SMART Home IOT power consumption

January 28, 2020

1 SMART Home IOT Power Consumption Prediction

1.1 Introduction

The purpose of this workbook is to be able to predict the energy consumption of the house against the weather conditions.

The data are gathered using IOT sensors and have been pushed into a MongoDB Atlas Database.

The dataset contains the readings with a time span of 1 minute of house appliances in kW from a smart meter and weather conditions of that particular region.

The dataset can be reteived here: https://www.kaggle.com/taranvee/smart-home-dataset-with-weather-information

The column description is the following: * time * use [kW] : Total energy consumption * gen [kW] : Total energy generated by means of solar or other power generation resources * House overall [kW] : overall house energy consumption * Dishwasher [kW] : energy consumed by specific appliance * Furnace 1 [kW] : energy consumed by specific appliance * Furnace 2 [kW] : energy consumed by specific appliance * Home office [kW] : energy consumed by specific appliance * Fridge [kW] : energy consumed by specific appliance * Wine cellar [kW] : energy consumed by specific appliance * Garage door [kW] : energy consumed by specific appliance * Kitchen 12 [kW] : energy consumption in kitchen 1 * Kitchen 14 [kW] : energy consumption in kitchen 2 * Kitchen 38 [kW] : energy consumption in kitchen 3 * Barn [kW] : energy consumed by specific appliance * Well [kW] : energy consumed by specific appliance * Microwave [kW] : energy consumed by specific appliance * Living room [kW] : energy consumption in Living room * Solar [kW] : Solar power generation * temperature * icon * humidity * visibility * summary * apparentTemperature * pressure * windSpeed * cloudCover * windBearing * precipIntensity * dewPoint * precipProbability

1.2 Importing the necessary libraries

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima_model import ARIMA
import pymongo
from pymongo import MongoClient
import pprint
```

```
import json
import time
from sklearn.metrics import r2_score, median_absolute_error, mean_absolute_error
from sklearn.metrics import median_absolute_error, mean_squared_error,␣
 ↪mean_squared_log_error

%matplotlib inline
```

1.3 Setup common parameters

[2]:
```
plt.rcParams["figure.figsize"] = (25,5)
```

1.4 Loading the datasets

Connection to MongoDB database, and gathering the data.

[3]:
```
client = MongoClient()
# Point the client at mongo URI
client = MongoClient('Connection string')
# Select database
db = client['IOT']
# Select the collection within the database
smart_home = db.smart_home
# Convert entire collection to Pandas dataframe
dataset = pd.DataFrame(list(smart_home.find()))
```

1.5 Data visualisation

Let's have a look at name and data type of each feature (column)

[4]:
```
tmp_str = "Feature(attribute)     DataType";␣
 ↪print(tmp_str+"\n"+"-"*len(tmp_str))
print(dataset.dtypes)
```

```
Feature(attribute)     DataType
-------------------------------
_id                    object
time                   object
use [kW]               object
gen [kW]               object
House overall [kW]     object
Dishwasher [kW]        object
Furnace 1 [kW]         object
Furnace 2 [kW]         object
Home office [kW]       object
Fridge [kW]            object
Wine cellar [kW]       object
```

```
Garage door [kW]       object
Kitchen 12 [kW]        object
Kitchen 14 [kW]        object
Kitchen 38 [kW]        object
Barn [kW]              object
Well [kW]              object
Microwave [kW]         object
Living room [kW]       object
Solar [kW]             object
temperature            object
icon                   object
humidity               object
visibility             object
summary                object
apparentTemperature    object
pressure               object
windSpeed              object
cloudCover             object
windBearing            object
precipIntensity        object
dewPoint               object
precipProbability      object
dtype: object
```

Lets see the dimensionality of the DataFrame.

[5]:
```
print("Shape of the data: {} --> n_rows = {}, n_cols = {}".format(dataset.
↪shape, dataset.shape[0],dataset.shape[1]))
```

```
Shape of the data: (503911, 33) --> n_rows = 503911, n_cols = 33
```

Lets print the first lines of the dataset to get a vision of the data

[6]:
```
dataset.head(10)
```

[6]:
```
                        _id        time  use [kW]    gen [kW]  \
0  5e1f30de7445e5de64eff9ec  1451624402  0.931817  0.00346667
1  5e1f30de7445e5de64eff9ed  1451624400  0.932833  0.00348333
2  5e1f30de7445e5de64eff9ee  1451624403   1.02205  0.00348333
3  5e1f30de7445e5de64eff9ef  1451624404    1.1394  0.00346667
4  5e1f30de7445e5de64eff9f0  1451624405   1.39187  0.00343333
5  5e1f30de7445e5de64eff9f1  1451624407    1.4319  0.00341667
6  5e1f30de7445e5de64eff9f2  1451624408    1.6273  0.00341667
7  5e1f30de7445e5de64eff9f3  1451624409   1.73538  0.00341667
8  5e1f30de7445e5de64eff9f4  1451624410   1.58508  0.00341667
9  5e1f30de7445e5de64eff9f5  1451624401  0.934333  0.00346667

  House overall [kW] Dishwasher [kW] Furnace 1 [kW] Furnace 2 [kW]  \
0           0.931817        1.67e-05         0.0207      0.0623167
```

```
1           0.932833        3.33e-05         0.0207      0.0619167
2            1.02205        1.67e-05         0.1069      0.0685167
3             1.1394     0.000133333       0.236933      0.0639833
4            1.39187     0.000283333        0.50325      0.0636667
5             1.4319         0.00025       0.477867       0.178633
6             1.6273     0.000183333        0.44765         0.3657
7            1.73538        1.67e-05        0.17155         0.6825
8            1.58508           5e-05         0.0221       0.678733
9           0.934333               0      0.0207167      0.0638167

  Home office [kW] Fridge [kW]  … visibility summary apparentTemperature  \
0         0.446067    0.123533  …         10   Clear               29.26
1         0.442633     0.12415  …         10   Clear               29.26
2         0.446583    0.123133  …         10   Clear               29.26
3         0.446533     0.12285  …         10   Clear               29.26
4         0.447033      0.1223  …         10   Clear               29.26
5         0.444283      0.1218  …         10   Clear               29.26
6         0.441467    0.121617  …         10   Clear               29.26
7         0.438733    0.121633  …         10   Clear               29.26
8           0.4402     0.12145  …         10   Clear               29.26
9         0.444067       0.124  …         10   Clear               29.26

  pressure windSpeed  cloudCover windBearing precipIntensity dewPoint  \
0  1016.91      9.18  cloudCover         282               0     24.4
1  1016.91      9.18  cloudCover         282               0     24.4
2  1016.91      9.18  cloudCover         282               0     24.4
3  1016.91      9.18  cloudCover         282               0     24.4
4  1016.91      9.18  cloudCover         282               0     24.4
5  1016.91      9.18  cloudCover         282               0     24.4
6  1016.91      9.18  cloudCover         282               0     24.4
7  1016.91      9.18  cloudCover         282               0     24.4
8  1016.91      9.18  cloudCover         282               0     24.4
9  1016.91      9.18  cloudCover         282               0     24.4

  precipProbability
0                 0
1                 0
2                 0
3                 0
4                 0
5                 0
6                 0
7                 0
8                 0
9                 0

[10 rows x 33 columns]
```

And the latest rows of the dataset.

[7]:
```
dataset.tail(10)
```

[7]:
```
                             _id        time  use [kW]    gen [kW]  \
503901  5e1f31387445e5de64f7aa49  1452128301   1.53738  0.00318333
503902  5e1f31387445e5de64f7aa4a  1452128302   1.55182      0.0032
503903  5e1f31387445e5de64f7aa4b  1452128303   1.59962  0.00321667
503904  5e1f31387445e5de64f7aa4c  1452128304   1.60887  0.00321667
503905  5e1f31387445e5de64f7aa4d  1452128305   1.60123  0.00318333
503906  5e1f31387445e5de64f7aa4e  1452128306   1.59933  0.00323333
503907  5e1f31387445e5de64f7aa4f  1452128307   1.92427  0.00321667
503908  5e1f31387445e5de64f7aa50  1452128308    1.9782  0.00321667
503909  5e1f31387445e5de64f7aa51           \
503910  5e1f31387445e5de64f7aa52  1452128309   1.99095  0.00323333

