Machine Learning with Energy Dataset

INFO 7390

Advance Data Science & Architecture

Professor: Srikanth Krishnamurthy

By: Team 5

Akash Jagtap

Jerin Rajan

Nitin Prince Reuben

Contents

Part 1: Research Papers Referred

Paper 1: Data driven prediction model of energy use of appliances in a low energy house

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

Summary :

The research paper aims at understanding appliances as they represent a significant part of the electrical energy demand with measurements of temperature and humidity sensors in multiple areas in the house. It states that televisions and consumer electronics operating in *standby* were attributed to a 10.2% increase in the electricity consumption. Best model is GBM (Gradient Boosting Model) which explain 97% of the variance in training set and 57% in the testing set. Data from kitchen, laundry and living room has the highest importance for the energy prediction. The various appliances used during the research are Space Heating, Space Cooling, Water Heating, Refrigerator, Cooking, Clothes dryers, Freezers, Lighting, Clothes Washers, Dishwashers, Color TV and boxes, personal computer and others.

The energy efficiency and usage depends on the type and number of electrical appliances and the use of the appliances by the occupants. Data from 454 houses and 140 commercial building in the Pacific Northwest was used in one example which uses Markov Chain Monte Carlo technique to generate synthetic data. According to the research, the refrigerators have a uniform load profile, while clothes washers, clothes dryers and dishwashers are very user-dependent and thus vary from household to household and time of day. The article ranked highest the clothes dryers for the potential of demand response mostly because of their large power demand.

The electricity load prediction 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 and external weather. The regression test used heating degree days, cooling degree days, gross domestic product, and humidity (when available) and could explain the variability of the monthly demand between 91 and 95%. One-minute interval power measurement in 12 houses, including individual devices (furnace, air conditioner, range/cooker, clothes dryer, dishwasher and domestic hot water heater). Another data testing of 1628 households. 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 among the most important determinants of daily minimum consumption. Statistics also observed that being at home during the day correlated with lower appliance efficiency which means increased use of appliances due to occupants in the house.

In UK, appliance ownership and their use was studied for 183 dwellings using an odds ratio analysis. The study suggested that households owning more than thirty appliances have an increased probability of having a high electrical energy demand due to which these households

own more than four or more IT devices, more than five entertainment items, an electric oven, hob or range, two or more preservation and cooling appliances or three or more laundry machines.

The data was collected from energy counters every 10min which includes the heat recovery ventilation unit, domestic hot water heat pump, the energy consumption of appliances, lighting, and electric baseboard heaters.

The data set obtained was split in training and test validation using CARET's methodology. The highest correlation was found between T1 and T3 which are the two different observations and the lowest correlation is between lights and T2 and lights and RH_3. These analyses were depicted in pairs plot of bivariate scatter plot and histogram. Also, the same appliances energy use was demonstrated in heat map visual in weekly basis. The finding from heat map for the 4 consecutive week is that during the evening the appliances energy usage is higher.

The data set consists of multiple features so feature filtering and selection is performed using the Boruta package. This package finds the importance of attributes with its shadow attributes obtained from shuffling the original dataset. This finds one such appliance NSM with highest importance. This is obtained by finding the optimal variable by using RFE to minimize the RMSE. Various techniques were used like the regression, SVM with tuning parameters.

The appliances energy consumption during the exploratory data analysis, training and model selection tells us the various relationships. For example if the humidity in the room increased when occupants arrived in the room and would drop if it was empty. There is a negative correlation between rooms and appliances there is almost negative correlation. Various filtering methods was used for feature extraction. The best models were obtained by RF and GBM by evaluating it RMSE and R square

This research paper analyses various relationships between parameters using the pairplots. Using different model to get better RMSE and Rsquare and data from weather station helps to improve the accuracy. The future scope could involve gettting solar radiation and precipitation, CO2, noise into consideration for better prediction

Paper 2: Renewable and Sustainable Energy Reviews

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

Summary :

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:

- AI based method also called as the black box predicts energy use without knowing the internal relationship of the building and the individual components.
- Hybrid is called as grey box which requires detailed building information for simulation for model development.

There are 2 prediction method which are widely used for energy systems. They are as follows:

- Single Prediction – utilize one learning algorithm
- 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.

Paper 3: Prediction of appliances energy use in smart home

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

Summary :

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.

Part 2: Exploratory Data Analysis

Data - https://github.com/LuisM78/Appliances-energy-prediction-data

To perform EDA, we should import few python libraries which is important and will help us perform calculations.

Importing Libraries

In [66]:
```
import pandas as pd
import datetime
import calendar
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from matplotlib.pylab import rcParams
import seaborn as sns
import tkinter as Tk
from tkinter import filedialog
from tkinter import *
import sklearn
from sklearn.preprocessing import StandardScaler
```

Each of these libraries have its own uses. For example:

- Pandas – used to import 'csv' file
- Datetime – used to differentiate hours from minutes
- Calender – used to convert month from number to word
- Numpy – used to solve matrix
- Matplotlib – used to draw graph
- Seaborn – it is also a visualization library based on matplotlib
- Tkinter – it used for browsing file.

We also have to understand how much data does the file contains.

Checking data content

In [4]:
```
print("rows,columns: ",f.shape)
f.head()
```

```
rows,columns:  (19735, 29)
```

Out[4]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_mm_hg	RH_out	Windspe
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	7.0000
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	6.6666
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	6.3333
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	6.0000
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	5.6666

5 rows × 29 columns

List of features in the dataset. Please refer the variable description txt file to know what are these features.

In [5]:
```
f.columns
```

Out[5]:
```
Index(['date', 'Appliances', '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'],
      dtype='object')
```

Geting statistics of data present in each column of dataframe

In [6]:
```
f.describe()
```

Out[6]:

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	...	19735
mean	97.694958	3.801875	21.686571	40.259739	20.341219	40.420420	22.267611	39.242500	20.855335	39.026904	...	19
std	102.524891	7.935988	1.606066	3.979299	2.192974	4.069813	2.006111	3.254576	2.042884	4.341321	...	2
min	10.000000	0.000000	16.790000	27.023333	16.100000	20.463333	17.200000	28.766667	15.100000	27.660000	...	14
25%	50.000000	0.000000	20.760000	37.333333	18.790000	37.900000	20.790000	36.900000	19.530000	35.530000	...	18
50%	60.000000	0.000000	21.600000	39.656667	20.000000	40.500000	22.100000	38.530000	20.666667	38.400000	...	19
75%	100.000000	0.000000	22.600000	43.066667	21.500000	43.260000	23.290000	41.760000	22.100000	42.156667	...	20
max	1080.000000	70.000000	26.260000	63.360000	29.856667	56.026667	29.236000	50.163333	26.200000	51.090000	...	24

8 rows × 28 columns

Analysing similar Features

In [7]:
```
# Columns for temperature sensors
temp_cols = ["T1", "T2", "T3", "T4", "T5", "T6", "T7", "T8", "T9", "T_out", "Tdewpoint"]

# Columns for humidity sensors
rho_cols = ["RH_1", "RH_2", "RH_3", "RH_4", "RH_5", "RH_6", "RH_7", "RH_8", "RH_9", "RH_out"]

# Columns for weather data
weather_cols = ["T_out", "Tdewpoint", "RH_out","Press_mm_hg", "Windspeed", "Visibility"]

# Target variable column
target = ["Appliances", "lights"]
```

In [9]:
```
f[temp_cols].corr().abs()
```

Out[9]:

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T_out	Tdewpoint
T1	1.000000	0.836834	0.892402	0.877001	0.885247	0.654769	0.838705	0.825413	0.844777	0.682846	0.571309
T2	0.836834	1.000000	0.735245	0.762066	0.720550	0.801186	0.663660	0.578191	0.675535	0.792255	0.582602
T3	0.892402	0.735245	1.000000	0.852778	0.888169	0.686882	0.847374	0.795283	0.901324	0.699417	0.645886
T4	0.877001	0.762066	0.852778	1.000000	0.871813	0.652350	0.877763	0.796256	0.889439	0.663478	0.519471
T5	0.885247	0.720550	0.888169	0.871813	1.000000	0.629161	0.870624	0.824981	0.911055	0.651321	0.588362
T6	0.654769	0.801186	0.686882	0.652350	0.629161	1.000000	0.619085	0.482836	0.667177	0.974787	0.764242
T7	0.838705	0.663660	0.847374	0.877763	0.870624	0.619085	1.000000	0.882123	0.944776	0.631293	0.466625
T8	0.825413	0.578191	0.795283	0.796256	0.824981	0.482836	0.882123	1.000000	0.869338	0.502842	0.391810
T9	0.844777	0.675535	0.901324	0.889439	0.911055	0.667177	0.944776	0.869338	1.000000	0.668220	0.581483
T_out	0.682846	0.792255	0.699417	0.663478	0.651321	0.974787	0.631293	0.502842	0.668220	1.000000	0.790661
Tdewpoint	0.571309	0.582602	0.645886	0.519471	0.588362	0.764242	0.466625	0.391810	0.581483	0.790661	1.000000

The temperature range for alomst all recorded reading is between 14.89 to 29.86 degree Celcius. However the temperature range for T6, Tdewpoint and T_out is -6.6 to 28.9 degree Celcius, which may be due the fact that the T6 and T_out are the readings taken from the outside of the building. All the temperatures are very well correlated with each other.

In [12]:
```
f[rho_cols].describe()
```

Out[12]:

	RH_1	RH_2	RH_3	RH_4	RH_5	RH_6	RH_7	RH_8	RH_9	RH_out
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	40.259739	40.420420	39.242500	39.026904	50.949283	54.609083	35.388200	42.936165	41.552401	79.750418
std	3.979299	4.069813	3.254576	4.341321	9.022034	31.149806	5.114208	5.224361	4.151497	14.901088
min	27.023333	20.463333	28.766667	27.660000	29.815000	1.000000	23.200000	29.600000	29.166667	24.000000
25%	37.333333	37.900000	36.900000	35.530000	45.400000	30.025000	31.500000	39.066667	38.500000	70.333333
50%	39.656667	40.500000	38.530000	38.400000	49.090000	55.290000	34.863333	42.375000	40.900000	83.666667
75%	43.066667	43.260000	41.760000	42.156667	53.663333	83.226667	39.000000	46.536000	44.338095	91.666667
max	63.360000	56.026667	50.163333	51.090000	96.321667	99.900000	51.400000	58.780000	53.326667	100.000000

In [11]:
```
f[rho_cols].corr().abs()
```

Out[11]:

	RH_1	RH_2	RH_3	RH_4	RH_5	RH_6	RH_7	RH_8	RH_9	RH_out
RH_1	1.000000	0.797535	0.844677	0.880359	0.303258	0.245126	0.801122	0.736196	0.764001	0.274126
RH_2	0.797535	1.000000	0.678326	0.721435	0.250271	0.389933	0.690584	0.679777	0.676467	0.584911
RH_3	0.844677	0.678326	1.000000	0.898978	0.375422	0.514912	0.832685	0.828822	0.833538	0.356192
RH_4	0.880359	0.721435	0.898978	1.000000	0.352591	0.392178	0.894301	0.847259	0.856591	0.336813
RH_5	0.303258	0.250271	0.375422	0.352591	1.000000	0.263797	0.325808	0.359840	0.272197	0.185941
RH_6	0.245126	0.389933	0.514912	0.392178	0.263797	1.000000	0.357222	0.489580	0.391943	0.718587
RH_7	0.801122	0.690584	0.832685	0.894301	0.325808	0.357222	1.000000	0.883984	0.858686	0.378519
RH_8	0.736196	0.679777	0.828822	0.847259	0.359840	0.489580	0.883984	1.000000	0.855812	0.487355
RH_9	0.764001	0.676467	0.833538	0.856591	0.272197	0.391943	0.858686	0.855812	1.000000	0.359377
RH_out	0.274126	0.584911	0.356192	0.336813	0.185941	0.718587	0.378519	0.487355	0.359377	1.000000

We have almost all humidity features which are ranging from 20.46% to 63.36%. However for the RH_5 & RH_out and RH_6 we have range of 24% to 100% and 1% to 99.9%, respectively. This is due to the fact that RH_5 is the humidity recorded at the bathroom and RH_6 & RH_out are the humidities recorded at outside the building. Thus the correlation of the humidity features RH_1, RH_2, RH_3, RH_4, RH_7, RH_8, RH_9 has significant correlation with each other and very less correlation to other humidity features.

In [13]:
```
f[weather_cols].describe()
```

Out[13]:

	T_out	Tdewpoint	RH_out	Press_mm_hg	Windspeed	Visibility
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	7.411665	3.760707	79.750418	755.522602	4.039752	38.330834
std	5.317409	4.194648	14.901088	7.399441	2.451221	11.794719
min	-5.000000	-6.600000	24.000000	729.300000	0.000000	1.000000
25%	3.666667	0.900000	70.333333	750.933333	2.000000	29.000000
50%	6.916667	3.433333	83.666667	756.100000	3.666667	40.000000
75%	10.408333	6.566667	91.666667	760.933333	5.500000	40.000000
max	26.100000	15.500000	100.000000	772.300000	14.000000	66.000000

In [14]:
```
f[weather_cols].corr().abs()
```

Out[14]:

	T_out	Tdewpoint	RH_out	Press_mm_hg	Windspeed	Visibility
T_out	1.000000	0.790661	0.574197	0.143249	0.192936	0.077367
Tdewpoint	0.790661	1.000000	0.036506	0.244098	0.125972	0.042190
RH_out	0.574197	0.036506	1.000000	0.092017	0.176458	0.083125
Press_mm_hg	0.143249	0.244098	0.092017	1.000000	0.235032	0.040315
Windspeed	0.192936	0.125972	0.176458	0.235032	1.000000	0.007516
Visibility	0.077367	0.042190	0.083125	0.040315	0.007516	1.000000

We have pressure data ranges from 729.3 mmHg to 772.30 mmHg We have Windspeed recorded ranges from 0 to 14 m/s Also, the visbility recorded was ranges from 1 to 66 kms. This features have very low correlation factors among each other.

In [15]:
```
f[["rv1","rv2"]].describe()
```

Out[15]:

	rv1	rv2
count	19735.000000	19735.000000
mean	24.988033	24.988033
std	14.496634	14.496634
min	0.005322	0.005322
25%	12.497889	12.497889
50%	24.897653	24.897653
75%	37.583769	37.583769
max	49.996530	49.996530

In [16]:
```
f[["rv1","rv2"]].corr().abs()
```

Out[16]:

	rv1	rv2
rv1	1.0	1.0
rv2	1.0	1.0

In [18]:
```
(f['rv1']-f['rv2']==0).count() - f.shape[0]
```

Out[18]:
```
0
```

We have rv1 and rv2, 2 random variable which was just added to check validate the performance of the feature selections models. We found the perfect correlation between these two variables. Also, it is proved that the both columns are identical, giving redundancy in the dataset.

