11/4/2018

Machine learning with Energy datasets

Assignment 2- Report

Team

ANKIT YADAV
PREETAM JAIN
RITU AGARWAL

Content

1. Research Paper Analysis

RESEARCH PAPER 1: Data driven prediction models of energy use of appliances in a low-energy house

Link - https://www.sciencedirect.com/science/article/pii/S0378778816308970?via%3Dihub

Abstract

This paper presents and discusses data-driven predictive models for the energy use of appliances. Data used include measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station and recorded energy use of lighting fixtures. The paper discusses data filtering to remove non-predictive parameters and feature ranking.

1. Introduction

The electricity consumption in domestic buildings is explained by two main factors: the type and number of electrical appliances and the use of the appliances by the occupants. The occupancy level of the building in different locations could also help to determine the use of the appliances. Specifically, data from a nearby airport weather station, temperature and humidity in different rooms in the house from a wireless sensor network and one sub-metered electrical energy consumption have been used to predict the energy use by appliances. Four regression models have been tested, namely (a) multiple linear regression model(lm), (b) support vector machine with radial basis function kernel(SVM-radial), (c) random forest(RF) and (d) gradient boosting machines(GBM) with different combinations of predictors.

- 1.1 Literature review

1.1.1 Appliances' loads in buildings and numerical modeling of their consumption

This section reviews some articles regarding modeling of appliances and other socio economic factors that help to understand the different data and methodologies that have been used in the past to understand appliances' energy use.

1.1.2 Electricity load prediction

This section presents and discusses research addressing electricity load prediction to identify the parameters, models and other methods that have been useful for energy prediction.

Models such as multiple regression, neural networks, forecasting methods, engineering methods, support vector machines, time series techniques and forecasting methods have been used to predict the electricity demand. The models usually have considered parameters such as the time of day, outdoor temperature, month, weekend, holidays, yesterday's consumption, rainfall index, global solar radiation, wind speed and occupancy.

From the data available to the researchers, an extensive list of variables was studied: weather, location (ZIP code), age of building, ownership, presence of double pane windows, energy efficient light fixtures, floor area, pet ownership, number of refrigerators and entertainment devices, number of occupants and income level were studied. The researchers concluded that the most important variables were weather, location, and floor area. Also, the number of refrigerators and entertainment appliances are among the most important determinants of daily minimum consumption. Another study found that being at home during the day correlated with lower appliance efficiency. The provided explanation by the researchers is the lower efficiency was likely due to the increased use of appliances when the house is occupied more often.

A prediction system for the problem of individual appliance prediction was presented in. The system used information such as past consumption, hour, day, season and month. The system is capable of learning from past data. One of the main conclusions was that the last 24 h are the most relevant for prediction.

- 1.2 Research objectives and methodology outline

The purpose of this work is to understand the relationships between appliances energy consumption and different predictors.

2. House description

A house located in Stambruges is taken into consideration for analyzing the aggregated electric energy consumption per month. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network.

3. Recorded data and description

The energy (Wh) data logged every 10 min for the appliances is the focus of this analysis. The 10 min reporting interval was chosen to be able to capture quick changes in energy consumption.

- 3.1 Data sets and exploratory analysis

The combined data set is split in training and test validation using CARET'S (Classification and Regression Training Package) create data partition function. 75% of the data is used for the training of the models and the rest is used for testing.

- 3.2 Data features filtering and importance

The Boruta package is used here to select all the relevant variables. Several researchers have used this package for variable filtering. To test the Boruta algorithm, two random variables were introduced in the data sets. Moreover, this feature or variable selection helps in model interpretability and reduces complexity of the model.

The Boruta package compares importance of attributes with importance of shadow attributes that are created by shuffling original ones. The Boruta algorithm is capable of detecting the two random variables that have no predicting power for the appliances' energy consumption.

To test how many variables are optimal to minimize the RMSE the recursive feature elimination (RFE) is used to select the optimal inputs.

- 3.3 The performance of regression models

In order to compare the performance of each of the regression models, different performance evaluation indices are used here: the root mean squared error (RMSE), the coefficient of determination or R-squared/R2, the mean absolute error (MAE) and the mean absolute percentage error (MAPE).

- 3.4 Model selection

The best models are the ones that provide the lower RMSE and highest R-square values.

4. Conclusion

For all the models, the time information was ranked as the most important to predict the appliances' consumption.

When using all the predictors the light consumption was ranked highly. However, when studying different predictor subsets, removing the light consumption appeared not to have a significant impact. This may be an indication that other features are correlated well with the light energy consumption.

This study has found curious relationships between variables. Future work could include considering weather data such as solar radiation and precipitation.

RESEARCH PAPER 2: A review of artificial intelligence based building energy use prediction: Contrasting the capabilities of single and ensemble prediction models

Link - https://www.sciencedirect.com/science/article/pii/S1364032116307420

Building energy prediction can be broadly classified into engineering, Artificial Intelligence (AI) based, and hybrid approaches. While engineering and hybrid approaches use thermodynamic equations to estimate energy use, the AI-based approach uses historical data to predict future energy use under constraints. Tall buildings and skyscrapers accounts for 30% of total electricity consumption. This gave wide spectrum to data analyst to analyze this scenario which can be exercised through engineering methods. There are 2 major methods which are as follows: (1) AI based method also called as the black box predicts energy use without knowing the internal relationship of the building and the individual components. (2) Hybrid is called as grey box which requires detailed building information for simulation for model development.

There are 2 prediction methods which are widely used for energy systems. They are as follows: (1) Single Prediction – utilize one learning algorithm (2) Ensemble Prediction – integrate some of the single prediction to improve accuracy of prediction.

There are many parameters which can be used for predicting energy usage by buildings like building type which can be commercial, residential, educational or research. We can assume them to be one among educational, research or commercial due to availability of data. Residential building has privacy issues. The predicting model used for energy usage varied from one prediction algorithm to ensemble. Ensemble model is preferred due to its demonstrated superiority over one prediction after much research and development into it over the years. Artificial Neural Network (ANN) is used because of ease of implementation and reliable prediction performance besides regression, SVR, ARMAX, CHAID for building energy use prediction. The energy type was divided into 5 categories i.e. whole building energy/electricity (35%), heating and cooling energy (11%), heating energy (11%), cooling energy (13%) and all others (8%). Other considerable parameter was Prediction time scale which represents the time resolution of the prediction which is often impacted by the sampling interval of sensors and the research. Preferred time scale was Hour (49%), Day (19%), Year (8%) and Other (24% - min-by-min, week, month). Based on knowledge researchers collect data. Meteorology (60%), Occupancy (29%), Other (54%). Different patterns could be identified and analyzed.

The various AI-based prediction models involve methods like Data collection, data preprocessing, model training and model testing. In the single prediction method, I use multiple linear regression where inputs consist of shape factor, envelope U-value, window-to-floor area ratio, building time constant and climate which is defined as a function of sol-air temperature and heating set-point. These models were easy and efficient forecast tools for calculating heating demand of residential buildings. Catalina et al. simplified the MLR model by introducing only three inputs namely, building global heat loss coefficient, south equivalent surface, and the difference between the indoor set point temperature and sol-air temperature. Their results indicated that the proposed method closely predicted future building heat demand. Jacob et al. improved the performance of regression model by introducing the rate of change of the indoor air temperature as an independent variable. Their study indicated that the performance of MLR could be improved by introducing appropriate independent variables. In ANN, a nonlinear statistical technique which consists of Input, output and hidden layers interconnected. Ben-Nakhi and Mahmoud applied General Regression Neural Network (GRNN) to predict the cooling load for commercial buildings. Multiple results show ANNs could perform better than regression method for short-term forecasting. To detect complex nonlinear relationships between inputs and outputs implicitly which is good for real time monitoring. However, ANN method fails to establish any interconnection relationship between building physical parameters and building energy use, which limits the model's fitting ability when changes are made to building components or systems. In Support Vector Regression where input data is mapped via a nonlinear function. It finds the most deviation from the obtained target. Selection of kernel function is important as it affects the learning ability of SVR. SVR demonstrates its ability to predict hourly cooling load in the building. SVR helps in optimization

In the Ensemble prediction method, the aim is to provide best possible prediction performance by automatically managing the strengths and weaknesses of each base model. It has multiple learning models depending on base model resampling, manipulation or randomization of training data, learning algorithm and parameters. Ensemble models can be Homogenous and Heterogenous models. Homogenous model uses bagging and boosting. The various ways to implement it is firstly via Input feature identification, Data monitoring and preprocessing, Learning algorithm selection, Base model generation, Model integration.

