Assignment 2 Report

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

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

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

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

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.

```
In [2]: 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
```

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

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

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

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

```
In [6]: f.isnull().sum()

Out[6]: 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
```

```
In [10]: f.dtypes

Out[10]: 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
```

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.

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

```
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)
```

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

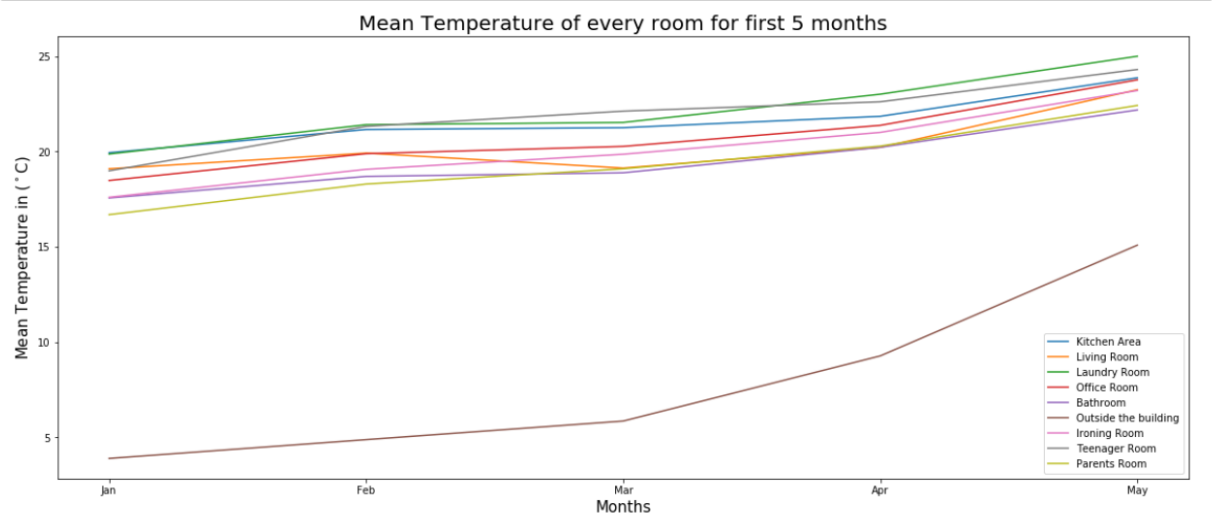
```
week1=range(1,8)
week2=range(8,15)
week3=range(15,22)
week4=range(22,29)
```

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.

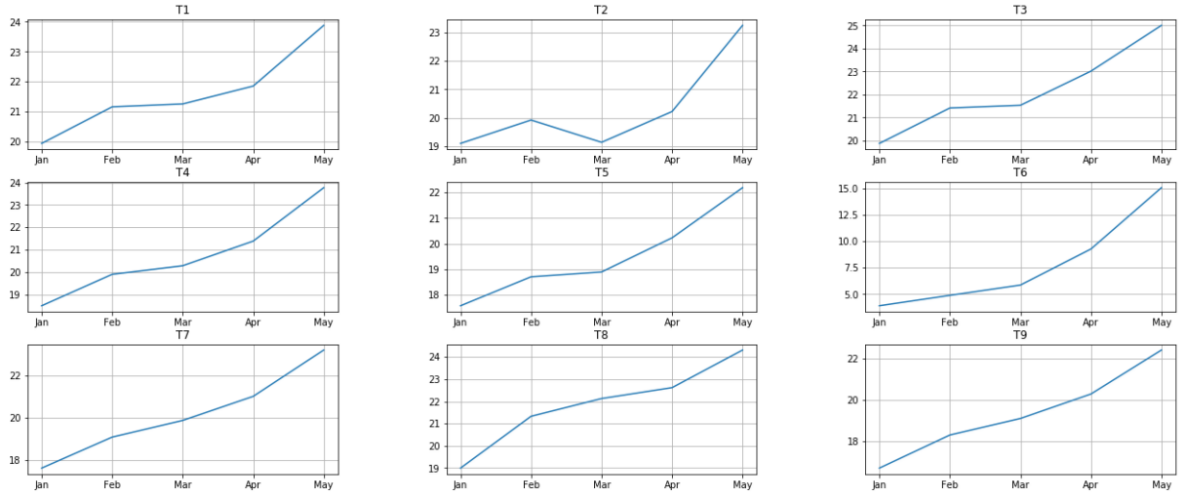
The two-summary matrix are as follows:

- ‘df_summ’: It contains sum and mean of all the relevant values.
- ‘df_ min_max_summ’: It contains minimum and maximum vales of all relevant data.

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



In the above graph, we can see how mean temperature is changing over the course of time for various rooms. Differentiation of rooms are given at the bottom right corner of the graph.



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

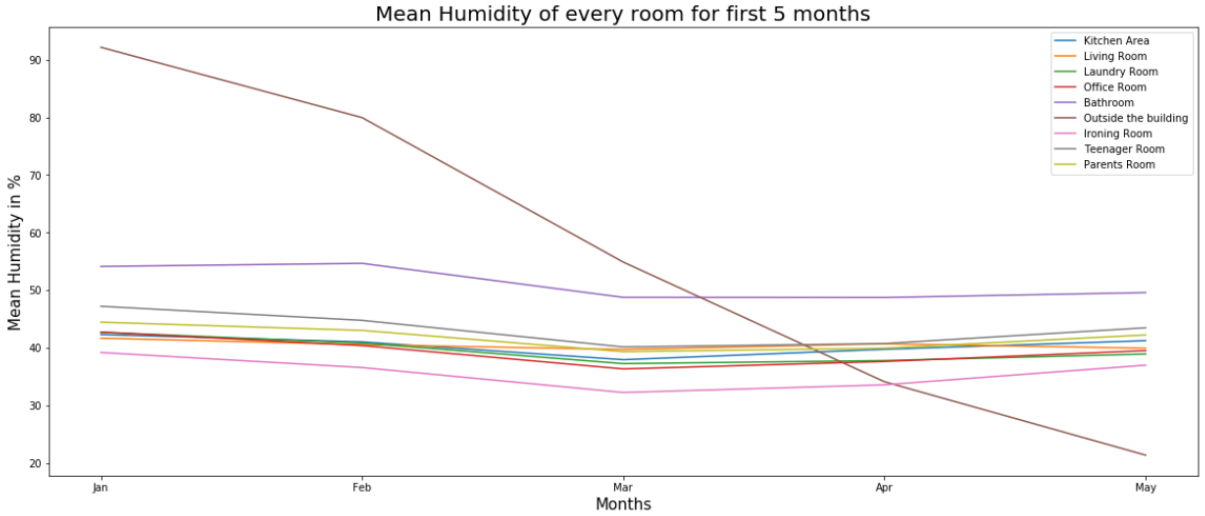
T6: Temperature outside the building (north side), in Celsius

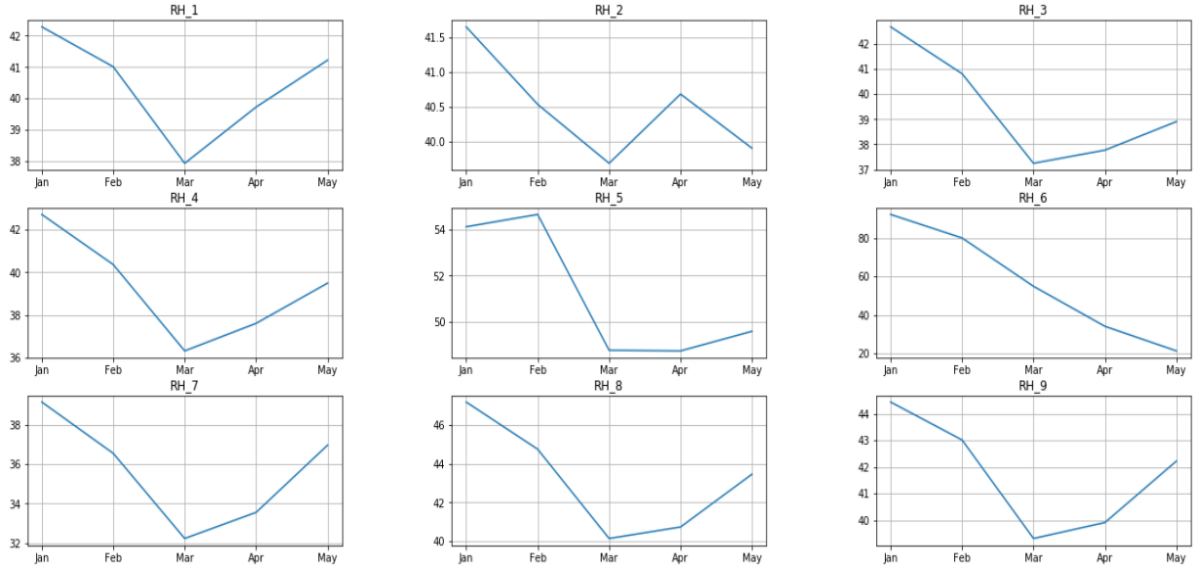
T7: Temperature in ironing room, in Celsius

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.