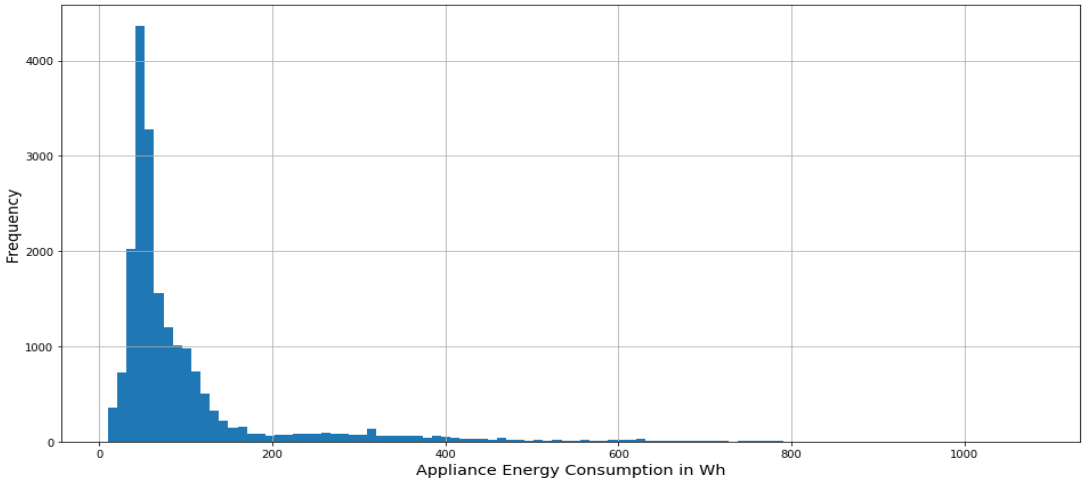
Histogram of Appliances:

In [21]:
```
# Histogram for appliances

plt.xlabel("Appliance Energy Consumption in Wh", fontsize="x-large")
plt.ylabel("Frequency", fontsize="x-large")

f["Appliances"].hist(figsize=(16, 8), bins=100)
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x233db30eda0>
```

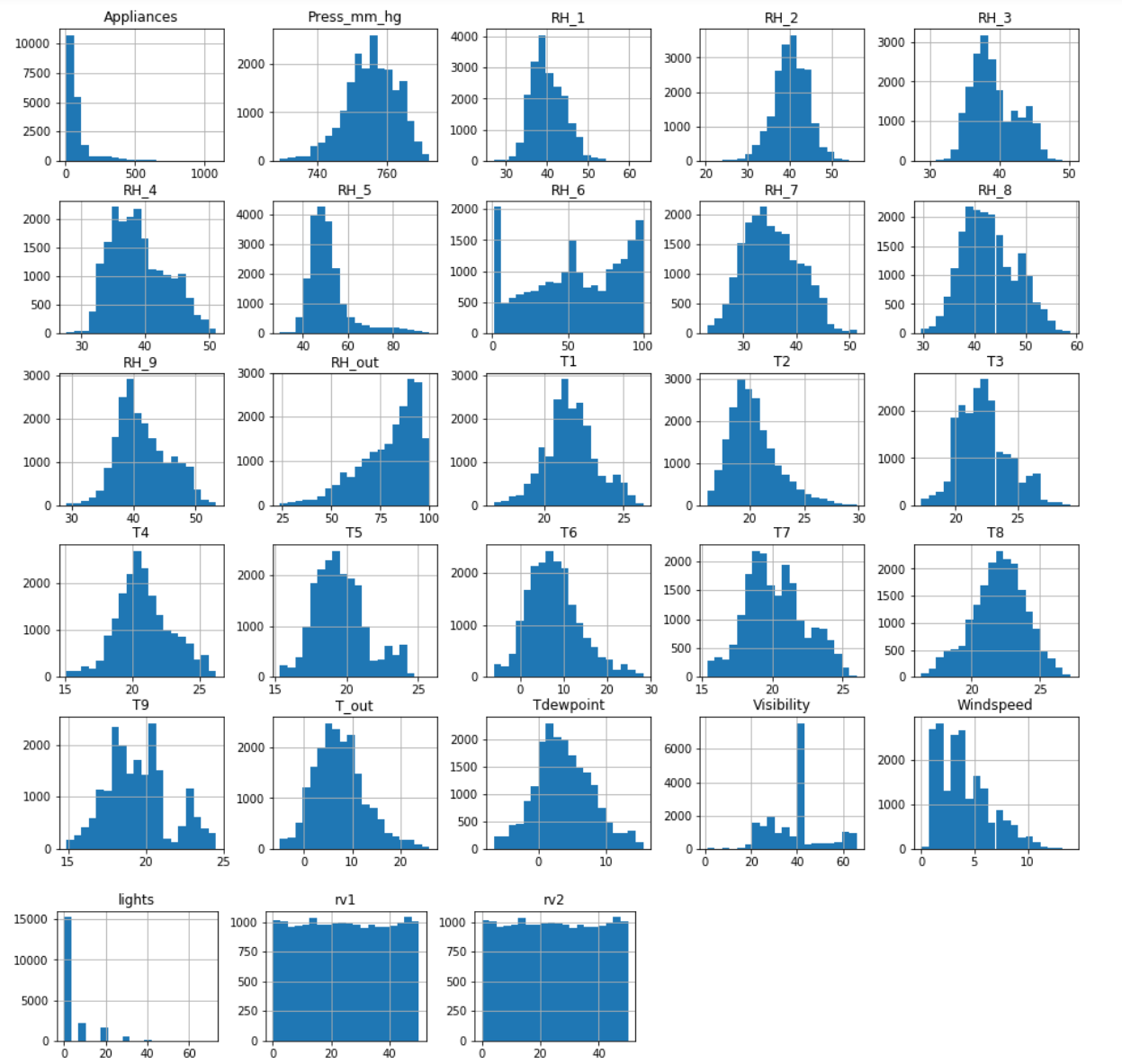


In [23]:
```
print("Percentage of dataset in range of 0-200 Wh")
print("{:.3f}%".format(
    (f[f.Appliances <= 200]["Appliances"].count()*100.0) / f.shape[0]
))
```

```
Percentage of dataset in range of 0-200 Wh
90.291%
```

We can see that Maximum appliance consumption is 1080 Wh. However, we have 90.21% of data for which the appliance energy consumption is less than 200 Wh.

Histograms of all features:



All temperature features follows normal distribution, but T9 and T7. Similarly, almost all humidity features follow normal distribution except RH_6 and RH_out. From the distribution plot, fetures visibility, windspeed, lights and appliances are skewed. Also, no features follows the similar distribution as appliances.

Checking for the high correlation combinations within our energy dataset.

In [30]:
```
f.columns
```

Out[30]:
```
Index(['date', 'Appliances', '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'],
      dtype='object')
```

In [31]:
```
corr_temp = f[['Appliances', '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']]
```

In [34]:
```
# To generate all pairs for given columns
from pydoc import help
from scipy.stats.stats import pearsonr
from itertools import combinations

for pair in combinations(corr_temp.columns, 2):
    col_1, col_2 = pair
    # Calculate the coefficient and p-value
    corr_coef, p_val = pearsonr(corr_temp[col_1], corr_temp[col_2])
    # Check for high correlation
    if abs(corr_coef) > 0.88:
        # Print details for pairs with high correlation
        print("Column pair : {}, {}".format(*pair))
        print("Correlation coefficient : {}".format(corr_coef))
        #print("p-value : {}".format(p_val))
```

```
Column pair : T1, T3
Correlation coefficient : 0.8924022851105653
Column pair : T1, T5
Correlation coefficient : 0.8852468734266528
Column pair : RH_1, RH_4
Correlation coefficient : 0.8803585351891954
Column pair : T3, T5
Correlation coefficient : 0.8881689127749807
Column pair : T3, T9
Correlation coefficient : 0.901323585082566
Column pair : RH_3, RH_4
Correlation coefficient : 0.8989782902830767
Column pair : T4, T9
Correlation coefficient : 0.8894391511402545
Column pair : RH_4, RH_7
Correlation coefficient : 0.8943012488817835
Column pair : T5, T9
Correlation coefficient : 0.9110551178067928
Column pair : T6, T_out
Correlation coefficient : 0.9747866900664536
Column pair : T7, T8
Correlation coefficient : 0.8821232262844055
Column pair : T7, T9
Correlation coefficient : 0.9447764235687443
Column pair : RH_7, RH_8
Correlation coefficient : 0.8839839643177076
Column pair : rv1, rv2
Correlation coefficient : 1.0
```

rv1 and rv2 shows perfect correlation. And as derived in above observations, rv2 is redundant feature. Feature T9 shows high correlation with T3, T4, T5, T7. Thus T9 can be consider as a overhead in the dataset. Similarly, T5 has a high correlation with the T9, T3 and T1 which thus acts added redundancy. Also, we know T6 and T_out are exterior temperatue features, and shows high degree of correlations. We can get rid of either of the above two fetures to lower the data redundancy.

It is important for us to analyze whether there are null values in the dataset and how are values distributed.

Checking for any null vaules

```
In [47]: f.isnull().sum() #checking null values

Out[47]: date           0
         Appliances     0
         lights         0
         T1             0
         RH_1           0
         T2             0
         RH_2           0
         T3             0
         RH_3           0
         T4             0
         RH_4           0
         T5             0
         RH_5           0
         T6             0
         RH_6           0
         T7             0
         RH_7           0
         T8             0
         RH_8           0
         T9             0
         RH_9           0
         T_out          0
         Press_mm_hg    0
         RH_out         0
         Windspeed      0
         Visibility     0
         Tdewpoint      0
         rv1            0
         rv2            0
         dtype: int64
```

Data does not have any null values. Now check for all datatypes within the data.

```
In [48]: f.dtypes

Out[48]: date            object
         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
         Visibility     float64
         Tdewpoint      float64
         rv1            float64
         rv2            float64
         dtype: object
```

All values are in 'float' except date column which needs to be converted to datetime format. Creating additional features using existing ones in the data so as to get more insights.

From the above two images, we can understand that there are no null values in the dataset and also all-important values are in ‘float’. This makes the data calculation part easy.

Creating additional features using existing ones in the data so as to get more insights.

In [35]:
```
f['date']=pd.to_datetime(f['date'])
f['year']=f['date'].dt.year
f['month']=f['date'].dt.month
f['day']=f['date'].dt.day
f['day_of_week']=f['date'].dt.weekday_name
f['time_hr_24']=f['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'

f['time_slot']=f['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'
f['week']=f['day'].map(week_num)
weekend = ['Saturday','Sunday']
def week_day_type(x):
    if x in weekend:
        return 'weekends'
    else:
        return 'weekdays'
f['week_day_type']=f['day_of_week'].map(week_day_type)
```

We are diving the time into four groups. Morning, afternoon, evening and night. This is just categorizing of data.

In the same way, we are categorizing the days into weeks as first 7 days are week 1, 2nd 7 days are week 2 and so on.

To analyze the dataset better, we have created 2 summary matrixes. This will make the calculations easy.

The two-summary matrix are as follows:

- 'df_summ': It contains sum and mean of all the relevant values.

```
In [37]: df_summ
```

Out[37]:

	Month	Tot_Appliance	Avg_Appliance	Tot_light	Avg_light	Tot_Energy	Avg_Energy	Mean_T1	Mean_RH_1	Mean_T2	...	Weekdays_Mean_T2	Weekdays_I
0	1.0	283510.0	97.026010	13800.0	4.722793	297310.0	101.748802	19.945175	42.284480	19.102295	...	18.860596	1
1	2.0	421550.0	100.945881	26490.0	6.343391	448040.0	107.289272	21.163817	41.010174	19.924704	...	19.978110	2
2	3.0	432800.0	96.953405	17480.0	3.915771	450280.0	100.869176	21.259760	37.923927	19.141096	...	19.202354	2
3	4.0	427200.0	98.888889	10440.0	2.416667	437640.0	101.305556	21.856511	39.722240	20.224405	...	20.249074	2
4	5.0	362950.0	94.199325	6820.0	1.770049	369770.0	95.969375	23.877728	41.219759	23.253625	...	23.256463	2

5 rows × 381 columns

From the above dataframe we can get insights like Tot_Energy consumed was highest in March where as it was lowest in January and so on.

- 'df_ min_max_summ': It contains minimum and maximum vales of all relevant data.

```
In [38]: df_min_max_summ
```

Out[38]:

	Month	Max_App	Max_light	Max_T1	Max_RH_1	Max_T2	Max_RH_2	Max_T3	Max_RH_3	Max_T4	...	Week5_Min_T8	Week5_Min_RH_8	Week5_Min_
0	1.0	1080.0	70.0	23.790000	63.360000	22.790000	50.260000	22.790	50.163333	22.463333	...	16.89	45.000000	16
1	2.0	770.0	50.0	24.100000	57.663333	23.600000	47.700000	25.500	49.363333	23.760000	...	20.00	29.600000	18
2	3.0	880.0	50.0	23.700000	50.330000	22.856667	47.060000	25.700	43.026667	23.200000	...	22.79	36.663333	19
3	4.0	900.0	30.0	23.666667	57.496667	24.600000	56.026667	27.600	44.363333	23.760000	...	20.29	36.790000	18
4	5.0	850.0	30.0	26.260000	54.666667	29.856667	54.090000	29.236	47.693333	26.200000	...	0.00	0.000000	0

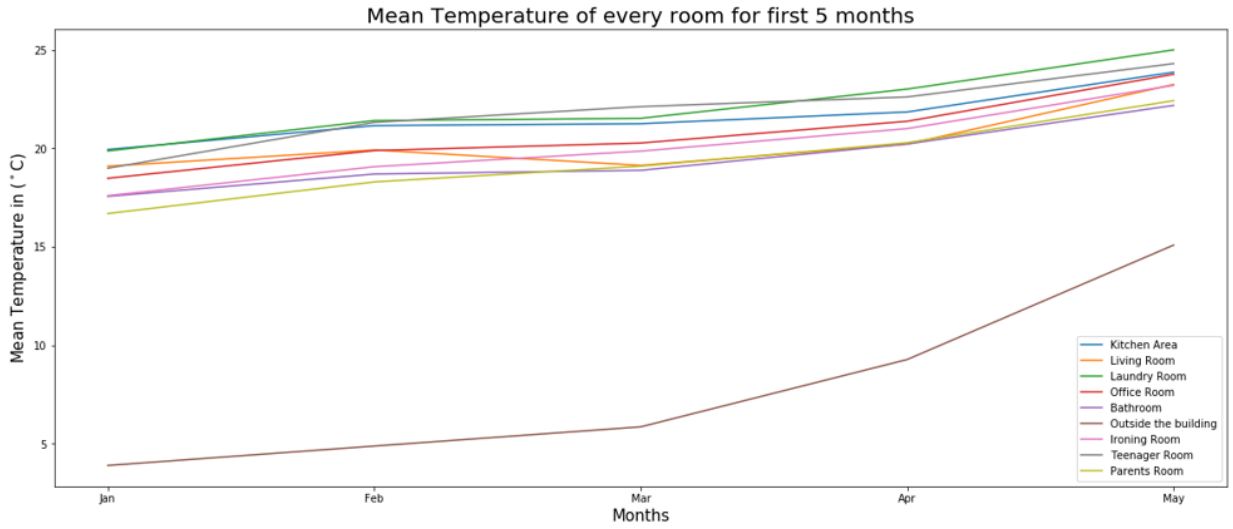
5 rows × 313 columns

From the above dataframe we can get insights like the max temperature T1 (i.e temperature in kitchen area) was 26.26 degree Celsius and it was recored in month of May.