Proposed by Hansen and Salamon 1990 to solve classification problems. They state that the collective decision produced by the ensemble model is less likely to be in error than the decision made by any of the individual models. Multiple classification algorithms and integration schemes were used to develop ensemble models. heterogeneous ensemble classifier that consisted of five different classification algorithms to solve health-related short text classification problem. A parameter sensitivity analysis was carried out to obtain the best possible features. Multiple model integration schemes such as multi-staging, reverse multi-staging, majority voting, and weighted probability averaging were used to combine the classification results of the base classifiers. The result indicated that the proposed ensemble classifier performs better than the single SVM classifier in the studied problem. Three combination techniques such as the majority voting, the LSE-based weighting, and the double-layer hierarchical combining were used to aggregate the individual SVMs. Three typical classification problems: data classification, handwritten digit recognition, and fraud detection were used to test the efficacy of the proposed ensemble model.

Their results indicated that the ensemble model outperforms single SVM model in terms of classification accuracy which is performed on buildings around the globe.

RESEARCH PAPER 3: Prediction of appliances energy use in smart home

Link - https://www.sciencedirect.com/science/article/pii/S0360544212002903

This paper has been written with an aim to predict the energy consumption in household for the next day. To achieve this, they have collected data of homes of France and have analyzed and have come up with certain predictors. It has been studied that residential sector is the biggest sector in electricity consumption. And it is required that we understand the pattern of electricity consumption in household so that Industries can generate and transfer only that amount of energy to the household area to better circulate power. The energy market is divided into distinct categories, but the Day Ahead Market or Spot Market is of great interest. This type of energy market involves bidding the energy consumption of the next day. It is a very complex mechanism, which requires a very good knowledge of the demand for the power suppliers There were lots of theories which were proposed but it is important to understand the each and every criterion like number of appliances, usage of these appliances, day of week, etc. So, this paper concentrates more over discrimination of usage of electricity on appliance level which would make things easier to understand the pattern of usage of electricity over the course of time. In order to get a better load control, the energy prediction has to go down from total household energy consumption to electrical device consumption. The concept of smart grid has been introduced to tackle power system challenges. Smart grid initiatives seek to improve operations, maintenance and planning using modern technology to better manage energy use and costs. This would help industries to smartly circulate the generated electricity to different industry which would be much more efficient than present method. There have been lot of expectations which was not getting met as the usage of appliance differ over period of time on daily basis. A reliable model was required as usage of appliances on peak time was different as compared to other times. Thus, they came up with a concept called demand dispatch which is ability to control individual loads in precise manner at all the times and not only during peak times. This load management id of two types.

• **Direct Control**: This method refers to classical method of load control which involves increasing the energy production in case of higher load demand.

• **Control by cost**: This method refers to change the load curve shape in such a way that energy consumption peak decreases, even though the total energy consumption for the specific house stays the same. When it has been understood that we have to consider the usage of appliances to get a better picture of electricity consumption and load balancing, there are 4 different type of predictors which can be considered to calculate the same.

• **The "will always consume" predictor**: According to this predictor, we assume that an appliance is always running and consuming electricity.

• **The "will never consume" predictor**: According to this predictor, we assume that an appliance is not at all being used and is not consuming electricity.

• **The ARMA predictor**: ARMA stands for Autoregressive Moving Average. According to this method current value of a time variable is assumed to be a function of its past values and it is expressed as a weighted sum (moving average).

• **The proposed predictor**: According to this model, an inhabitant in the house interacts with various electrical devices as part of his routine activities. Thus, energy consumption can be modeled as a process which is having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

Improving the precision of prediction is highly necessary. It is important for us to understand the pattern of usage of electricity. The segmentation of data can be made considering various aspects such as the season, month, period of the day (day/night), type of day (weekday/weekend). The objective of this operation is to reduce the average dispersion to improve the prediction. In such conditions, k-means clustering method can be precise to cluster similar data together. At last, I would like to conclude by saying forecasting the energy consumption in homes is an important aspect in the power management of the grid, as the consumption in the residential sector represents a significant percentage in the total electricity demand. The development of the smart grid is not possible without a good prediction of energy consumption. The trend nowadays is to get the prediction of energy consumption not only at house level, but at household appliance level. The prediction of energy consumption in housing is very dependent on inhabitants' behavior, so a stochastic method for prediction has been presented in this paper.

2. Exploratory Data Analysis

We imported following libraries

In [1]:
```
import numpy as np
```

In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Then we read the data and viewed its head.

There were mainly variables for temperature and humidity at various parts of house.

In [3]:
```
df=pd.read_csv("../Part_2/energydata_complete.csv")
df.head(10)
df.tail(20)
```

Out[3]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_n
19715	2016-05-27 14:50:00	60	0	25.500000	46.060000	26.350000	41.000000	28.426667	40.590000	24.666667	...	23.100000	46.590000	21.833333	
19716	2016-05-27 15:00:00	60	0	25.500000	45.933333	26.277143	41.000000	28.356667	40.560000	24.666667	...	23.100000	46.590000	21.800000	
19717	2016-05-27 15:10:00	70	0	25.500000	45.760000	26.200000	41.000000	28.290000	40.433333	24.700000	...	23.133333	46.590000	21.966667	
19718	2016-05-27 15:20:00	80	0	25.500000	45.626667	26.171429	41.000000	28.260000	40.260000	24.700000	...	23.133333	46.590000	22.133333	
19719	2016-05-27 15:30:00	80	0	25.500000	45.590000	26.100000	41.000000	28.200000	40.126667	24.700000	...	23.166667	46.590000	22.300000	
19720	2016-05-27 15:40:00	70	0	25.500000	45.522500	26.100000	41.051429	28.200000	40.200000	24.700000	...	23.100000	46.590000	22.466667	
19721	2016-05-27 15:50:00	100	0	25.500000	45.633333	26.080000	41.196000	28.133333	40.260000	24.700000	...	23.200000	46.590000	22.633333	

We also checked its shape along with the variable information.

```
In [4]: df.shape
```

```
Out[4]: (19735, 29)
```

```
In [8]: data_description = pd.read_csv("../Part_2/variables_description.txt", delimiter='\t')
        data_description.head(40)
```

Out[8]:

	Variable Description
0	===================
1	date time year-month-day hour:minute:second
2	Appliances, energy use in Wh
3	lights, energy use of light fixtures in the ho...
4	T1, Temperature in kitchen area, in Celsius
5	RH_1, Humidity in kitchen area, in %
6	T2, Temperature in living room area, in Celsius
7	RH_2, Humidity in living room area, in %
8	T3, Temperature in laundry room area
9	RH_3, Humidity in laundry room area, in %
10	T4, Temperature in office room, in Celsius
11	RH_4, Humidity in office room, in %
12	T5, Temperature in bathroom, in Celsius
13	RH_5, Humidity in bathroom, in %

We also needed to check the variable information for further use.

```
In [9]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 29 columns):
date           19735 non-null object
Appliances     19735 non-null int64
lights         19735 non-null int64
T1             19735 non-null float64
RH_1           19735 non-null float64
T2             19735 non-null float64
RH_2           19735 non-null float64
T3             19735 non-null float64
RH_3           19735 non-null float64
T4             19735 non-null float64
RH_4           19735 non-null float64
T5             19735 non-null float64
RH_5           19735 non-null float64
T6             19735 non-null float64
RH_6           19735 non-null float64
T7             19735 non-null float64
RH_7           19735 non-null float64
T8             19735 non-null float64
RH_8           19735 non-null float64
T9             19735 non-null float64
RH_9           19735 non-null float64
T_out          19735 non-null float64
Press_mm_hg    19735 non-null float64
```

Then we change the date time according to month, week and time of day.

In [11]:

```python
import datetime
import calendar
df['date']=pd.to_datetime(df['date'])
df['year']=df['date'].dt.year
df['month']=df['date'].dt.month
df['day']=df['date'].dt.day
df['day_of_week']=df['date'].dt.weekday_name
df['time_hr_24']=df['date'].dt.hour
morning=range(6,12)
afternoon=range(12,17)
evening=range(17,22)
def time_slot(x):
    if x in morning:
        return 'morning'
    elif x in afternoon:
        return 'afternoon'
    elif x in evening:
        return 'evening'
    else:
        return 'night'

df['day_slot']=df['time_hr_24'].map(time_slot)

week1=range(1,8)
week2=range(8,15)
week3=range(15,22)
week4=range(22,29)

def week_num(x):
    if x in week1:
        return 'week1'
    elif x in week2:
        return 'week2'
    elif x in week3:
        return 'week3'
    elif x in week4:
        return 'week4'
    else:
        return 'week5'
df['week']=df['day'].map(week_num)
```

According to that we added new columns into our data frame

```python
df.drop(['date'],axis=1,inplace=True)
df=pd.get_dummies(df,prefix=['DOW','TS','WDT','W'],columns=['day_of_week','day_slot','week_day_type','week'])
print(df.shape)

df.dtypes
```

```
(19735, 50)
```

