CAPSTONE 2

Supervised Learning

Research Question

What is the prediction on the total consumption of gas and electricity (the sum of both), based on the building type?

- 2 Regression models
- Target: is going to be the total KWH plus the total THERMS
- Variables: is going to be the top correlated non-month variables with target plus some other low correlation features so there will be less of an overfitting. Also building type will be the non-numeric (categorical) feature.
  - THERMS TOTAL SQFT, KWH TOTAL SQFT, THERMS SQFT MAXIMUM 2010, KWH SQFT MAXIMUM 2010, ZERO KWH ACCOUNTS, KWH MAXIMUM 2010,THERM MAXIMUM 2010, AVERAGE HOUSESIZE, OCCUPIED UNITS, TOTAL POPULATION
  - 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

# 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]:
```
#unique values
df.BUILDING_TYPE.unique()
```

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

As shown above there is 77 missing values and NAN us use to inplace, with about an 11%

We will fill in the NAN values with 'Other'

In [10]:
```
df['BUILDING_TYPE']= df['BUILDING_TYPE'].fillna('Other')
```

In [11]:
```
#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(['COMMUNITY AREA NAME', 'ELECTRICITY ACCOUNTS', 'GAS ACCOUNTS','BUILDING_SUBTYPE','CENSUS
BLOCK'], 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. For later on, to make a new variable with the sum of both and get a total. That total will be our target (y)

In [12]:
```
df['TOTAL THERMS'] = df['TOTAL THERMS'].fillna(0)
df['TOTAL KWH'] = df['TOTAL KWH'].fillna(0)
```

In [13]:
```
#checking the mean for one column
df['KWH FEBRUARY 2010'].mean()
```

Out[13]:
```
17376.51384103959
```

In [14]:
```
#fill NAN values with the mean for its corresponding column
df= df.fillna(df.mean())
df
```

Out[14]:

	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	K OCTOI 2
0	Residential	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
1	Residential	7334.000	7741.000	4214.000	4284.000	2518.000	4273.000	4566.000	2787.000	3357.000	5540.
2	Commercial	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
3	Commercial	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
4	Commercial	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
...	...	...	...	...	...	...	...	...	...	...	
67046	Residential	2705.000	1318.000	1582.000	1465.000	1494.000	2990.000	2449.000	2351.000	1213.000	2174.
67047	Commercial	1005.000	1760.000	1521.000	1832.000	2272.000	2361.000	3018.000	3030.000	2886.000	3833.
67048	Residential	3567.000	3031.000	2582.000	2295.000	7902.000	4987.000	5773.000	3996.000	3050.000	3103.
67049	Residential	1208.000	1055.000	1008.000	1109.000	1591.000	1367.000	1569.000	1551.000	1376.000	1236.
67050	Residential	2717.000	3057.000	2695.000	3793.000	4237.000	5383.000	5544.000	6929.000	5280.000	5971.

67051 rows × 68 columns

In [15]:
```
#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[15]:

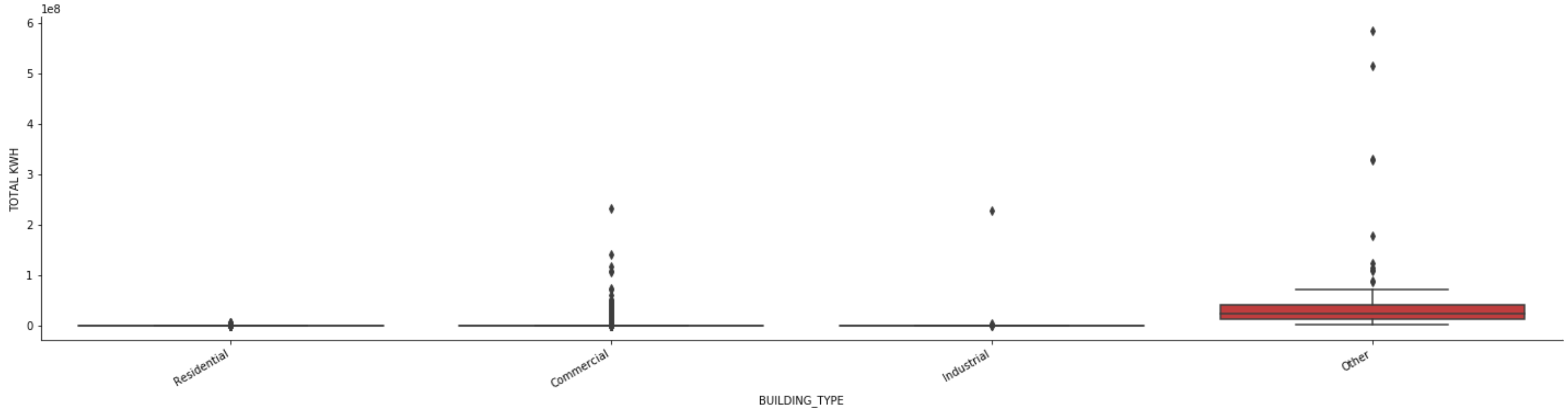
	Total	Percent
OCCUPIED HOUSING UNITS	0	0.000
THERM OCTOBER 2010	0	0.000
TERM APRIL 2010	0	0.000
THERM MAY 2010	0	0.000
THERM JUNE 2010	0	0.000
THERM JULY 2010	0	0.000
THERM AUGUST 2010	0	0.000
THERM SEPTEMBER 2010	0	0.000
THERM NOVEMBER 2010	0	0.000
RENTER-OCCUPIED HOUSING PERCENTAGE	0	0.000