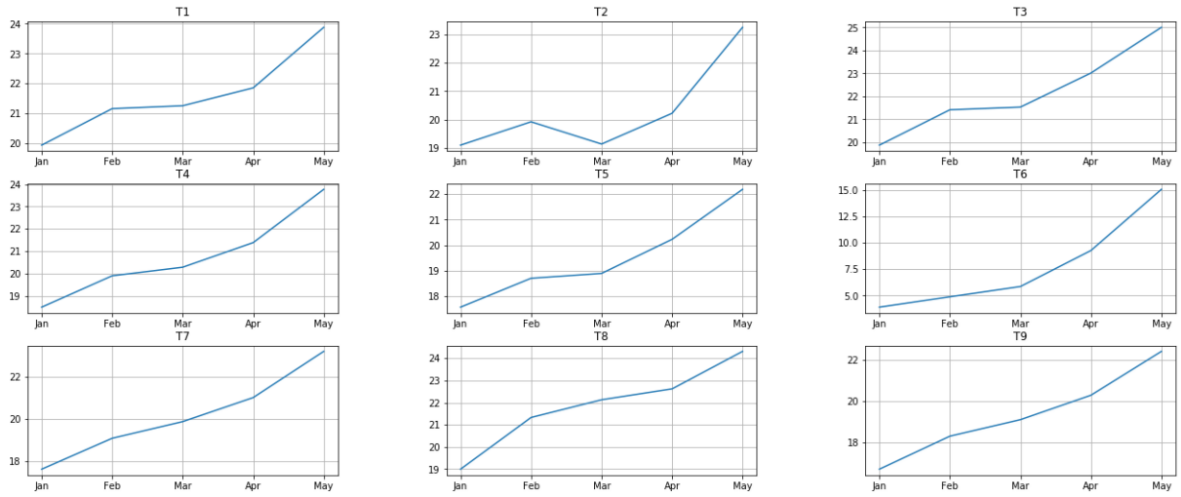
Its better to save these dataframes in a 'csv' file which will make it easy for us to refer.

```
In [39]: df_summ.to_csv('MeanSummaryDataOnMonthBasis.csv')
         df_min_max_summ.to_csv('MinMaxSummaryDataOnMonthBasis.csv')
```

Let’s start plot the graphs which will make this data easier to underdtand.



From the above graph we can say that average temperature in each room and outside the building kept on increasing from jan to may.Also, over the months the average temperature outside the bulding was very less as compared to other rooms.



This graph is like the previous graph. The mean temperature is separated and is plotted in different graphs to make it easier to understand.

T1: Temperature in kitchen area, in Celsius

T2: Temperature in living room area, in Celsius

T3: Temperature in laundry room area

T4: Temperature in office room, in Celsius

T5: Temperature in bathroom, in Celsius

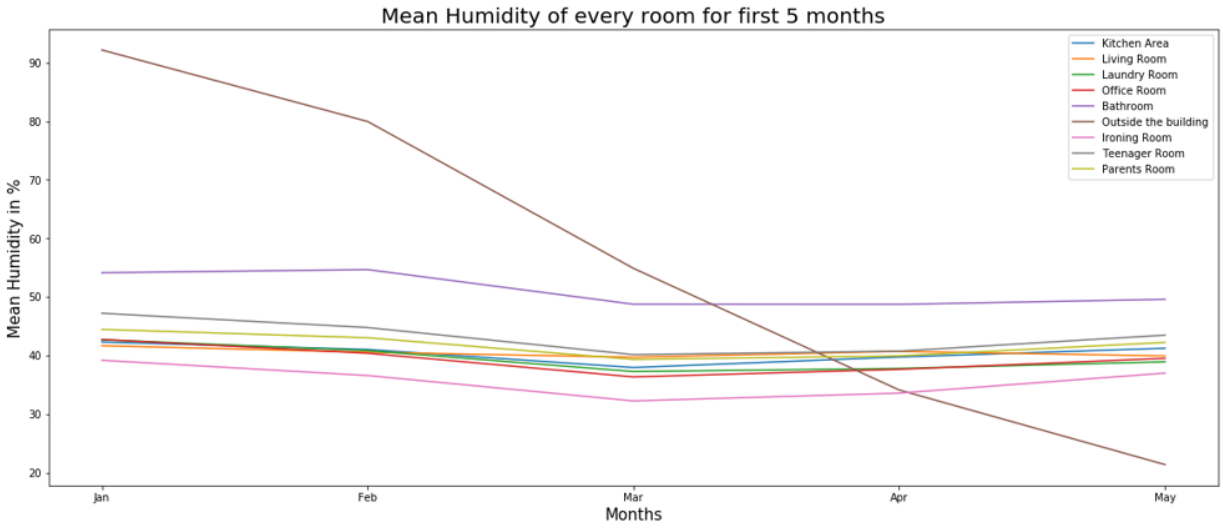
T6: Temperature outside the building (north side), in Celsius

T7: Temperature in ironing room, in Celsius

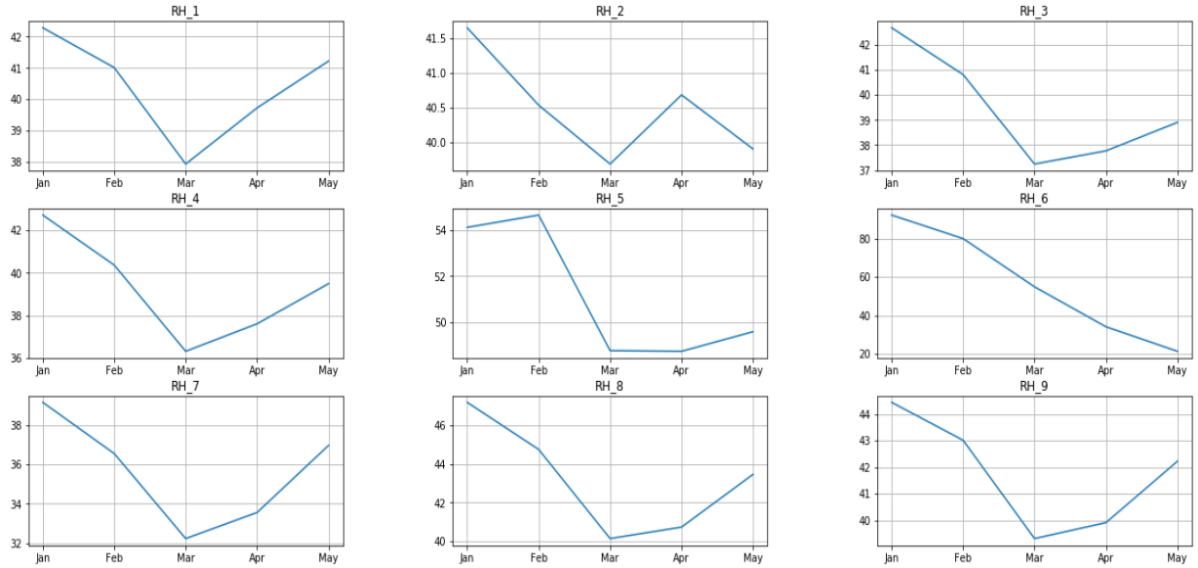
T8: Temperature in teenager room 2, in Celsius

T9: Temperature in parent's room, in Celsius

Humidity in the room is studied in the same way.



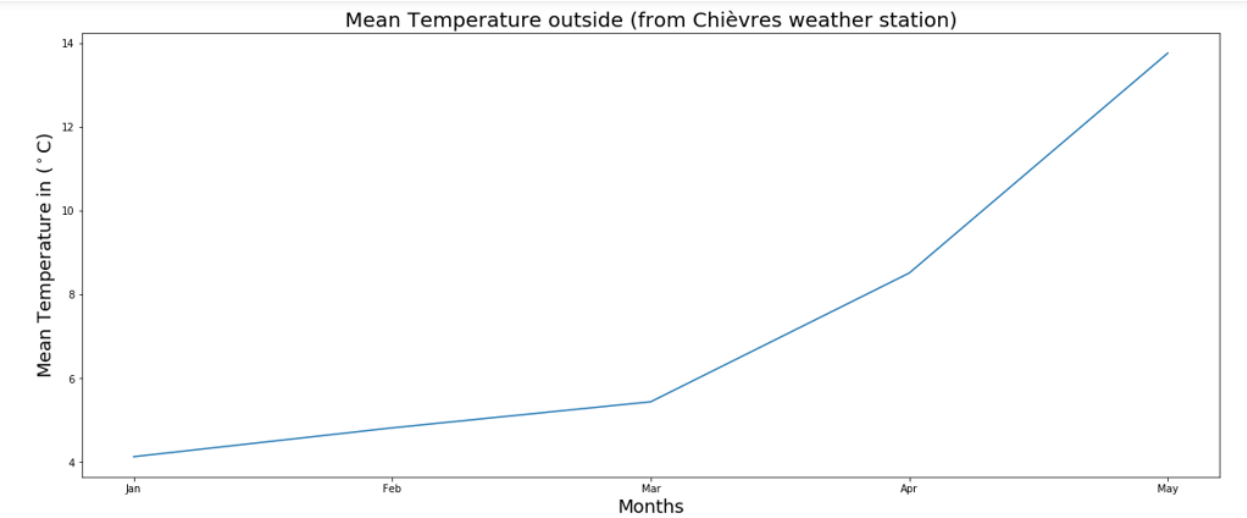
From the above graph we can say that on an average the humidity % Outside the building droped drastically from Jan (around 90 %) to May (around 20%) as compared to humidity % of other rooms.



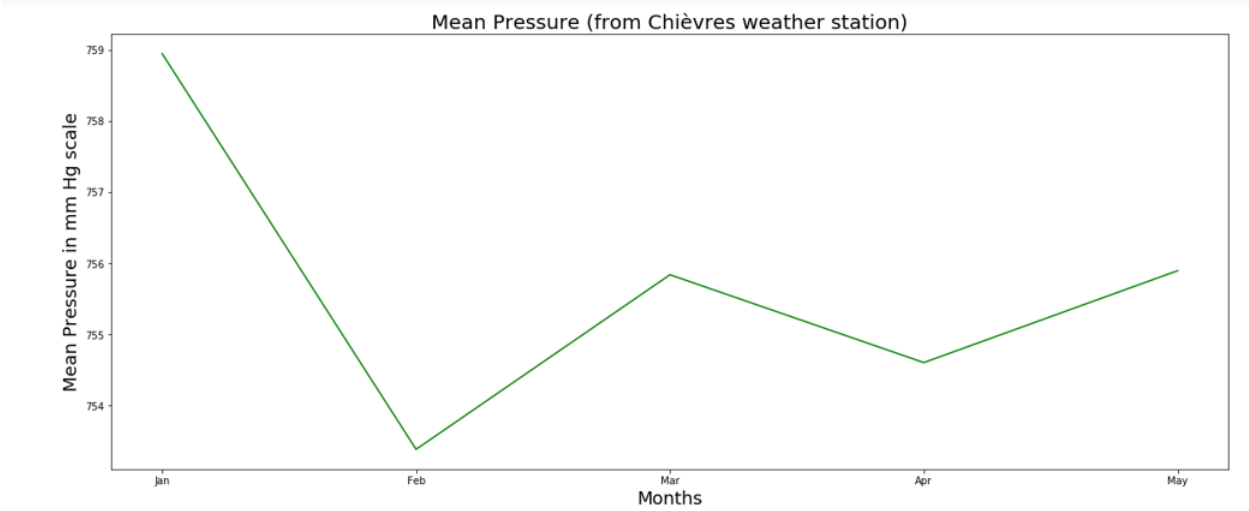
From the above two graphs, we can understand the properties of humidity in different rooms.

RH_1: Humidity in kitchen area, in %
RH_2: Humidity in living room area, in %
RH_3: Humidity in laundry room area, in %
RH_4: Humidity in office room, in %
RH_5: Humidity in bathroom, in %
RH_6: Humidity outside the building (north side), in %
RH_7: Humidity in ironing room, in %
RH_8: Humidity in teenager room 2, in %
RH_9: Humidity in parent's room, in %

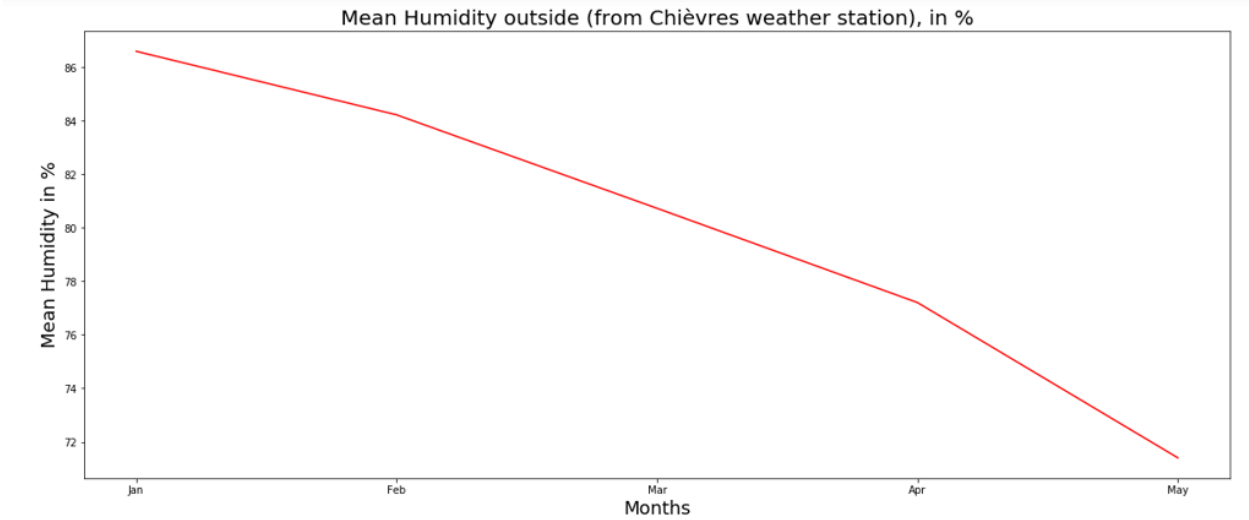
There are 6 other features present in dataset which are: Outside temperature, outside Pressure, outside Humidity, Windspeed, visibility and total dew point. Let's plot each of them in individual graphs to understand their characteristics.