Out[11]:

```
Appliances       int64
lights           int64
T1             float64
RH_1           float64
T2             float64
RH_2           float64
T3             float64
RH_3           float64
T4             float64
RH_4           float64
T5             float64
RH_5           float64
T6             float64
RH_6           float64
T7             float64
RH_7           float64
T8             float64
RH_8           float64
T9             float64
RH_9           float64
T_out          float64
Press_mm_hg    float64
RH_out         float64
Windspeed      float64
```

After this we checked the correlation check for redundant variable .

```
In [12]: def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df.select_dtypes(include=['float64','int64']))
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations")
print(get_top_abs_correlations(df.select_dtypes(include=['float64','int64']), 20))
```

```
Top Absolute Correlations
rv1   rv2      1.000000
T6    T_out    0.974787
T7    T9       0.944776
T5    T9       0.911055
T3    T9       0.901324
RH_3  RH_4     0.898978
RH_4  RH_7     0.894301
T1    T3       0.892402
T9    month    0.890605
T4    T9       0.889439
T3    T5       0.888169
T1    T5       0.885247
RH_7  RH_8     0.883984
T7    T8       0.882123
RH_1  RH_4     0.880359
T4    T7       0.877763
T1    T4       0.877001
T4    T5       0.871813
T5    T7       0.870624
T8    T9       0.869338
dtype: float64
```

We found that rv1 and rv2, t6 and t_out etc are highly correlated .

So, we removed these variables from our data.

From the above correlation matrix it could be found:

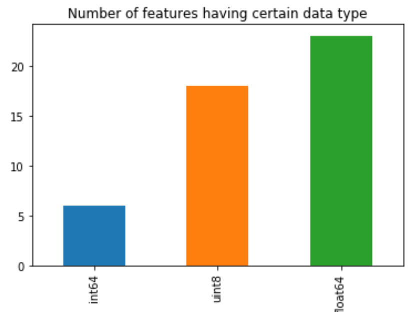
1. rv1 and rv2 are highly correlated.
2. T-out and T6 are highly correlated.
3. T9 and T7 are highly correlated

Thus, one of them must be rejected.

```
In [19]: df=df.drop(['rv2','T7','T_out'],axis=1)
```

```
In [20]: from matplotlib import cm as cm
```

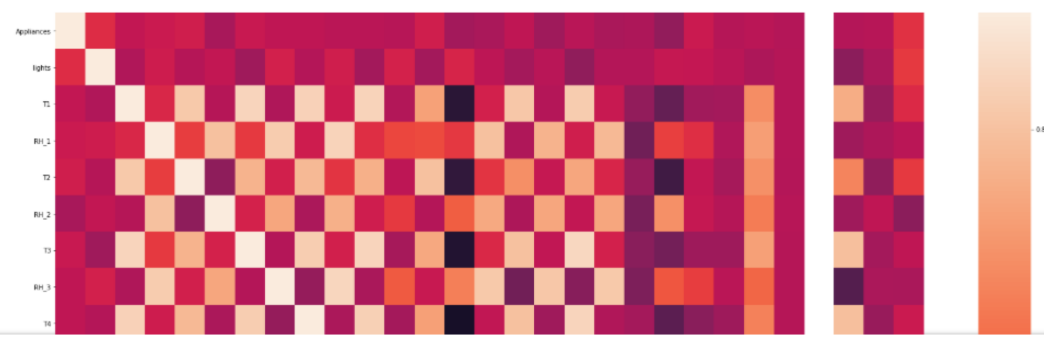
```
In [21]: # correlation_matrix(df.select_dtypes(include=['float64','int64']))
         df.dtypes.value_counts().sort_values().plot(kind='bar')
         plt.title('Number of features having certain data type')
         plt.show()
```




```
In [22]: fig = plt.figure(figsize = (30,30))
```

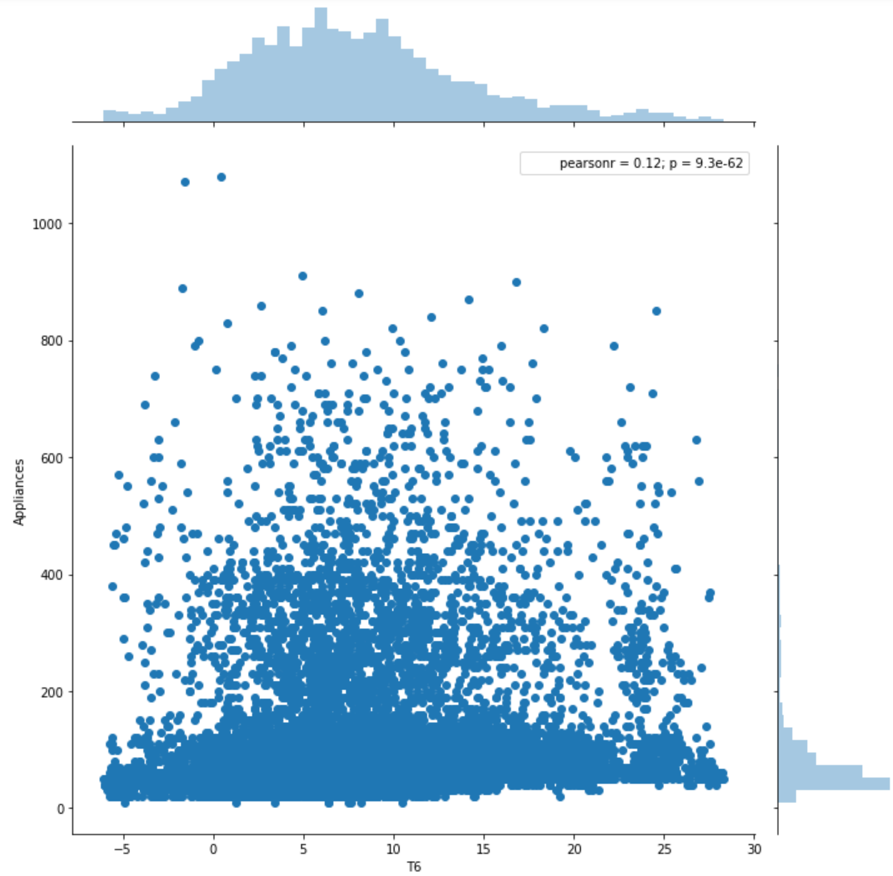
We also checked by using heatmap of variables.

```
In [22]: fig = plt.figure(figsize = (30,30))
         ax1 = fig.add_subplot(111)
         sns.heatmap(df.select_dtypes(include=['float64','int64']).corr())

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x21a00ae7438>
```



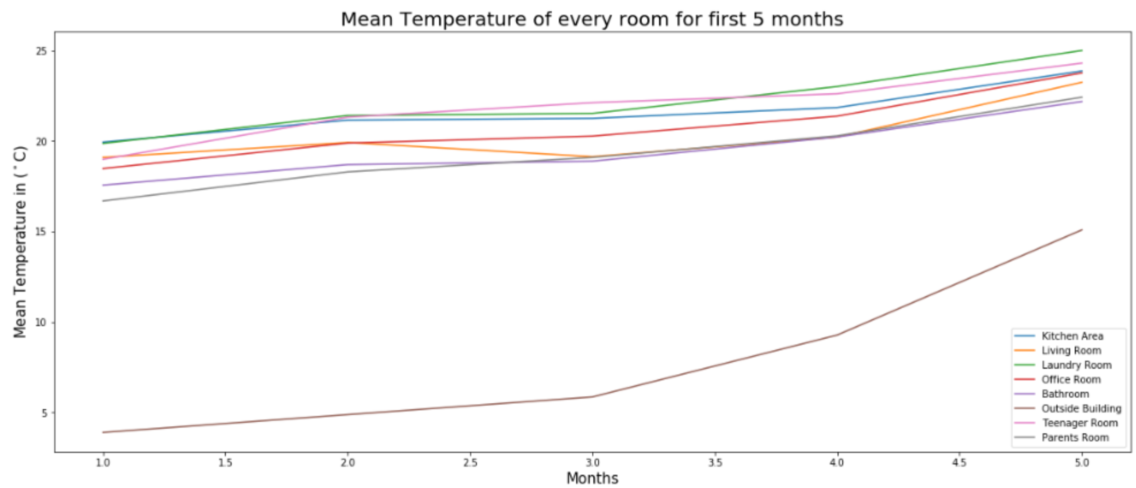
We also made following plot.

```
In [23]: # df['Total_Usage'] = df['Appliances'] + df['lights']
         sns.jointplot(x='T6',y='Appliances',data=df,size=10)
```



We also plotted the graph to look at the changes in temperature over time for various appliances

```
plt.xlabel("Months" , fontsize=15)
plt.ylabel("Mean Temperature in ($^\circ$C)" , fontsize=15)
plt.title("Mean Temperature of every room for first 5 months" , fontsize=20)
y = ['Jan' , 'Feb' , 'Mar' , 'Apr' , 'May']

plt.legend()
plt.show()
```