2.2 Outliers

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

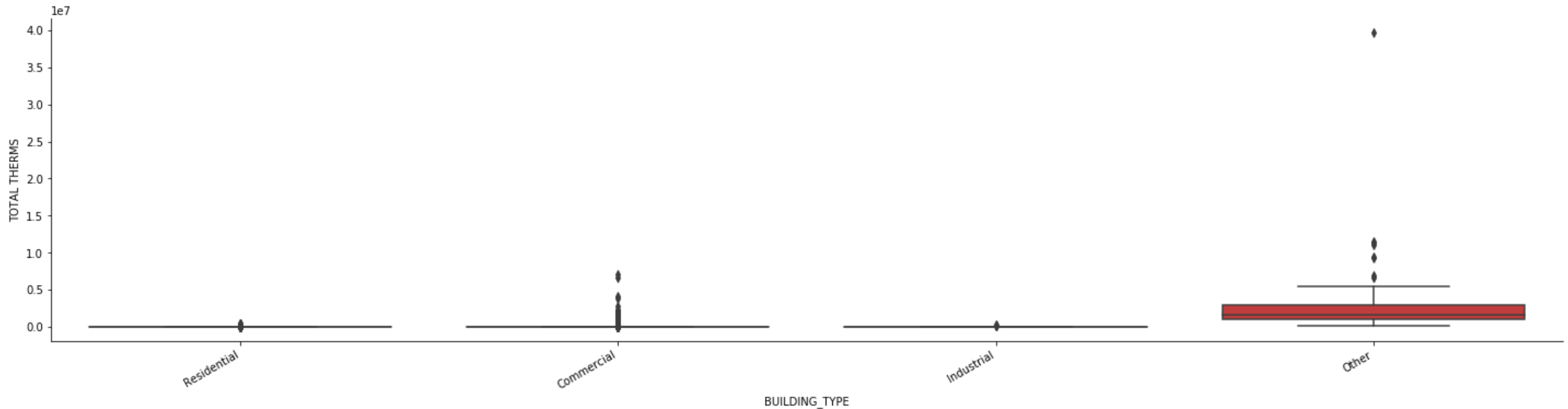
In [16]:
```
chart=sns.catplot(
    data=df,
    y='TOTAL KWH',
    x='BUILDING_TYPE',
    kind='box',
    height=5, # make the plot 5 units high
    aspect=4)
chart.set_xticklabels(rotation=30, horizontalalignment='right')
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x1163df550>



In [17]:
```
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[17]: <seaborn.axisgrid.FacetGrid at 0x12d685390>



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 [18]:
```
# See the numeric variables to use on winsorize
numeric_columns = df.select_dtypes(['int64', 'float64']).columns
```

In [19]:
```
#zcore to clean outliers
from scipy import stats
df[(np.abs(stats.zscore(df[numeric_columns]))<3).all(axis=1)]
```

Out[19]:

	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	K OCTOI 2
0	Residential	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
1	Residential	7334.000	7741.000	4214.000	4284.000	2518.000	4273.000	4566.000	2787.000	3357.000	5540.
2	Commercial	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
3	Commercial	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
4	Commercial	17581.588	17376.514	16242.122	15956.964	19066.228	23004.853	24828.907	22675.264	18564.097	17241.
...	...	...	...	...	...	...	...	...	...	...	
67046	Residential	2705.000	1318.000	1582.000	1465.000	1494.000	2990.000	2449.000	2351.000	1213.000	2174.
67047	Commercial	1005.000	1760.000	1521.000	1832.000	2272.000	2361.000	3018.000	3030.000	2886.000	3833.
67048	Residential	3567.000	3031.000	2582.000	2295.000	7902.000	4987.000	5773.000	3996.000	3050.000	3103.
67049	Residential	1208.000	1055.000	1008.000	1109.000	1591.000	1367.000	1569.000	1551.000	1376.000	1236.
67050	Residential	2717.000	3057.000	2695.000	3793.000	4237.000	5383.000	5544.000	6929.000	5280.000	5971.

65080 rows × 68 columns

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

3.1 Target and Variables

Now that the data is clean, lets pick the target and find some correlation with it to pick some features that will explain better our target

To make the prediction of the total consumption of gas and electricity, we will use the following:

- Target (y): total_kwh_therms(TOTAL KWH + TOTAL THERMS)

In [20]:
```
#make a new variable for our target
df['TOTAL_KWH_THERMS'] = df['TOTAL KWH'] + df ['TOTAL THERMS']
```

3.2 Correlation between Variables and target

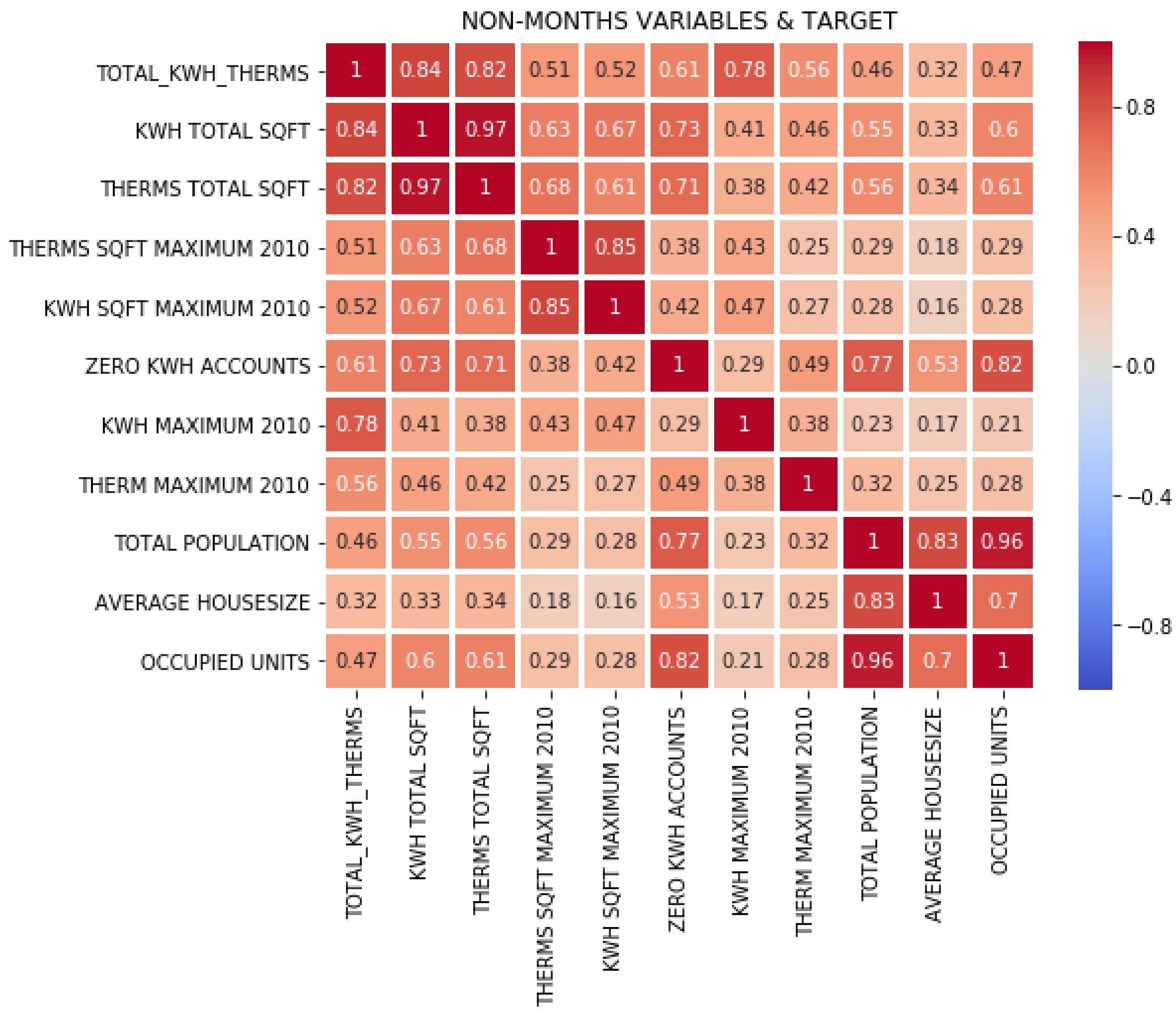
Our threshold for correlation will be .5, which in this case are the top variables that correlate with the target: TOTAL_KWH_THERMS