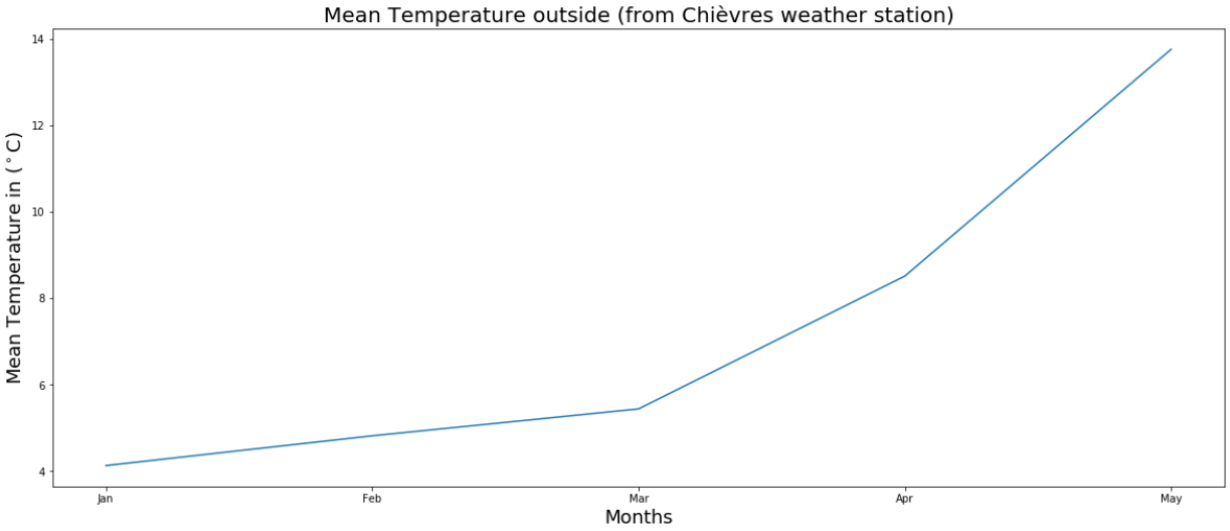


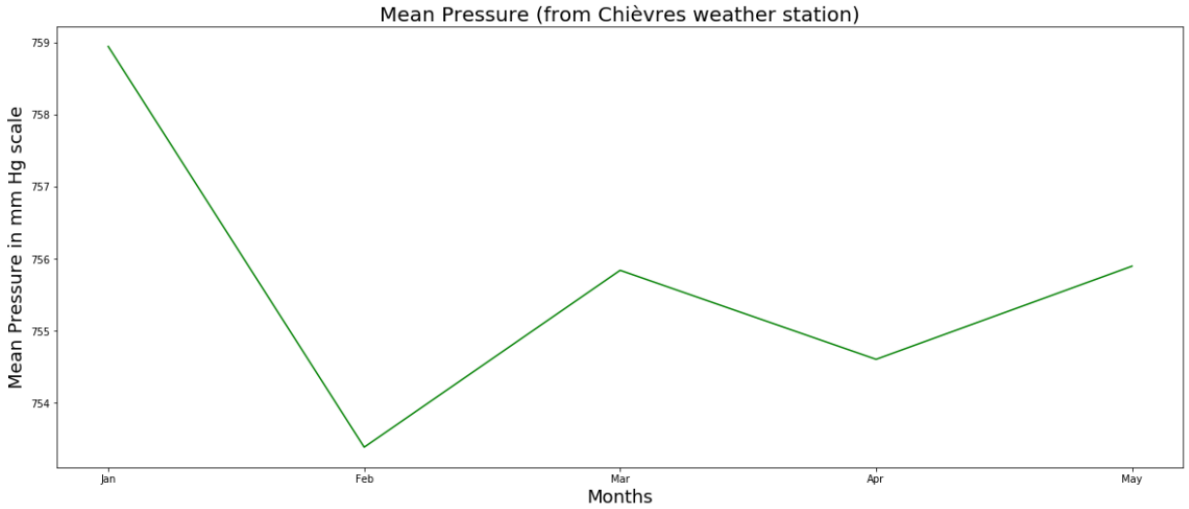
From the above two graphs, we can understand the properties of humidity in different rooms.

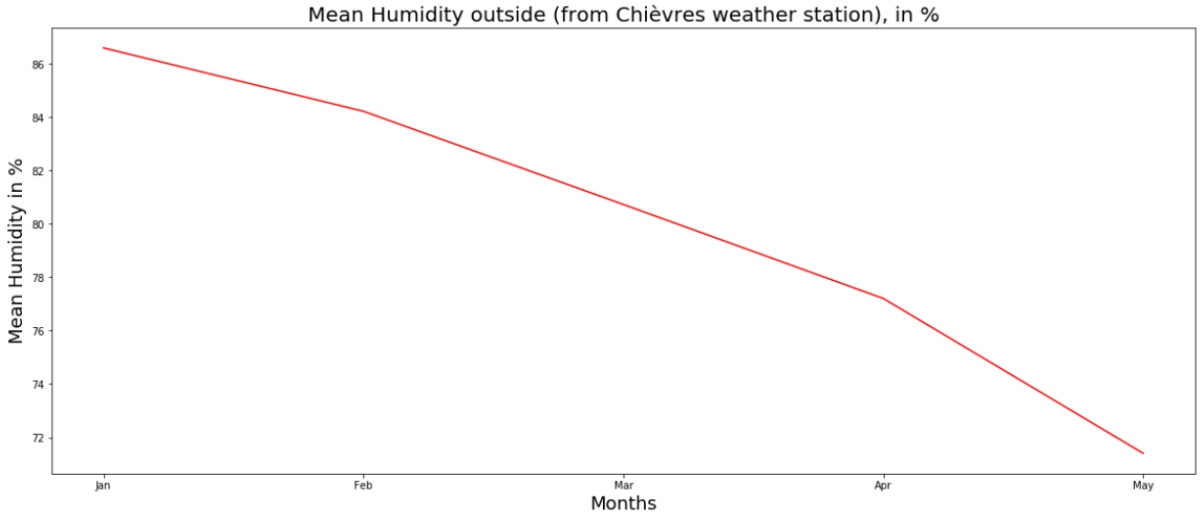
RH_1: Humidity in kitchen area, in %
RH_2: Humidity in living room area, in %
RH_3: Humidity in laundry room area, in %
RH_4: Humidity in office room, in %

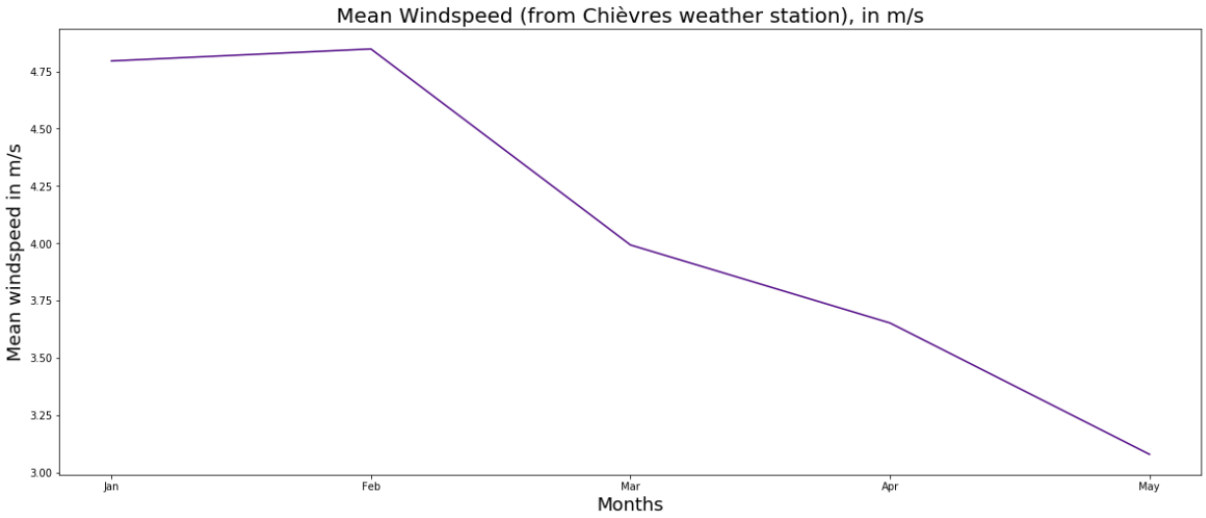
RH_5: Humidity in bathroom, in %
RH_6: Humidity outside the building (north side), in %
RH_7: Humidity in ironing room, in %
RH_8: Humidity in teenager room 2, in %
RH_9: Humidity in parent's room, in %

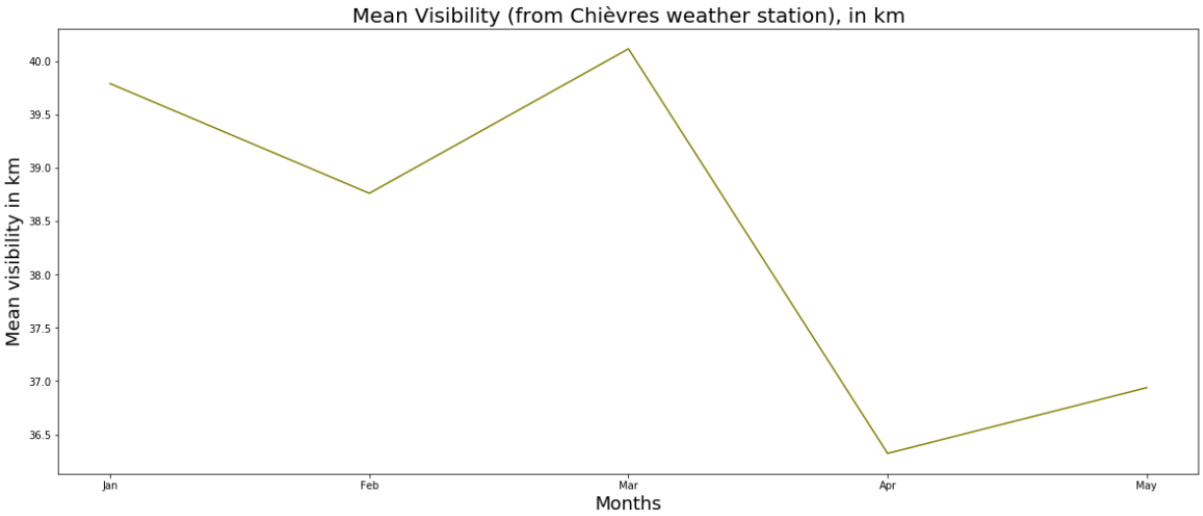
There are 6 other features present in dataset which are: Outside temperature, outside Pressure, outside Humidity, Windspeed, visibility and total dew point. Let's plot each of them in individual graphs to understand their characteristics.