       House overall [kW] Dishwasher [kW] Furnace 1 [kW] Furnace 2 [kW]  \
503901            1.53738     0.000133333      0.0216833       0.642733
503902            1.55182           5e-05         0.0562       0.624783
503903            1.59962         6.67e-05      0.0892167        0.63865
503904            1.60887         3.33e-05         0.1143       0.623283
503905            1.60123           5e-05      0.0852667       0.642417
503906            1.59933           5e-05       0.104017       0.625033
503907            1.92427         3.33e-05       0.422383       0.637733
503908             1.9782           5e-05       0.495667       0.620367
503909
503910            1.99095           5e-05         0.4947       0.634133

       Home office [kW] Fridge [kW]  … visibility     summary  \
503901        0.0420333  0.00528333  …       8.74  Light Rain
503902          0.04175     0.00525  …       8.74  Light Rain
503903          0.04175  0.00561667  …       8.74  Light Rain
503904        0.0418167  0.00521667  …       8.74  Light Rain
503905        0.0417833  0.00526667  …       8.74  Light Rain
503906          0.04175  0.00523333  …       8.74  Light Rain
503907        0.0420333  0.00498333  …       8.74  Light Rain
503908           0.0421  0.00533333  …       8.74  Light Rain
503909                               …
503910           0.0421  0.00491667  …       8.74  Light Rain

       apparentTemperature pressure windSpeed cloudCover windBearing  \
503901               29.45  1011.49      6.72       0.31         186
503902               29.45  1011.49      6.72       0.31         186
503903               29.45  1011.49      6.72       0.31         186
503904               29.45  1011.49      6.72       0.31         186
503905               29.45  1011.49      6.72       0.31         186
503906               29.45  1011.49      6.72       0.31         186
503907               29.45  1011.49      6.72       0.31         186
```

```
503908                29.45  1011.49      6.72       0.31          186
503909
503910                29.45  1011.49      6.72       0.31          186

       precipIntensity dewPoint precipProbability
503901          0.0101    31.27              0.51
503902          0.0101    31.27              0.51
503903          0.0101    31.27              0.51
503904          0.0101    31.27              0.51
503905          0.0101    31.27              0.51
503906          0.0101    31.27              0.51
503907          0.0101    31.27              0.51
503908          0.0101    31.27              0.51
503909
503910          0.0101    31.27              0.51

[10 rows x 33 columns]
```

We can see that the row 503909 is invalid. Lets remove it.

[8]:
```
dataset = dataset.drop(503909)
dataset.tail(5)
```

[8]:
```
                             _id        time use [kW]    gen [kW]  \
503905  5e1f31387445e5de64f7aa4d  1452128305  1.60123  0.00318333
503906  5e1f31387445e5de64f7aa4e  1452128306  1.59933  0.00323333
503907  5e1f31387445e5de64f7aa4f  1452128307  1.92427  0.00321667
503908  5e1f31387445e5de64f7aa50  1452128308   1.9782  0.00321667
503910  5e1f31387445e5de64f7aa52  1452128309  1.99095  0.00323333

       House overall [kW] Dishwasher [kW] Furnace 1 [kW] Furnace 2 [kW]  \
503905            1.60123           5e-05      0.0852667       0.642417
503906            1.59933           5e-05       0.104017       0.625033
503907            1.92427        3.33e-05       0.422383       0.637733
503908             1.9782           5e-05       0.495667       0.620367
503910            1.99095           5e-05         0.4947       0.634133

       Home office [kW] Fridge [kW]  … visibility     summary  \
503905        0.0417833  0.00526667  …       8.74  Light Rain
503906          0.04175  0.00523333  …       8.74  Light Rain
503907        0.0420333  0.00498333  …       8.74  Light Rain
503908           0.0421  0.00533333  …       8.74  Light Rain
503910           0.0421  0.00491667  …       8.74  Light Rain

       apparentTemperature pressure windSpeed cloudCover windBearing  \
503905                29.45  1011.49      6.72       0.31          186
503906                29.45  1011.49      6.72       0.31          186
```

```
503907                29.45  1011.49      6.72        0.31          186
503908                29.45  1011.49      6.72        0.31          186
503910                29.45  1011.49      6.72        0.31          186

       precipIntensity dewPoint precipProbability
503905          0.0101    31.27              0.51
503906          0.0101    31.27              0.51
503907          0.0101    31.27              0.51
503908          0.0101    31.27              0.51
503910          0.0101    31.27              0.51

[5 rows x 33 columns]
```

Lets cleanup the column name by removing the units [kW] in the column name.

[9]:
```python
dataset.columns = [col.replace(' [kW]', '') for col in dataset.columns]
dataset.columns
```

[9]:
```
Index(['_id', 'time', 'use', 'gen', 'House overall', 'Dishwasher', 'Furnace 1',
       'Furnace 2', 'Home office', 'Fridge', 'Wine cellar', 'Garage door',
       'Kitchen 12', 'Kitchen 14', 'Kitchen 38', 'Barn', 'Well', 'Microwave',
       'Living room', 'Solar', 'temperature', 'icon', 'humidity', 'visibility',
       'summary', 'apparentTemperature', 'pressure', 'windSpeed', 'cloudCover',
       'windBearing', 'precipIntensity', 'dewPoint', 'precipProbability'],
      dtype='object')
```

Lets aggregate the following columns: * 'Furnace 1' and 'Furnace 2' in a new column named totalFurnace * 'Kitchen 12', 'Kitchen 14' and 'Kitchen 38' in a new column named avgKitchen

[10]:
```python
dataset['sum_Furnace'] = dataset[['Furnace 1','Furnace 2']].sum(axis=1)
dataset['avg_Kitchen'] = dataset[['Kitchen 12','Kitchen 14','Kitchen 38']].
 ↪mean(axis=1)
dataset.head(5)
```

[10]:
```
                         _id        time       use         gen House overall  \
0  5e1f30de7445e5de64eff9ec  1451624402  0.931817  0.00346667      0.931817
1  5e1f30de7445e5de64eff9ed  1451624400  0.932833  0.00348333      0.932833
2  5e1f30de7445e5de64eff9ee  1451624403   1.02205  0.00348333       1.02205
3  5e1f30de7445e5de64eff9ef  1451624404    1.1394  0.00346667        1.1394
4  5e1f30de7445e5de64eff9f0  1451624405   1.39187  0.00343333       1.39187

    Dishwasher Furnace 1  Furnace 2 Home office    Fridge  …  \
0     1.67e-05    0.0207  0.0623167    0.446067  0.123533  …
1     3.33e-05    0.0207  0.0619167    0.442633   0.12415  …
2     1.67e-05    0.1069  0.0685167    0.446583  0.123133  …
3  0.000133333  0.236933  0.0639833    0.446533   0.12285  …
4  0.000283333   0.50325  0.0636667    0.447033    0.1223  …
```

```
  apparentTemperature pressure windSpeed  cloudCover windBearing  \
0               29.26  1016.91      9.18  cloudCover         282
1               29.26  1016.91      9.18  cloudCover         282
2               29.26  1016.91      9.18  cloudCover         282
3               29.26  1016.91      9.18  cloudCover         282
4               29.26  1016.91      9.18  cloudCover         282

  precipIntensity dewPoint precipProbability sum_Furnace avg_Kitchen
0               0     24.4                 0    0.083017    0.000206
1               0     24.4                 0    0.082617    0.000189
2               0     24.4                 0    0.175417    0.000217
3               0     24.4                 0    0.300917    0.000261
4               0     24.4                 0    0.566917    0.000350

[5 rows x 35 columns]
```

By Looking at the time column, we can observe that it is stored as UNIX timestamp. As this is time series records, lets figure out when took place the first record.

[11]:
```
print(' start ' , time.strftime('%Y-%m-%d %H:%M:%S', time.
 ↪localtime(int(dataset['time'].iloc[0]))))
```