In [21]:
```
# 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_THERMS']).sort_values(ascending=False).
head(30)
```

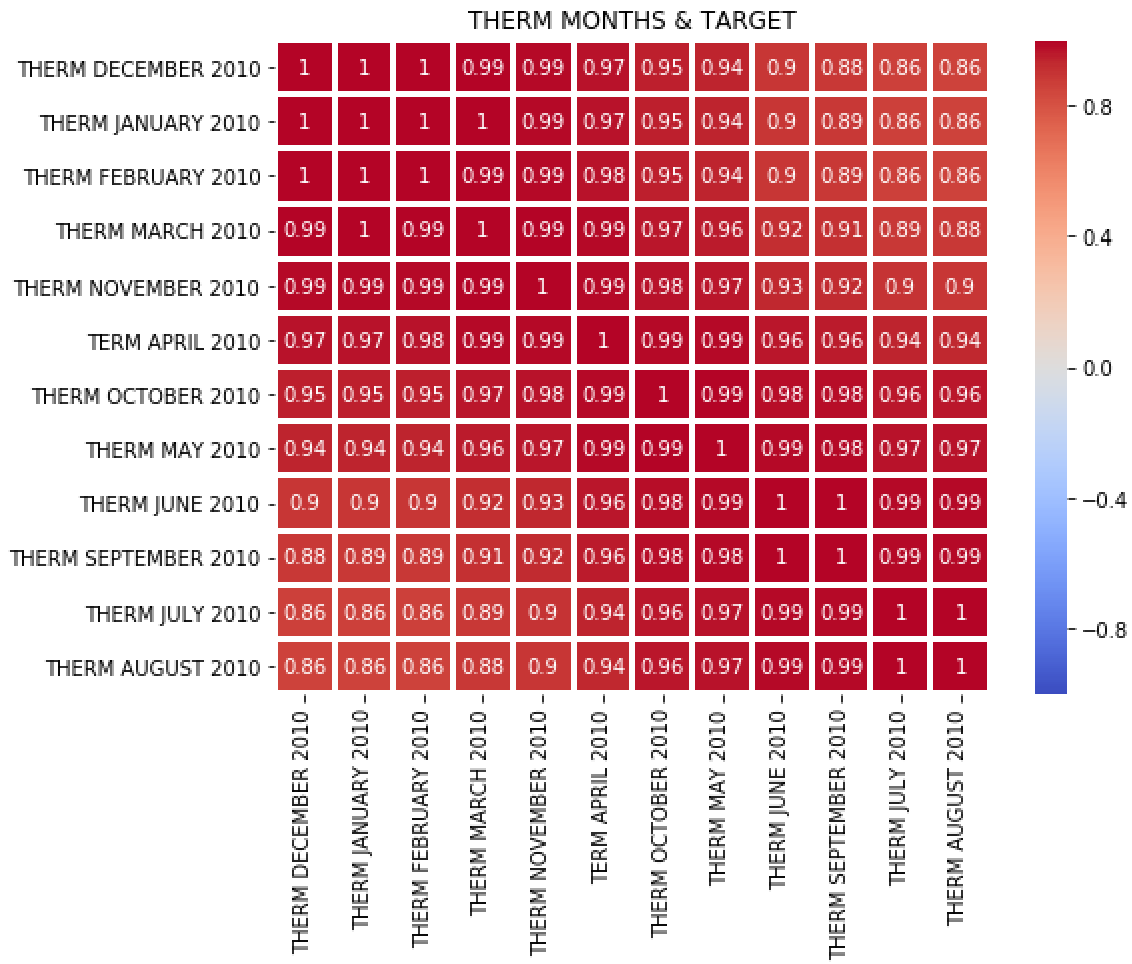
Out[21]:
```
TOTAL_KWH_THERMS       1.000
TOTAL KWH              0.999
KWH APRIL 2010         0.998
KWH MARCH 2010         0.998
KWH OCTOBER 2010       0.997
KWH SEPTEMBER 2010     0.997
KWH MAY 2010           0.996
KWH JUNE 2010          0.995
KWH AUGUST 2010        0.994
KWH NOVEMBER 2010      0.994
KWH FEBRUARY 2010      0.993
KWH JULY 2010          0.992
KWH DECEMBER 2010      0.980
KWH TOTAL SQFT         0.845
THERMS TOTAL SQFT      0.825
KWH MAXIMUM 2010       0.778
THERM DECEMBER 2010    0.746
THERM JANUARY 2010     0.735
THERM FEBRUARY 2010    0.729
THERM MARCH 2010       0.719
THERM NOVEMBER 2010    0.718
TOTAL THERMS           0.712
TERM APRIL 2010        0.688
THERM OCTOBER 2010     0.670
THERM MAY 2010         0.663
THERM JUNE 2010        0.621
THERM SEPTEMBER 2010   0.612
ZERO KWH ACCOUNTS      0.609
THERM JULY 2010        0.598
THERM AUGUST 2010      0.595
Name: TOTAL_KWH_THERMS, dtype: float64
```