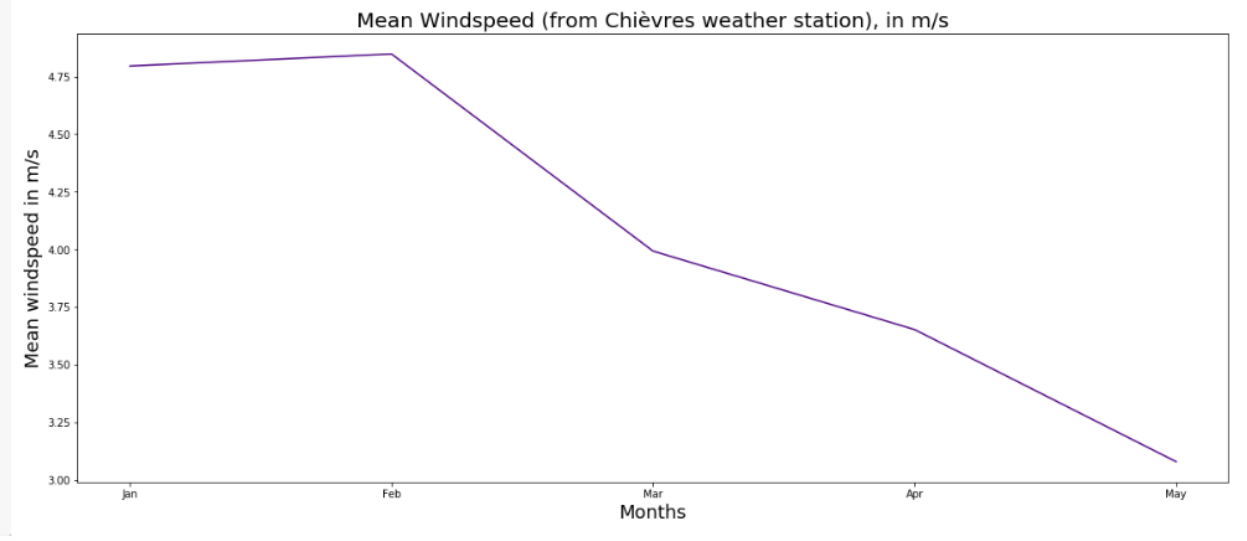
We can see that on an average outside temperature increased gradually from Jan(around 4) to Mar(around 5.7) but increased drastically from March(around 5.7) to May(around 14)



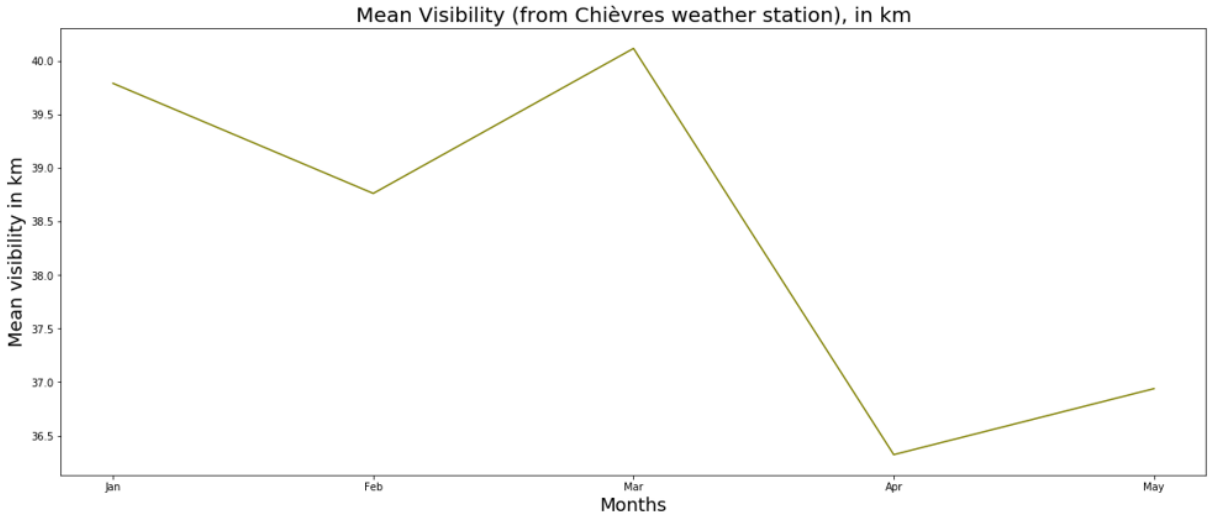
On an average Pressure was lowest in the Feb where as it was highest in Jan



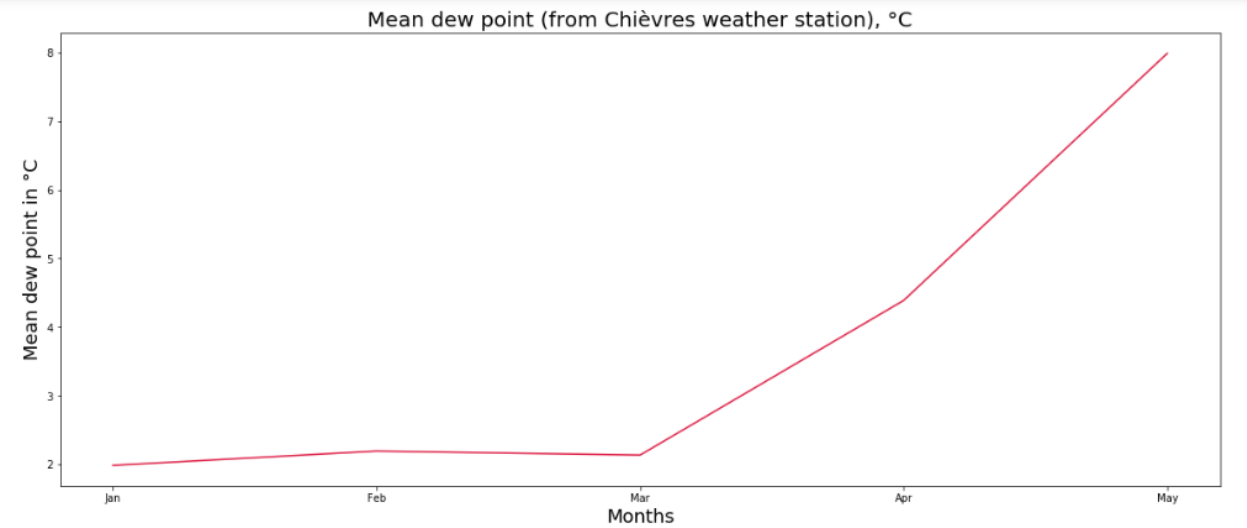
On an average the humidity % kept on decreasing from Jan(around 87%) to May(around 71%)



On an average there was a slight increase in windspeed from Jan(around 4.75 m/s) to Feb(around 4.85 m/s) but then it started decreasing from Feb to May (around 3.1 m/s)

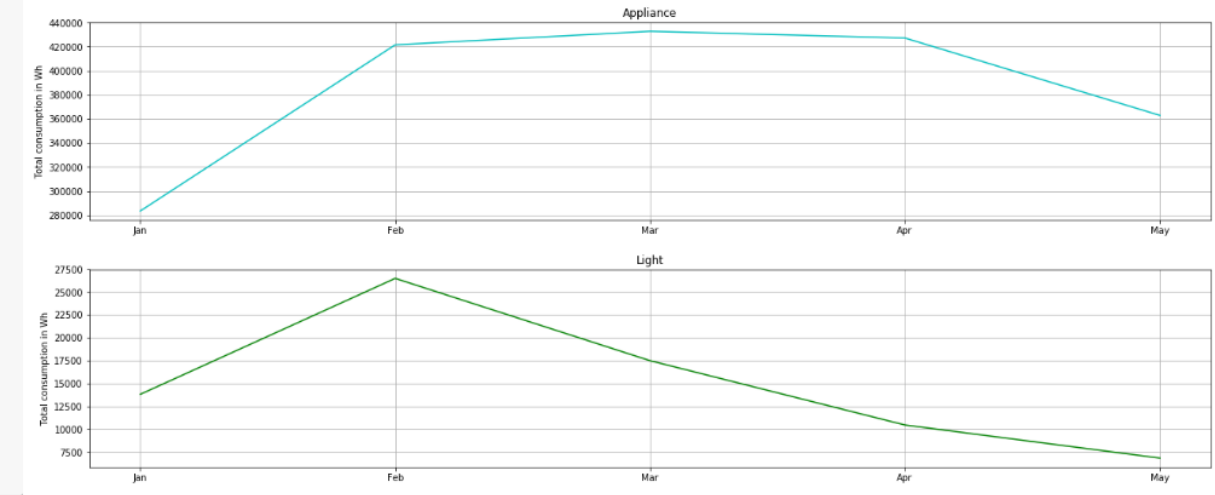


On an average Visibility was highest in the month of Mar(around 39.75km) and the lowest was in the month of Apr(around 36.3km)



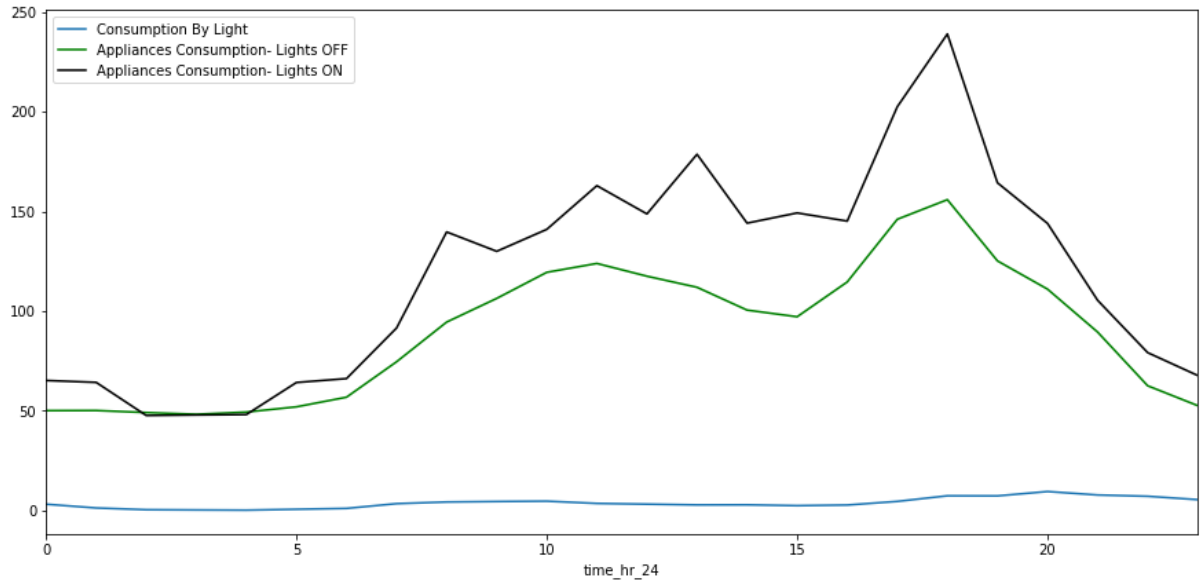
The dew point temperature on an average was around 2 degree celsius from Jan to Mar where as it increased to around 8 degree celsius in month of May

We will now try to find the total consumption of energy by 'lights' and all the appliances at home.

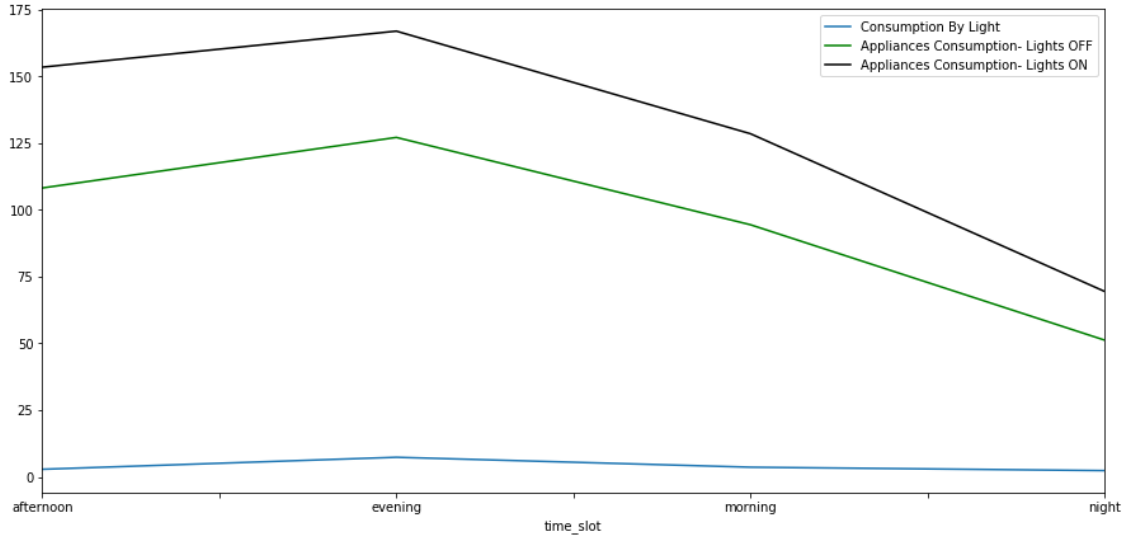


From the above graph the lowest enery cosumption was in January where as it was highest in March. Energy consumed by lights was highest in Feb where as it was lowest in May

Let's plot the graph for the appliances energy utilization with respect to the light usage.