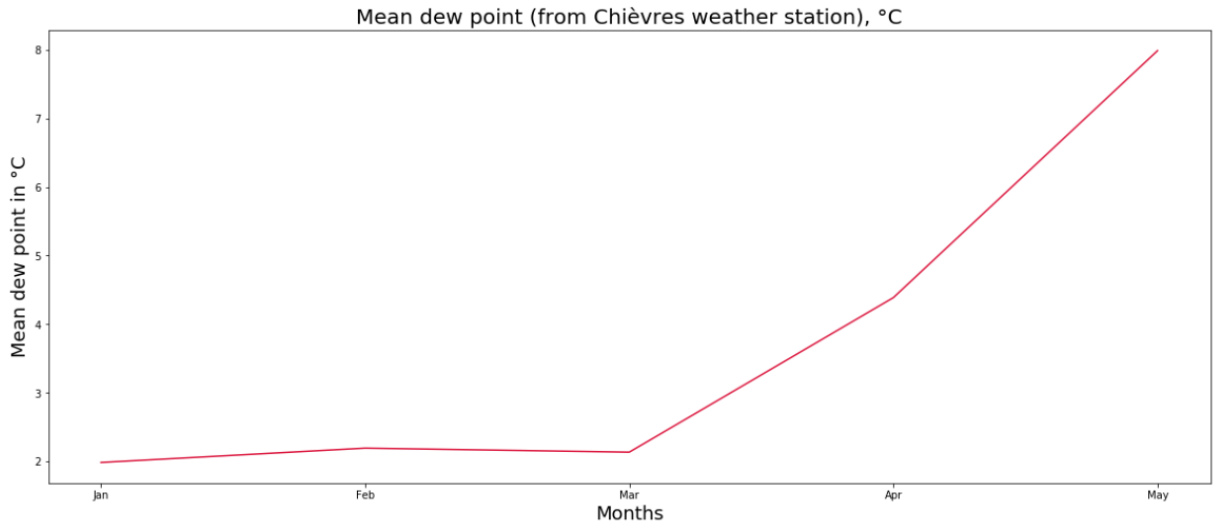




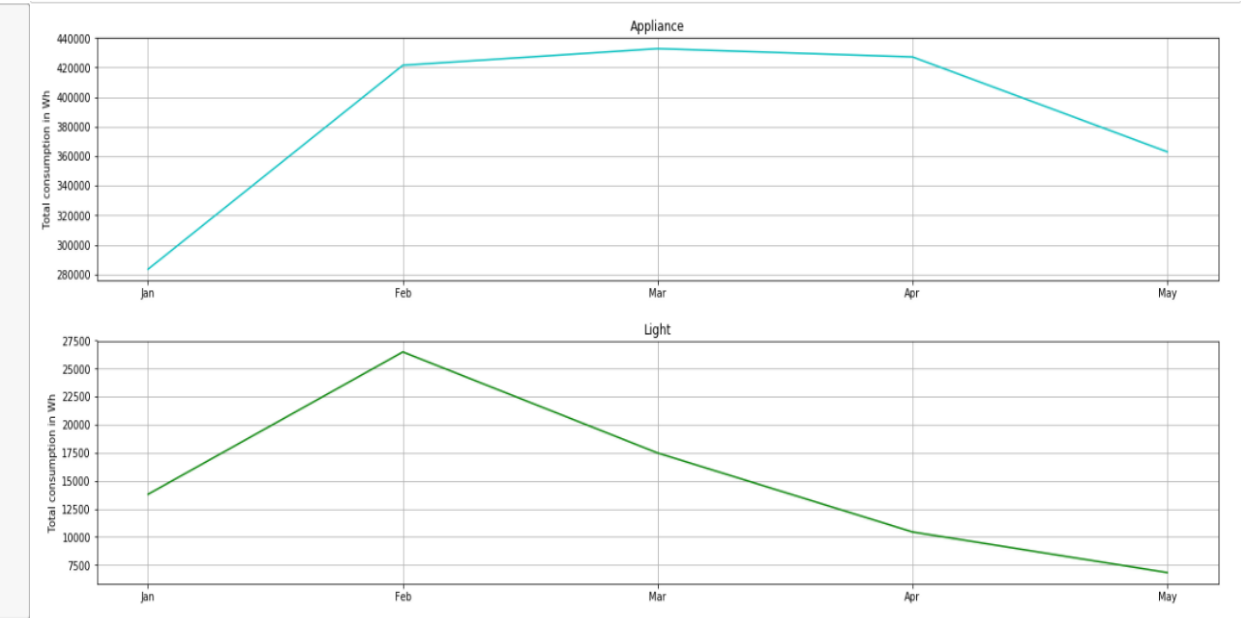

Mean Humidity outside (from Chièvres weather station), in %
Mean Humidity in %
86
84
82
80
78
76
74
72
Jan
Feb
Mar
Apr
May
Months


Mean Windspeed (from Chièvres weather station), in m/s
Mean windspeed in m/s
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
Jan
Feb
Mar
Apr
May
Months

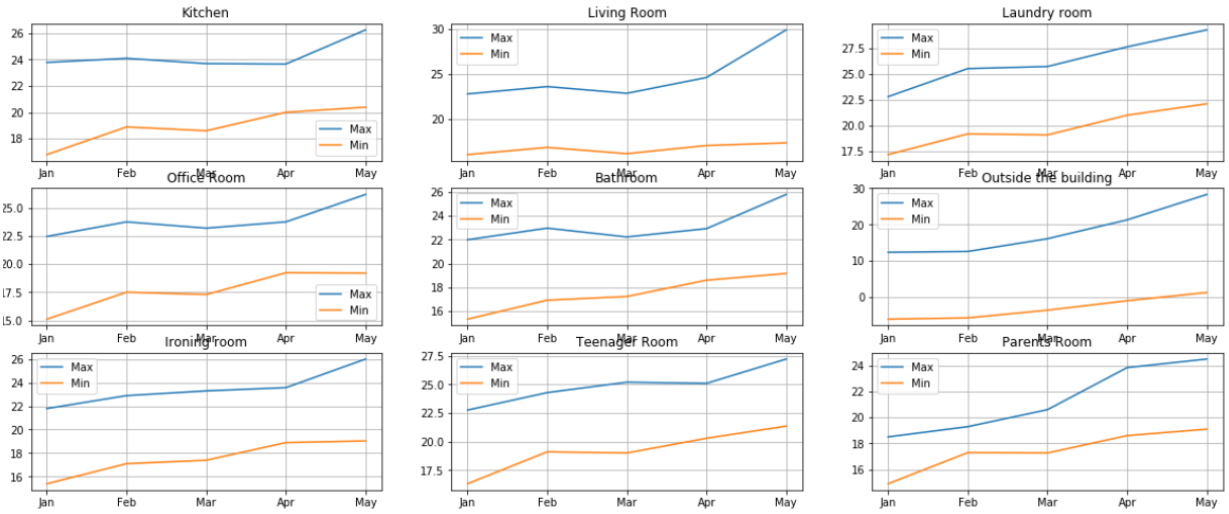

Mean Visibility (from Chièvres weather station), in km
Mean visibility in km
40.0
39.5
39.0
38.5
38.0
37.5
37.0
36.5
Jan
Feb
Mar
Apr
May
Months



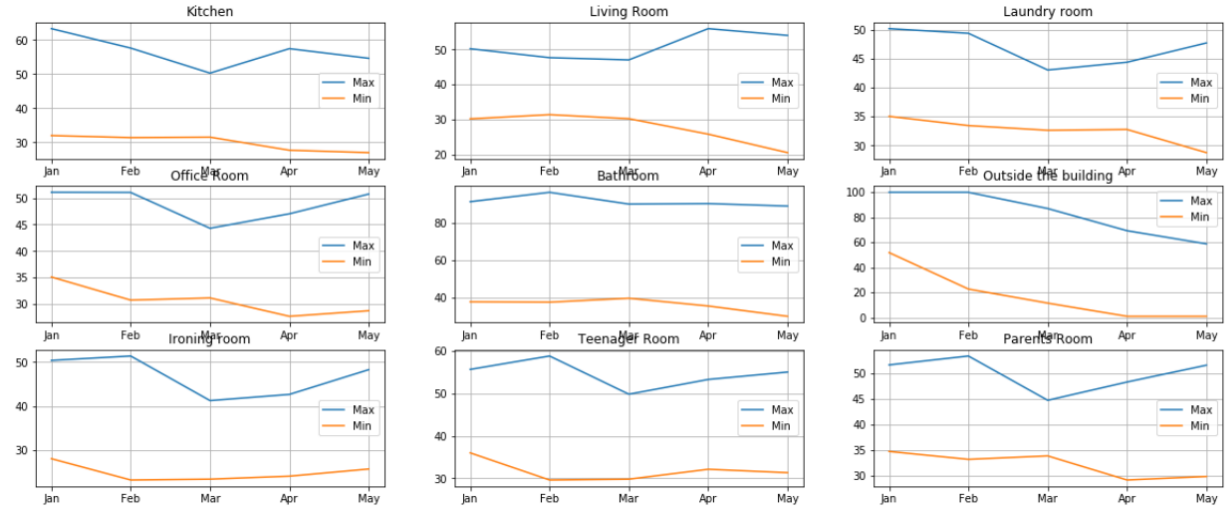
We will now try to find the total consumption of energy by 'lights' and all the appliances at home.