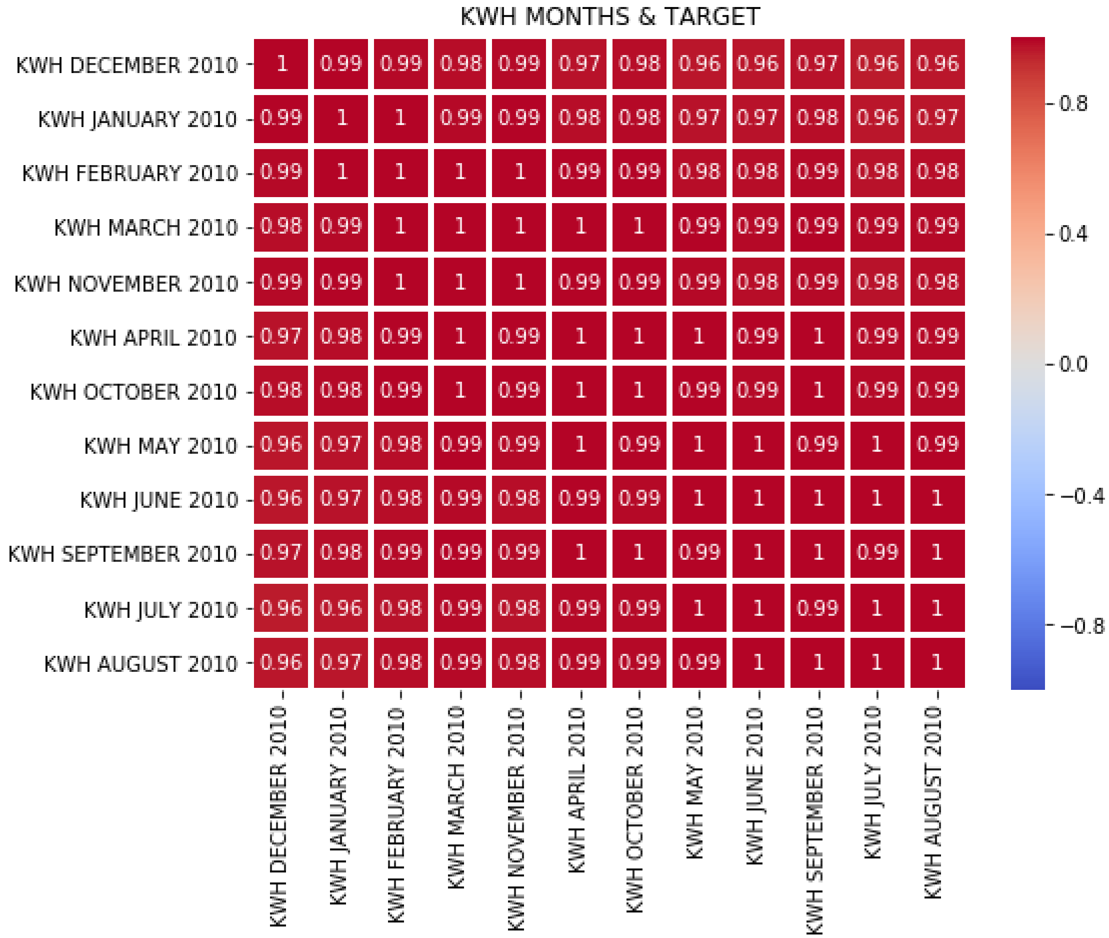
In [22]:
```
#df with each non-month variable
df2= df[['TOTAL_KWH_THERMS','KWH TOTAL SQFT','THERMS TOTAL SQFT','THERMS SQFT MAXIMUM 2010','KWH SQFT MAXIMUM 2010','ZERO KWH ACCOUNTS','KWH MAXIMUM 2010','THERM MAXIMUM 2010','TOTAL POPULATION','AVERAGE HOUSESIZE','OCCUPIED UNITS']]
#df with each month for GAS
df3= df[['THERM DECEMBER 2010','THERM JANUARY 2010','THERM FEBRUARY 2010', 'THERM MARCH 2010','THERM NOVEMBER 2010',
             'TERM APRIL 2010', 'THERM OCTOBER 2010', 'THERM MAY 2010', 'THERM JUNE 2010', 'THERM SEPTEMBER 2010',
            'THERM JULY 2010','THERM AUGUST 2010']]
#df with each month for KWH
df4= df[['KWH DECEMBER 2010','KWH JANUARY 2010','KWH FEBRUARY 2010', 'KWH MARCH 2010','KWH NOVEMBER 2010',
             'KWH APRIL 2010', 'KWH OCTOBER 2010', 'KWH MAY 2010', 'KWH JUNE 2010', 'KWH SEPTEMBER 2010',
            'KWH JULY 2010','KWH AUGUST 2010']]
```