We also used pandas profiling to check for various variables.

```
In [40]: import pandas_profiling
         pandas_profiling.ProfileReport(df)
```

Out[40]:

Overview

Dataset info

Number of variables	29
Number of observations	19735
Total Missing (%)	0.0%
Total size in memory	4.4 MiB
Average record size in memory	232.0 B

Variables types

Numeric	25
Categorical	2
Boolean	0
Date	1
Text (Unique)	0
Rejected	1
Unsupported	0

Warnings

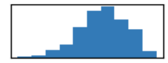
Variables

Appliances
Numeric

Distinct count	92	Mean	97.695
Unique (%)	0.5%	Minimum	10
Missing (%)	0.0%	Maximum	1080
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Press_mm_hg
Numeric

Distinct count	2189	Mean	755.52
Unique (%)	11.1%	Minimum	729.3
Missing (%)	0.0%	Maximum	772.3



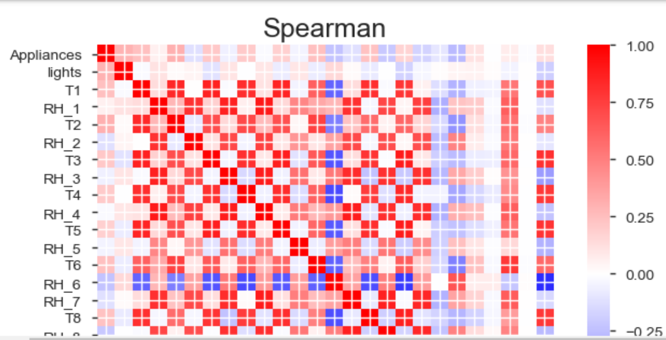
RH_2
Numeric

Distinct count	3376	Mean	40.42
Unique (%)	17.1%	Minimum	20.463
Missing (%)	0.0%	Maximum	56.027
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

RH_3
Numeric

Distinct count	2618	Mean	39.243
Unique (%)	13.3%	Minimum	28.767
Missing (%)	0.0%	Maximum	50.163
Missing (n)	0	Zeros (%)	0.0%

In [40]:
```
import pandas_profiling
pandas_profiling.ProfileReport(df)
```



In [41]:
```
df.columns
```

3. Feature Engineering:

There are lots of features in our dataset. Temperature of 8 different rooms are recorded in degree
Celsius. Humidity of 8 different rooms are recorded in percentage. Outside temperature is also recorded in degree Celsius and humidity in percentage. Along with it pressure has been recorded
in millimeter scale in mercury, visibility in kilometer, dew point in degree Celsius and windspeed in m/s.
Before performing any kind of test, we have to analyses the data and look into it.
We also have to understand how much data does the file contains.
It is important for us to analyze whether there are null values in the dataset and how are values distributed.

```
In [38]: data=pd.read_csv("../Part_2/energydata_complete.csv")

In [39]: data.head()
```

Out[39]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_mm_hg	RH_out	Windsp
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...	17.033333	45.53	6.600000	733.5	92.0	7.00
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	17.066667	45.56	6.483333	733.6	92.0	6.66
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	17.000000	45.50	6.366667	733.7	92.0	6.33
3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	...	17.000000	45.40	6.250000	733.8	92.0	6.00
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	...	17.000000	45.40	6.133333	733.9	92.0	5.66

5 rows × 29 columns

```
In [40]: analysed_data=pd.read_csv("../Part_2/Data_analysed.csv")

In [41]: analysed_data.head()
```

Out[41]:

Unnamed: 0	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	...	month_mean_RH1	month_mean_RH2	month_mean_RH3	m

We also did linear regression on the data.

Linear Regression Model

In [48]:
```
lm=linear_model.LinearRegression()
mod=lm.fit(x_train_sc,y_train)
print(mod.coef_)

print(x_train.columns)
```

```
[ 1.27311327e+01 -3.55718214e+00  4.82855496e+01 -3.86638652e+01
 -4.99836910e+01  5.07160517e+01  2.27692226e+01  3.82525914e+00
 -6.79021158e+00 -2.61521677e+00  3.43333957e-01  2.88166878e+01
 -7.34441736e-01  2.44386866e+00 -6.96897198e+00  1.82222358e+01
 -1.37056743e+01 -2.69899999e+01 -4.90771902e+00 -4.09994697e+01
  1.17023936e+00 -6.85337123e+00  3.74400084e+00  2.32604519e+00
  1.85177640e+01  4.09382986e-01  4.09382986e-01 -3.90798505e-14
 -1.17913554e+01 -1.42626392e+00 -1.17279793e+00  1.77942571e+00
  5.20350987e+00  1.13050837e+00  3.87232266e+00 -2.38662247e+00
 -2.52393895e+00 -3.63926792e+00 -1.56566165e+00  1.38663924e+00
  1.01841853e+01  5.50861544e+00 -1.50336310e+01 -1.12071555e+00
  1.12071555e+00]
Index(['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T5',
       'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8', 'RH_8', 'T9', 'RH_9', 'T_out',
       'Press_mm_hg', 'RH_out', 'Windspeed', 'Visibility', 'Tdewpoint', 'rv1',
       'rv2', 'year', 'month', 'day', 'time_hr_24', 'time_min', 'DOW_Friday',
       'DOW_Monday', 'DOW_Saturday', 'DOW_Sunday', 'DOW_Thursday',
       'DOW_Tuesday', 'DOW_Wednesday', 'TS_afternoon', 'TS_evening',
       'TS_morning', 'TS_night', 'WDT_weekdays', 'WDT_weekends'],
      dtype='object')
```

Finally, We found the importance of various features in our dataset.

In [77]:
```
d = pd.DataFrame(feature_importances)
```

In [79]:
```
d.columns = ['Feature', 'Value']
```

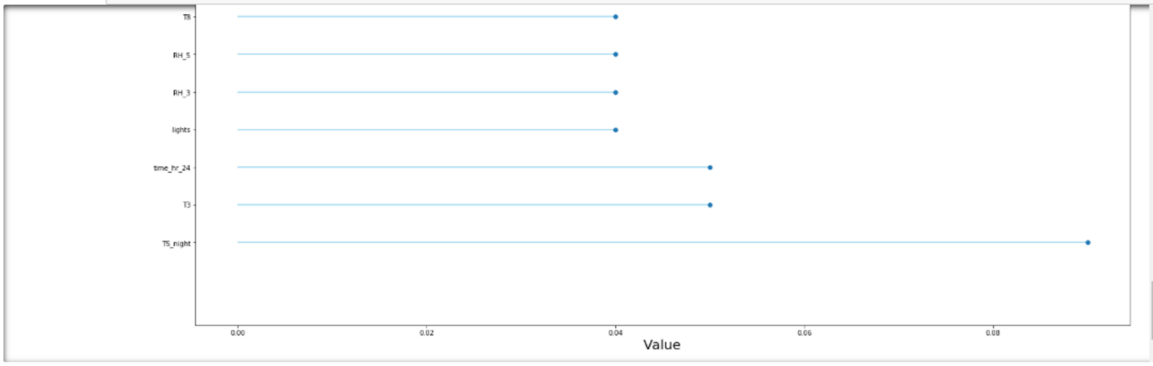
In [80]:
```
d
```

Out[80]:

	Feature	Value
0	TS_night	0.09
1	T3	0.05
2	time_hr_24	0.05
3	lights	0.04
4	RH_3	0.04
5	RH_5	0.04
6	T8	0.04
7	Press_mm_hg	0.04
8	RH_1	0.03
9	T2	0.03
10	RH_2	0.03
11	T4	0.03
12	RH_4	0.03
13	T7	0.03
14	RH_7	0.03
15	RH_8	0.03
16	RH_9	0.03
17	RH_out	0.03
18	Windspeed	0.03
19	Tdewpoint	0.03
20	T1	0.02

We also plotted a graph to check the same.