```
 start  2016-01-01 06:00:02
```

Now that we have the first timestamp of the series, and as we also from the dataset publisher that the frequency of publication is 1 minute, then we can convert it in a way we can read it.

[12]:
```
time_index = pd.date_range('2016-01-01 06:00', periods=len(dataset), ␣
 ↪freq='min')
time_index = pd.DatetimeIndex(time_index)
dataset = dataset.set_index(time_index)
dataset = dataset.drop(['time'], axis=1)
dataset.iloc[np.r_[0:5,-5:0]].iloc[:,0]
```

[12]:
```
2016-01-01 06:00:00    5e1f30de7445e5de64eff9ec
2016-01-01 06:01:00    5e1f30de7445e5de64eff9ed
2016-01-01 06:02:00    5e1f30de7445e5de64eff9ee
2016-01-01 06:03:00    5e1f30de7445e5de64eff9ef
2016-01-01 06:04:00    5e1f30de7445e5de64eff9f0
2016-12-16 04:25:00    5e1f31387445e5de64f7aa4d
2016-12-16 04:26:00    5e1f31387445e5de64f7aa4e
2016-12-16 04:27:00    5e1f31387445e5de64f7aa4f
2016-12-16 04:28:00    5e1f31387445e5de64f7aa50
2016-12-16 04:29:00    5e1f31387445e5de64f7aa52
Name: _id, dtype: object
```

As we have 503910, we should have nearly a year of data points. (There are 525 600 minutes in a year). Lets verify this.

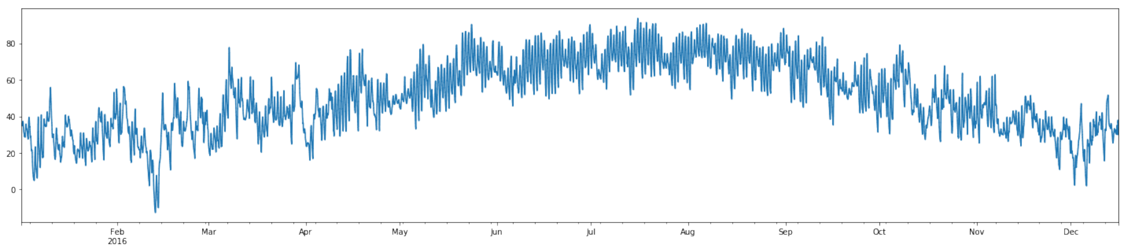
[13]:
```
dataset.tail(5)
```

[13]:
```
                                          _id      use         gen  \
2016-12-16 04:25:00  5e1f31387445e5de64f7aa4d  1.60123  0.00318333
2016-12-16 04:26:00  5e1f31387445e5de64f7aa4e  1.59933  0.00323333
2016-12-16 04:27:00  5e1f31387445e5de64f7aa4f  1.92427  0.00321667
2016-12-16 04:28:00  5e1f31387445e5de64f7aa50   1.9782  0.00321667
2016-12-16 04:29:00  5e1f31387445e5de64f7aa52  1.99095  0.00323333

                    House overall Dishwasher  Furnace 1 Furnace 2 Home office  \
2016-12-16 04:25:00       1.60123      5e-05  0.0852667  0.642417   0.0417833
2016-12-16 04:26:00       1.59933      5e-05   0.104017  0.625033     0.04175
2016-12-16 04:27:00       1.92427   3.33e-05   0.422383  0.637733   0.0420333
2016-12-16 04:28:00        1.9782      5e-05   0.495667  0.620367      0.0421
2016-12-16 04:29:00       1.99095      5e-05     0.4947  0.634133      0.0421

                         Fridge Wine cellar  … apparentTemperature pressure  \
2016-12-16 04:25:00  0.00526667  0.00866667  …               29.45  1011.49
2016-12-16 04:26:00  0.00523333  0.00843333  …               29.45  1011.49
2016-12-16 04:27:00  0.00498333  0.00846667  …               29.45  1011.49
2016-12-16 04:28:00  0.00533333  0.00823333  …               29.45  1011.49
2016-12-16 04:29:00  0.00491667  0.00813333  …               29.45  1011.49

                    windSpeed cloudCover windBearing precipIntensity dewPoint  \
2016-12-16 04:25:00      6.72       0.31         186          0.0101    31.27
2016-12-16 04:26:00      6.72       0.31         186          0.0101    31.27
2016-12-16 04:27:00      6.72       0.31         186          0.0101    31.27
2016-12-16 04:28:00      6.72       0.31         186          0.0101    31.27
2016-12-16 04:29:00      6.72       0.31         186          0.0101    31.27

                    precipProbability sum_Furnace avg_Kitchen
2016-12-16 04:25:00              0.51    0.727683    0.000211
2016-12-16 04:26:00              0.51    0.729050    0.000200
2016-12-16 04:27:00              0.51    1.060117    0.000200
2016-12-16 04:28:00              0.51    1.116033    0.000217
2016-12-16 04:29:00              0.51    1.128833    0.000217

[5 rows x 34 columns]
```

Therefore a daily resampling should be more accurante to visualize the data. Lets verify this by looking at the temperature.

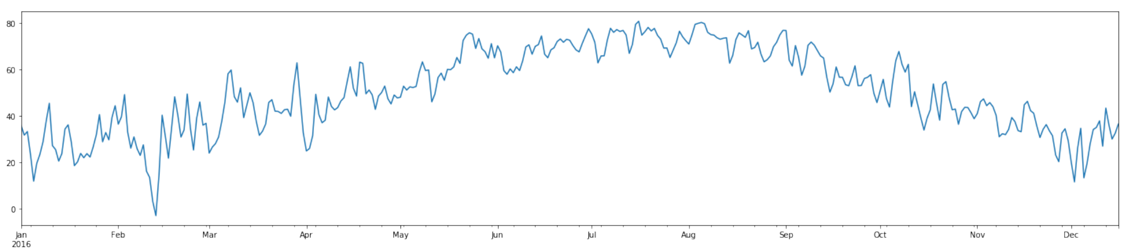
[14]:
```
# Apply numeric values to temperature column to avoid errors during the resample
dataset['temperature'] = dataset['temperature'].apply(pd.to_numeric,␣
 ↪errors='coerce')
dataset['temperature'].plot()
```

[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1c231be4d0>



[15]:
```
dataset['temperature'].resample(rule='D').mean().plot()
```

[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1c25231210>



1.6 Data cleanup

Lets have a lookup at the different columns of this dataset and see if some cleanup needs to be done and where. Have a look at the proposed columns within this dataset.

[16]:
```
dataset.columns
```

[16]:
```
Index(['_id', 'use', 'gen', 'House overall', 'Dishwasher', 'Furnace 1',
       'Furnace 2', 'Home office', 'Fridge', 'Wine cellar', 'Garage door',
       'Kitchen 12', 'Kitchen 14', 'Kitchen 38', 'Barn', 'Well', 'Microwave',
       'Living room', 'Solar', 'temperature', 'icon', 'humidity', 'visibility',
       'summary', 'apparentTemperature', 'pressure', 'windSpeed', 'cloudCover',
       'windBearing', 'precipIntensity', 'dewPoint', 'precipProbability',
       'sum_Furnace', 'avg_Kitchen'],
      dtype='object')
```

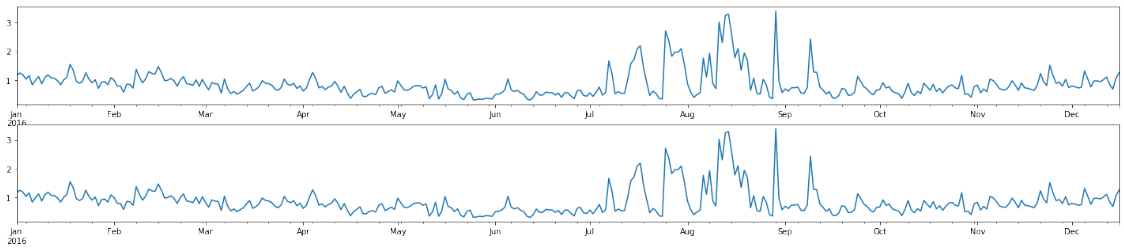
Lets start by removing the columns which have been aggregated (Kitchen and Furnace).

[17]:
```
dataset = dataset.drop(['Kitchen 12','Kitchen 14','Kitchen 38'], axis=1)
dataset = dataset.drop(['Furnace 1','Furnace 2'], axis=1)
```

Lets look at the column 'use', and 'House overall' columns. The definition of the columns are the following ones and looks close: * use [kW] : Total energy consumption * House overall [kW] : overall house energy consumption

[18]:
```
fig, axes = plt.subplots(nrows=2, ncols=1)
# Apply numeric values to columns to avoid errors during the resample
dataset['use'] = dataset['use'].apply(pd.to_numeric, errors='coerce')
dataset['House overall'] = dataset['House overall'].apply(pd.to_numeric,␣
↪errors='coerce')
# Graph de columns
dataset['use'].resample('D').mean().plot(ax=axes[0])
dataset['House overall'].resample('D').mean().plot(ax=axes[1])
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c252809d0>
```



The data looks similar. Lets delete the column 'House overall'.

[19]:
```
dataset = dataset.drop(columns=['House overall'])
```

Lets look at the 'Summary' and 'icons' column.

[20]:
```
dataset['summary'].value_counts()
```

[20]:
```
Clear                       376730
Partly Cloudy                62268
Light Rain                   27368
Drizzle                      10370
Overcast                      6041
Rain                          5169
Mostly Cloudy                 4548
Light Snow                    4323
Flurries                      1789
Breezy                        1561
Snow                          1152
Breezy and Partly Cloudy      1041
Foggy                          974
Rain and Breezy                174
Heavy Snow                     171
```