In [23]:
```
fig, ax = plt.subplots(figsize=(8,6))

sns.heatmap(df2.corr(),annot = True,vmin=-1, vmax=1, center= 0, cmap= 'coolwarm',linewidths=2, lineco
lor='white')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_title('NON-MONTHS VARIABLES & TARGET')
plt.show()

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

sns.heatmap(df3.corr(),annot = True,vmin=-1, vmax=1, center= 0, cmap= 'coolwarm',linewidths=2, lineco
lor='white')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_title('THERM MONTHS & TARGET')
plt.show()

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

sns.heatmap(df4.corr(),annot = True,vmin=-1, vmax=1, center= 0, cmap= 'coolwarm',linewidths=2, lineco
lor='white')
bottom, top = ax.get_ylim()
ax.set_title('KWH MONTHS & TARGET')
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



	TOTAL_KWH_THERMS	KWH TOTAL SQFT	THERMS TOTAL SQFT	THERMS SQFT MAXIMUM 2010	KWH SQFT MAXIMUM 2010	ZERO KWH ACCOUNTS	KWH MAXIMUM 2010	THERM MAXIMUM 2010	TOTAL POPULATION	AVERAGE HOUSESIZE	OCCUPIED UNITS
TOTAL_KWH_THERMS	1	0.84	0.82	0.51	0.52	0.61	0.78	0.56	0.46	0.32	0.47
KWH TOTAL SQFT	0.84	1	0.97	0.63	0.67	0.73	0.41	0.46	0.55	0.33	0.6
THERMS TOTAL SQFT	0.82	0.97	1	0.68	0.61	0.71	0.38	0.42	0.56	0.34	0.61
THERMS SQFT MAXIMUM 2010	0.51	0.63	0.68	1	0.85	0.38	0.43	0.25	0.29	0.18	0.29
KWH SQFT MAXIMUM 2010	0.52	0.67	0.61	0.85	1	0.42	0.47	0.27	0.28	0.16	0.28
ZERO KWH ACCOUNTS	0.61	0.73	0.71	0.38	0.42	1	0.29	0.49	0.77	0.53	0.82
KWH MAXIMUM 2010	0.78	0.41	0.38	0.43	0.47	0.29	1	0.38	0.23	0.17	0.21
THERM MAXIMUM 2010	0.56	0.46	0.42	0.25	0.27	0.49	0.38	1	0.32	0.25	0.28
TOTAL POPULATION	0.46	0.55	0.56	0.29	0.28	0.77	0.23	0.32	1	0.83	0.96
AVERAGE HOUSESIZE	0.32	0.33	0.34	0.18	0.16	0.53	0.17	0.25	0.83	1	0.7
OCCUPIED UNITS	0.47	0.6	0.61	0.29	0.28	0.82	0.21	0.28	0.96	0.7	1


THERM MONTHS & TARGET
THERM DECEMBER 2010
THERM JANUARY 2010
THERM FEBRUARY 2010
THERM MARCH 2010
THERM NOVEMBER 2010
TERM APRIL 2010
THERM OCTOBER 2010
THERM MAY 2010
THERM JUNE 2010
THERM SEPTEMBER 2010
THERM JULY 2010
THERM AUGUST 2010
0.8
0.4
0.0
−0.4
−0.8


KWH MONTHS & TARGET
KWH DECEMBER 2010
KWH JANUARY 2010
KWH FEBRUARY 2010
KWH MARCH 2010
KWH NOVEMBER 2010
KWH APRIL 2010
KWH OCTOBER 2010
KWH MAY 2010
KWH JUNE 2010
KWH SEPTEMBER 2010
KWH JULY 2010
KWH AUGUST 2010
0.8
0.4
0.0
−0.4
−0.8

Setting Y and X

- For the X variables, dummies represent the building type and the other varibales that have a high correlation with the target. Those variables will be added as the features to be tested on the models
- Also, three variable with low correlation with the target, to lower the bias and overfitting of the model
    - AVERAGE HOUSESIZE
    - OCCUPIED UNITS
    - TOTAL POPULATION
- The correlation between the choosen variables and target can be seen at the top heat map

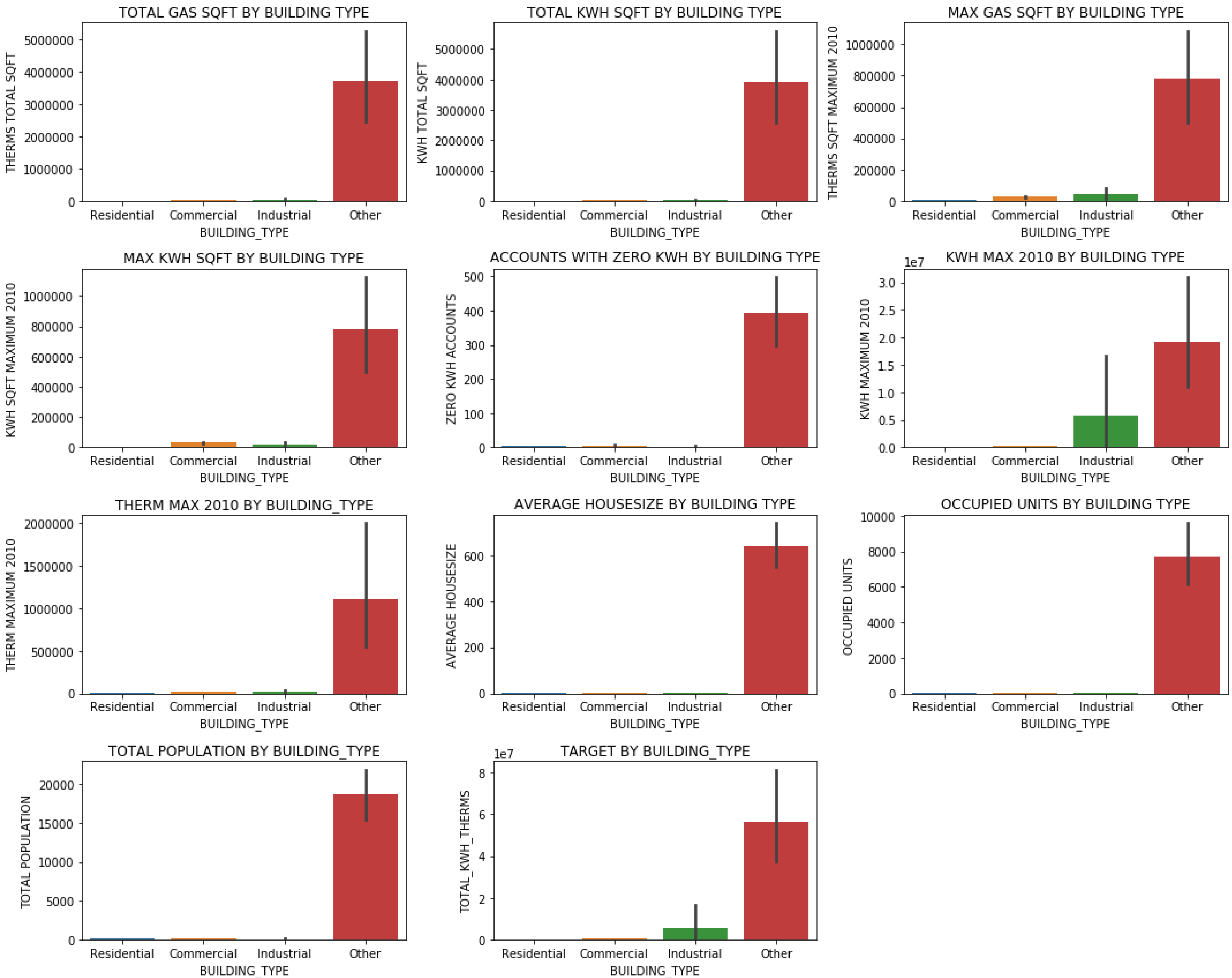
```
    - (The NON-MONTHS VARIABLES & TARGET)
```

In [24]:
```
#setting Y and X
df= pd.concat([df, pd.get_dummies(df.BUILDING_TYPE, prefix= 'Type', drop_first=True)], axis=1)
dummy_column_names = list(pd.get_dummies(df.BUILDING_TYPE, prefix= 'Type', drop_first=True).columns)