In [89]:
```
a=d['Feature']
plt.xlabel("Value" , fontsize=20)
plt.ylabel("Feature" , fontsize=20)
plt.legend()
plt.title("Importance of Features" , fontsize=40)
plt.hlines(y=a, xmin=0, xmax=d['Value'], color='skyblue')
plt.plot(d['Value'], a, "o")
plt.rcParams['figure.figsize'] = (25 , 50)
plt.show()
```



In [ ]:

4. Prediction Algorithms

```
import pandas as pd
import datetime
import numpy as np
import sklearn
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.grid_search import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import *
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

Splitting data

```
In [219]: from sklearn.model_selection import train_test_split
```

```
In [220]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Linear Regression

```
In [56]: from sklearn.linear_model import LinearRegression
```

```
In [57]: lm=LinearRegression()
```

```
In [218]:
X = df.drop(['Appliances'],axis=1)
y = energy_data_complete['Appliances']
```

```
In [221]: lm.fit(X_train,y_train)

Out[221]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                   normalize=False)
```

```
In [225]: # The coefficients
          print('Coefficients: \n', lm.coef_)

          Coefficients:
           [ 1.58639142e+00 -1.81851521e+00  1.32188205e+01 -1.78769245e+01
           -1.27339698e+01  2.57171413e+01  6.65504364e+00  1.44641205e+00
           -1.46278964e+00 -1.82584390e+00  1.37015448e-02  3.31368293e+00
           -1.26498195e-01 -1.24450234e+00  1.02147795e+01 -2.88283445e+00
           -1.35422575e+01 -1.16098376e+00  1.89036791e-01  9.85002954e-01
            1.88603123e+00  1.75320013e-01 -2.13444209e+00  4.24421146e-02
           -3.55271368e-15 -8.81883689e+00  6.49793896e-01 -1.92191360e-01
            1.46465722e+01  3.74873514e+00  1.05233084e+01 -7.92293360e+00
           -6.89346415e+00 -9.13086101e+00 -4.97135705e+00  1.51364888e+00
            2.19771521e+01  1.11796838e+01 -3.46704849e+01 -2.60037485e+00
            2.60037485e+00  8.58817418e+00  6.60952869e+00  2.53724874e+00
           -7.25238248e+00 -1.04825691e+01]
```

Using Random Forest

```
In [231]: from sklearn.ensemble import RandomForestRegressor
```

```
In [232]: # Random Forest Regressor
          # model = RandomForestRegressor(n_estimators=300, max_features = 11)
          #     # create the RFE model and select 3 attributes
          # rfe = RFE(model)
          # rfe = rfe.fit(X_train, y_train)
          #     # summarize the selection of the attributes
          # print(rfe.support_)
          # print(rfe.ranking_)
          # print(rfe.n_features_)
          #     #Check the accuracy of the model
          # rfe.score(X_train, y_train)
          rand_forest_model = RandomForestRegressor(max_depth=5, random_state=0) # build model
          rand_forest_model = rand_forest_model.fit(X_train, y_train.values.ravel()) # train model
          RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=5,
                     max_features='auto', max_leaf_nodes=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=10, n_jobs=1,
                     oob_score=False, random_state=0, verbose=0, warm_start=False)
          feature_importance = rand_forest_model.feature_importances_
          R2 = rand_forest_model.score(X_train, y_train) # coefficient of determination
          rand_forest_model_predictions = rand_forest_model.predict(X_test) # make predictions

          C:\Users\ritua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ensemble\forest.py:248: FutureWarning: The default v
          alue of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
            "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

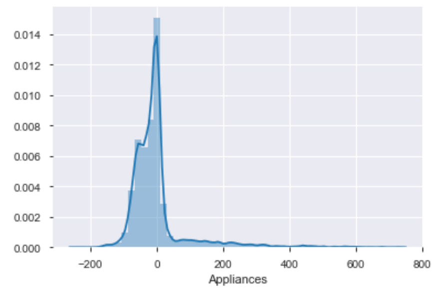
```
In [233]: # rand_forest_model_metrics = view_metrics(y_test, rand_forest_model_predictions,'Random Forest Regressor', samples = 50, total_s
          plt.scatter(y_test,rand_forest_model_predictions)
          plt.xlabel('Y Test')
          plt.ylabel('Predicted Y')

Out[233]: Text(0,0.5,'Predicted Y')
```

In [234]:
```
print('MAE:', metrics.mean_absolute_error(y_test, rand_forest_model_predictions))
print('MSE:', metrics.mean_squared_error(y_test, rand_forest_model_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, rand_forest_model_predictions)))
```

```
MAE: 48.96025500004364
MSE: 7862.571568228872
RMSE: 88.67114281562448
```

In [235]:
```
sns.distplot((y_test-rand_forest_model_predictions),bins=50)
```

Out[235]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2c04d320>
```



Neural Network Models

Neural Network Model

In [22]:
```
mlp = MLPRegressor(hidden_layer_sizes=(365,365,365),max_iter=500,alpha=1.00000000e-06,random_state=42)
mlp.fit(x_train_sc,y_train)
```

Out[22]:
```
MLPRegressor(activation='relu', alpha=1e-06, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(365, 365, 365), learning_rate='constant',
       learning_rate_init=0.001, max_iter=500, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=42, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

Neural Network on Training Dataset

In [23]:
```
y_train_pred=mlp.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
print("MAPE :",mean_absolute_percentage_error(y_train,y_train_pred))
```

```
R2   : 0.7592006178700873
MAE  : 27.93967498755577
RMSE : 50.52214265806127
MAPE : 30.235642921597133
```

Neural Network on Testing Dataset

In [24]:
```
y_test_pred=mlp.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
print("MAPE :",mean_absolute_percentage_error(y_test,y_test_pred))
```

```
R2   : 0.4078561865573932
MAE  : 39.471749093105366
RMSE : 78.10655339647808
MAPE : 39.30815270527815
```

5. Feature Selection

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features variables, predictors) for use in model construction. Feature selection techniques are used for four reasons:

- simplification of models to make them easier to interpret by researchers/users,
- shorter training times,
- enhanced generalization by reducing overfitting (formally, reduction of variance)

The central premise when using a feature selection technique is that the data contains many features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information. Redundant or irrelevant features are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

TPOT:

The Tree-Based Pipeline Optimization Tool (TPOT) was one of the very first AutoML methods and open-source software packages developed for the data science community. The goal of TPO
is to automate the building of ML pipelines by combining a flexible expression tree representation of pipelines with stochastic search algorithms such as genetic programming. TPOT makes use of the Python-based scikit-learn library as its ML menu

Tpot

In [251]:
```
from tpot import TPOTRegressor
pipeline_optimizer = TPOTRegressor()
```

In [252]:
```
pipeline_optimizer = TPOTRegressor(generations=10, population_size=20, cv = 3,
                                    random_state=42, verbosity=2)
```

In [253]:
```
pipeline_optimizer.fit(X_train,y_train)
```

Failed to display Jupyter Widget of type `HBox`.

If you're reading this message in the Jupyter Notebook or JupyterLab Notebook, it may mean that the widgets JavaScript is still loading. If this message persists, it likely means that the widgets JavaScript library is either not installed or not enabled. See the Jupyter Widgets Documentation for setup instructions.

If you're reading this message in another frontend (for example, a static rendering on GitHub or NBViewer), it may mean that your frontend doesn't currently support widgets.

```
Generation 1 - Current best internal CV score: -5225.879046662546
Generation 2 - Current best internal CV score: -5225.879046662546
Generation 3 - Current best internal CV score: -5222.948696227385
Generation 4 - Current best internal CV score: -5222.948696227385
Generation 5 - Current best internal CV score: -5222.948696227385
Generation 6 - Current best internal CV score: -5222.948696227385
Generation 7 - Current best internal CV score: -5222.948696227385
Generation 8 - Current best internal CV score: -5222.948696227385
Generation 9 - Current best internal CV score: -5222.948696227385
Generation 10 - Current best internal CV score: -5222.948696227385

Best pipeline: ElasticNetCV(RandomForestRegressor(input_matrix, bootstrap=False, max_features=0.25, min_samples_leaf=1, min_sam
ples_split=2, n_estimators=100), l1_ratio=0.05, tol=0.01)
```

Out[253]:
```
TPOTRegressor(config_dict=None, crossover_rate=0.1, cv=3,
        disable_update_check=False, early_stop=None, generations=10,
        max_eval_time_mins=5, max_time_mins=None, memory=None,
        mutation_rate=0.9, n_jobs=1, offspring_size=None,
        periodic_checkpoint_folder=None, population_size=20,
        random_state=42, scoring=None, subsample=1.0, use_dask=False,
        verbosity=2, warm_start=False)
```

Info of model

```
In [254]: energy_data_complete.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 27 columns):
date           19735 non-null datetime64[ns]
Appliances     19735 non-null int64
lights         19735 non-null int64
T1             19735 non-null float64
RH_1           19735 non-null float64
T2             19735 non-null float64
RH_2           19735 non-null float64
T3             19735 non-null float64
RH_3           19735 non-null float64
T4             19735 non-null float64
RH_4           19735 non-null float64
T5             19735 non-null float64
RH_5           19735 non-null float64
T6             19735 non-null float64
RH_6           19735 non-null float64
RH_7           19735 non-null float64
T8             19735 non-null float64
RH_8           19735 non-null float64
T9             19735 non-null float64
RH_9           19735 non-null float64
Press_mm_hg    19735 non-null float64
RH_out         19735 non-null float64
Windspeed      19735 non-null float64
Visibility     19735 non-null float64
Tdewpoint      19735 non-null float64
rv1            19735 non-null float64
month          19735 non-null int64
dtypes: datetime64[ns](1), float64(23), int64(3)
memory usage: 4.1 MB
```

When we get the output, we check the accuracy of score.