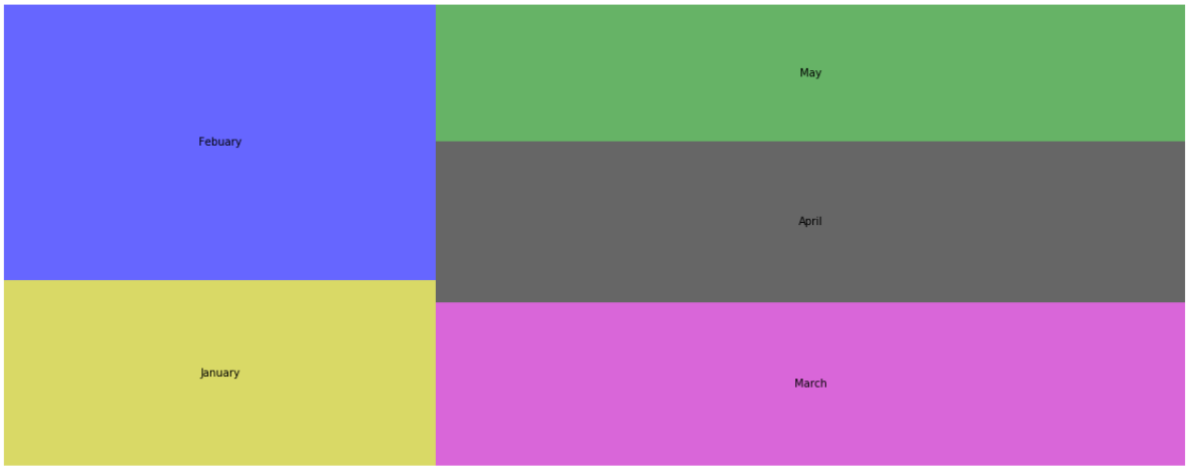
The temperature and humidity is varying a lot over the given period. We should also analyze deviation between minimum and maximum values of these over the period



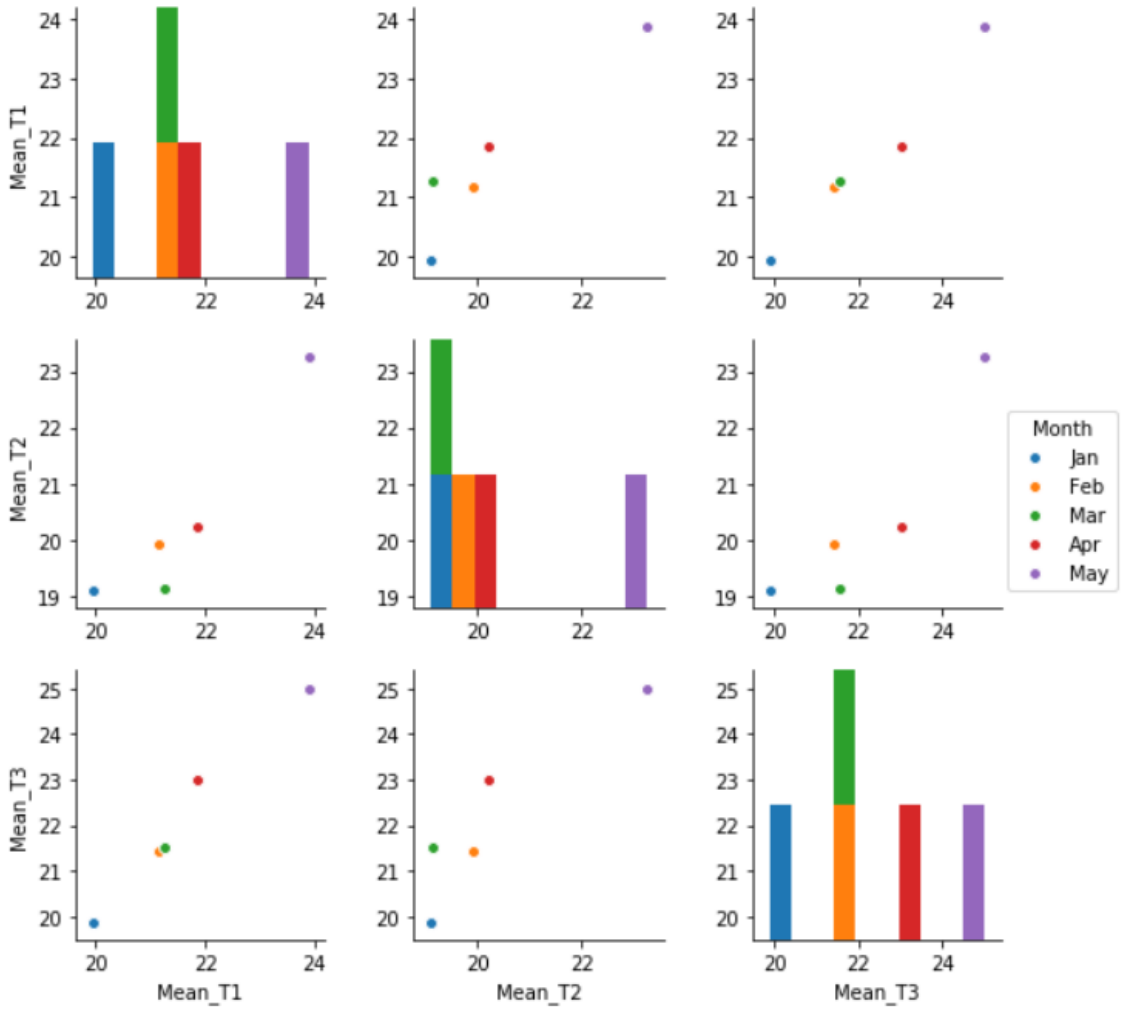
Above graph represents temperature and below represents humidity.