```
Flurries and Breezy            115
Dry                             58
Breezy and Mostly Cloudy        58
Name: summary, dtype: int64
```

[21]:
```
dataset['icon'].value_counts()
```

[21]:
```
clear-night            194536
clear-day              182252
rain                    43081
partly-cloudy-day       39492
partly-cloudy-night     27324
snow                     7550
cloudy                   6041
wind                     2660
fog                       974
Name: icon, dtype: int64
```

As this is not numerial values, lets remove them from the dataset. In case of a full machine learning, we could use these 2 columns by mapping the text fields into numerical values to provide a more acurrate model.

[22]:
```
dataset = dataset.drop(columns=['icon'])
dataset = dataset.drop(columns=['summary'])
```

Lets analyse the quality of the differents numeric fields.

[23]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 503910 entries, 2016-01-01 06:00:00 to 2016-12-16 04:29:00
Freq: T
Data columns (total 26 columns):
_id                      503910 non-null object
use                      503910 non-null float64
gen                      503910 non-null object
Dishwasher               503910 non-null object
Home office              503910 non-null object
Fridge                   503910 non-null object
Wine cellar              503910 non-null object
Garage door              503910 non-null object
Barn                     503910 non-null object
Well                     503910 non-null object
Microwave                503910 non-null object
Living room              503910 non-null object
Solar                    503910 non-null object
temperature              503910 non-null float64
humidity                 503910 non-null object
```

```
visibility             503910 non-null object
apparentTemperature    503910 non-null object
pressure               503910 non-null object
windSpeed              503910 non-null object
cloudCover             503910 non-null object
windBearing            503910 non-null object
precipIntensity        503910 non-null object
dewPoint               503910 non-null object
precipProbability      503910 non-null object
sum_Furnace            503910 non-null float64
avg_Kitchen            503910 non-null float64
dtypes: float64(4), object(22)
memory usage: 103.8+ MB
```

[24]:
```
dataset['gen'].unique()
```

[24]:
```
array([0.003466667, 0.003483333, 0.003433333, …, 0.25275, 0.1532,
       0.2099], dtype=object)
```

[25]:
```
dataset['Dishwasher'].unique()
```

[25]:
```
array([1.67e-05, 3.33e-05, 0.000133333, …, 1.37925, 1.368333333,
       1.136833333], dtype=object)
```

[26]:
```
dataset['Home office'].unique()
```

[26]:
```
array([0.446066667, 0.442633333, 0.446583333, …, 0.26505, 0.233166667,
       0.378083333], dtype=object)
```

[27]:
```
dataset['Fridge'].unique()
```

[27]:
```
array([0.123533333, 0.12415, 0.123133333, …, 0.078716667, 0.060083333,
       0.053383333], dtype=object)
```

[28]:
```
dataset['Wine cellar'].unique()
```

[28]:
```
array([0.006983333, 0.00685, 0.006716667, …, 0.097733333, 0.05885,
       0.058466667], dtype=object)
```

[29]:
```
dataset['Garage door'].unique()
```

[29]:
```
array([0.013083333, 0.013, 0.012783333, …, 0.1201, 0.1993, 0.0168],
      dtype=object)
```

[30]:
```
dataset['Barn'].unique()
```

[30]: array([0.031516667, 0.03135, 0.0315, …, 0.054983333, 0.0535,
0.022433333], dtype=object)

[31]:
```
dataset['Well'].unique()
```

[31]: array([0.001, 0.001016667, 0.001033333, …, 1.593616667, 1.498583333,
0.866816667], dtype=object)

[32]:
```
dataset['Microwave'].unique()
```

[32]: array([0.004066667, 0.0042, 0.004116667, …, 0.987516667, 0.601666667,
0.002533333], dtype=object)

[33]:
```
dataset['Living room'].unique()
```

[33]: array([0.00165, 0.001516667, 0.001616667, …, 0.05915, 0.299183333,
0.048783333], dtype=object)

[34]:
```
dataset['Solar'].unique()
```

[34]: array([0.003466667, 0.003483333, 0.003433333, …, 0.25275, 0.1532,
0.2099], dtype=object)

[35]:
```
dataset['humidity'].unique()
```

[35]: array([0.62, 0.61, 0.64, 0.65, 0.66, 0.68, 0.59, 0.58, 0.6, 0.7, 0.63,
0.56, 0.53, 0.51, 0.48, 0.47, 0.55, 0.57, 0.72, 0.73, 0.67, 0.52,
0.54, 0.69, 0.76, 0.8, 0.78, 0.49, 0.46, 0.45, 0.4, 0.34, 0.31,
0.5, 0.74, 0.75, 0.36, 0.29, 0.24, 0.22, 0.25, 0.35, 0.44, 0.71,
0.77, 0.79, 0.33, 0.3, 0.27, 0.81, 0.84, 0.83, 0.85, 0.82, 0.86,
0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.43, 0.41, 0.39, 0.37, 0.42,
0.38, 0.32, 0.28, 0.93, 0.21, 0.2, 0.26, 0.23, 0.18, 0.17, 0.19,
0.94, 0.96, 0.95, 0.15, 0.14, 0.13, 0.16, 0.97, 0.98], dtype=object)

[36]:
```
dataset['visibility'].unique()
```

[36]: array([10, 9.07, 9.9, 9.05, 8.52, 8.06, 9.8, 9.81, 9.91, 9.76, 9.72, 9.37,
9.58, 9.75, 9.62, 9.03, 9.54, 9.73, 9.92, 9.86, 9.85, 9.94, 9.7,
9.82, 9.96, 8.67, 9.87, 9.88, 9.74, 9.71, 8.84, 8.86, 8.78, 7.62,
7.66, 8.63, 9.42, 9.55, 9.61, 9.63, 9.43, 9.27, 9.22, 9.17, 9.09,
9.16, 8.95, 8.49, 8.55, 7.75, 8.32, 7.98, 8.74, 8.64, 6.64, 4.78,
4.3, 3.47, 2.91, 4.25, 4.71, 5.28, 5.15, 6.15, 7.77, 9.15, 9.38,
3.38, 1.92, 3.9, 6.33, 7.52, 6.78, 8.16, 6.6, 5.42, 9.06, 9.78,
9.26, 9.5, 7.5, 8.82, 9.21, 8.75, 8.93, 9.08, 8.34, 7.87, 7.24,
8.48, 9.97, 9.84, 9.83, 9, 8.66, 6.17, 5.2, 6.93, 7.33, 7.28, 4.77,
3.43, 4.2, 6.3, 9.67, 9.64, 8.89, 9.33, 8.09, 7.06, 8.47, 8.71,
7.45, 6.67, 5.29, 7.22, 7.58, 8.83, 9.66, 9.12, 9.68, 9.14, 8.42,