```
In [255]: print(pipeline_optimizer.score(X_test, y_test))
```

```
-3693.3057863344193
```

```
In [ ]:
```

TSFresh

tsfresh is a python package. It automatically calculates a large number of time series characteristics, the so called features. Further the package contains methods to evaluate the explaining power and importance of such characteristics for regression or classification tasks.

We start by creating the model for testing.

Performing tsfresh

In [9]:
```
from tsfresh.utilities.dataframe_functions import roll_time_series
```

In [10]:
```
df_shift, y = make_forecasting_frame(x, kind="price", max_timeshift=10, rolling_direction=1)
df_shift
```

Out[10]:

	time	value	id	kind
154008	2016-01-11 21:30:00	100.0	2016-01-11 21:40:00	price
136892	2016-01-11 21:30:00	100.0	2016-01-11 21:50:00	price
154009	2016-01-11 21:40:00	100.0	2016-01-11 21:50:00	price
119777	2016-01-11 21:30:00	100.0	2016-01-11 22:00:00	price
136893	2016-01-11 21:40:00	100.0	2016-01-11 22:00:00	price
154010	2016-01-11 21:50:00	100.0	2016-01-11 22:00:00	price
102663	2016-01-11 21:30:00	100.0	2016-01-11 22:10:00	price
119778	2016-01-11 21:40:00	100.0	2016-01-11 22:10:00	price
136894	2016-01-11 21:50:00	100.0	2016-01-11 22:10:00	price
154011	2016-01-11 22:00:00	110.0	2016-01-11 22:10:00	price
85550	2016-01-11 21:30:00	100.0	2016-01-11 22:20:00	price
102664	2016-01-11 21:40:00	100.0	2016-01-11 22:20:00	price
119779	2016-01-11 21:50:00	100.0	2016-01-11 22:20:00	price
136895	2016-01-11 22:00:00	110.0	2016-01-11 22:20:00	price
154012	2016-01-11 22:10:00	400.0	2016-01-11 22:20:00	price
68438	2016-01-11 21:30:00	100.0	2016-01-11 22:40:00	price

In [12]:
```
y.head()
```

Out[12]:
```
date
2016-01-11 21:40:00    100
2016-01-11 21:50:00    100
2016-01-11 22:00:00    110
2016-01-11 22:10:00    400
2016-01-11 22:20:00    400
Name: value, dtype: int64
```

In [13]:
```
from tsfresh import extract_relevant_features
```

Extracting relevant features.

In [14]:
```
from tsfresh import select_features
from tsfresh.utilities.dataframe_functions import impute

impute(X)
features_filtered = select_features(X, y)
```

```
WARNING:tsfresh.feature_selection.relevance:Infered classification as machine learning task
```

In [15]:
```
features_filtered.head()
```

2016-01-11 21:40:00	91.421879	100.0	111.541685	100.0	87.
2016-01-11 21:50:00	91.421879	100.0	111.541685	100.0	87.
2016-01-11 22:00:00	91.421879	100.0	111.541685	100.0	87.
2016-01-11 22:10:00	91.421879	100.0	111.541685	100.0	87.
2016-01-					

Getting final accuracy.

```python
    # write obtained data
    TestModels = TestModels.append([tmp])

TestModels.set_index('Model', inplace=True)

print(TestModels)
```

```
                          MAE_Test     MAE_Train      MSE_Test     MSE_Train  \
Model
LinearRegression       2.622187e+01  2.409460e+01  6.085721e+03  2.077187e+03
RandomForestRegressor  2.465078e+01  9.751281e+00  2.382641e+03  4.056849e+02
MLPRegressor           1.188270e+08  3.977830e+08  3.407596e+19  6.007033e+20

                           R2_Test      R2_Train     RMSE_Test    RMSE_Train
Model
LinearRegression      -4.474209e-01  5.351238e-01  7.801103e+01  4.557617e+01
RandomForestRegressor  4.333154e-01  9.092074e-01  4.881230e+01  2.014162e+01
MLPRegressor          -8.104589e+15 -1.344379e+17  5.837462e+09  2.450925e+10
```

```
In [ ]:
```

Boruta

An all relevant feature selection wrapper algorithm. It finds relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies (shadows).

First, build the model.

Building Model

```
In [2]: df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
        x_train=df_train.iloc[:,1:]
        y_train=df_train['Appliances']
        scaler.fit(x_train)
        x_train_sc=scaler.transform(x_train)
        x_test=df_test.iloc[:,1:]
        y_test=df_test['Appliances']
        x_test_sc=scaler.transform(x_test)
```

Running Boruta on this model

Running the boruta.

```
In [4]: import pandas as pd
        #from sklearn.ensemble import RandomForestClassifier
        from boruta import BorutaPy

        # load X and y
        # NOTE BorutaPy accepts numpy arrays only, hence the .values attribute
        X = x_train_sc
        y = y_train

        # define random forest classifier, with utilising all cores and
        # sampling in proportion to y labels
        rf = RandomForestRegressor(n_jobs=-1, max_depth=25)

        # define Boruta feature selection method
        feat_selector = BorutaPy(rf, n_estimators='auto', verbose=2)

        # find all relevant features
        feat_selector.fit(X, y)
```

```
Iteration:      1 / 100
Confirmed:      0
Tentative:      45
Rejected:       0
Iteration:      2 / 100
Confirmed:      0
Tentative:      45
Rejected:       0
Iteration:      3 / 100
Confirmed:      0
Tentative:      45
Rejected:       0
Iteration:      4 / 100
Confirmed:      0
Tentative:      45
Rejected:       0
Iteration:      5 / 100
Confirmed:      0
Tentative:      45
```

We need to select only helping features from our model.

We have to select only few features from our dataset which would help us give better results in our prediction.

In [10]:
```
# check selected features
feat_selector.support_
```

Out[10]:
```
array([ True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False, False,  True,  True,  True,  True,
        True, False,  True,  True, False, False, False, False, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False])
```

Implementing the optimised features which we got from above step after performing boruta.

In [11]:
```
column_list =['lights','RH_1','T2','RH_2','T3','RH_3','T4','RH_4','T5','RH_5','T6','RH_7','T8','RH_8','T9','RH_9','Press_mm_hg','
x_train=df_train.iloc[:,1:]
x_train= x_train[column_list]
print(x_train.shape)
y_train=df_train['Appliances']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:]
x_test = x_test[column_list]
print(x_test.shape)
y_test=df_test['Appliances']
x_test_sc=scaler.transform(x_test)
```

```
(11982, 20)
(5136, 20)
```

Making a model for random forest

In [12]:
```
rf=RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=25,
           max_features='sqrt', max_leaf_nodes=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=350, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
rf.fit(x_train_sc,y_train)
```

Out[12]:
```
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=25,
           max_features='sqrt', max_leaf_nodes=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=350, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Checking the accuracy of training and testing data

Checking the accuracy of data for training set.

In [13]:
```
y_train_pred=rf.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
print("MAPE :",mean_absolute_percentage_error(y_train,y_train_pred))
```

```
R2   : 0.999365728603491
MAE  : 0.978722739191654
RMSE : 1.6866280335790647
MAPE : 1.5898191622381659
```

Checking the accuracy of data for testing set.

In [14]:
```
y_test_pred=rf.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
print("MAPE :",mean_absolute_percentage_error(y_test,y_test_pred))
```

```
R2   : 0.5493916369664105
MAE  : 21.217272013210984
RMSE : 43.3163158169478
MAPE : 25.665825612557814
```