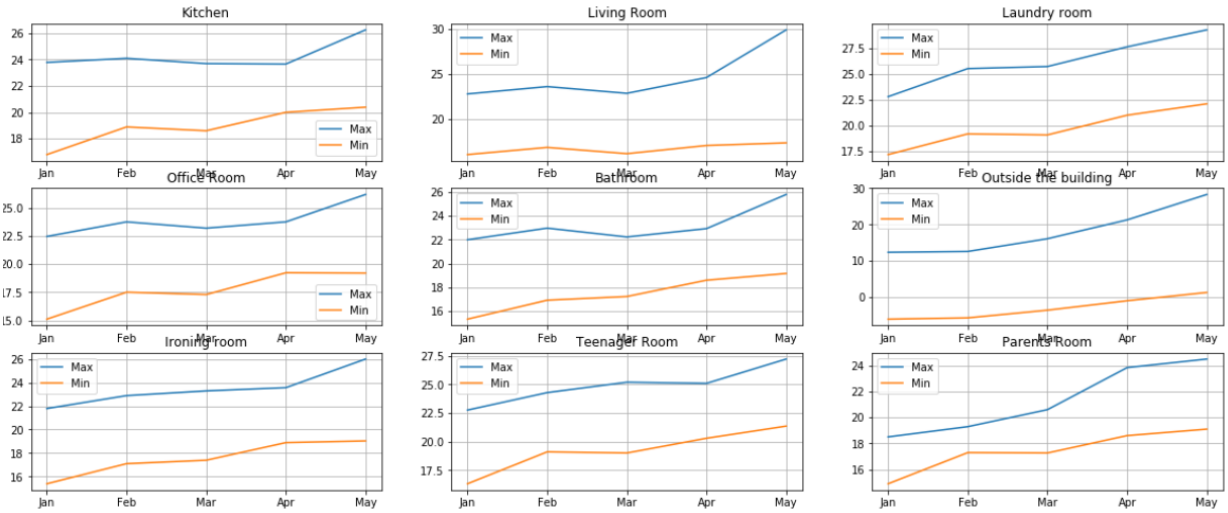


We can see that average energy utilization by Appliances is low when ligths are OFF when compared with that of when ligths are ON

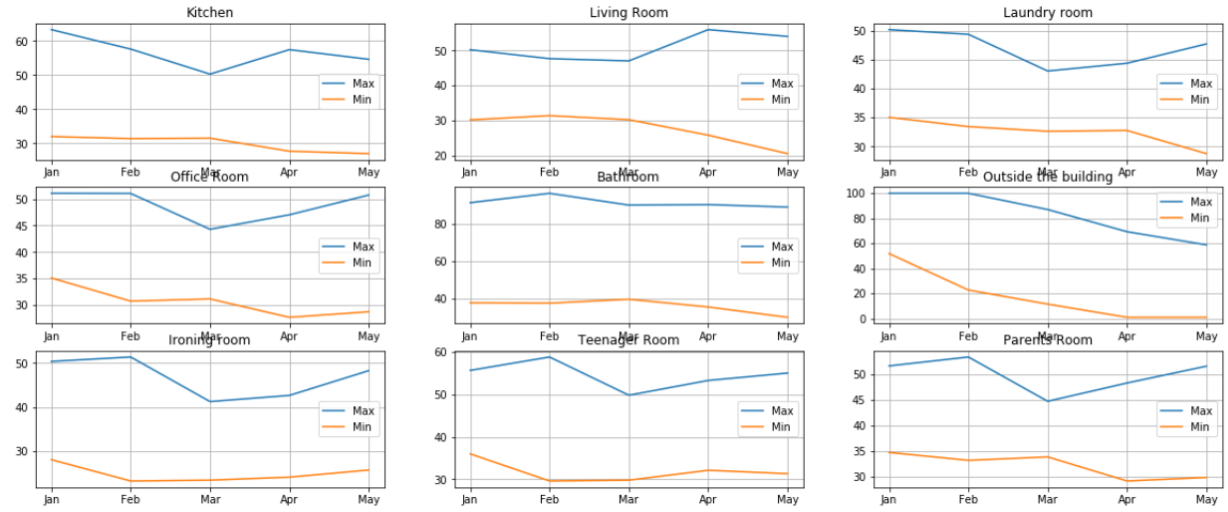


We can see that energy consumed by appliances and lights is minimum during night hours and highest during evening hours. Here morning = 6 - 11:59, afternoon = 12 - 16:59, evening = 17 - 21:59, night = 22 - 5:59.

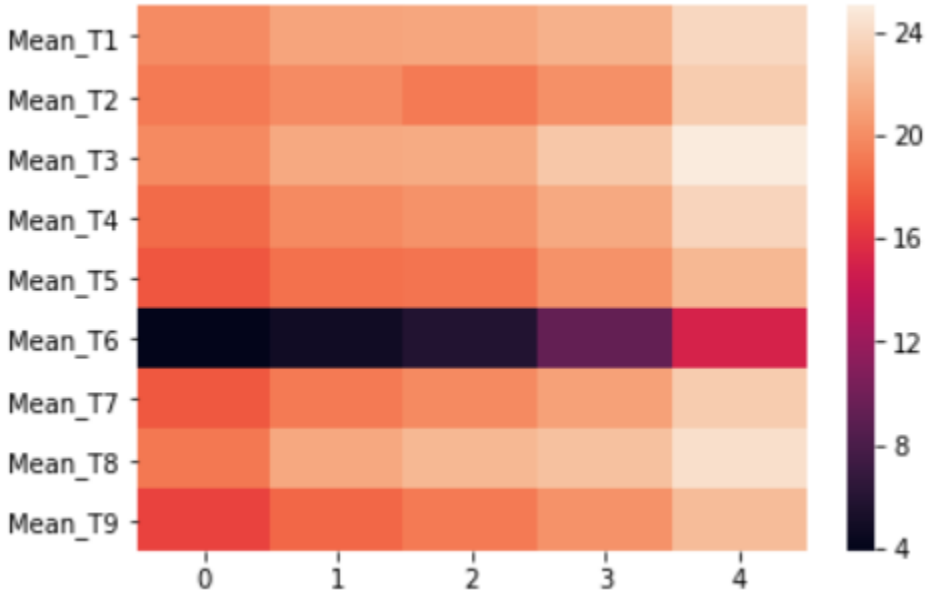
The temperature and humidity is varying a lot over the given period. We should also analyze deviation between minimum and maximum values of these over the period



Above graph represents temperature and below represents humidity.



Let's try to understand the pattern of change in mean temperature through a heat map.



When we see the above graphs of data, we can say that on an average temperature in T6 (i.e temperature outside the building) was low around 4-8 degree Celsius from Jan to April and it rose to around 14 degree Celsius in May.

Part 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.

```
In [5]: f.shape
Out[5]: (19735, 29)
```

There are 19735 rows and 29 columns in the dataset.

Creating Additional features and dummy columns to get more insights.

Creating additional features and dummy columns in the data so as to get more insights.

```
In [30]: 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
df['time_min']=df['date'].dt.minute
df['week_day_type']=df['day_of_week'].map(week_day_type)
df['time_slot']=df['time_hr_24'].map(time_slot)
df.drop(['date'],axis=1,inplace=True)
df=pd.get_dummies(df,prefix=['DOW','TS','WDT'],columns=['day_of_week','time_slot','week_day_type'])
print("rows,columns :",df.shape)
```

```
rows,columns : (19735, 46)
```

Let's check what the dataset contains.

```
In [31]: df.head()
```

Out[31]:

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	DOW_Sunday	DOW_Thursday	DOW_Tuesday	DOW_Wedn
0	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	...	0	0	0	
1	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	...	0	0	0	
2	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	45.890000	...	0	0	0	
3	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	45.723333	...	0	0	0	
4	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	45.530000	...	0	0	0	

5 rows × 46 columns

It is important for us to analyze whether there are null values in the dataset and how are values distributed.

```
In [33]: df.isnull().sum()

Out[33]: Appliances       0
         lights           0
         T1               0
         RH_1             0
         T2               0
         RH_2             0
         T3               0
         RH_3             0
         T4               0
         RH_4             0
         T5               0
         RH_5             0
         T6               0
         RH_6             0
         T7               0
         RH_7             0
         T8               0
         RH_8             0
         T9               0
         RH_9             0
         T_out            0
         Press_mm_hg      0
         RH_out           0
         Windspeed        0
         Visibility       0
         Tdewpoint        0
         rv1              0
         rv2              0
         year             0
         month            0
         day              0
         time_hr_24       0
         time_min         0
         DOW_Friday       0
         DOW_Monday       0
         DOW_Saturday     0
         DOW_Sunday       0
         DOW_Thursday     0
         DOW_Tuesday      0
         DOW_Wednesday    0
         TS_afternoon     0
         TS_evening       0
         TS_morning       0
         TS_night         0
         WDT_weekdays     0
         WDT_weekends     0
         dtype: int64
```

As we can see, we don't have any null values present in any row of our dataset.

We also have to understand the nature of these values before performing any mathematical calculations.

```
In [34]: df.dtypes

Out[34]: 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
         Visibility       float64
         Tdewpoint        float64
         rv1              float64
         rv2              float64
         year               int64
         month              int64
         day                int64
         time_hr_24         int64
         time_min           int64
         DOW_Friday         uint8
         DOW_Monday         uint8
         DOW_Saturday       uint8
         DOW_Sunday         uint8
         DOW_Thursday       uint8
         DOW_Tuesday        uint8
         DOW_Wednesday      uint8
         TS_afternoon       uint8
         TS_evening         uint8
         TS_morning         uint8
         TS_night           uint8
         WDT_weekdays       uint8
         WDT_weekends       uint8
         dtype: object
```

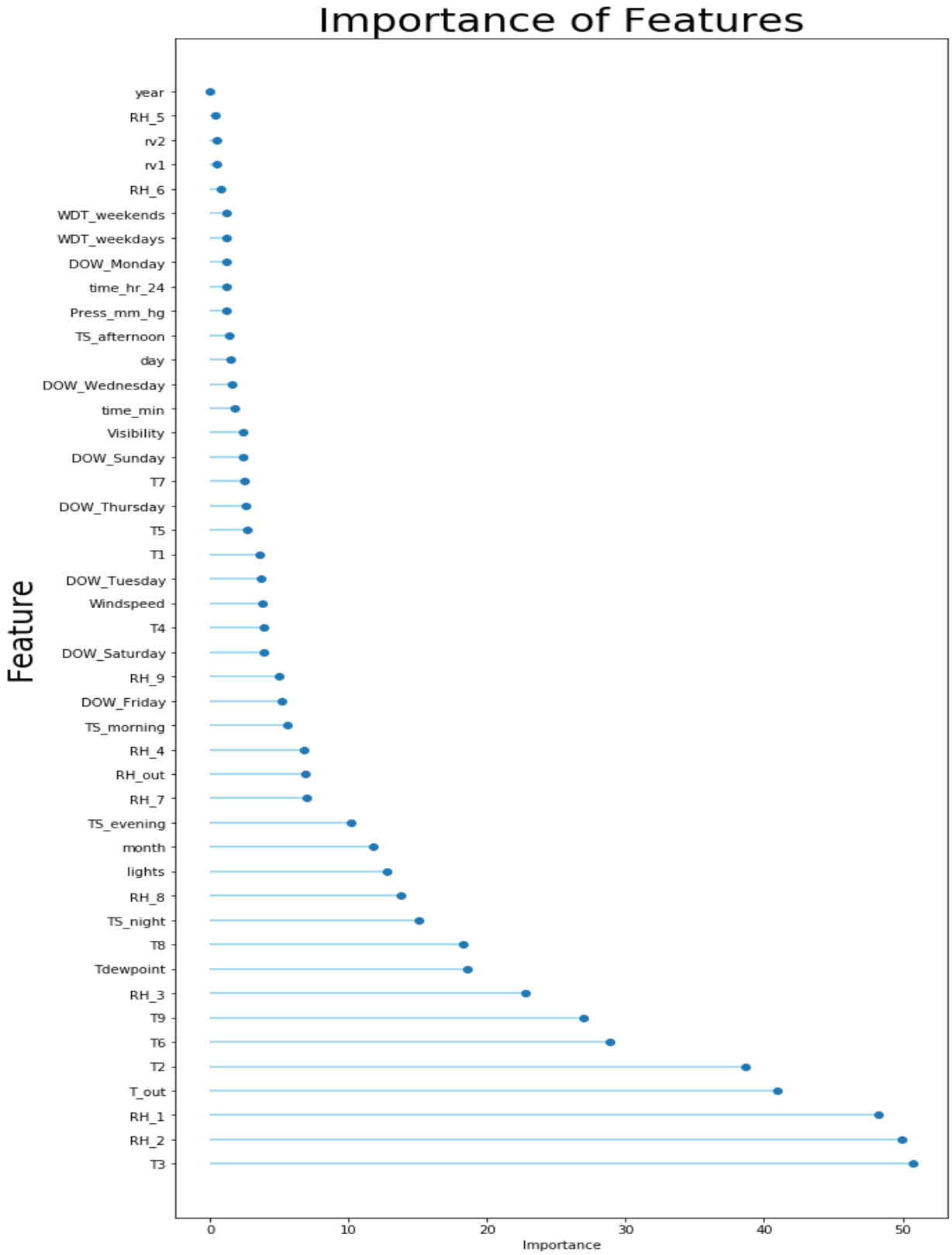
We can see that all important data are in 'float' format. We can also observe that few columns have Boolean data. This is the result of 'One-hot encoding'.

But it is also necessary for us to make scaling of the data as some values like pressure in millimeter is in hundred's and all other values are in ten's. We have to make sure that just because some features have higher numerical value than other, it should not prevail in our calculations and predictions.

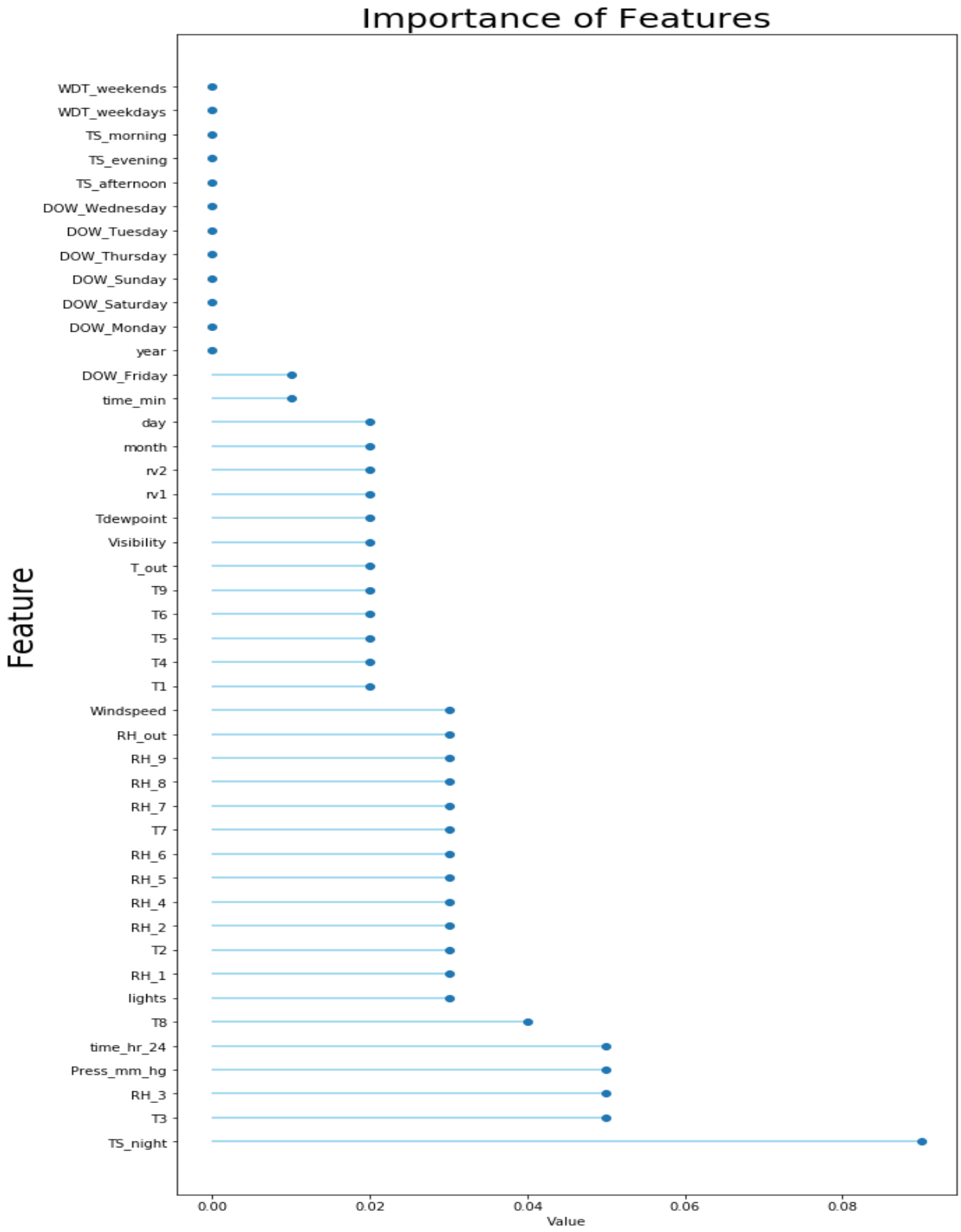
Spliting and scaling data using Standerscaler