7.71, 7.54, 9.13, 9.46, 9.28, 9.52, 9.69, 9.23, 8.97, 9.36, 8.43,
8.44, 7.93, 7.84, 7.1, 6.65, 6.73, 6.61, 7.12, 7.7, 9.95, 9.77,
9.25, 9.04, 9.31, 9.57, 9.6, 8.39, 8.96, 6.39, 8.1, 7.94, 8.08,
9.1, 9.79, 9.48, 8.88, 5.61, 6.06, 4.23, 3.03, 2.87, 4.6, 6.22,
6.53, 7.51, 9.29, 8.45, 8.92, 7.47, 3.97, 3.09, 1.47, 0.83, 0.84,
2.37, 2.46, 7.82, 9.49, 9.47, 9.35, 9.34, 8.35, 8.87, 7.6, 8.26,
7.65, 7.72, 7.96, 2.92, 2.75, 0.96, 1.33, 1.48, 1.17, 1.59, 1.86,
2.28, 4.1, 7.02, 7.31, 7.91, 5.17, 4.42, 5.87, 5.7, 8.8, 8.94,
5.32, 4.75, 3.55, 1.99, 1.02, 1.3, 1.95, 3.52, 7.76, 8.37, 7.79,
8.72, 8.91, 8.62, 8.24, 8.73, 9.02, 7.9, 8.17, 8.59, 9.18, 9.53,
7.59, 7.21, 6.38, 5.26, 6.68, 4.85, 2.67, 1.39, 1.91, 2.48, 2.8,
3.69, 3.74, 4.04, 4.43, 6.08, 7.04, 4.12, 4.24, 3.87, 3.95, 3.79,
4.32, 7.14, 4.9, 3.34, 3.29, 4.46, 8.21, 9.51, 9.24, 7.68, 7.01,
6.85, 5.22, 5.9, 6.81, 6.46, 6.84, 7.18, 4.59, 4.33, 5.65, 4.96,
4.83, 3.44, 3.01, 4.72, 3.81, 2.7, 2.95, 3.08, 3.78, 3.75, 2.94,
3.48, 4.08, 4.47, 4.94, 4.38, 7.49, 8.46, 8.9, 7.92, 8.56, 7.44,
8.11, 6.57, 5.18, 4.81, 5.58, 5.44, 8.65, 4.67, 5.67, 9.01, 9.44,
9.45, 9.32, 9.56, 9.11, 7.95, 7.89, 6.34, 8.01, 6.14, 8.25, 9.59,
9.65, 7.53, 8.28, 7.78, 8.38, 8.05, 8.57, 8.81, 6.45, 5.64, 5.71,
6.11, 5.72, 6.25, 6.21, 5.93, 3.73, 4.8, 5.08, 5.36, 5.98, 6.28,
6.99, 2.69, 2.47, 2.08, 0.98, 1.46, 2.82, 6.91, 9.41, 5.6, 4.98,
4.4, 2.5, 4.13, 4.34, 4.68, 9.3, 5.59, 4.92, 3.83, 3.13, 3.94,
3.98, 3.56, 3.63, 4.07, 5.83, 9.89, 7.41, 4.18, 3.21, 3.3, 4.48,
5.04, 4.86, 6.9, 8.3, 6.44, 7.64, 2.43, 2.88, 3.11, 5.11, 2.76,
1.15, 5.52, 5.56, 1.08, 2.38, 1.26, 2.2, 5.07, 6.47, 6.29, 6.23,
4.27, 4.35, 8.99, 8.98, 8.5, 9.4, 5.85, 5.01, 7.88, 8.19, 7.55,
5.14, 4.44, 8.29, 8.14, 6.86, 7.39, 9.39, 9.2, 6.02, 6.27, 6.66,
8.23, 8.31, 7.2, 4.39, 4.36, 4.61, 7.37, 6.82, 6.09, 5.5, 3.99,
7.08, 8.58, 7.81, 7.13, 5.82, 5.51, 6.97, 6.42, 6.12, 6.56, 5.39,
9.19, 8.02, 7.74, 8, 8.03, 8.53, 9.93, 6.98, 7.99, 7.38, 7.4, 8.68,
7.63, 5.13, 8.4, 5.94, 4.99, 6.55, 6.89, 4.76, 5.21, 3.72, 8.13,
8.69, 5.66, 5.57, 5.16, 4.88, 4.05, 3.86, 5.05, 7.25, 6.52, 8.2,
4.79, 3.02, 3.4, 2.25, 2.27, 1.34, 1.29, 4.45, 8.33, 8.79, 7.17,
6.03, 6.79, 8.7, 5.75, 7.69, 8.6, 8.51, 7.83, 3.7, 3.16, 2.65,
3.42, 7.35, 5.38, 4.69, 6.24, 8.27, 6.4, 5.53, 6.74, 7.57, 3.51,
4.06, 2.17, 1.55, 2.86, 4.37, 7.03, 8.77, 8.15, 6.76, 6.2, 4.5,
3.96, 7.3, 6.75, 7.05, 7.23, 6.96, 4.28, 5.95, 6.95, 7.46, 6.26,
6.36, 6.1, 8.54, 7.86, 5.46, 5.37, 5.1, 7.61, 7.34, 3.91, 2.97,
4.62, 6.37, 6.5, 6.51, 8.36, 7.11, 7.09, 7.42, 8.76, 5.24, 7.73,
8.04, 5.86, 5.55, 5.48, 4.02, 6.69, 4.89, 6.01, 6.87, 2.9, 7.15,
4.21, 8.61, 8.41, 7.56, 8.18, 7.43, 6.63, 7.85, 7.8, 7.67, 6.48,
7.48, 6.13, 5.31, 5.97, 5.4, 5.89, 4.93, 2.89, 2.09, 3.14, 2.34,
2.57, 2, 5.68, 5.03, 4.15, 5.47, 4.58, 6.94, 6.05, 5.91, 8.07,
5.74, 4.91, 7.32, 8.22, 6.35, 7.26, 7.27, 5.69, 6.41, 5.8, 4.57,
5.3, 5, 6.88, 7, 5.34, 2.93, 3.15, 3.65, 7.97, 6.32, 6.18, 5.62,
5.19, 7.07, 5.35, 7.19, 2.1, 1.8, 3.33, 4.49, 5.12, 5.79, 8.12,
5.09, 5.43, 5.99, 4.74, 4.66, 2.45, 2.44, 2.83, 2.77, 3.18, 3.62,

```
       3.27, 2.22, 1.54, 1.42, 1.2, 1.38, 1.41, 1.65, 1.49, 2.14, 6.59,
       3.59, 3.17, 4.11, 3.53, 3.49, 3.82, 4.73, 6, 5.06, 5.02, 6.49,
       4.64, 3.46, 1.87, 1.78, 5.92, 6.54, 4.51, 1.89, 1.32, 2.33, 3.28,
       6.77, 3.05, 2.3, 2.05, 4.01, 4.19, 1.07, 1.05, 1.37, 1.79, 0.54,
       0.34, 0.27, 0.35, 1.21, 4.52, 3.07, 2.31, 1.5, 2.49, 2.41, 3.93,
       3.64, 3.39, 3.06, 2.99, 3.31, 5.96, 2.12, 6.58], dtype=object)
```

[37]:
```
dataset['apparentTemperature'].unique()
```

[37]:
```
array([29.26, 29.4, 28.87, …, 26.16, 21.89, 30], dtype=object)
```

[38]:
```
dataset['pressure'].unique()
```

[38]:
```
array([1016.91, 1016.25, 1015.98, …, 1000.16, 1002.1, 1003.22],
      dtype=object)
```