X = df[['THERMS TOTAL SQFT', 'KWH TOTAL SQFT', 'THERMS SQFT MAXIMUM 2010','KWH SQFT MAXIMUM 2010','ZE
RO KWH ACCOUNTS','KWH MAXIMUM 2010','THERM MAXIMUM 2010','AVERAGE HOUSESIZE','OCCUPIED UNITS','TOTAL
 POPULATION'] + dummy_column_names ]
Y = df.TOTAL_KWH_THERMS
```

Display bar graphs to show the variables (X) and building type. As well as the target and building type.

In [25]:
```
import seaborn as sns
plt.figure(figsize=(15,12))

#variables
plt.subplot(4,3,1)
sns.barplot(df["BUILDING_TYPE"], df["THERMS TOTAL SQFT"])
plt.title("TOTAL GAS SQFT BY BUILDING TYPE")

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

plt.subplot(4,3,3)
sns.barplot(df["BUILDING_TYPE"], df["THERMS SQFT MAXIMUM 2010"])
plt.title("MAX GAS SQFT BY BUILDING TYPE")

plt.subplot(4,3,4)
sns.barplot(df["BUILDING_TYPE"], df["KWH SQFT MAXIMUM 2010"])
plt.title("MAX KWH SQFT BY BUILDING TYPE")

plt.subplot(4,3,5)
sns.barplot(df["BUILDING_TYPE"], df["ZERO KWH ACCOUNTS"])
plt.title("ACCOUNTS WITH ZERO KWH BY BUILDING TYPE")

plt.subplot(4,3,6)
sns.barplot(df["BUILDING_TYPE"], df["KWH MAXIMUM 2010"])
plt.title("KWH MAX 2010 BY BUILDING TYPE")

plt.subplot(4,3,7)
sns.barplot(df["BUILDING_TYPE"], df["THERM MAXIMUM 2010"])
plt.title("THERM MAX 2010 BY BUILDING_TYPE")

plt.subplot(4,3,8)
sns.barplot(df["BUILDING_TYPE"], df["AVERAGE HOUSESIZE"])
plt.title("AVERAGE HOUSESIZE BY BUILDING TYPE")

plt.subplot(4,3,9)
sns.barplot(df["BUILDING_TYPE"], df["OCCUPIED UNITS"])
plt.title("OCCUPIED UNITS BY BUILDING TYPE")

plt.subplot(4,3,10)
sns.barplot(df["BUILDING_TYPE"], df["TOTAL POPULATION"])
```

```
plt.title("TOTAL POPULATION BY BUILDING_TYPE")

#target
plt.subplot(4,3,11)
sns.barplot(df["BUILDING_TYPE"], df["TOTAL_KWH_THERMS"])
plt.title("TARGET BY BUILDING_TYPE")


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


TOTAL GAS SQFT BY BUILDING TYPE
THERMS TOTAL SQFT
BUILDING_TYPE
TOTAL KWH SQFT BY BUILDING TYPE
KWH TOTAL SQFT
MAX GAS SQFT BY BUILDING TYPE
THERMS SQFT MAXIMUM 2010
MAX KWH SQFT BY BUILDING TYPE
KWH SQFT MAXIMUM 2010
ACCOUNTS WITH ZERO KWH BY BUILDING TYPE
ZERO KWH ACCOUNTS
KWH MAX 2010 BY BUILDING TYPE
KWH MAXIMUM 2010
THERM MAX 2010 BY BUILDING_TYPE
THERM MAXIMUM 2010
AVERAGE HOUSESIZE BY BUILDING TYPE
AVERAGE HOUSESIZE
OCCUPIED UNITS BY BUILDING TYPE
OCCUPIED UNITS
TOTAL POPULATION BY BUILDING_TYPE
TOTAL POPULATION
TARGET BY BUILDING_TYPE
TOTAL_KWH_THERMS
Residential
Commercial
Industrial
Other

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

In [26]:
```
#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)
```

In [27]:
```
print("The number of observations in training set is {}".format(X_train.shape[0]))
print("The number of observations in test set is {}".format(X_test.shape[0]))
```

```
The number of observations in training set is 53640
The number of observations in test set is 13411
```

5- Regression (OLS) Model

5.1 OLS MODEL: CHECK TRAIN RESULT

In [28]:
```
#Train with OLS
import statsmodels.api as sm
X_train_sm = sm.add_constant(X_train)
# We fit an OLS model using statsmodels
results = sm.OLS(y_train, X_train).fit()

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

```
                                OLS Regression Results
=======================================================================================
Dep. Variable:       TOTAL_KWH_THERMS   R-squared (uncentered):                   0.993
Model:                            OLS   Adj. R-squared (uncentered):              0.993
Method:                 Least Squares   F-statistic:                          5.992e+05
Date:                Thu, 26 Mar 2020   Prob (F-statistic):                        0.00
Time:                        21:01:55   Log-Likelihood:                     -7.6356e+05
No. Observations:               53640   AIC:                                  1.527e+06
Df Residuals:                   53627   BIC:                                  1.527e+06
Df Model:                          13
Covariance Type:            nonrobust
============================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
THERMS TOTAL SQFT         1.5882      0.058     27.163      0.000       1.474       1.703
KWH TOTAL SQFT           11.1850      0.054    208.140      0.000      11.080      11.290
THERMS SQFT MAXIMUM 2010 -2.0276      0.068    -29.712      0.000      -2.161      -1.894
KWH SQFT MAXIMUM 2010   -11.4243      0.064   -179.619      0.000     -11.549     -11.300
ZERO KWH ACCOUNTS      1008.5449    153.627      6.565      0.000     707.434    1309.656
KWH MAXIMUM 2010          1.0186      0.001   1284.399      0.000       1.017       1.020
THERM MAXIMUM 2010        1.9846      0.016    127.899      0.000       1.954       2.015
AVERAGE HOUSESIZE      1298.1040    200.655      6.469      0.000     904.819    1691.389
OCCUPIED UNITS        -1817.1780     32.636    -55.680      0.000   -1881.145   -1753.211
TOTAL POPULATION        272.4864     18.690     14.579      0.000     235.853     309.120
Type_Industrial       -8.643e+04   6.31e+04     -1.369      0.171    -2.1e+05    3.73e+04
Type_Other             1.673e+06   1.02e+05     16.386      0.000    1.47e+06    1.87e+06
Type_Residential      -4.671e+04   1930.287    -24.197      0.000   -5.05e+04   -4.29e+04
==============================================================================
Omnibus:                   145615.117   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):      39061447738.730
Skew:                          32.664   Prob(JB):                         0.00
Kurtosis:                    4183.059   Cond. No.                     1.60e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.6e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