In [98]:
```
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']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Importance of features based on coefficients generated by linear model



Importance of features obtained through Random Forest model



Part 4: Prediction Algorithm

Algorithms are the single most important toolbox for anyone who must solve problems by writing computer programs. Algorithms are used not only by computer scientists and computer engineers, but also by many in other engineering and science disciplines.

Models build on data whose Outliers are removed

```
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
```

From the above image, we can see the important libraries which have to be included in order to perform the selected algorithms.

We also should divide the data for training and testing.

Spliting data and normalization

```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
scaler = StandardScaler()
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']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Linear Regression

In statistics, linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

Linear Regression Model

```
lm=linear_model.LinearRegression()
lm.fit(x_train_sc,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Checking R2, MAE, RMSE and MAPE values for linear model on training and testing data.

Linear Regression on Training dataset

```
y_train_pred=lm.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   : 0.246677578831
MAE  : 34.8186933258
RMSE : 58.126235611
MAPE : 45.8231905339
```

Linear Regression on Testing dataset

```
y_test_pred=lm.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   : 0.229768149054
MAE  : 34.5174622076
RMSE : 56.6320945638
MAPE : 47.0933502516
```

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Random Forest Model

```
rf=RandomForestRegressor()
rf.fit(x_train_sc, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           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=None, verbose=0, warm_start=False)
```

Checking R2, MAE, RMSE and MAPE values for linear model on training and testing data.

Random Forest on Training dataset

```
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   : 0.902155365581
MAE  : 10.07110666
RMSE : 20.9483710913
MAPE : 11.8698136903
```

Random Forest on Testing dataset

```
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   : 0.465035848574
MAE  : 25.0737928349
RMSE : 47.1969819082
MAPE : 31.3659338504
```

Building Model by dropping least significant features

Initially we were using entire dataset but now we are trying to optimize the data to get better results by dropping less significant features of the dataset .

We now expect to get better results in our previously performed algorithms.

Linear Regression

Let's build the linear regression model first.

Building Linear Regression Model by drop least singnificant features.(refer Part3:feature engineering and rough work's folder)

```
drop_col_list=['year','rv1','rv2','Press_mm_hg','DOW_Wednesday','T1','RH_6','T5','WDT_weekends','RH_5','time_min'
#              ,'Visibility','DOW_Monday','WDT_weekdays','TS_afternoon','DOW_Sunday','day','T_out'
              ]
x_train=df_train.iloc[:,1:]
y_train=df_train['Appliances']
x_train.drop(drop_col_list,axis=1,inplace=True)
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Appliances']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Linear Regression Model

```
lm=linear_model.LinearRegression()
lm.fit(x_train_sc,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Now, will check its performance on testing and training dataset.

Linear Regression on training dataset

```
y_train_pred=lm.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))
#Base Model's evaluating parameters for training dataset
# R2   : 0.246677578831
# MAE  : 34.8186933258
# RMSE : 58.126235611
# MAPE : 45.8231905339
```

```
R2   : 0.246387010104
MAE  : 34.8147378947
RMSE : 58.1374446481
MAPE : 45.8116274684
```

Linear Regression on Testing dataset

```
y_test_pred=lm.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))
#Base Model's evaluating parameters for testing dataset
# R2   : 0.229768149054
# MAE  : 34.5174622076
# RMSE : 56.6320945638
# MAPE : 47.0933502516
```

```
R2   : 0.230087785103
MAE  : 34.4835361606
RMSE : 56.6203425587
MAPE : 47.0720474117
```

There is very slight improvement in linear model after filtering out the least significant features.

Random Forest

Building Random Forest Models Based on Features selected by thier importance (refer Part3:feature engineering and rough work's folder)

```
drop_col_list=['year','DOW_Friday','DOW_Monday','DOW_Saturday','DOW_Sunday','DOW_Thursday','DOW_Tuesday','DOW_Wednesday','TS_afte
               'WDT_weekdays','WDT_weekends','TS_morning','month','time_min' ,'TS_evening','day','rv1','rv2','Visibility','Windspe
                'T9','lights','T7','Tdewpoint']
x_train=df_train.iloc[:,1:]
y_train=df_train['Appliances']
x_train.drop(drop_col_list,axis=1,inplace=True)
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Appliances']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Random Forest Model After using tuned hyperparameters which we got using random grid search (rough work's folder)

```
rf=RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=35,
           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=258, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
rf.fit(x_train_sc, y_train)
```

```
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=35,
           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=258, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Note: The rough work folder is under Part-4.

Now, will check its performance on testing and training dataset.

Random Forest on Training dataset

```
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))
#Base Model's evaluating parameters for training dataset
# R2   : 0.902155365581
# MAE  : 10.07110666
# RMSE : 20.9483710913
# MAPE : 11.8698136903
```

```
R2   : 0.999997098311
MAE  : 0.0374766366221
RMSE : 0.11407937878
MAPE : 0.0670381095246
```

Random Forest on Testing dataset

```
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))
#Base Model's evaluating parameters for testing dataset
# R2   : 0.465035848574
# MAE  : 25.0737928349
# RMSE : 47.1969819082
# MAPE : 31.3659338504
```

```
R2   : 0.562746622814
MAE  : 21.0750245205
RMSE : 42.6695903922
MAPE : 25.5898964878
```

There is significant improvement but it is comming at the cost of overfitting.

Neural Network

Neural networks (NNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" (i.e. progressively improve performance on) tasks by considering examples, generally without task-specific programming.

Unlike Random Forest and Liner Regression Models, Neutral network selects the best features by itself so we don't exclude any feature of dataset.

Neural Network

```
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']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Neural Network Model

```
mlp = MLPRegressor(hidden_layer_sizes=(155),max_iter=500,alpha=1.00000000e-06,random_state=42)
#65>,105--3 layer  #155 --1 layer #115 --2
mlp.fit(x_train_sc,y_train)
```

```
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=155, 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)
```

Our model is ready. Now, we check its performance on training and testing dataset.

Neural Network on Training Dataset

```
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   : 0.534484272556
MAE  : 26.2079151933
RMSE : 45.6929218734
MAPE : 32.7787131344
```

Neural Network on Testing Dataset

```
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   : 0.394636874831
MAE  : 29.2237202142
RMSE : 50.2064910124
MAPE : 37.5725864014
```

It gives better results as compared to Linear Regression Model but the results not better than Random Forest Model.

Best Model

Out of linear,RandomForest,Neural networks model's Random Forest model gave us the best results,

Base Model:

```
For training dataset:-

    R2   : 0.902155365581
    MAE  : 10.07110666
    RMSE : 20.9483710913
    MAPE : 11.8698136903

For testing dataset:-

    R2   : 0.465035848574
    MAE  : 25.0737928349
    RMSE : 47.1969819082
    MAPE : 31.3659338504
```

Optimized Model:

```
For training dataset:-

    R2   : 0.999997098311
    MAE  : 0.0374766366221
    RMSE : 0.11407937878
    MAPE : 0.0670381095246

For testing dataset:-

    R2   : 0.562746622814
    MAE  : 21.0750245205
    RMSE : 42.6695903922
    MAPE : 25.5898964878
```

Models build on complete dataset that is without removing Outliers.

In statistics, an outlier is an observation point that is distant from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. An outlier can cause serious problems in statistical analyses.

Outliers can occur by chance in any distribution, but they often indicate either measurement error or that the population has a heavy-tailed distribution. In the former case one wishes to discard them or use statistics that are robust to outliers, while in the latter case they indicate that the distribution has high skewness and that one should be very cautious in using tools or intuitions that assume a normal distribution. A frequent cause of outliers is a mixture of two distributions, which may be two distinct sub-populations, or may indicate 'correct trial' versus 'measurement error'; this is modeled by a mixture model.

Before we start performing, normalization of data is necessary.

Spliting data and normalization

```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
scaler = StandardScaler()
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']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Linear Regression

In statistics, linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

Linear Regression Model

```
lm=linear_model.LinearRegression()
lm.fit(x_train_sc,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Linear Regression on Training dataset

```
y_train_pred=lm.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   : 0.20492495517
MAE  : 52.04917031
RMSE : 91.8032672546
MAPE : 59.5590692172
```

Linear Regression on Testing dataset

```
y_test_pred=lm.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   : 0.202656060622
MAE  : 52.1348358691
RMSE : 90.6351639674
MAPE : 61.0346218181
```

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Random Forest Model

```
rf=RandomForestRegressor()
rf.fit(x_train_sc, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           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=None, verbose=0, warm_start=False)
```

Random Forest on Training dataset

```
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   : 0.903342360976
MAE  : 14.3318372665
RMSE : 32.0090045424
MAPE : 14.2999986786
```

Random Forest on Testing dataset

```
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   : 0.455901439969
MAE  : 36.8596520858
RMSE : 74.8708328696
MAPE : 37.6541418787
```

Building Model by dropping least significant features

We now expect to get better results in our previously performed algorithms

Linear Regression

Building Linear Regression Model by drop least singnificant features.(refer Part3:feature engineering and rough work's folder)

```
drop_col_list=['year','RH_5','rv2','rv1' ,'RH_6','time_hr_24','WDT_weekends','DOW_Monday','WDT_weekdays','Press_mm_hg','day'
               ,'TS_afternoon','time_min'
#               ,'DOW_Sunday','DOW_Wednesday'
              ]
x_train=df_train.iloc[:,1:]
y_train=df_train['Appliances']
x_train.drop(drop_col_list,axis=1,inplace=True)
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Appliances']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Linear Regression Model

```
lm=linear_model.LinearRegression()
lm.fit(x_train_sc,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Checking its performance on training and testing data.

Linear Regression on training dataset

```
y_train_pred=lm.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))
#Base Model's evaluating parameters for training dataset
# R2   : 0.20492495517
# MAE  : 52.04917031
# RMSE : 91.8032672546
# MAPE : 59.5590692172
```

```
R2   : 0.204335590571
MAE  : 52.0751956124
RMSE : 91.8372864162
MAPE : 59.6210484115
```

Linear Regression on Testing dataset

```
y_test_pred=lm.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))
#Base Model's evaluating parameters for testing dataset
# R2   : 0.202656060622
# MAE  : 52.1348358691
# RMSE : 90.6351639674
# MAPE : 61.0346218181
```

```
R2   : 0.203865241079
MAE  : 52.0925576441
RMSE : 90.5664133033
MAPE : 60.9601606995
```

There is very slight improvement in linear model after dropping least significant features.

Random Forest

Building Random Forest Models Based on Features selected by thier importance (refer Part3:feature engineering and rough work's folder)

```
drop_col_list=['year','DOW_Monday','DOW_Saturday','DOW_Sunday','DOW_Thursday','DOW_Tuesday','DOW_Wednesday','TS_afternoon','TS_mo
               'WDT_weekdays','WDT_weekends','month','time_min','DOW_Friday','TS_evening','day','rv1','rv2','Visibility'
               ,'T9','T7','lights']
x_train=df_train.iloc[:,1:]
y_train=df_train['Appliances']
x_train.drop(drop_col_list,axis=1,inplace=True)
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Appliances']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Random Forest Model After using tuned hyperparameters which we got using random grid search (refer rough work's folder)

```
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)
```

```
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)
```

Note : The rough work folder is under Part-4.

Checking its performance on training and testing dataset.

Random Forest on Training dataset

```
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))
#Base Model's evaluating parameters for training dataset
# R2   : 0.903342360976
# MAE  : 14.3318372665
# RMSE : 32.0090045424
# MAPE : 14.2999986786
```

```
R2   : 0.999360449368
MAE  : 1.36231948386
RMSE : 2.60370359672
MAPE : 1.80088335883
```

Random Forest on Testing dataset

```
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))
#Base Model's evaluating parameters for testing dataset
# R2   : 0.455901439969
# MAE  : 36.8596520858
# RMSE : 74.8708328696
# MAPE : 37.6541418787
```

```
R2   : 0.615763350576
MAE  : 28.9173039919
RMSE : 62.9177624391
MAPE : 28.1695894996
```

There is significant improvement but it is comming at the cost of overfitting.

Neural Network

Unlike Random Forest and Liner Regression Models, Neutral network selects the best features by itself so we don't exclude any feature of dataset.

Neural Network

```
x_train=df_train.iloc[:,1:]
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:]
y_test=df_test['Appliances']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

```
(13814, 45)
```

Neural Network Model

```
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)
```

```
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

```
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   : 0.759200617 87
MAE  : 27.9396749876
RMSE : 50.5221426581
MAPE : 30.2356429216
```

Neural Network on Testing Dataset

```
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   : 0.405482606628
MAE  : 39.4471746019
RMSE : 78.262940013
MAPE : 39.1624511579
```

It gives better results as compared to Linear Regression but not better than Random Forest.

Best Model

Out of linear,RandomForest,Neural networks model's Random Forest model gave us the best results,

Base Model:

```
For training dataset:-

    R2   : 0.903342360976
    MAE  : 14.3318372665
    RMSE : 32.0090045424
    MAPE : 14.2999986786

For testing dataset:-

     R2   : 0.455901439969
     MAE  : 36.8596520858
     RMSE : 74.8708328696
     MAPE : 37.6541418787
```

Optimized Model:

```
For training dataset:-

    R2   : 0.999360449368
    MAE  : 1.36231948386
    RMSE : 2.60370359672
    MAPE : 1.80088335883

For testing dataset:-

    R2   : 0.615763350576
    MAE  : 28.9173039919
    RMSE : 62.9177624391
    MAPE : 28.1695894996
```

Part 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 TPOT 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.

To Perform tpot, we have to install the library in our local computer.

