n_estimators= 2)
regr.fit(X_train, y_train)
```

Out[43]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=2, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [44]:
```
#accuracy
regr.score(X_test, y_test)
```

Out[44]:
```
-1.876942634004389
```

In [45]:
```
#TARGET 2
#random forest regression model
#number of leaf (levels)= 5
#random state set at 0 to get the same random picking consistent
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=5, random_state=0)
regr.fit(X_train2, y_train2)
```

Out[45]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [46]:
```
#accuracy
regr.score(X_test2, y_test2)
```

Out[46]: 0.7127274640919277

Since this random forest was the one that gave us the best result, lest test the cross validation score

In [47]:
```
# CV FOR TARGET 1: TOTAL KWH
from sklearn.model_selection import cross_val_score
cross_val= cross_val_score(regr, X, Y, cv=10)
print(cross_val)
print ('cv_score mean:{}'.format(np.mean(cross_val)))
```

```
[ 0.41633997 -2.12415902  0.58201797  0.75473265  0.22096259  0.71850575
  0.51098585  0.38120564  0.81510389  0.09094932]
cv_score mean:0.2366644609358583
```

In [48]:
```
# CV FOR TARGET 2: TOTAL THERMS
from sklearn.model_selection import cross_val_score
cross_val2= cross_val_score(regr, X2, Y2, cv=10)
print(cross_val2)
print ('cv_score mean:{}'.format(np.mean(cross_val2)))
```

```
[ 0.17859304  0.65721866  0.77484326  0.87005857  0.38978478 -0.16561426
  0.36128666  0.52461839  0.73845082  0.78928679]
cv_score mean:0.5118526702056964
```