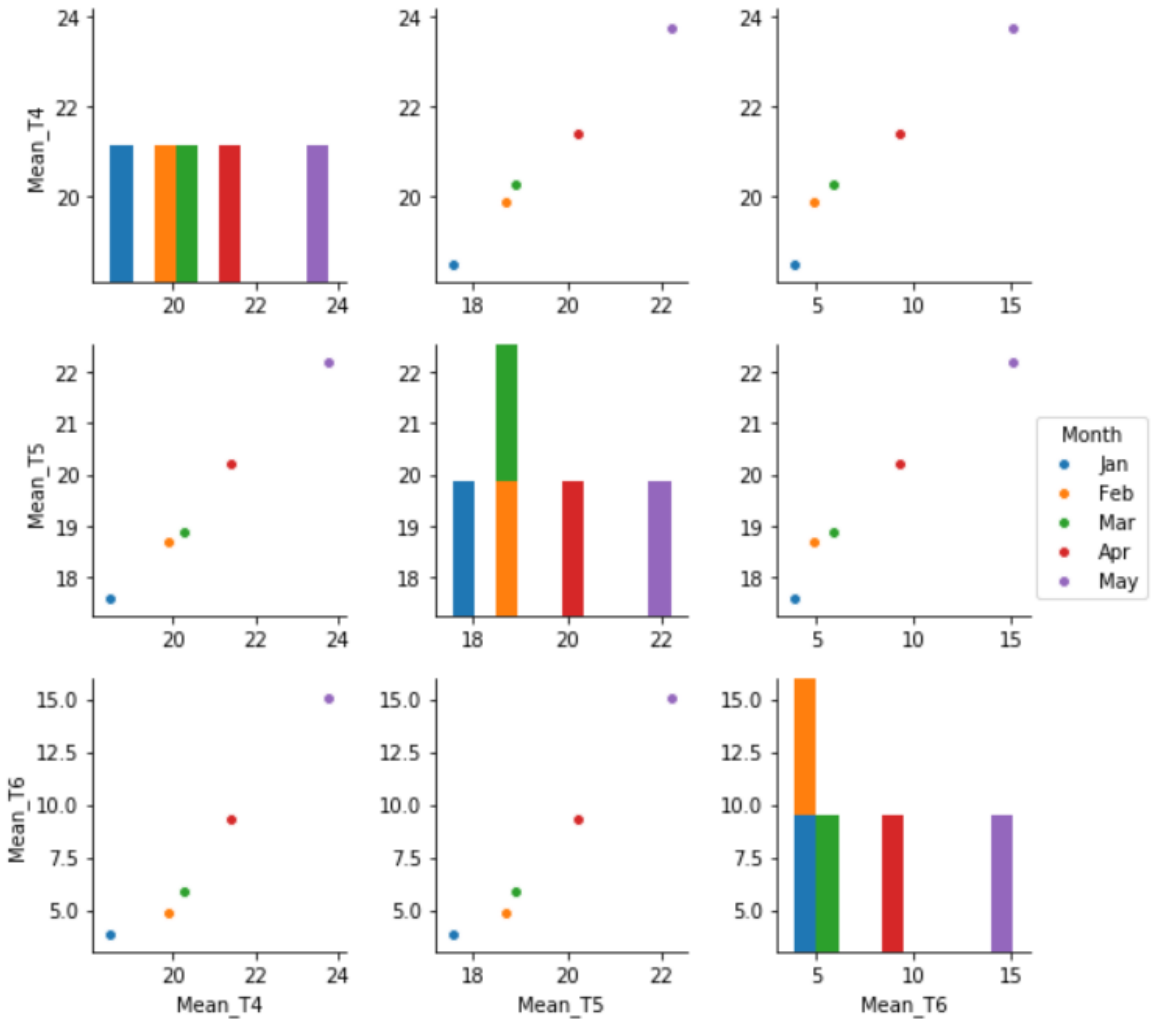


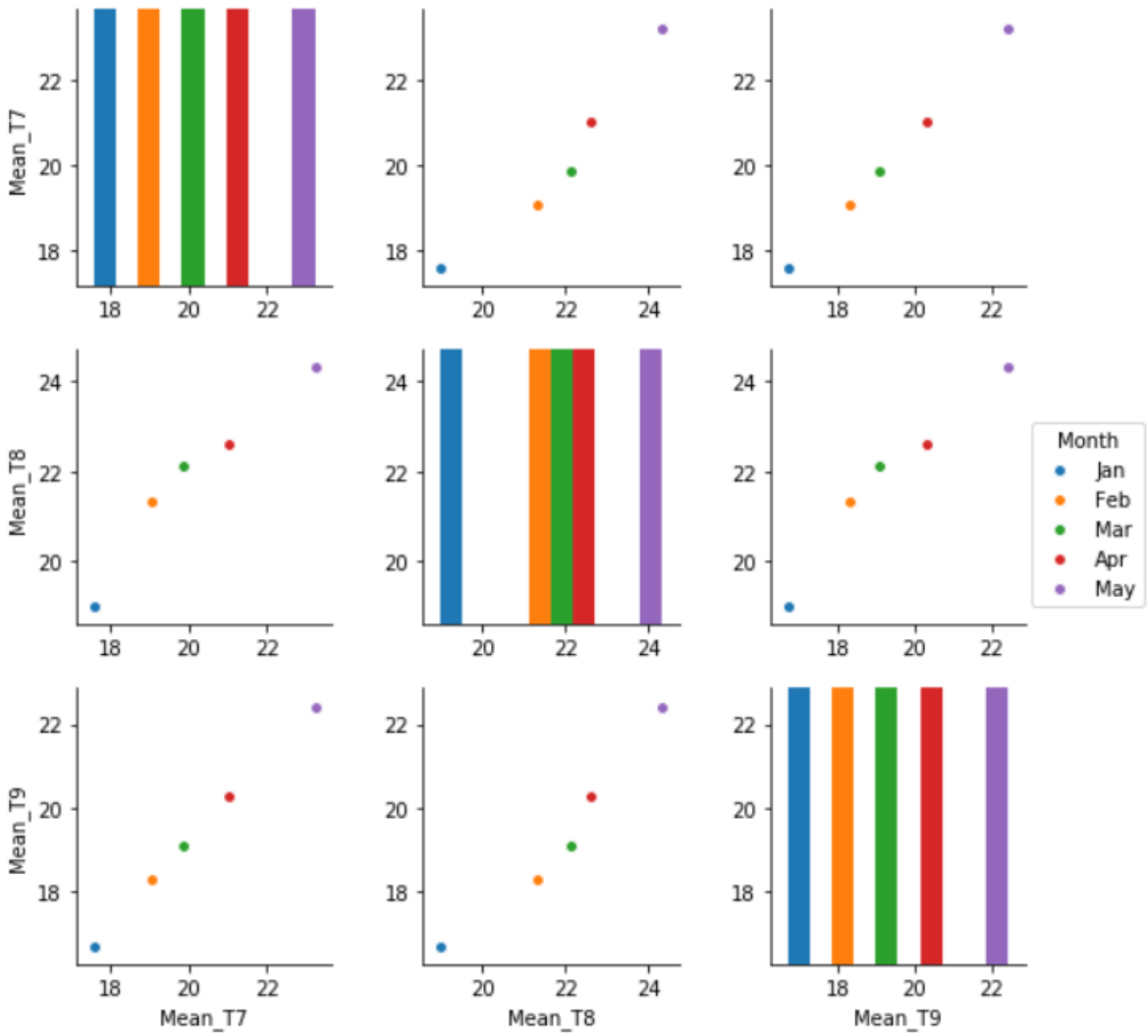
The following is the square plot representation of total energy consumption of appliances over the period.



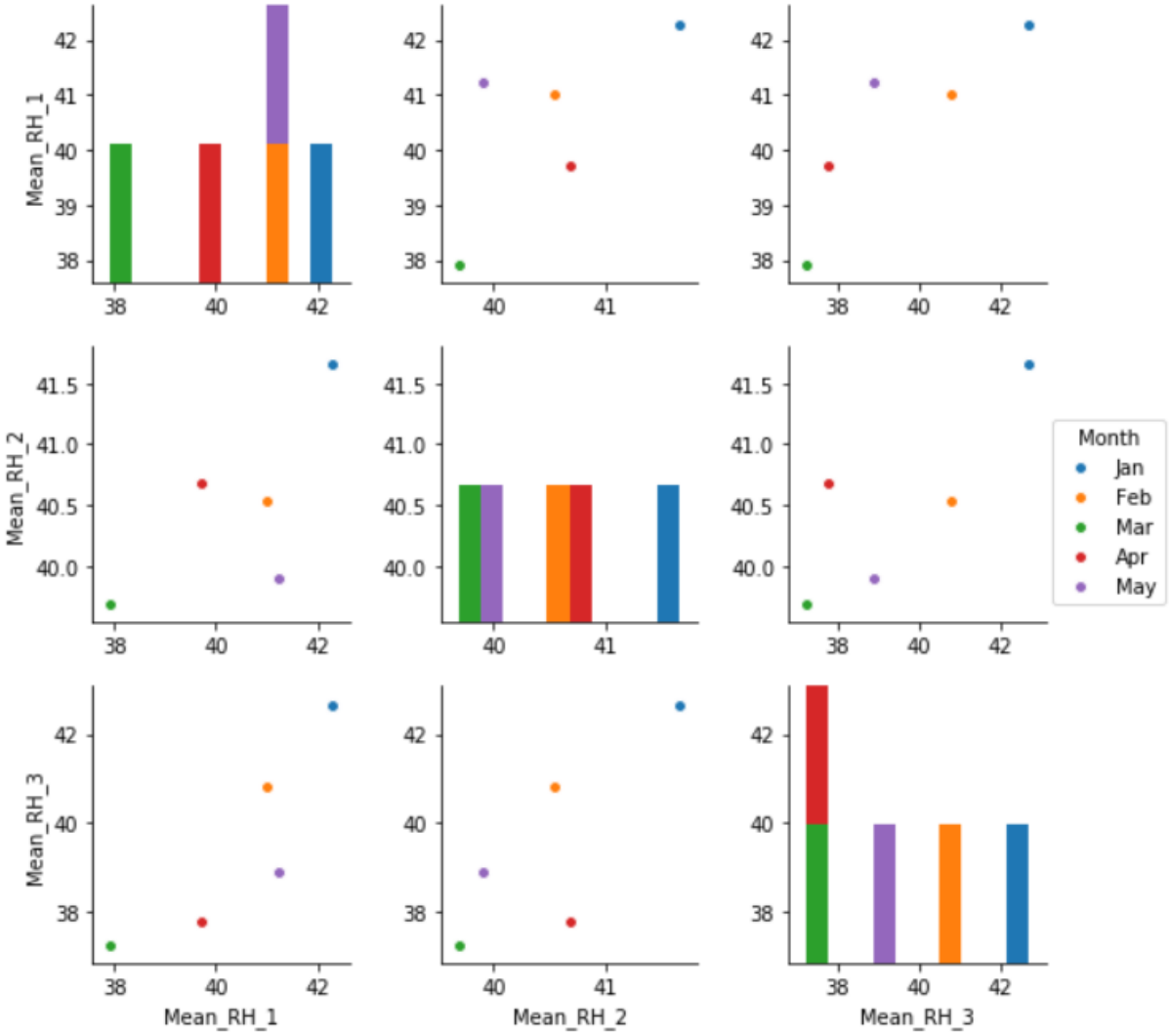
It is important to understand the change in mean of temperature with respect to others and over the period. So, we are comparing mean of temperature in 3 rooms at one time with months.