[39]:
```
dataset['windSpeed'].unique()
```

[39]:
```
array([9.18, 8.29, 8.2, …, 13.58, 14.71, 16.38], dtype=object)
```

[40]:
```
dataset['cloudCover'].unique()
```

[40]:
```
array(['cloudCover', 0.75, 0, 1, 0.31, 0.44, 0.13, 0.19, 0.25, 0.16, 0.21,
       0.15, 0.14, 0.27, 0.28, 0.17, 0.05, 0.1, 0.26, 0.29, 0.11, 0.09,
       0.12, 0.06, 0.02, 0.08, 0.04, 0.35, 0.22, 0.23, 0.54, 0.39, 0.03,
       0.07, 0.76, 0.62, 0.18, 0.79, 0.48, 0.24, 0.57, 0.41, 0.78, 0.2,
       0.77, 0.46, 0.55, 0.01, 0.51, 0.47, 0.5, 0.4, 0.3, 0.43, 0.33, 0.6,
       0.68, 0.66, 0.45, 0.34, 0.52, 0.67, 0.49, 0.37, 0.36, 0.61, 0.38,
       0.42, 0.53, 0.63, 0.32, 0.56, 0.58, 0.72, 0.73, 0.71, 0.64, 0.59],
      dtype=object)
```

[41]:
```
dataset['dewPoint'].unique()
```

[41]:
```
array([24.4, 23.9, 23.39, …, 28.73, 31.01, 31.27], dtype=object)
```

[42]:
```
dataset['precipProbability'].unique()
```

[42]:
```
array([0, 0.02, 0.3, 0.62, 0.74, 0.76, 0.77, 0.81, 0.7, 0.64, 0.56, 0.5,
       0.53, 0.69, 0.54, 0.07, 0.11, 0.31, 0.37, 0.08, 0.03, 0.01, 0.4,
       0.58, 0.61, 0.6, 0.71, 0.18, 0.04, 0.21, 0.06, 0.13, 0.29, 0.57,
       0.67, 0.66, 0.73, 0.79, 0.55, 0.65, 0.59, 0.1, 0.26, 0.15, 0.41,
       0.28, 0.17, 0.2, 0.39, 0.22, 0.42, 0.46, 0.09, 0.05, 0.12, 0.52,
       0.63, 0.24, 0.14, 0.47, 0.83, 0.35, 0.51, 0.44, 0.75, 0.72, 0.84,
       0.82, 0.27, 0.25, 0.48, 0.33, 0.49, 0.36, 0.43, 0.19, 0.32, 0.16,
       0.34, 0.68, 0.38, 0.23, 0.78, 0.45, 0.8], dtype=object)
```

[43]:
```
dataset['windBearing'].unique()
```

[43]: array([282, 285, 281, 265, 268, 260, 259, 258, 255, 238, 239, 272, 273,
       256, 278, 275, 266, 269, 270, 267, 249, 254, 253, 262, 279, 280,
       283, 263, 243, 244, 233, 220, 219, 214, 221, 211, 200, 195, 197,
       198, 196, 207, 213, 218, 215, 246, 250, 264, 252, 261, 217, 294,
       302, 308, 318, 322, 321, 320, 328, 346, 340, 344, 338, 335, 347,
       352, 350, 351, 359, 1, 358, 0, 355, 341, 339, 329, 349, 354, 353,
       324, 306, 232, 248, 226, 191, 228, 231, 257, 274, 202, 206, 201,
       212, 205, 189, 199, 179, 190, 110, 216, 193, 240, 5, 316, 348, 357,
       15, 33, 12, 325, 28, 4, 57, 21, 2, 20, 13, 22, 16, 17, 9, 19, 27,
       26, 40, 37, 39, 41, 52, 54, 49, 43, 59, 36, 45, 64, 55, 30, 62, 56,
       38, 46, 71, 66, 115, 108, 120, 89, 70, 78, 85, 74, 47, 34, 44, 50,
       58, 87, 101, 91, 131, 144, 158, 159, 166, 247, 284, 291, 288, 289,
       293, 290, 286, 271, 276, 242, 164, 153, 210, 169, 161, 160, 171,
       154, 167, 165, 141, 184, 208, 225, 277, 287, 292, 295, 296, 229,
       227, 223, 237, 14, 114, 83, 3, 92, 48, 31, 76, 53, 51, 25, 343,
       336, 330, 298, 303, 305, 307, 313, 304, 312, 319, 10, 24, 7, 356,
       299, 297, 301, 300, 309, 310, 317, 314, 327, 315, 326, 323, 8, 11,
       23, 18, 35, 334, 311, 186, 174, 183, 182, 168, 177, 172, 178, 188,
       185, 192, 173, 155, 135, 147, 187, 194, 176, 156, 143, 181, 157,
       170, 180, 331, 235, 251, 140, 42, 109, 121, 113, 102, 96, 134, 142,
       152, 333, 126, 175, 204, 137, 146, 63, 29, 80, 104, 98, 122, 73,
       99, 234, 236, 209, 222, 241, 86, 132, 106, 32, 77, 117, 345, 342,
       337, 203, 224, 68, 81, 100, 105, 90, 65, 82, 93, 245, 151, 97, 6,
       163, 150, 230, 107, 112, 103, 84, 61, 69, 129, 148, 72, 128, 136,
       149, 95, 79, 123, 88, 60, 116, 94, 139, 332, 162, 118, 127, 67,
       130, 138, 125, 145, 111, 133, 124, 119, 75], dtype=object)

```
[44]: dataset['precipIntensity'].unique()
```