```
C:\Users\nitin>pip install tpot
Collecting tpot
  Downloading TPOT-0.9.2.tar.gz (888kB)
    100% |████████████████████████████████| 890kB 6.8MB/s
Requirement already satisfied: numpy>=1.12.1 in c:\programdata\anaconda3\lib\site-packages (from tpot)
Requirement already satisfied: scipy>=0.19.0 in c:\programdata\anaconda3\lib\site-packages (from tpot)
Requirement already satisfied: scikit-learn>=0.18.1 in c:\programdata\anaconda3\lib\site-packages (from tpot)
Collecting deap>=1.0 (from tpot)
  Downloading deap-1.2.2.tar.gz (936kB)
    100% |████████████████████████████████| 942kB 3.6MB/s
Collecting update_checker>=0.16 (from tpot)
  Downloading update_checker-0.16-py2.py3-none-any.whl
Collecting tqdm>=4.11.2 (from tpot)
  Downloading tqdm-4.19.7-py2.py3-none-any.whl (52kB)
    100% |████████████████████████████████| 61kB 6.1MB/s
Collecting stopit>=1.1.1 (from tpot)
  Downloading stopit-1.1.2.tar.gz
Requirement already satisfied: pandas>=0.20.2 in c:\programdata\anaconda3\lib\site-packages (from tpot)
Requirement already satisfied: requests>=2.3.0 in c:\programdata\anaconda3\lib\site-packages (from update_checker>=0.16->tpot)
Requirement already satisfied: python-dateutil>=2 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.20.2->tpot)
Requirement already satisfied: pytz>=2011k in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.20.2->tpot)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.3.0->update_checker>=0.16->tpot)
Requirement already satisfied: idna<2.7,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.3.0->update_checker>=0.16->tpot)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.3.0->update_checker>=0.16->tpot)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.3.0->update_checker>=0.16->tpot)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2->pandas>=0.20.2->tpot)
Building wheels for collected packages: tpot, deap, stopit
  Running setup.py bdist_wheel for tpot ... done
  Stored in directory: C:\Users\nitin\AppData\Local\pip\Cache\wheels\d2\54\33\7549c05095a6a38d3de610f88f2d075e56617ff887dce6d54e
  Running setup.py bdist_wheel for deap ... done
  Stored in directory: C:\Users\nitin\AppData\Local\pip\Cache\wheels\82\aa\67\2c93e17c84646c86099fda53ee0b3329372dcf94dd8789fd13
  Running setup.py bdist_wheel for stopit ... done
  Stored in directory: C:\Users\nitin\AppData\Local\pip\Cache\wheels\95\fc\6b\0289a3bce1635be994845f61cbaa91a7ac93dfc453229f0442
Successfully built tpot deap stopit
Installing collected packages: deap, update-checker, tqdm, stopit, tpot
Successfully installed deap-1.2.2 stopit-1.1.2 tpot-0.9.2 tqdm-4.19.7 update-checker-0.16
```

Now we have to build the model.

Building the model

```
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)
```

Now we can run the tpot code.

Tpot Code

```
my_tpot = TPOTRegressor(generations=10)
my_tpot.fit(x_train_sc, y_train)
```

```
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap.py:219: ImportWarning: can't resolve package from __spec__ or __package__, fa
lling back on __name__ and __path__
  return f(*args, **kwds)
```

```
Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.

TPOTRegressor(config_dict={'sklearn.linear_model.ElasticNetCV': {'l1_ratio': array([ 0.  ,  0.05,  0.1 ,  0.15,  0.2 ,  0.25,
0.3 ,  0.35,  0.4 ,
        0.45,  0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,
        0.9 ,  0.95,  1.  ]), 'tol': [1e-05, 0.0001, 0.001, 0.01, 0.1]}, 'sklearn.ensemble.ExtraT....45,
        0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,  0.9 ,
        0.95,  1.  ])}}}},
       crossover_rate=0.1, cv=5, 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=100, periodic_checkpoint_folder=None,
       population_size=100, random_state=None, scoring=None, subsample=1.0,
       verbosity=0, warm_start=False)
```

Now, when we got output, we can find the accuracy of training and testing data.

Accuracy for training data.

```
print(my_tpot.score(x_train_sc, y_train))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\skl
encountered in less
  return (self.pvalues_ < self.alpha / len(sel
```

```
-46.8504652813
```

Accuracy for testing data.

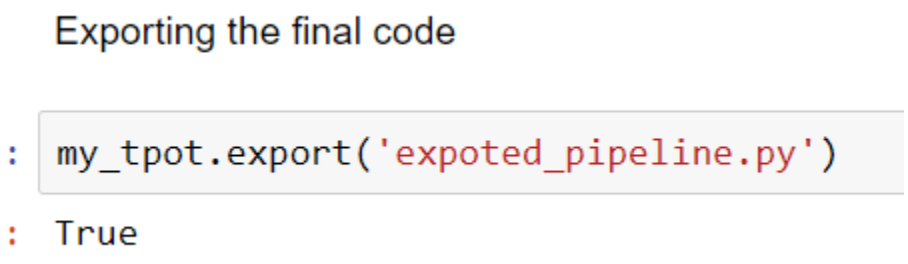
```
print(my_tpot.score(x_test_sc, y_test))
```

```
C:\ProgramData\Anaconda3\lib\site-packages
encountered in less
  return (self.pvalues_ < self.alpha / len
```

```
-1820.58627833
```

Exporting the final code

We can now export the final pipeline code which we received from tpot.



TSFresh

Tsfresh is a python package. It automatically calculates many 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 should start by creating a model for testing.

Then we run the following set of codes to derive the output.

Performing tsfresh

```
: from tsfresh.utilities.dataframe_functions import roll_time_series
```

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

```
:                    time  value              id  kind
```

It will have details of time series.

Now we should extract relevant data from the model.

Extracting relevant features.

```
: from tsfresh import select_features
  from tsfresh.utilities.dataframe_functions import impute

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

When our final model is ready, we can extract the necessary output by running the following code.

```
for model in models:
    # get model name
    m = str(model)
    tmp['Model'] = m[:m.index('(')]
    # fit model on training dataset
    model.fit(Xtrn, Ytrn)
    # predict consumption
    predictions = model.predict(Xtest)
    #Evaluation for Testing set
    #R2 score
    tmp['R2_Test'] = r2_score(Ytest,predictions)
    #Mean Absolute Error(MAE)
    tmp['MAE_Test']= mean_absolute_error(Ytest,predictions)
    #Mean Squared Error(MSE)
    tmp['MSE_Test']= mean_squared_error(Ytest,predictions)
    #Root Mean Squared Error (RMSE)
    tmp['RMSE_Test'] = np.sqrt(mean_squared_error(Ytest,predictions))
    #Evaluation for Training test
    predictions_trn = model.predict(Xtrn)
    #R2_Score
    tmp['R2_Train'] = r2_score(Ytrn,predictions_trn)
    #Mean Absolute Error(MAE)
    tmp['MAE_Train']= mean_absolute_error(Ytrn,predictions_trn)
    #Mean Squared Error(MSE)
    tmp['MSE_Train']= mean_squared_error(Ytrn,predictions_trn)
    #Root Mean Squared Error (RMSE)
    tmp['RMSE_Train'] = np.sqrt(mean_squared_error(Ytrn,predictions_trn))
```

And the accuracy is

Model	MAE_Test	MAE_Train	MSE_Test	MSE_Train
LinearRegression	2.464584e+01	2.303663e+01	3.519243e+03	1.979770e+03
RandomForestRegressor	2.449775e+01	9.890073e+00	2.298906e+03	4.064006e+02
MLPRegressor	5.106404e+08	4.727550e+08	6.369965e+20	3.810890e+20

Model	R2_Test	R2_Train	RMSE_Test	RMSE_Train
LinearRegression	1.629873e-01	5.569258e-01	5.932321e+01	4.449461e+01
RandomForestRegressor	4.532309e-01	9.090472e-01	4.794691e+01	2.015938e+01
MLPRegressor	-1.515025e+17	-8.528804e+16	2.523879e+10	1.952150e+10

Boruta

Boruta is an all relevant feature selection wrapper algorithm, capable of working with any classification method that output variable importance measure (VIM); by default, Boruta uses Random Forest. The method performs a top-down search for relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies, and progressively eliminating irrelevant features to stabilize that test.

First, we should build a model.

Running the boruta.

```
X = x_train_sc
y = y_train
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:      100 / 100
Confirmed:      20
Tentative:      0
Rejected:       24
```

```
C:\Users\Akash\Anaconda3\lib\site-packages\boruta\boruta_py.py:418: RuntimeWarning: invalid value encountered in greater
  hits = np.where(cur_imp[0] > imp_sha_max)[0]
```

```
BorutaPy(alpha=0.05,
     estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           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=25, n_jobs=-1,
           oob_score=False,
           random_state=<mtrand.RandomState object at 0x000001645E8D60D8>,
           verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x000001645E8D60D8>,
     two_step=True, verbose=2)
```

Best features as per Boruta

```
x_train.columns[feat_selector.support_]
```

```
Index(['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',
       'RH_out', 'time_hr_24', 'TS_night'],
      dtype='object')
```

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

```
column_list = x_train.columns[feat_selector.support_]
x_train=df_train.iloc[:,1:]
x_train= x_train[column_list]
print("Training :",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("Testing :",x_test.shape)
y_test=df_test['Appliances']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

```
Training : (11982, 20)
Testing : (5136, 20)
```

Using Random Forest Model that we got from Boruta

```
rf=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           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=25, n_jobs=-1,
           oob_score=False,
           random_state=42,
           verbose=0, warm_start=False)
rf.fit(x_train_sc, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           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=25, n_jobs=-1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Checking models performance on training and testing data.

Checking the accuracy of data for training set.

```
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   : 0.923882174473
MAE  : 9.54768679196
RMSE : 18.4767225661
MAPE : 11.6103280941
```

Checking the accuracy of data for testing set.

```
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   : 0.488583802533
MAE  : 24.0691245528
RMSE : 46.1465382137
MAPE : 29.3761530006
```

Comparing Part-4 and Part-5 results:

Using the best features extracted from boruta and hyperparameters for random forest model, we got better result than the base model of random forest that we developed in Part4.

Base Model(From Part4):

```
For training dataset:-

    R2   : 0.902155365581
    MAE  : 10.07110666
    RMSE : 20.9483710913
    MAPE : 11.8698136903

For testing dataset:-

    R2   : 0.465035848574
    MAE  : 25.0737928349
    RMSE : 47.1969819082
    MAPE : 31.3659338504
```

Model Based on the features and hyperparameters we got from boruta:-

```
For training dataset:-

   R2   : 0.923882174473
   MAE  : 9.54768679196
   RMSE : 18.4767225661
   MAPE : 11.6103280941

For testing dataset:-

    R2   : 0.488583802533
    MAE  : 24.0691245528
    RMSE : 46.1465382137
    MAPE : 29.3761530006
```

Part 6: Model Validation and Selection

Cross Validation Technique

Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation set), in order to limit problems like overfitting[citation needed], give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

Here, cross validation is split into two parts. Training data and for testing data. And we are splitting our data into 10 equal parts.

Training data.

```
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
```

```
array([ 0.23969467,  0.20955204,  0.27998882,  0.24029214,  0.24212669,
        0.21562057,  0.23401395,  0.24889118,  0.24022395,  0.26513223])
```

```
accuracy_train.mean()
```

```
0.24155362389574181
```

Testing data.

```
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
```

```
array([ 0.15171928,  0.20987585,  0.25722266,  0.20690283,  0.26778345,
        0.25304469,  0.28914837,  0.22071825,  0.18029891,  0.23000607])
```

```
accuracy_test.mean()
```

```
0.22667203553439866
```

Regularization

In mathematics, statistics, and computer science, particularly in the fields of machine learning and inverse problems, regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.

First, we have make a model.

```
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)
```

There are 3 distinct types of test which we have to perform. They are

- Lasso
- Ridge
- ElasticNet

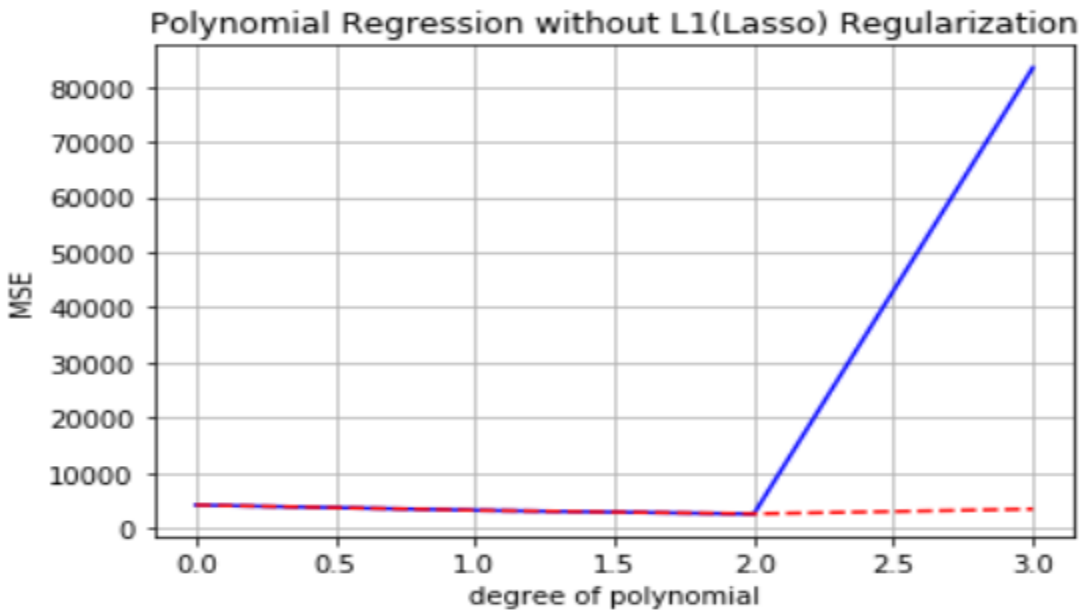
Lasso(L1)

```
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()
```

From this, we will get values of R2, MAE, RMSE & MAPE on training dataset for degree of polynomial 0-3 and we receive the following graph.