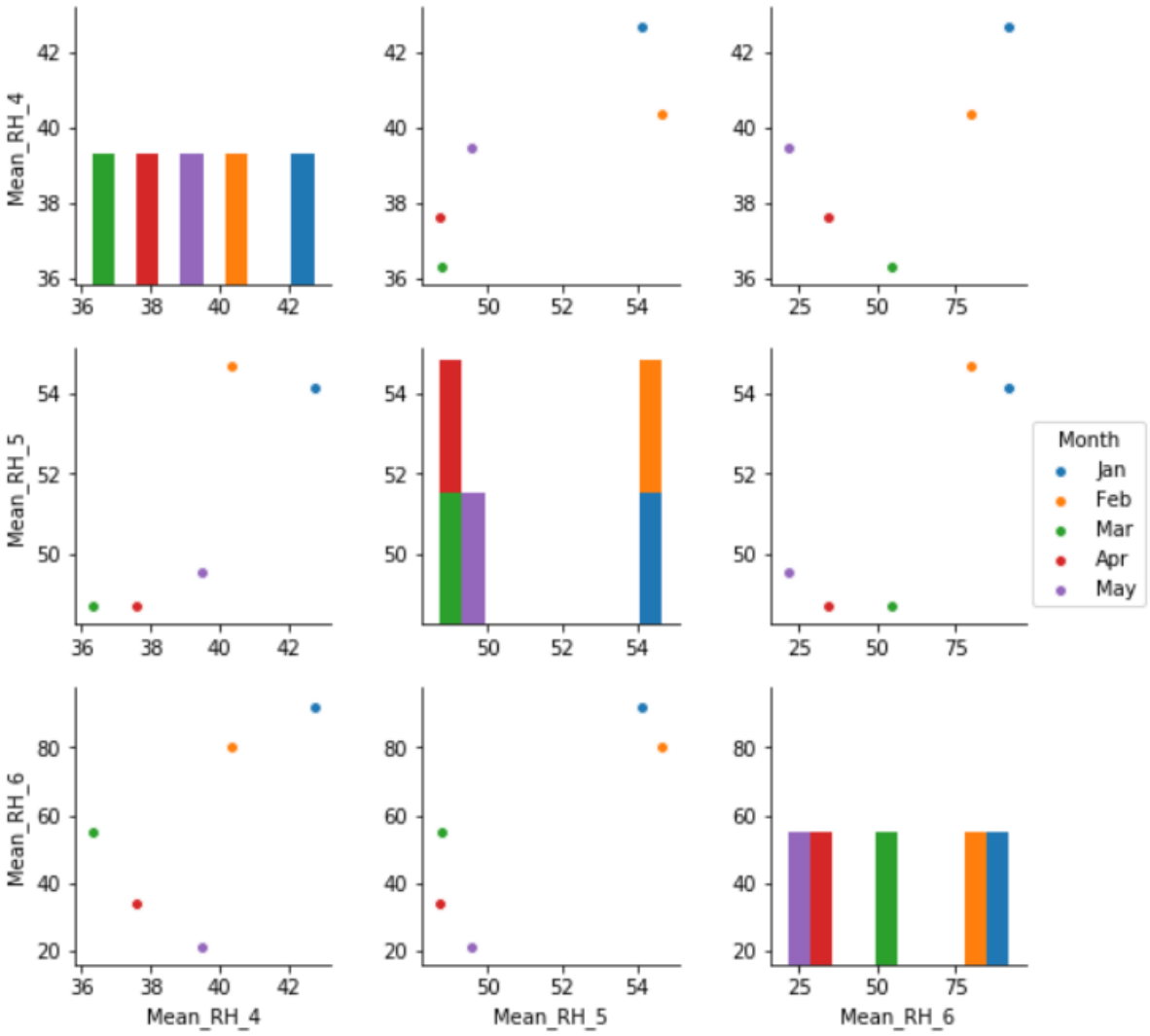


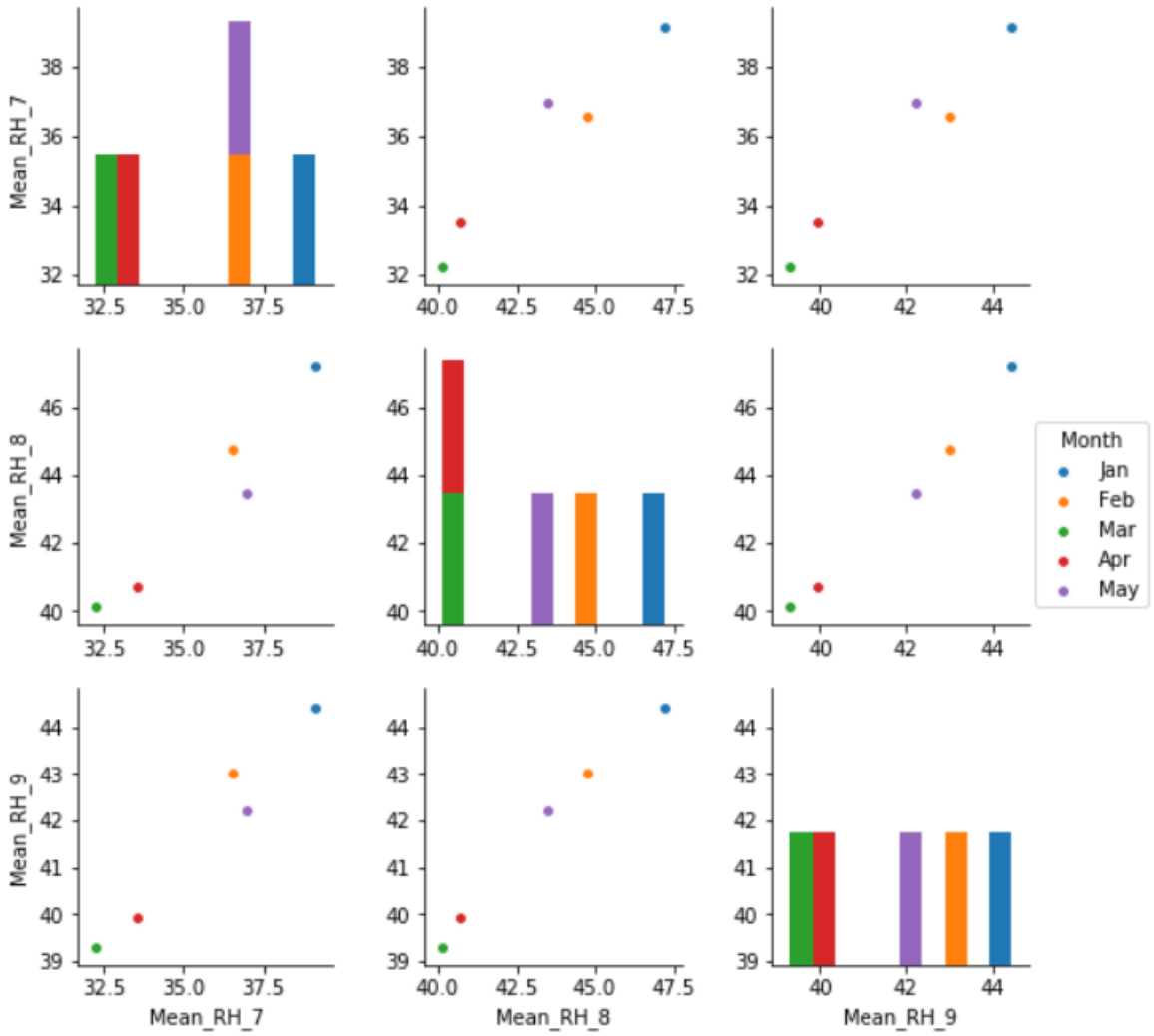

Mean_T4
Mean_T5
Mean_T6
Mean_T4
Mean_T5
Mean_T6
Month
Jan
Feb
Mar
Apr
May


Mean_T7
Mean_T8
Mean_T9
Mean_T7
Mean_T8
Mean_T9
Month
Jan
Feb
Mar
Apr
May

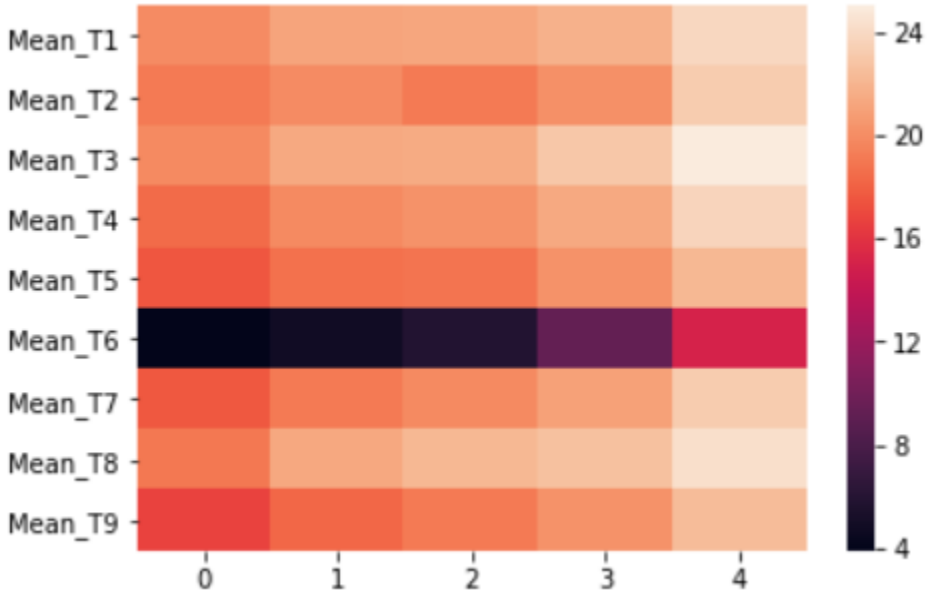
Similarly, comparing the change in mean of humidity of 3 room at a time with respect to months.




Mean_RH_4
Mean_RH_5
Mean_RH_6
Month
Jan
Feb
Mar
Apr
May


Mean_RH_7
Mean_RH_8
Mean_RH_9
Month
Jan
Feb
Mar
Apr
May

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 understand the pattern. Also, we get a view of how the data is varying and which features would be important to be selected to run selected algorithms to get the desired data.