[44]: array([0, 0.0011, 0.0064, 0.0225, 0.0666, 0.0758, 0.0822, 0.1298, 0.0628,
       0.046, 0.0282, 0.0153, 0.0097, 0.0123, 0.0421, 0.0132, 0.0025,
       0.0032, 0.0013, 0.0065, 0.0075, 0.0026, 0.0012, 0.0015, 0.0008,
       0.0078, 0.0168, 0.0217, 0.0206, 0.0432, 0.0496, 0.0415, 0.0199,
       0.0044, 0.0018, 0.0049, 0.0023, 0.0035, 0.0063, 0.0016, 0.001,
       0.016, 0.0152, 0.0353, 0.0326, 0.0336, 0.0597, 0.1031, 0.0898,
       0.0451, 0.0141, 0.0306, 0.0219, 0.0191, 0.0024, 0.0031, 0.0036,
       0.0058, 0.0039, 0.008, 0.0155, 0.0135, 0.0173, 0.0186, 0.006,
       0.0027, 0.0043, 0.0047, 0.0077, 0.0051, 0.0082, 0.0088, 0.0074,
       0.0028, 0.002, 0.0014, 0.012, 0.0029, 0.0033, 0.0116, 0.011,
       0.0205, 0.0119, 0.0246, 0.0251, 0.0174, 0.0055, 0.0038, 0.0045,
       0.0089, 0.0133, 0.0408, 0.1058, 0.179, 0.0609, 0.0122, 0.0071,
       0.0202, 0.004, 0.0009, 0.0037, 0.01, 0.0247, 0.0112, 0.0084,
       0.0061, 0.0134, 0.0113, 0.0114, 0.0285, 0.1316, 0.0753, 0.067,
       0.0327, 0.0616, 0.0832, 0.0456, 0.0521, 0.0477, 0.0652, 0.1259,
       0.182, 0.1431, 0.0541, 0.0139, 0.0167, 0.0059, 0.0102, 0.0056,
       0.0127, 0.0022, 0.0085, 0.0489, 0.0172, 0.018, 0.0281, 0.041,
       0.0636, 0.0605, 0.0019, 0.0104, 0.0128, 0.0163, 0.0195, 0.0184,

```
0.0208, 0.0066, 0.009, 0.0103, 0.031, 0.04, 0.0235, 0.005, 0.0144,
0.0081, 0.0068, 0.0212, 0.0215, 0.0254, 0.0136, 0.0092, 0.0079,
0.0275, 0.0233, 0.0052, 0.0182, 0.0271, 0.0689, 0.0138, 0.0017,
0.0034, 0.003, 0.0073, 0.0083, 0.0175, 0.0213, 0.0242, 0.0096,
0.0146, 0.0228, 0.025, 0.0667, 0.0704, 0.062, 0.0533, 0.0095,
0.0067, 0.0042, 0.0157, 0.0237, 0.0243, 0.0145, 0.0131, 0.0046,
0.0169, 0.0249, 0.0094, 0.0444, 0.0294, 0.0106, 0.0121, 0.007,
0.0183, 0.0149, 0.0021, 0.042, 0.0443, 0.0383, 0.0778, 0.0164,
0.0048, 0.0111, 0.0241, 0.0234, 0.0319, 0.02, 0.0296, 0.0631,
0.0335, 0.017, 0.0365, 0.0563, 0.0555, 0.0257, 0.0192, 0.0093,
0.0324, 0.1517, 0.0808, 0.015, 0.0253, 0.0229, 0.03, 0.0041,
0.0118, 0.0107, 0.0305, 0.0372, 0.0318, 0.0221, 0.0413, 0.0439,
0.0218, 0.0431, 0.0072, 0.0057, 0.0142, 0.0101, 0.0087, 0.0466,
0.0198, 0.0126, 0.0569, 0.0436, 0.0407, 0.0194, 0.0277, 0.0185,
0.0216, 0.0238, 0.0162, 0.0115, 0.0196, 0.0497, 0.0464, 0.0403,
0.0907, 0.1445, 0.0527, 0.0263, 0.0086, 0.0442, 0.0587, 0.1205,
0.1006, 0.0151, 0.0279, 0.0193, 0.0266, 0.0458, 0.0273, 0.021,
0.0307, 0.0877, 0.0681, 0.0124, 0.0091, 0.0062, 0.0252, 0.0256,
0.0293, 0.0201, 0.0462, 0.0529, 0.0197, 0.0248, 0.0147, 0.0076,
0.0069, 0.1313, 0.0659, 0.0161, 0.0108, 0.0488, 0.0362, 0.0209,
0.028, 0.0181, 0.0343, 0.191, 0.1522, 0.0053, 0.013, 0.0696,
0.1039, 0.048, 0.0259, 0.0148, 0.0261, 0.0852, 0.1359, 0.1581,
0.138, 0.065, 0.0166, 0.0129, 0.0295, 0.0679, 0.087, 0.0712,
0.0117, 0.0156, 0.0416, 0.0367, 0.0453, 0.0346, 0.023, 0.019,
0.1097, 0.0938, 0.0371, 0.0224, 0.0098, 0.0552, 0.0317, 0.0664,
0.0828, 0.1513, 0.1201, 0.0304, 0.0054, 0.0772, 0.0751, 0.0409,
0.0812, 0.0494, 0.1241, 0.076, 0.064, 0.0645, 0.0425, 0.0465, 0.06,
0.106, 0.0179, 0.0811, 0.0991, 0.0143, 0.0187, 0.0125, 0.0236,
0.0178, 0.0288, 0.0311, 0.0323, 0.0245, 0.0105, 0.033, 0.0398,
0.0301, 0.0658, 0.0109, 0.0159, 0.0657, 0.0551, 0.0211, 0.0405,
0.0598, 0.0404, 0.0471, 0.0349, 0.0227, 0.0655, 0.0618, 0.0203,
0.0748, 0.0691, 0.0528, 0.0447, 0.0393, 0.0749, 0.1089, 0.0806,
0.0158, 0.0188, 0.0331, 0.0525, 0.0844, 0.1353, 0.077, 0.0342,
0.0485, 0.0558, 0.0492, 0.0276, 0.0647, 0.0765, 0.0486, 0.0272,
0.0391, 0.0389, 0.0434, 0.0591, 0.0707, 0.0368, 0.0344, 0.0417,
0.0648, 0.0467, 0.0315, 0.022, 0.0418, 0.0265], dtype=object)
```

Except ‘cloudCover’ column which has some unexpected values, lets convert all the numerical field type into numerical data type. Removing also all NaN values with the next value found in the dataset.

[45]:
```
dataset['gen'] = dataset['gen'].apply(pd.to_numeric, errors='coerce')
#dataset['gen'].replace([''], method='bfill', inplace=True)
dataset['Dishwasher'] = dataset['Dishwasher'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['Dishwasher'].replace([''], method='bfill', inplace=True)
```

```
dataset['Home office'] = dataset['Home office'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['Home office'].replace([''], method='bfill', inplace=True)
dataset['Fridge'] = dataset['Fridge'].apply(pd.to_numeric, errors='coerce')
#dataset['Fridge'].replace([''], method='bfill', inplace=True)
dataset['Wine cellar'] = dataset['Wine cellar'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['Wine cellar'].replace([''], method='bfill', inplace=True)
dataset['Garage door'] = dataset['Garage door'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['Garage door'].replace([''], method='bfill', inplace=True)
dataset['Barn'] = dataset['Barn'].apply(pd.to_numeric, errors='coerce')
#dataset['Barn'].replace([''], method='bfill', inplace=True)
dataset['Well'] = dataset['Well'].apply(pd.to_numeric, errors='coerce')
#dataset['Well'].replace([''], method='bfill', inplace=True)
dataset['Microwave'] = dataset['Microwave'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['Microwave'].replace([''], method='bfill', inplace=True)
dataset['Living room'] = dataset['Living room'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['Living room'].replace([''], method='bfill', inplace=True)
dataset['Solar'] = dataset['Solar'].apply(pd.to_numeric, errors='coerce')
#dataset['Solar'].replace([''], method='bfill', inplace=True)
dataset['humidity'] = dataset['humidity'].apply(pd.to_numeric, errors='coerce')
#dataset['humidity'].replace([''], method='bfill', inplace=True)
dataset['visibility'] = dataset['visibility'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['visibility'].replace([''], method='bfill', inplace=True)
dataset['apparentTemperature'] = dataset['apparentTemperature'].apply(pd.
↪to_numeric, errors='coerce')
#dataset['apparentTemperature'].replace([''], method='bfill', inplace=True)
dataset['pressure'] = dataset['pressure'].apply(pd.to_numeric, errors='coerce')
#dataset['pressure'].replace([''], method='bfill', inplace=True)
dataset['windSpeed'] = dataset['windSpeed'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['windSpeed'].replace([''], method='bfill', inplace=True)
dataset['dewPoint'] = dataset['dewPoint'].apply(pd.to_numeric, errors='coerce')
#dataset['dewPoint'].replace([''], method='bfill', inplace=True)
dataset['precipProbability'] = dataset['precipProbability'].apply(pd.
↪to_numeric, errors='coerce')
#dataset['precipProbability'].replace([''], method='bfill', inplace=True)
dataset['windBearing'] = dataset['windBearing'].apply(pd.to_numeric,␣
↪errors='coerce')
#dataset['windBearing'].replace([''], method='bfill', inplace=True)
dataset['precipIntensity'] = dataset['precipIntensity'].apply(pd.to_numeric,␣
↪errors='coerce')
```

```
#dataset['precipIntensity'].replace([''], method='bfill', inplace=True)
```

Lets cleanup 'cloudCover' column by replacing the text value CloudCover by the next value which is found in the dataset.

[46]:
```
dataset['cloudCover'].unique()
```

[46]:
```
array(['cloudCover', 0.75, 0, 1, 0.31, 0.44, 0.13, 0.19, 0.25, 0.16, 0.21,
       0.15, 0.14, 0.27, 0.28, 0.17, 0.05, 0.1, 0.26, 0.29, 0.11, 0.09,
       0.12, 0.06, 0.02, 0.08, 0.04, 0.35, 0.22, 0.23, 0.54, 0.39, 0.03,
       0.07, 0.76, 0.62, 0.18, 0.79, 0.48, 0.24, 0.57, 0.41, 0.78, 0.2,
       0.77, 0.46, 0.55, 0.01, 0.51, 0.47, 0.5, 0.4, 0.3, 0.43, 0.33, 0.6,
       0.68, 0.66, 0.45, 0.34, 0.52, 0.67, 0.49, 0.37, 0.36, 0.61, 0.38,
       0.42, 0.53, 0.63, 0.32, 0.56, 0.58, 0.72, 0.73, 0.71, 0.64, 0.59],
      dtype=object)
```

[47]:
```
dataset['cloudCover'].replace(['cloudCover'], method='bfill', inplace=True)
dataset['cloudCover'].replace([''], method='bfill', inplace=True)
dataset['cloudCover'] = dataset['cloudCover'].astype('float')
dataset['cloudCover'].unique()
```