/usr/local/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2495: FutureWarning: Method .ptp is deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

R values are high and mostly all p- values are 0, which shows the features do have a significant impact on the prediction of the model

5.2 OLS MODEL: CHECK TRAIN RESULT

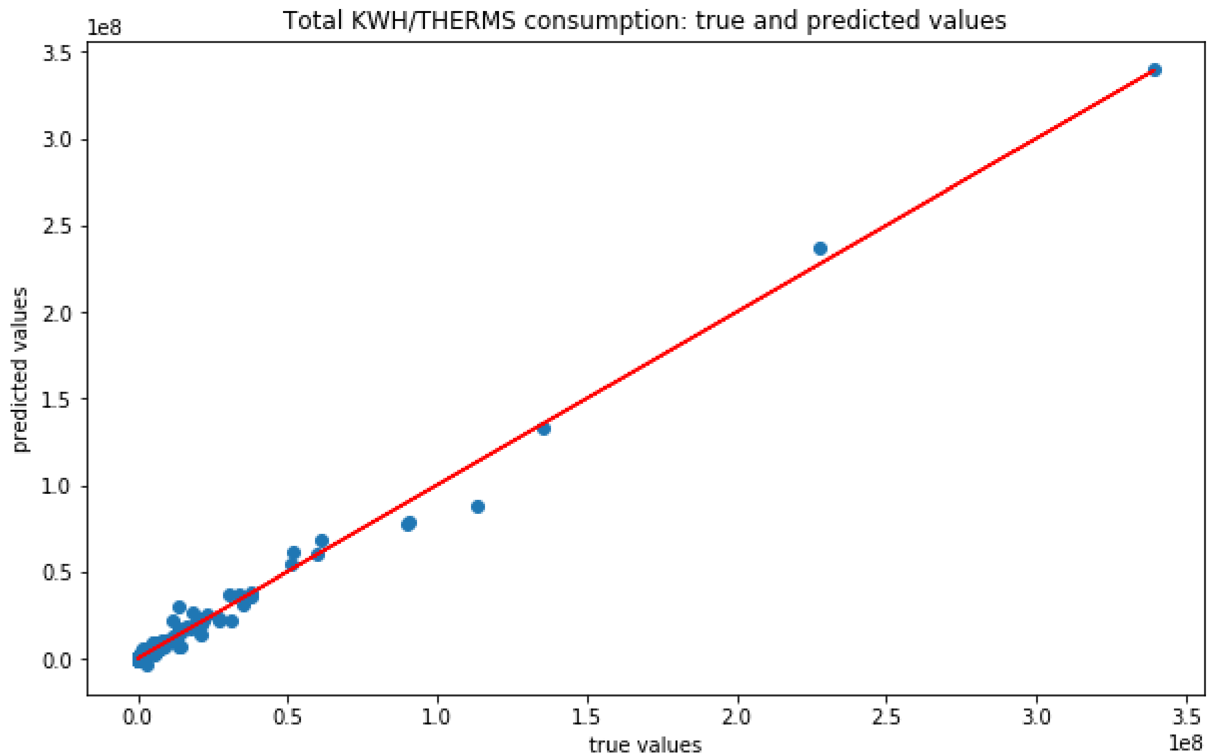
In [29]:
```
#RESULTS FOR TEST OLS
results = sm.OLS(y_test, X_test).fit()
print(results.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:        TOTAL_KWH_THERMS   R-squared (uncentered):                   0.990
Model:                             OLS   Adj. R-squared (uncentered):              0.990
Method:                  Least Squares   F-statistic:                          1.074e+05
Date:                 Thu, 26 Mar 2020   Prob (F-statistic):                        0.00
Time:                         21:01:55   Log-Likelihood:                     -1.9257e+05
No. Observations:                13411   AIC:                                  3.852e+05
Df Residuals:                    13398   BIC:                                  3.853e+05
Df Model:                           13
Covariance Type:             nonrobust
============================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
THERMS TOTAL SQFT           4.0382      0.238     16.974      0.000       3.572       4.504
KWH TOTAL SQFT              6.8721      0.240     28.594      0.000       6.401       7.343
THERMS SQFT MAXIMUM 2010   -3.3946      0.268    -12.680      0.000      -3.919      -2.870
KWH SQFT MAXIMUM 2010      -8.1427      0.264    -30.811      0.000      -8.661      -7.625
ZERO KWH ACCOUNTS       -2079.5657    352.699     -5.896      0.000   -2770.906   -1388.226
KWH MAXIMUM 2010            1.0452      0.002    574.505      0.000       1.042       1.049
THERM MAXIMUM 2010          1.4889      0.070     21.278      0.000       1.352       1.626
AVERAGE HOUSESIZE        2922.0961    305.118      9.577      0.000    2324.022    3520.170
OCCUPIED UNITS          -1170.9602     51.349    -22.804      0.000   -1271.612   -1070.308
TOTAL POPULATION          613.3339     19.804     30.970      0.000     574.515     652.152
Type_Industrial         -1.411e+06   1.56e+05     -9.029      0.000    -1.72e+06    -1.1e+06
Type_Other               1.233e+06   1.59e+05      7.777      0.000     9.22e+05    1.54e+06
Type_Residential        -6.327e+04   4345.208    -14.561      0.000    -7.18e+04   -5.48e+04
==============================================================================
Omnibus:                    26713.462   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):       1236610492.233
Skew:                          14.949   Prob(JB):                         0.00
Kurtosis:                    1490.318   Cond. No.                     1.02e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.02e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

R squared perform good in test, but perform less in training. Still with high score, and not significantly lower than training score. Most of all the p-values did stay under .05, meaning all did have a meaningful effect on prediction of the model.

5.3 DISPLAY PREDICTION OF VALUES

In [30]:
```
#display graph to show predicted values and true values
# We are making predictions here
y_preds = results.predict(X_test)
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()
```



As shown, the closer the dots are to the red line the better the predictions are. Our model does a good job predicting low and high values, it seems like some middle vales are not predict as well as the other values but not that far off.

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

In [31]:
```
#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[31]:
```
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 [32]:
```
#accuracy
regr.score(X_test, y_test)
```

Out[32]:
```
0.8214506057239951
```

2nd RF

In [33]:
```
#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_train, y_train)
```

Out[33]:
```
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 [34]:
```
#accuracy
regr.score(X_test, y_test)
```