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)
```

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

```
In [6]: f.isnull().sum()

Out[6]: 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
```

```
In [10]: f.dtypes

Out[10]: 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
```

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

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.

```
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)
```

This function is used to bring all values in all features in one scale. This helps us to perform calculations better and be fair towards all other features.

It is also important for us to break down the data.

```
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)
```

We have ‘timestamp’ in our data. To analyze things better, we have to divide the data int smaller formats like date, year, month, etc.

These data put lots of values of type ‘string’ in our dataset. This would make the computation tougher.

To solve that problem, we have to perform one hot encoding.

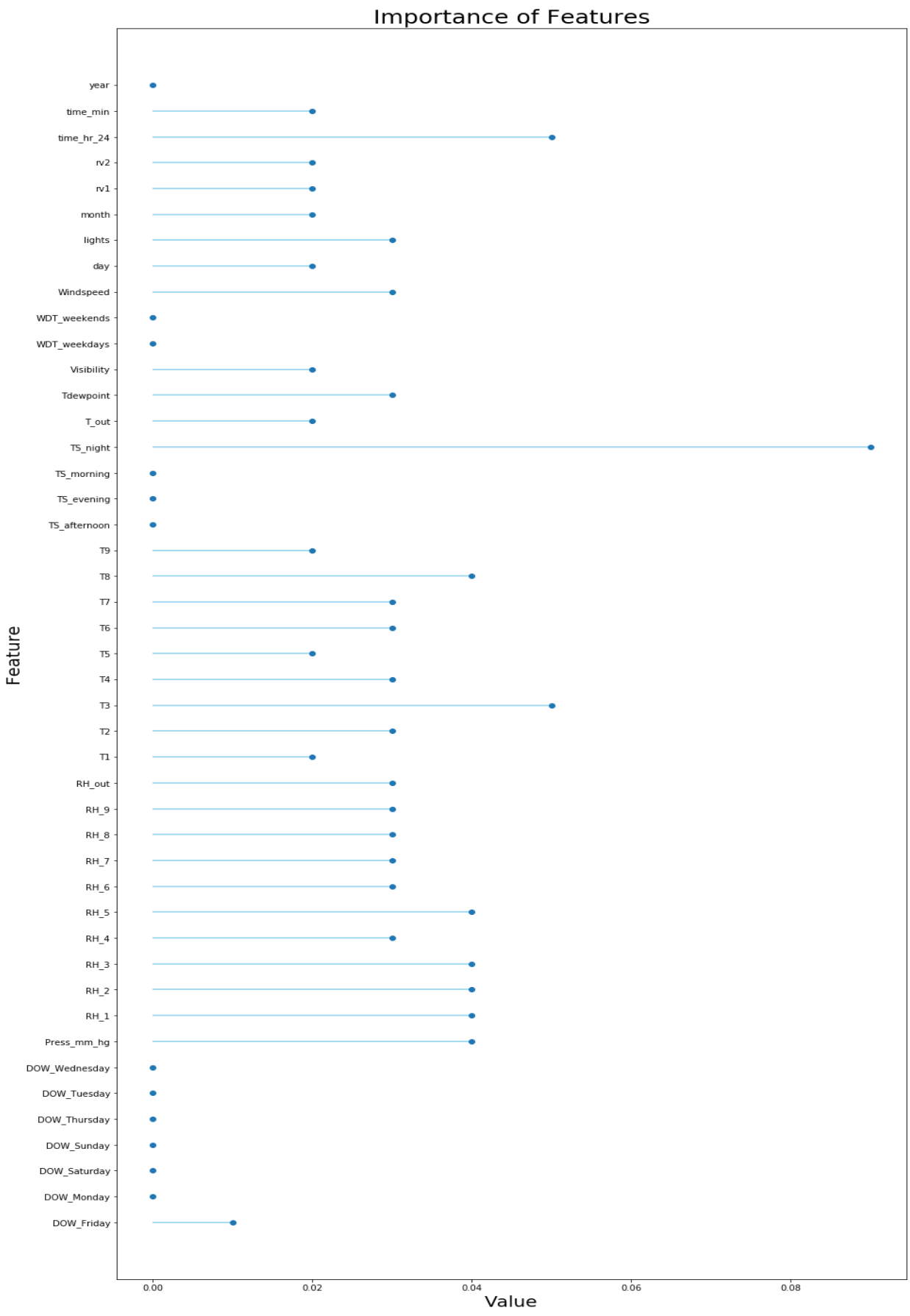
```
df=pd.get_dummies(df,prefix=['DOW','TS','WDT'],columns=['day_of_week','time_slot','week_day_type'])
```

This helps us create Boolean data which would make computation easier.

```
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
```

These are the newly created columns in our dataset. They contain Boolean data.

A graph is drawn to signify the importance of distinctive features in the dataset.


Importance of Features
year
time_min
time_hr_24
rv2
rv1
month
lights
day
Windspeed
WDT_weekends
WDT_weekdays
Visibility
Tdewpoint
T_out
TS_night
TS_morning
TS_evening
TS_afternoon
T9
T8
T7
T6
T5
T4
T3
T2
T1
RH_out
RH_9
RH_8
RH_7
RH_6
RH_5
RH_4
RH_3
RH_2
RH_1
Press_mm_hg
DOW_Wednesday
DOW_Tuesday
DOW_Thursday
DOW_Sunday
DOW_Saturday
DOW_Monday
DOW_Friday
Feature
0.00
0.02
0.04
0.06
0.08
Value

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.

Non-Outliers

```
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)
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)
```

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)
```

Now to perform algorithms, we first must perform it in training 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
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we have to perform the algorithms in thee testing data.

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   : 0.229974899459
MAE  : 34.4072707097
RMSE : 56.6244932848
MAPE : 46.8672903133
```

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.

Now to perform algorithms, we first must perform it in training 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   : 0.903749893974
MAE  : 9.94016024036
RMSE : 20.7769770217
MAPE : 11.8311980947
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we have to perform the algorithms in thee testing data.

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   : 0.460272022653
MAE  : 24.7200155763
RMSE : 47.4066593965
MAPE : 30.7304787396
```

Optimizing of Dataset

Initially we were using entire dataset but now we are trying to optimize the data to get better results.

Building Models Based on Selected Features

```
drop_col_list=['year','DOW_Friday','DOW_Monday','DOW_Saturday','DOW_Sunday','DOW_Thursday','DOW_Tuesday','DOW_Wednesday','TS_after
               'WDT_weekdays','WDT_weekends','TS_morning','month','time_min' ,'TS_evening','day','rv1','rv2','Visibility','Windspee
                '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']
x_test_sc=scaler.transform(x_test)
```

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

Linear Regression

Let’s build the linear regression model first.

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 perform tests on training 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   : 0.205607329539
MAE  : 35.740773535
RMSE : 59.689695776
MAPE : 47.9896147986
```

And now on testing data.

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   : 0.194025341463
MAE  : 35.2672418322
RMSE : 57.9312075245
MAPE : 48.7600182166
```

Random Forest

Before proceeding with testing the algorithm, we first have to tune the hyperparameters to get better result.

Random Forest Model After using tuned hyperparameters

```
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)
```

Will perform the tests on training 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   : 0.999997098311
MAE  : 0.0374766366221
RMSE : 0.11407937878
MAPE : 0.0670381095246
```

Now will perform the tests on testing data.

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   : 0.57042010372
MAE  : 20.5130126239
RMSE : 42.2935229173
MAPE : 24.750752227
```

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.

We have to check the data before starting the tests.

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

```
(11982, 45)
```

We must prepare a model for our tests.

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 can start performing tests. Initially we start with our training data.

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
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we should perform the algorithms in thee testing data.

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   : 0.395215589656
MAE  : 29.170573685
RMSE : 50.1824870825
MAPE : 37.5097758701
```

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)
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)
```

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.

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

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

Now to perform algorithms, we first must perform it in training data.

```
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
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we have to perform the algorithms in thee testing data.

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   : 0.202353758845
MAE  : 52.4473890861
RMSE : 90.6523438651
MAPE : 61.7795970369
```

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.

First, we build the 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)
```

Now to perform algorithms, we first must perform it in training 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   : 0.901313591729
MAE  : 14.3792529318
RMSE : 32.3431822664
MAPE : 14.2605523459
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we have to perform the algorithms in thee testing data.

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   : 0.47441832924
MAE  : 35.5865563249
RMSE : 73.5857942129
MAPE : 35.4129570787
```

Building Model

Initially we were using entire dataset but now we are trying to optimize the data to get better results.

Building Models Based on Selected Features

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

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

Linear Regression

We start with building the 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 to perform algorithms, we first must perform it in training 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   : 0.171048338471
MAE  : 53.2887700492
RMSE : 93.7386468196
MAPE : 62.3960338305
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we have to perform the algorithms in thee testing data.

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   : 0.171646282313
MAE  : 53.3113370386
RMSE : 92.3808148987
MAPE : 64.0116106896
```

Random Forest

First, we build the model

```
: 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)
```

Now to perform algorithms, we first must perform it in training 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   : 0.999360449368
MAE  : 1.36231948386
RMSE : 2.60370359672
MAPE : 1.80088335883
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we have to perform the algorithms in thee testing data.

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   : 0.631187334288
MAE  : 28.1314719464
RMSE : 61.6420097136
MAPE : 27.1339805346
```

Neural Network

We have to check the data before starting the tests.

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

```
(13814, 45)
```

We must prepare a model for our tests.