[47]:
```
array([0.75, 0.  , 1.  , 0.31, 0.44, 0.13, 0.19, 0.25, 0.16, 0.21, 0.15,
       0.14, 0.27, 0.28, 0.17, 0.05, 0.1 , 0.26, 0.29, 0.11, 0.09, 0.12,
       0.06, 0.02, 0.08, 0.04, 0.35, 0.22, 0.23, 0.54, 0.39, 0.03, 0.07,
       0.76, 0.62, 0.18, 0.79, 0.48, 0.24, 0.57, 0.41, 0.78, 0.2 , 0.77,
       0.46, 0.55, 0.01, 0.51, 0.47, 0.5 , 0.4 , 0.3 , 0.43, 0.33, 0.6 ,
       0.68, 0.66, 0.45, 0.34, 0.52, 0.67, 0.49, 0.37, 0.36, 0.61, 0.38,
       0.42, 0.53, 0.63, 0.32, 0.56, 0.58, 0.72, 0.73, 0.71, 0.64, 0.59])
```

Lets check that now the datatype are all sets correctly so that we can work on the prediction

[48]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 503910 entries, 2016-01-01 06:00:00 to 2016-12-16 04:29:00
Freq: T
Data columns (total 26 columns):
_id                      503910 non-null object
use                      503910 non-null float64
gen                      503910 non-null float64
Dishwasher               503910 non-null float64
Home office              503910 non-null float64
Fridge                   503910 non-null float64
Wine cellar              503910 non-null float64
Garage door              503910 non-null float64
Barn                     503910 non-null float64
Well                     503910 non-null float64
Microwave                503910 non-null float64
```

```
Living room             503910 non-null float64
Solar                   503910 non-null float64
temperature             503910 non-null float64
humidity                503910 non-null float64
visibility              503910 non-null float64
apparentTemperature     503910 non-null float64
pressure                503910 non-null float64
windSpeed               503910 non-null float64
cloudCover              503910 non-null float64
windBearing             503910 non-null int64
precipIntensity         503910 non-null float64
dewPoint                503910 non-null float64
precipProbability       503910 non-null float64
sum_Furnace             503910 non-null float64
avg_Kitchen             503910 non-null float64
dtypes: float64(24), int64(1), object(1)
memory usage: 103.8+ MB
```

As the data have now the expected format, let go to the prediction phase. ## Prediction A popular and widely used statistical method for time series forecasting is the ARIMA model.

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data. ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

- AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.
- I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- MA: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components are explicitly specified in the model as a parameter. A standard notation is used of ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

[49]:
```
# Configuration of the ARIMA model
def forcast_ts(data, tt_ratio):
    X = data.values
    size = int(len(X) * tt_ratio)
    train, test = X[0:size], X[size:len(X)]
    history = [x for x in train]
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=(5,1,0))
        model_fit = model.fit(disp=0)
        output = model_fit.forecast()
```

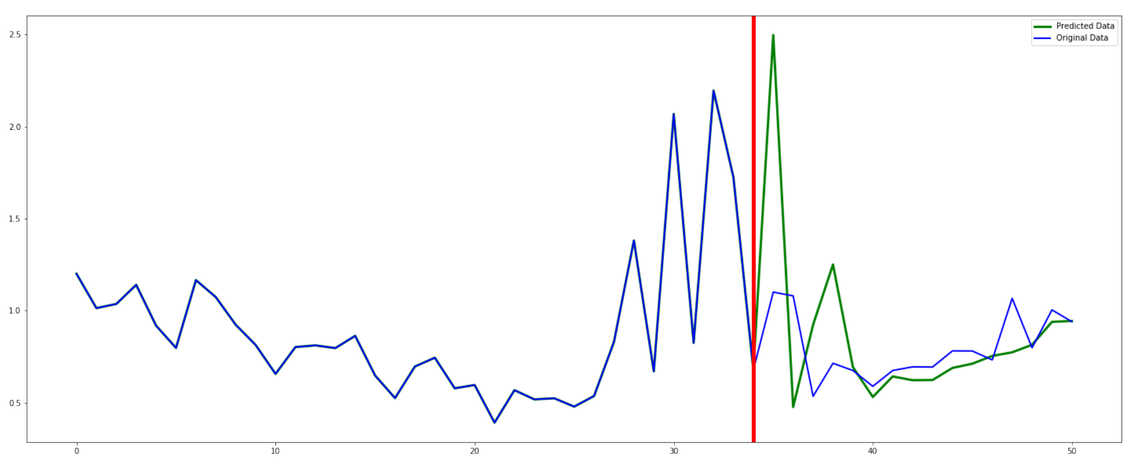
```
        yhat = output[0]
        predictions.append(yhat)
        obs = test[t]
        history.append(obs)
        print('progress:%',round(100*(t/len(test))),'\t predicted=%f, expected=%f' % (yhat, obs), end="\r")
    error = mean_squared_error(test, predictions)
    print('\n Test MSE: %.3f' % error)

    plt.rcParams["figure.figsize"] = (25,10)
    preds = np.append(train, predictions)
    plt.plot(list(preds), color='green', linewidth=3, label="Predicted Data")
    plt.plot(list(data), color='blue', linewidth=2, label="Original Data")
    plt.axvline(x=int(len(data)*tt_ratio)-1, linewidth=5, color='red')
    plt.legend()
    plt.show()
```

Lets start to apply the prediction for the power consumption (column ‘use’) by using weekly data.

[50]:
```
col = 'use'
dataset.dropna(inplace=True) # remove NaN values from the dataset
data = dataset[col].resample('w').mean()
data.shape
tt_ratio = 0.70 # Train to Test ratio
forcast_ts(data, tt_ratio)
```

```
/Users/frederic.favelin/opt/anaconda3/lib/python3.7/site-
packages/statsmodels/base/model.py:548: HessianInversionWarning: Inverting
hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
```

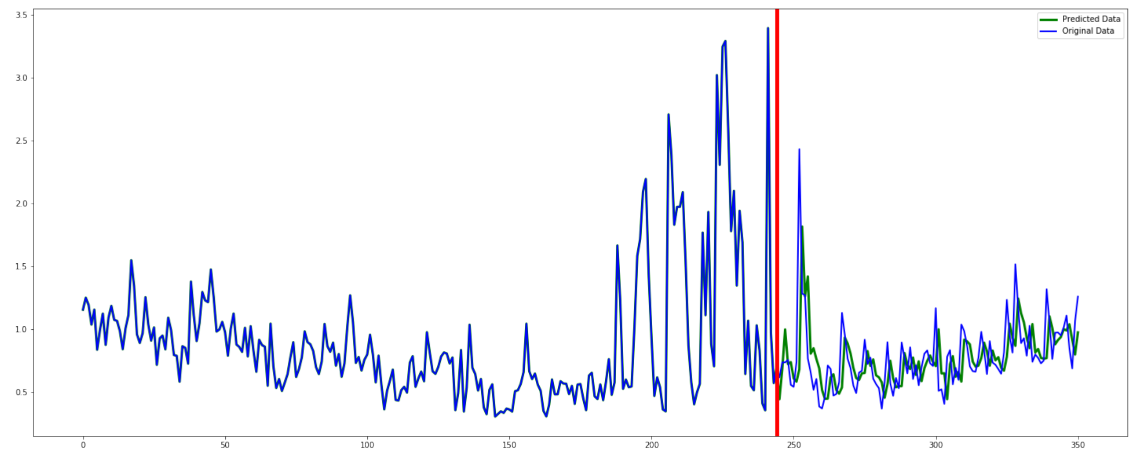
```
progress:% 94    predicted=0.944201, expected=0.939880
 Test MSE: 0.180
```



Weekly prediction are not accurate enough as MSE result is too high. Lets use daily figures instead of weekly.

[51]:
```
col = 'use'
dataset.dropna(inplace=True) # remove NaN values from the dataset
data = dataset[col].resample('d').mean()
data.shape
tt_ratio = 0.70 # Train to Test ratio
forcast_ts(data, tt_ratio)
```

```
progress:% 99    predicted=0.972182, expected=1.256528
 Test MSE: 0.075
```



Daily predictions are much accurate than weekly once. As we have removed some feature set from the model, maybe by using them we could improve the prediction.

[ ]:
```

```