Cross validation scores, are very low. Not very good prediction of the targets.

3rd RF

TARGET 1: (TOTAL KWH)

In [49]:
```
#random forest regression model
#number of leaf (levels)= 10
#random state set at 0 to get the same random picking consistent
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=10, random_state=0)
regr.fit(X_train, y_train)
```

Out[49]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=10, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [50]:
```
#accuracy
regr.score(X_test, y_test)
```

Out[50]:
```
0.5332547368915355
```

TARGET 2: (TOTAL THERMS)

In [51]:
```
#random forest regression model
#number of leaf (levels)= 10
#random state set at 0 to get the same random picking consistent
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=10, random_state=0)
regr.fit(X_train2, y_train2)
```

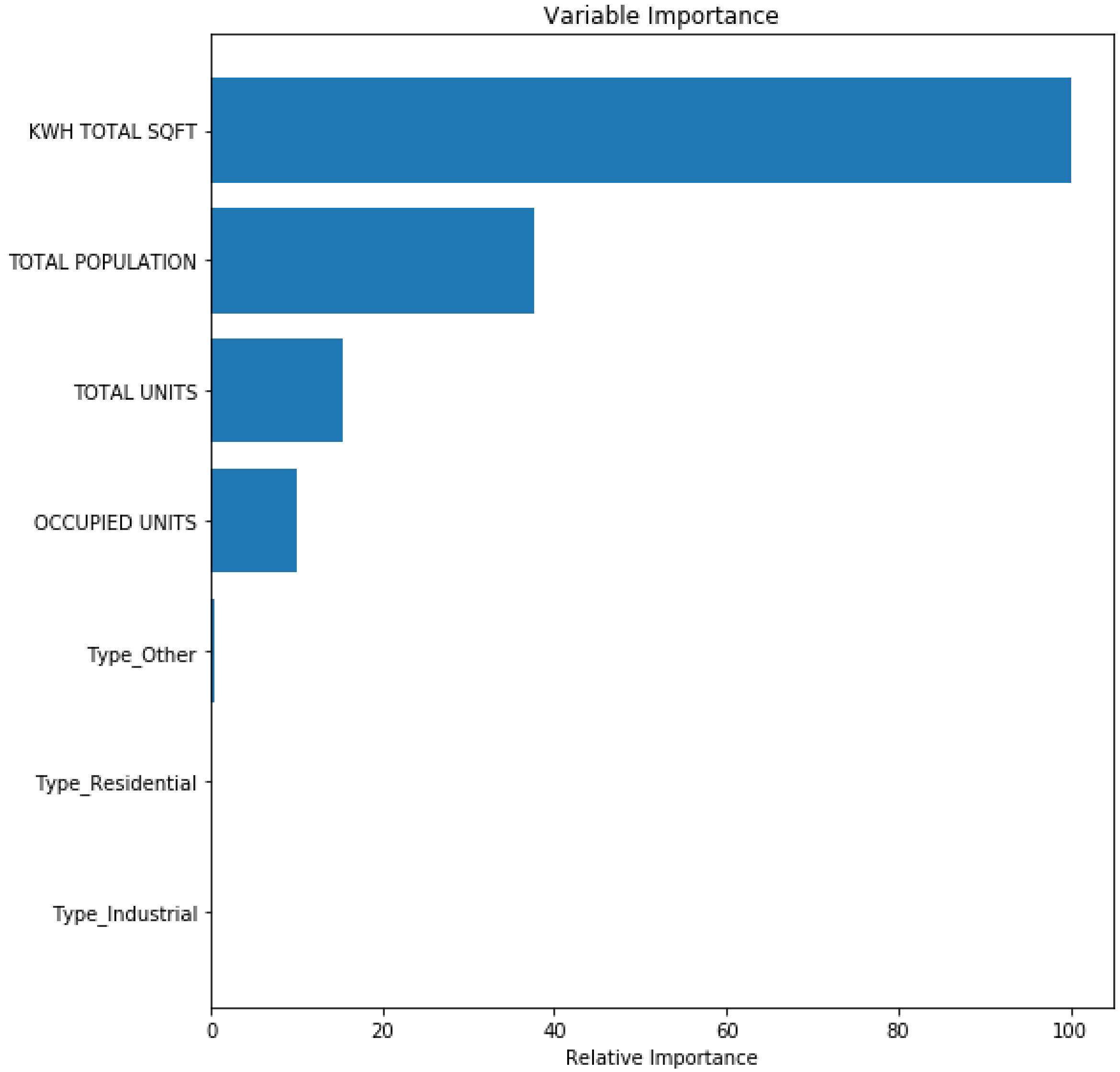
Out[51]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=10, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [52]:
```
#accuracy
regr.score(X_test2, y_test2)
```

Out[52]: 0.6700268740235923

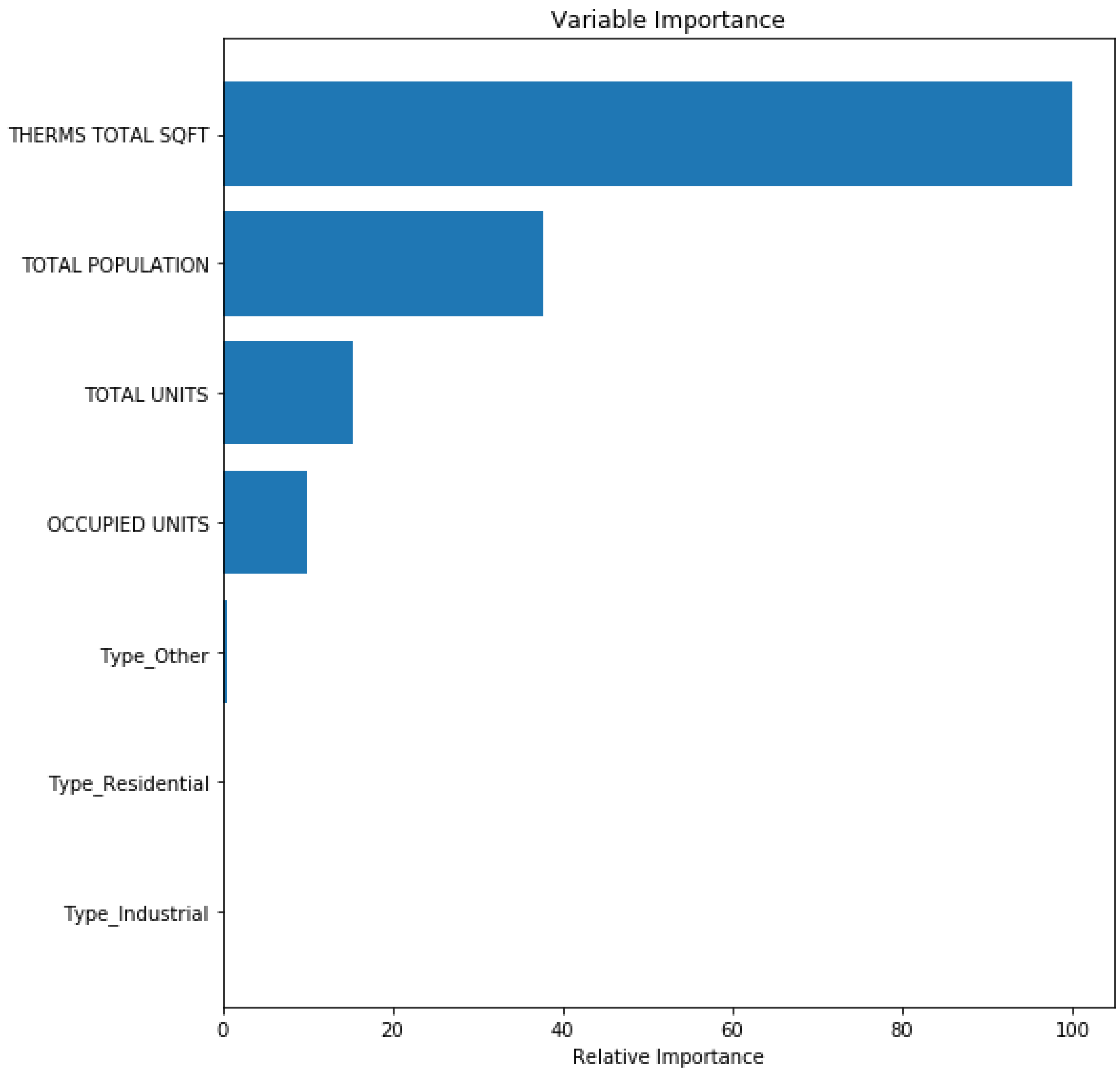
6.2 Importance of Variable

In [53]:
```
#counting how many times a feature is used over the course of many decision trees.
feature_importance = regr.feature_importances_
plt.figure(figsize=(15,8))
# Make importances relative to max importance.
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.tight_layout()
plt.show()
```



KWH total Sqft is the varible with the most importance. Is used over the course of many decision trees

In [54]:
```
#counting how many times a feature is used over the course of many decision trees.
feature_importance = regr.feature_importances_
plt.figure(figsize=(15,8))
# Make importances relative to max importance.
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X2.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.tight_layout()
plt.show()
```



6 -Conclusion

Models:

- In this capstone two regression models were used to show how well each perform in predicting the target. The first model was the Regression Model OLS (Ordinary least squares) and the second model was the Random Forest Regression.
- The model prediction scores where very low, but out of both model i will go with the OLS regression model. It had the best scores. From the graph many graphs were not predicted and way off the line. Also, the the Mean Squared error was very high, which shows the model does not predict very well the targets. Target 1, Total KWH, was predicted the best out of the two. This could mean there were higher consumption of KWH, and therefore the predictions were higher.

What is it useful for:

The model can show the prediction of what type of building will use what amount of energy/gas consumption based on SQFT, units, population, average housesize, and along with other features. It can help people who are looking into spending less energy/gas as much as possible, or what will be there average consumption base on what they have or are planning to have.

In [ ]:

```

```