Ridge(L2)

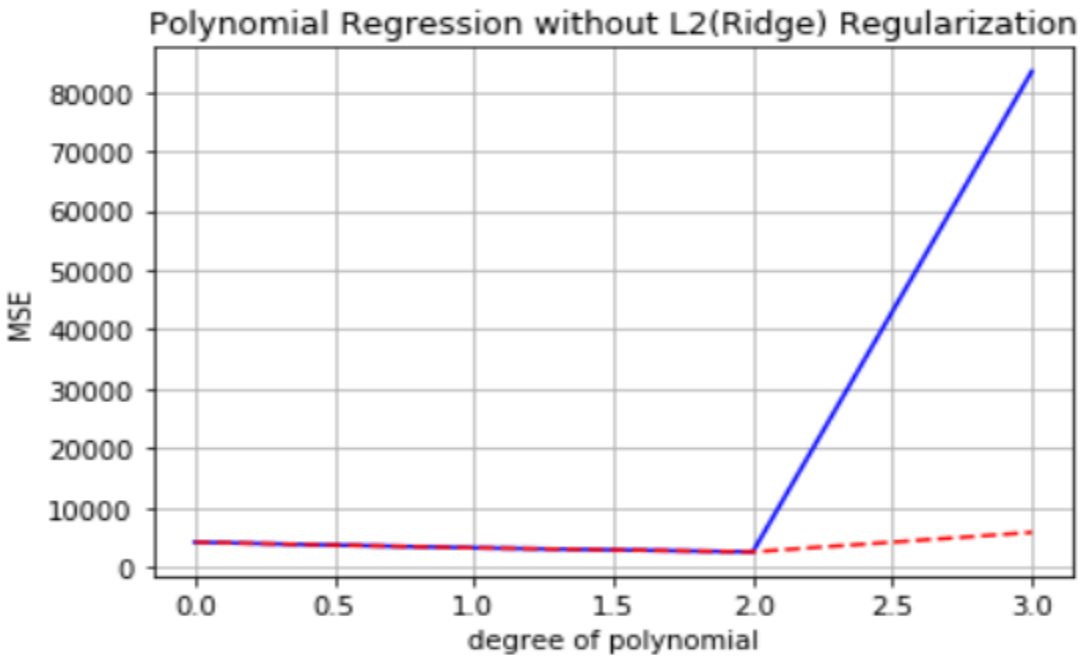
```
for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    X_train_ = model.fit_transform(X_train)
    X_test_ = model.fit_transform(X_test)

    l2reg.fit(X_train_, y_train)
    train_pred_l2 = l2reg.predict(X_train_)
    test_pred_l2 = l2reg.predict(X_test_)
    l2reg_test_mse_list.append(mean_squared_error(y_test, test_pred_l2))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_pred_l2))
    print("MAE  :",mean_absolute_error(y_train,train_pred_l2))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_pred_l2)))
    print("MAPE :",mean_absolute_percentage_error(y_train,train_pred_l2))
    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_pred_l2))
    print("MAE  :",mean_absolute_error(y_test,test_pred_l2))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_pred_l2)))
    print("MAPE :",mean_absolute_percentage_error(y_test,test_pred_l2))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without L2(Ridge) Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, l2reg_test_mse_list, '--r')
plt.show()

# Red dash line is the testing error after L2 regularization
```

From this, we will get values of R2, MAE, RMSE & MAPE on training dataset for degree of polynomial 0-3 and we receive the following graph.



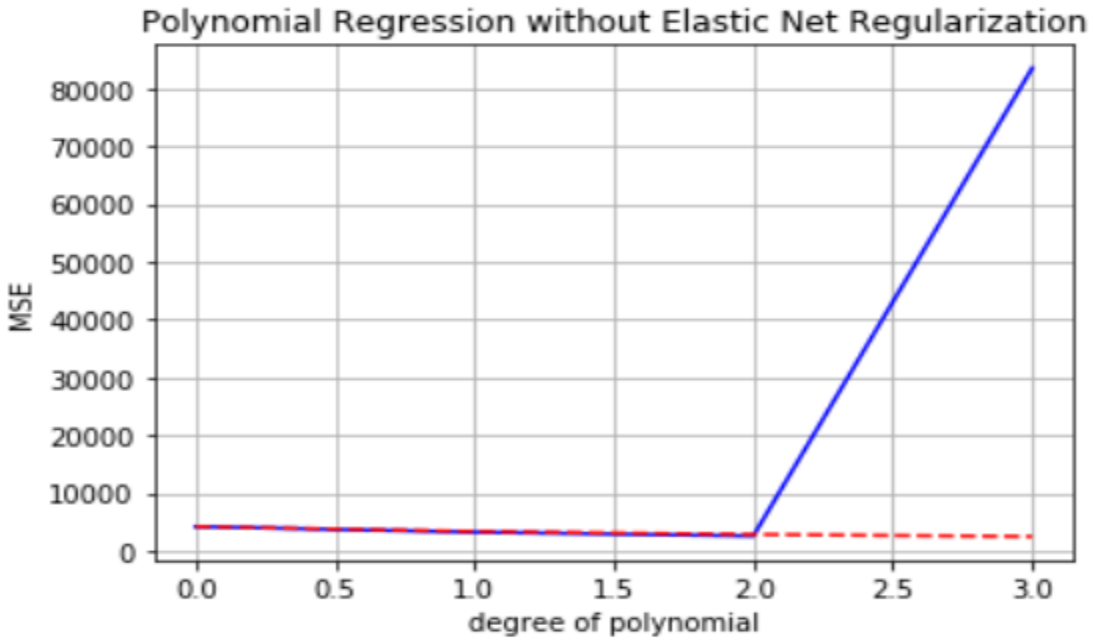
ElasticNet

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

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

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without Elastic Net Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, enreg_test_mse_list, '--r')
plt.show()
# Red dash line is the testing error after Elastic Net regularization
```

From this, we will get values of R2, MAE, RMSE & MAPE on training dataset for degree of polynomial 0-3 and we receive the following graph.



Now we should find the best value of alpha which will help us make the model better.

```
l1reg = Lasso(normalize=True)
for alpha in alphas:
    l1reg.alpha = alpha
    this_scores = model_selection.cross_val_score(l1reg, X_train_, y_train, n_jobs=1, cv=6)
    scores.append(np.mean(this_scores))
    scores_std.append(np.std(this_scores))

max_score = np.max(scores)
max_score_pos = scores.index(max_score)
optimal_alpha = alphas[max_score_pos]
std_err = np.array(scores_std) / np.sqrt(len(X_train_))
print ( 'The calculated optimal alpha is %f ' % optimal_alpha )
print ('The max generalization score of L1 regularized polynomial regression model is %f +- %f' \
      % (max_score, std_err[max_score_pos]))

plt.semilogx(alphas, np.array(scores), '-b')
# plot error lines showing +/- std. errors of the scores
plt.semilogx(alphas, np.array(scores) + std_err, '--b')
plt.semilogx(alphas, np.array(scores) - std_err, '--b')
plt.ylabel('CV score')
plt.xlabel('alpha')
plt.axhline(np.max(scores), linestyle='--', color='r')
plt.axhline(lm_score, linestyle='--', color='g')
plt.show()
```

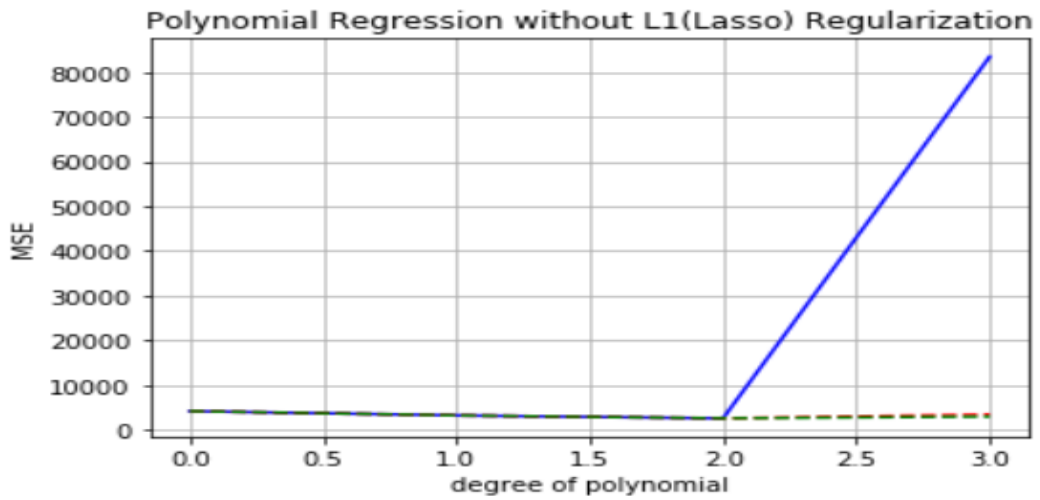
And the scores and the alpha value which we received are as follows.

```
The generalization score of linear regression model is 0.240693
The generalization score of quadratic regression model is 0.379625
```

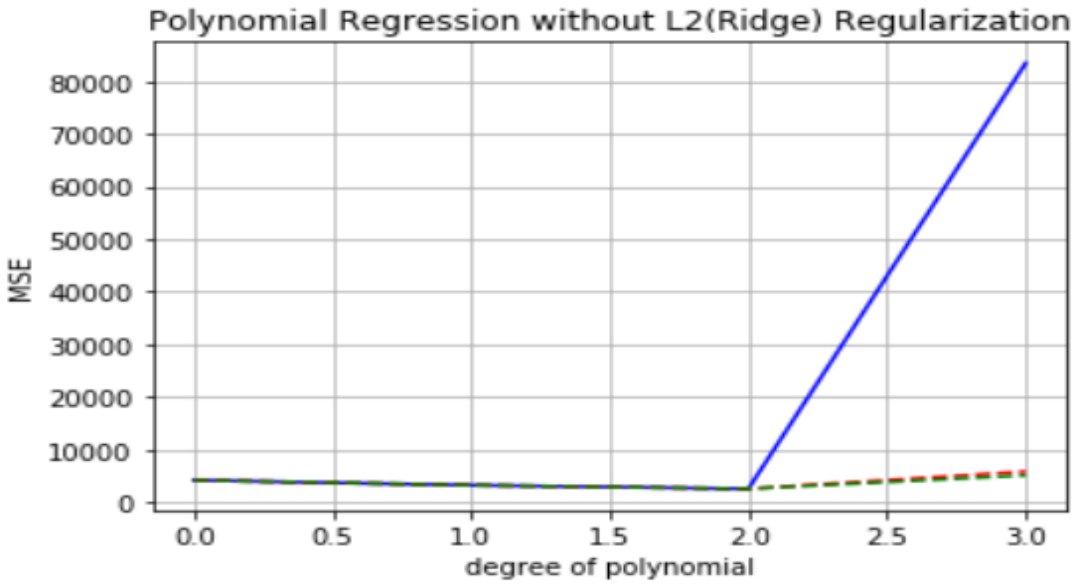
```
The calculated optimal alpha is 0.000050
The max generalization score of L1 regularized polynomial regression model is 0.377953 +- 0.000245
```

Now we should apply this alpha value in our previous code to make the model better.

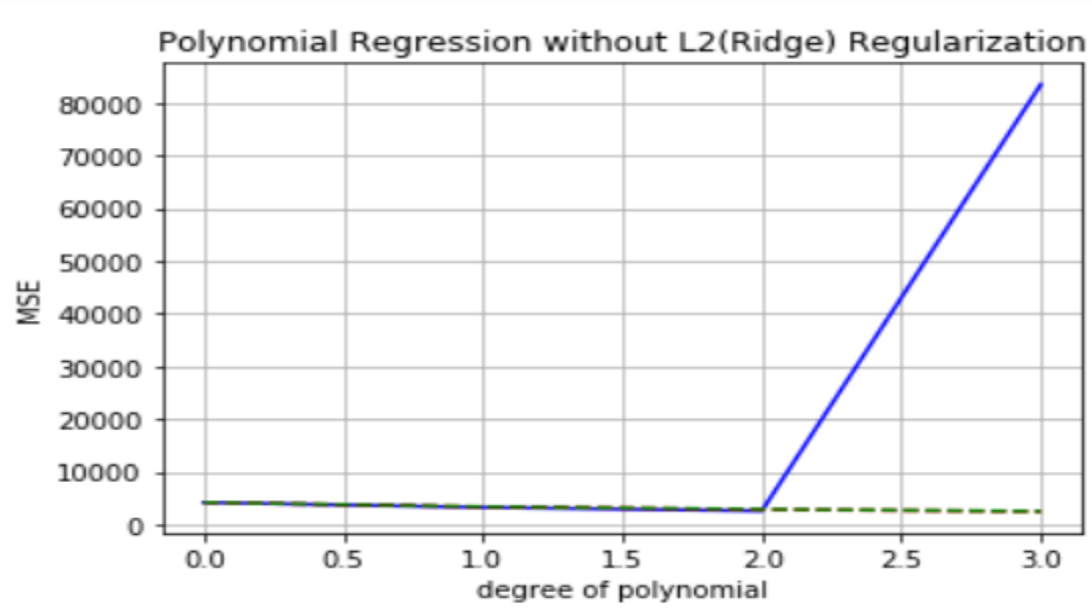
Lasso(L1)



Ridge(L2)



ElasticNet



In the above 3 graphs, dotted red line was previous model (without using optimum alpha value) and dotted green line is using the new alpha value. We can clearly see a change in regularization.

Result:

Parameters	Dataset	Without Regularization	L1	L2	Elastic-Net
R-sq	Training	0.246	0.467	0.478	0.348
	Testing	0.229	0.386	0.391	0.31
MAE	Training	34.818	29.594	29.38	31.377
	Testing	34.407	30.741	30.832	31.443
RMSE	Training	58.126	48.883	48.352	54.059
	Testing	56.624	50.55	50.355	53.575
MAPE	Training	45.823	38.698	38.57	40.259
	Testing	46.867	40.898	41.191	41.398

We can see that, L2 gave the best possible results as compared to Linear regression, L1 and Elastic-Net.

Part 7: Final Pipeline

Using the best features and hyperparameters that we got through Boruta (Part 5) we will create a pipeline.

Spliting data and normalization and selecting the best features that we got from boruta in Part 5

```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
scaler = StandardScaler()
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',
       'RH_out', 'time_hr_24', 'TS_night']
x_train=df_train.iloc[:,1:]
x_train= x_train[column_list]
print("Training :",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("Testing :",x_test.shape)
y_test=df_test['Appliances']  .
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

```
Training : (11982, 20)
Testing : (5136, 20)
```

Using the Random Forset model with hyperparameters that we got from boruta

```
pipe = Pipeline([('gfdh',RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           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=25, n_jobs=1,
           oob_score=False,
           random_state=42,
           verbose=0, warm_start=False))])
pipe.fit(x_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('gfdh', RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           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=25, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False))])
```

Finding R2, MAE , RMSE and MAPE on training data

```
y_train_pred=pipe.predict(x_train)
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   : 0.923875456927
MAE  : 9.54245751573
RMSE : 18.4775378515
MAPE : 11.6079758199
```

Finding R2, MAE , RMSE and MAPE on testing data

```
y_test_pred=pipe.predict(x_test)
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   : 0.498862650462
MAE  : 23.5739119962
RMSE : 45.6804394773
MAPE : 28.804898802
```

Conclusion

1. Best model Random Forest.
2. Boruta gave us the best features and hyperparameters for Random forest, optimizing for model further.
3. Linear Regression gave best results with L2 regularization.
4. Automated the process of feature engineering, feature selection and prediction using sklearn.pipeline and also dockerized it(Please refer to README file in Docker folder to know how to get the image and run it).