6. Model Validation and Selection

Creating a model

In [1]:
```
import pandas as pd
import datetime
import numpy as np
import sklearn
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.grid_search import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import *
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 6, 4
def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

Now the model is ready, let's find ut how the data is distributed.

In [2]:
```
energy.head()
```

Out[2]:

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	DOW_Sunday	DOW_Thursday	DOW_Tuesc
27	100	20	21.356667	45.826667	20.666667	45.163333	20.390000	46.090000	19.390000	47.500000	...	0	0	
28	100	20	21.390000	45.690000	20.700000	45.060000	20.390000	46.090000	19.426667	47.993333	...	0	0	
29	100	20	21.500000	45.333333	20.700000	44.933333	20.390000	46.060000	19.566667	48.466667	...	0	0	
30	110	20	21.500000	45.126667	20.790000	44.633333	20.390000	46.000000	19.666667	48.093333	...	0	0	
31	400	20	21.533333	44.966667	20.790000	44.360000	20.426667	45.933333	19.600000	47.500000	...	0	0	

5 rows × 46 columns

Now when we now how the initial data is distributed, let's find out whethere it has any null values. Also, we have to find what are the nature of the data in our dataset.

In [3]:
```
energy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17118 entries, 27 to 19697
Data columns (total 46 columns):
Appliances       17118 non-null int64
```

Detecting overfitting.

In [4]:
```
# Overfitting detecting
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
```

Making the model to help us better analyze the data better.

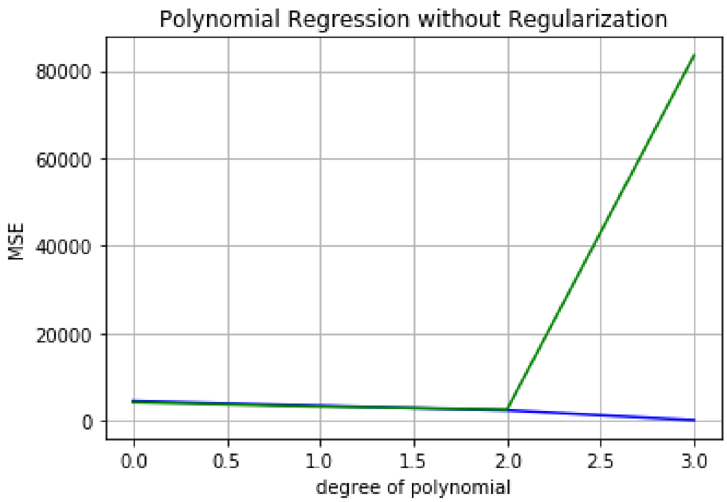
In [5]:
```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train.iloc[:,1:]
y_train=df_train['Appliances']
scaler.fit(x_train)
X_train=scaler.transform(x_train)
x_test=df_test.iloc[:,1:]
y_test=df_test['Appliances']
X_test=scaler.transform(x_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2026: FutureWarning: From version 0.21, test_size
will always complement train_size unless both are specified.
  FutureWarning)
```

Regularization

In [6]:
```
# Set random_state as a const to make sure the same split will be generated every time you run the code.
#X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=0)

# Containers for note down the MSE
train_mse_list = []
test_mse_list = []
degree_of_polynomial = []
```

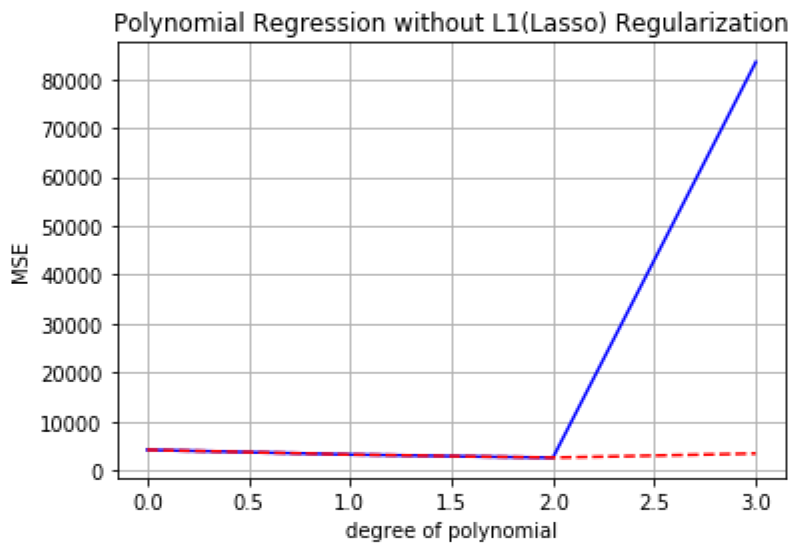
In [7]:
```
# MSE of quartic expression will be very large compared with the lower degree, so let's just end at 3.
for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    # Transfer the X to a polynomial form by using fit_transform
    X_train_ = model.fit_transform(X_train)
    X_test_ = model.fit_transform(X_test)
    lm = linear_model.LinearRegression()
    lm.fit(X_train_, y_train)
    train_pred = lm.predict(X_train_)
    train_mse_list.append(mean_squared_error(y_train, train_pred))
    test_pred = lm.predict(X_test_)
    test_mse_list.append(mean_squared_error(y_test, test_pred))
    degree_of_polynomial.append(i)
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_pred))
    print("MAE  :",mean_absolute_error(y_train,train_pred))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_pred)))
    print("MAPE :",mean_absolute_percentage_error(y_train,train_pred))
    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_pred))
    print("MAE  :",mean_absolute_error(y_test,test_pred))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_pred)))
    print("MAPE :",mean_absolute_percentage_error(y_test,test_pred))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without Regularization')
plt.plot(degree_of_polynomial, train_mse_list, '-b', degree_of_polynomial, test_mse_list, '-g')
plt.show()
```



Performing Lasso

Performing L1 (lasso)

In [8]:
```
from sklearn.linear_model import Lasso

# Randomly pick a alpha value for regularization
l1reg = Lasso(alpha=0.000003, normalize=True)

l1reg_test_mse_list = []

for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    X_train_ = model.fit_transform(X_train)
    X_test_ = model.fit_transform(X_test)

    l1reg.fit(X_train_, y_train)
    train_pred_l1 = l1reg.predict(X_train_)
    test_pred_l1 = l1reg.predict(X_test_)
    l1reg_test_mse_list.append(mean_squared_error(y_test, test_pred_l1))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_pred_l1))
    print("MAE  :",mean_absolute_error(y_train,train_pred_l1))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_pred_l1)))
    print("MAPE :",mean_absolute_percentage_error(y_train,train_pred_l1))
    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_pred_l1))
    print("MAE  :",mean_absolute_error(y_test,test_pred_l1))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_pred_l1)))
    print("MAPE :",mean_absolute_percentage_error(y_test,test_pred_l1))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without L1(Lasso) Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, l1reg_test_mse_list, '--r')
plt.show()
```



Cross Validation Technique

Testing for training dataset.

In [15]:
```
#df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
#x_train=df_train.iloc[:,1:]
#y_train=df_train['Appliances']
#scaler.fit(x_train)
#X_train=scaler.transform(x_train)
#x_test=df_test.iloc[:,1:]
#y_test=df_test['Appliances']
#X_test=scaler.transform(x_test)
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
accuracy_train = cross_val_score(estimator = LinearRegression() , X = X_train, y = y_train , cv = 10)

accuracy_train
```

Out[15]:
```
array([0.23939319, 0.20994031, 0.28080389, 0.24029031, 0.24235254,
       0.21612244, 0.23405993, 0.24911729, 0.24016138, 0.26527946])
```

Mean accuracy.

In [16]:
```
accuracy_train.mean()
```

Out[16]:
```
0.24175207193743198
```

Performing on testing dataset.

In [17]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
accuracy_test = cross_val_score(estimator = LinearRegression() , X = X_test, y = y_test , cv = 10)

accuracy_test
```

Out[17]:
```
array([0.15211076, 0.20928208, 0.25724492, 0.20743369, 0.26784327,
       0.25292395, 0.28916533, 0.22152727, 0.18031048, 0.22997972])
```

Getting Accuracy of model

In [18]:
```
accuracy_test.mean()
```

Out[18]:
```
0.2267821480096499
```

In [ ]:
```

```

7 Final Pipeline

In [238]:
```
from sklearn.preprocessing import MinMaxScaler
min_max=MinMaxScaler()
```

In [239]:
```
X_train_minmax = min_max.fit_transform(X_train)
```

```
C:\Users\ritua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323: DataConversionWarning: Da
ta with input dtype uint8, int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [240]:
```
X_test_minmax=min_max.fit_transform(X_test)
```

```
C:\Users\ritua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323: DataConversionWarning: Da
ta with input dtype uint8, int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [241]:
```
from sklearn.preprocessing import StandardScaler
```

In [242]:
```
scaler = StandardScaler()
```

In [243]:
```
X_train_scale=scaler.fit_transform(X_train)
```

```
C:\Users\ritua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\preprocessing\data.py:617: DataConversionWarning: Da
ta with input dtype uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Users\ritua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\base.py:462: DataConversionWarning: Data with input
dtype uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

Error metric

```python
In [245]: error_metric = pd.DataFrame({'r2_train': [],
                             'r2_test': [],
                              'rms_train':[],
                             'rms_test': [],
                             'mae_train': [],
                             'mae_test':[],
                             'mape_train':[],
                             'mape_test':[]})

rmse_dict = {}

def calc_error_metric(modelname, model, X_train_scale, y_train, X_test_scale, y_test):
    global error_metric
    y_train_predicted = model.predict(X_train)
    y_test_predicted = model.predict(X_test)