Out[34]:
```
0.946522518490696
```

3rd RF

In [35]:
```
#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[35]:
```
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 [36]:
```
#accuracy
regr.score(X_test, y_test)
```

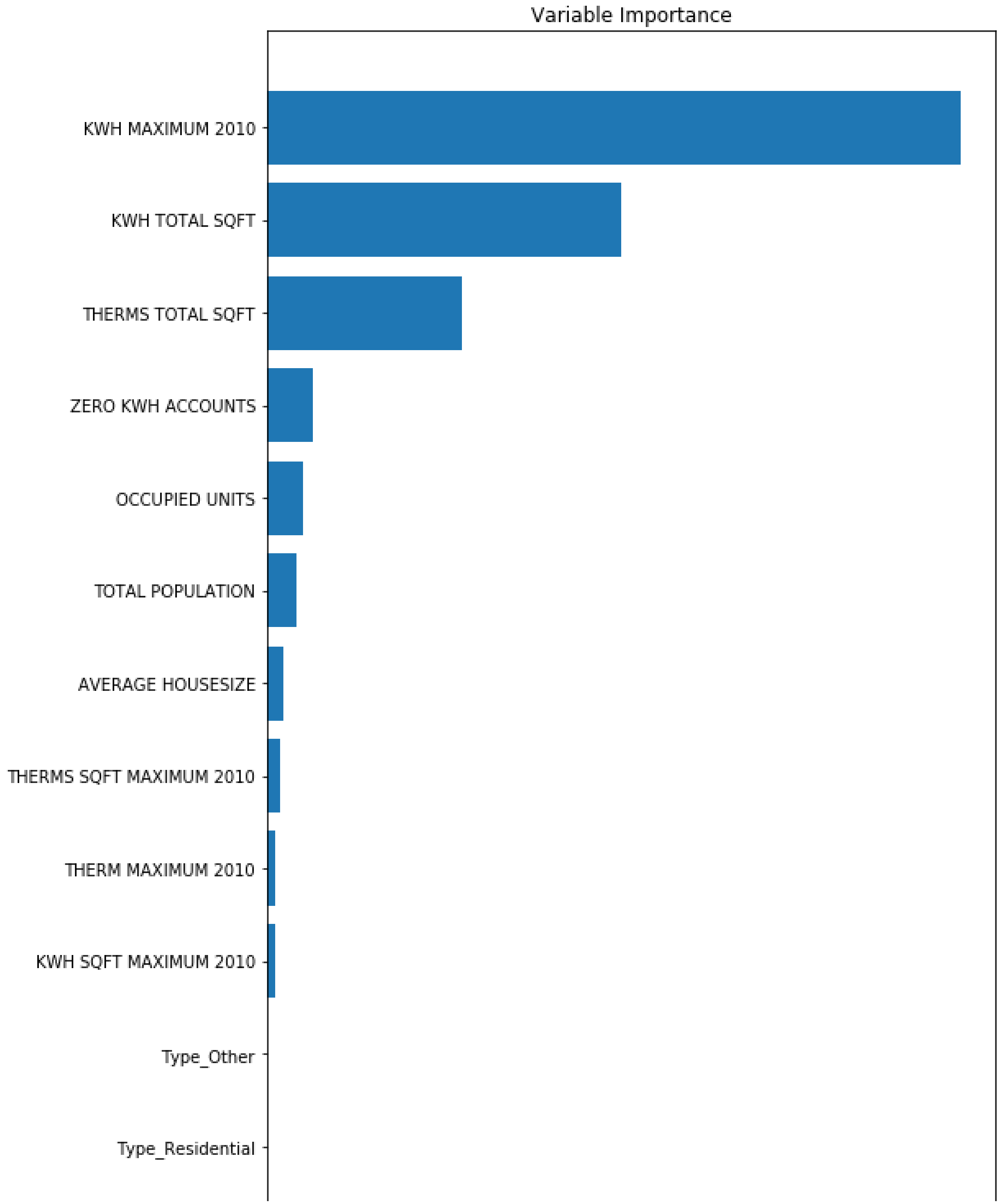
Out[36]: 0.9536594472750529

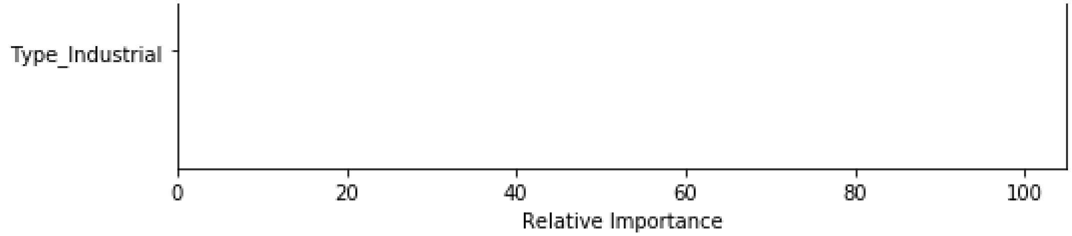
The more depth (level of leafs on each tree) that was added to the model, the greater the accuracy scores. The third model perform the best with 95% accuracy. Making model number 3 best choice.

6.2 Importance of Variable

Show the measure of the importance of each features, by counting how many times a feature is used over the course of many decision trees.

In [37]:
```
feature_importance = regr.feature_importances_
plt.figure(figsize=(15,12))
# 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()
```


Variable Importance
KWH MAXIMUM 2010
KWH TOTAL SQFT
THERMS TOTAL SQFT
ZERO KWH ACCOUNTS
OCCUPIED UNITS
TOTAL POPULATION
AVERAGE HOUSESIZE
THERMS SQFT MAXIMUM 2010
THERM MAXIMUM 2010
KWH SQFT MAXIMUM 2010
Type_Other
Type_Residential



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.
- Although both models perform very well with high scores of prediction, for this data set I will pick the Regression Model OLS, because it was the one with the highest accuracy score. The graph even shows how well the dots were close to the red line, showing the model did predict mostly all data points.

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.