    #MAE, RMS, MAPE, R2

    r2_train = r2_score(y_train, y_train_predicted)
    r2_test = r2_score(y_test, y_test_predicted)

    rms_train = sqrt(mean_squared_error(y_train, y_train_predicted))
    rms_test = sqrt(mean_squared_error(y_test, y_test_predicted))

    mae_train = mean_absolute_error(y_train, y_train_predicted)
    mae_test = mean_absolute_error(y_test, y_test_predicted)

    mape_train = np.mean(np.abs((y_train - y_train_predicted) / y_train)) * 100
    mape_test = np.mean(np.abs((y_test - y_test_predicted) / y_test)) * 100

    rmse_dict[modelname] = rms_test

    df_local = pd.DataFrame({'Model':[modelname],
                             'r2_train': [r2_train],
                             'r2_test': [r2_test],
                             'rms_train':[rms_train],
                             'rms_test': [rms_test],
                             'mae_train': [mae_train],
                             'mae_test': [mae_test],
                             'mape_train':[mape_train],
                             'mape_test':[mape_test]})
```

Getting best model.

In [246]:
```
from sklearn.neural_network import MLPRegressor
```

In [247]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [248]:
```
from math import sqrt
```

In [249]:
```
import operator
```

In [250]:
```
# Regression
clf = LinearRegression()
clf.fit(X_train, y_train)
calc_error_metric('Linear Regression', clf, X_train, y_train, X_test, y_test)
print('Linear Regression Peformed')

# Random Forest
rf = RandomForestRegressor(n_estimators=100, max_depth=7)
rf.fit(X_train, y_train)
calc_error_metric('RandomForest', rf, X_train, y_train, X_test, y_test)
print('RandomForest Performed')

# Neural network
nn = MLPRegressor()
nn.fit(X_train, y_train)
calc_error_metric('Neural Network', nn, X_train, y_train, X_test, y_test)
print('Neural Network performed')

#### Calculate best model
best_model =  min(rmse_dict.items(),key=operator.itemgetter(1))[0]
print('Best Model is-', best_model)

#### Write the error
error_metric.to_csv('Error_metrics.csv')
```

```
Linear Regression Peformed
RandomForest Performed
Neural Network performed
Best Model is  RandomForest
```

In [90]:
```
error_metric
```

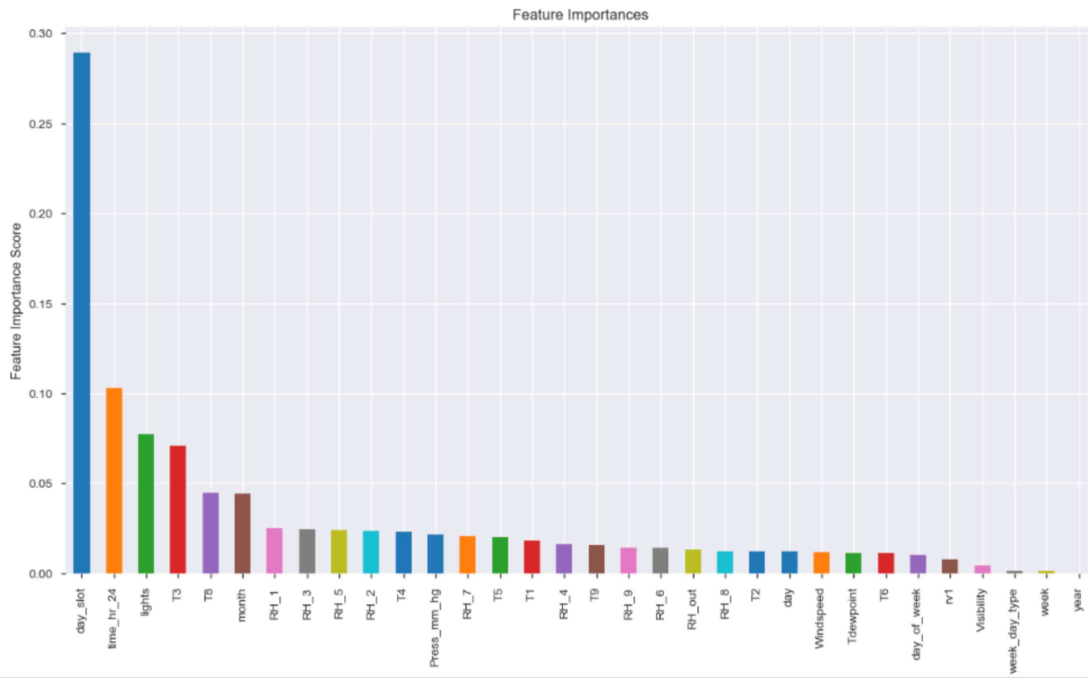
Out[90]:

	Model	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	Linear Regression	52.491344	52.625376	62.606520	60.304493	0.178557	0.189465	90.333257	93.139558
0	RandomForest	45.879759	44.400356	51.149829	47.843062	0.272879	0.359836	84.988958	82.773972
0	Neural Network	51.305092	51.746769	57.898395	55.929704	0.124915	0.122877	93.236107	96.889885

Getting feature importance of random forest

In [91]:
```
# Feature Importance in Random Forest
imp_feature = pd.Series(rf.feature_importances_,X_train.columns).sort_values(ascending=False)
imp_feature.plot(kind='bar', title='Feature Importances',figsize = (15,8))
plt.ylabel('Feature Importance Score')
```

Out[91]: Text(0,0.5,'Feature Importance Score')



Doing prediction by random forest.

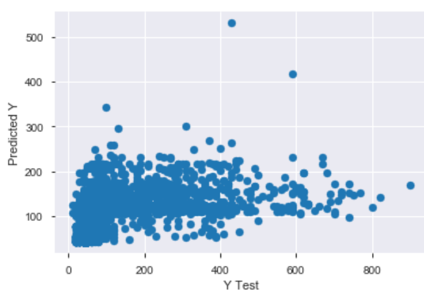
In [231]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [232]:
```
# Random Forest Regressor
# model = RandomForestRegressor(n_estimators=300, max_features = 11)
#     # create the RFE model and select 3 attributes
# rfe = RFE(model)
# rfe = rfe.fit(X_train, y_train)
#     # summarize the selection of the attributes
# print(rfe.support_)
# print(rfe.ranking_)
# print(rfe.n_features_)
#     #Check the accuracy of the model
# rfe.score(X_train, y_train)
rand_forest_model = RandomForestRegressor(max_depth=5, random_state=0) # build model
rand_forest_model = rand_forest_model.fit(X_train, y_train.values.ravel()) # train model
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=5,
           max_features='auto', max_leaf_nodes=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=10, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
feature_importance = rand_forest_model.feature_importances_
R2 = rand_forest_model.score(X_train, y_train) # coefficient of determination
rand_forest_model_predictions = rand_forest_model.predict(X_test) # make predictions
```

```
C:\Users\ritua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ensemble\forest.py:248: FutureWarning: The default v
alue of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [233]:
```
# rand_forest_model_metrics = view_metrics(y_test, rand_forest_model_predictions,'Random Forest Regressor', samples = 50, total_s
plt.scatter(y_test,rand_forest_model_predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

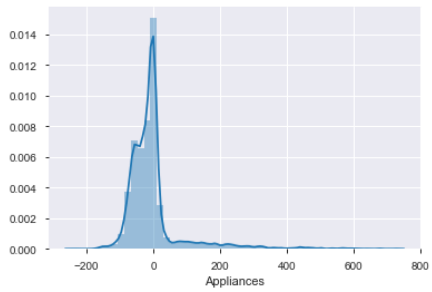
Out[233]: Text(0,0.5,'Predicted Y')



In [234]:
```
print('MAE:', metrics.mean_absolute_error(y_test, rand_forest_model_predictions))
print('MSE:', metrics.mean_squared_error(y_test, rand_forest_model_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, rand_forest_model_predictions)))
```

```
MAE: 48.96025500004364
MSE: 7862.571568228872
RMSE: 88.67114281562448
```

```
In [235]: sns.distplot((y_test-rand_forest_model_predictions),bins=50)
Out[235]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa2c04d320>
```



Predicting using Random forest algorithms

```
In [236]: from sklearn.feature_selection import RFE
```

```
In [237]: model = RandomForestRegressor(n_estimators=20, max_features = 11)
rfe = RFE(model)
rfe = rfe.fit(X_train, y_train)
    # summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
print(rfe.n_features_)
    #Check the accuracy of the model
rfe.score(X_train, y_train)
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True False False  True False
 False False False  True False False False False False False False False
 False False  True False False False False False False False]
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  3  4  1  2
 24  7  5  1  8  9 11 15 23 17 21 12  6 13  1 19 10 18 22 14 20 16]
23
```

```
Out[237]: 0.9224397516383712
```

So, our model gives 92 % accuracy

8. Summary

By the above analysis we have concluded following points regarding the data given to us.

- Best model to analyze and predict is Random Forest.
- There are many columns which are highly correlated, and they need to be removed in order to get good prediction
- The data have almost none outliers and no NULL valued column. So, the data is almost clean.