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)
```

Our model is ready. Now, we can start performing tests. Initially we start with our training data.

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   : 0.75920061787
MAE  : 27.9396749876
RMSE : 50.5221426581
MAPE : 30.2356429216
```

We can see the values of R2, MAE, RMSE and MAPE from the above screenshot. Now, we should perform the algorithms in thee testing data.

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   : 0.407856186557
MAE  : 39.4717490931
RMSE : 78.1065533965
MAPE : 39.3081527053
```

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.

Training data.

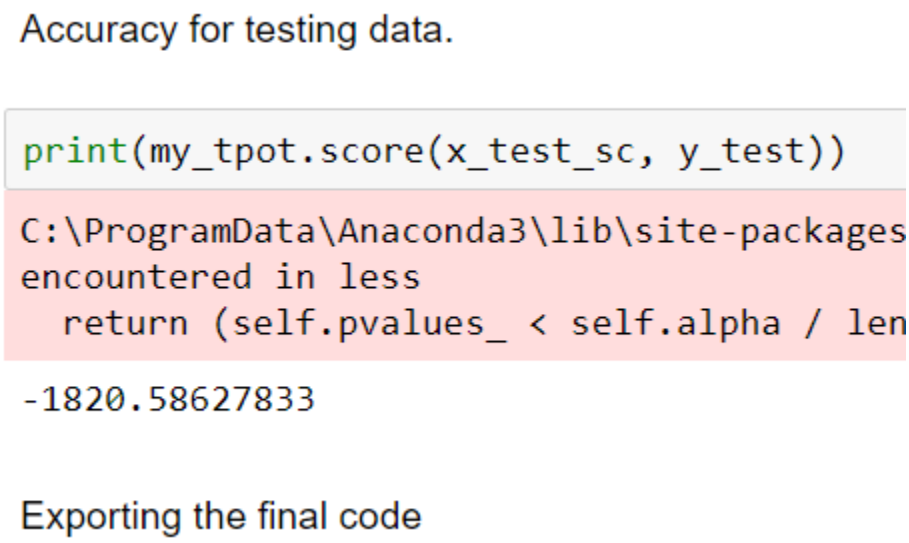
Accuracy for training data.

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

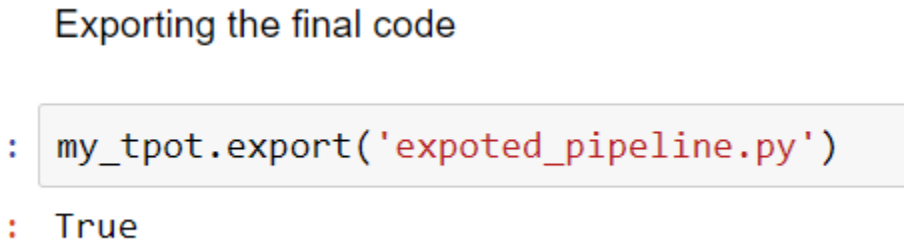
```
C:\ProgramData\Anaconda3\lib\site-packages\skl
encountered in less
  return (self.pvalues_ < self.alpha / len(sel
```

```
-46.8504652813
```

Testing Data.



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
```

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 elliminating irrelevant featurs to stabilise that test.

First, we should build a model.

Building 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 Boruta code.

Running the boruta.

```
import pandas as pd
#from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

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

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

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

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

After iterations, the output received was,

```
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 0x000002236F229E58>,
           verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x000002236F229E58>,
     two_step=True, verbose=2)
```

Checking for key features in our dataset.

```
# check selected features
feat_selector.support_
```

```
array([ True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False, False,  True,  True,  True,  True,
        True, False,  True,  True, False, False, False, False, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False], dtype=bool)
```

Implementing the selected features in the model to get better output.

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

Now we should make a model for random forest.

Making a model for Random Forest.

```
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)
```

For which we get the following output.

```
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)
```

Now we can check the accuracy of the training and testing dataset.

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   : 0.999365728603
MAE  : 0.978722739192
RMSE : 1.68662803358
MAPE : 1.58981916224
```

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   : 0.549177108565
MAE  : 21.2181672511
RMSE : 43.326625739
MAPE : 25.663435664
```

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

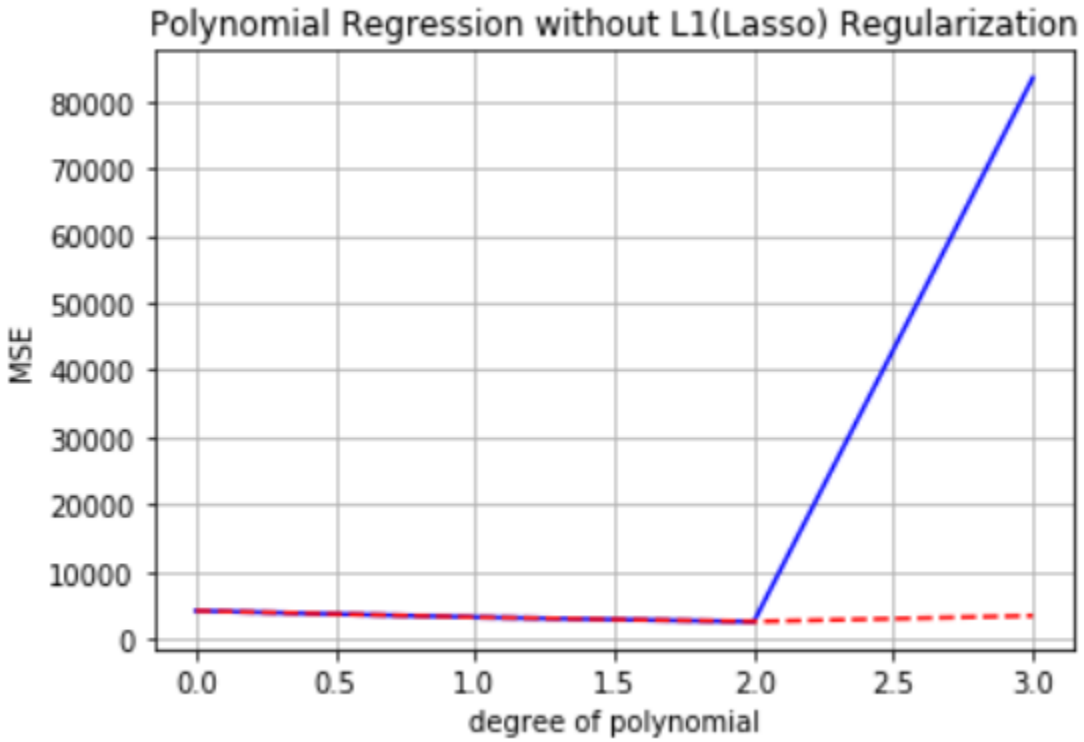
Lasso(L1)

```
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)

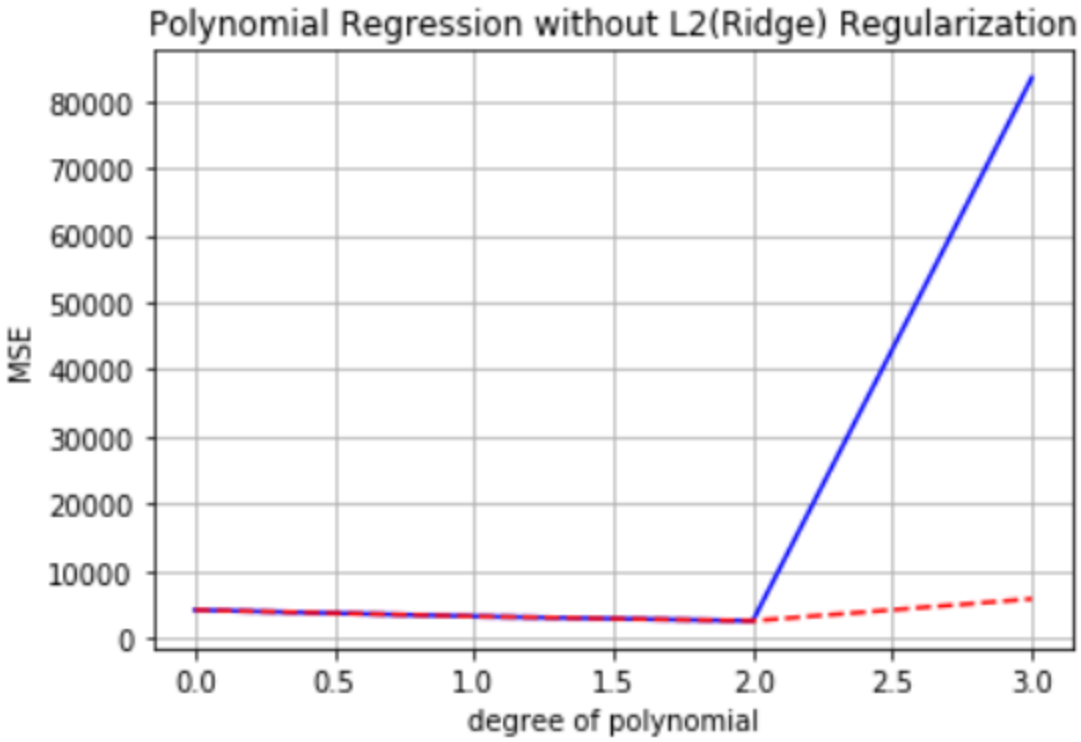
```
for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    X_train_ = model.fit_transform(X_train)
    X_test_ = model.fit_transform(X_test)

    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.



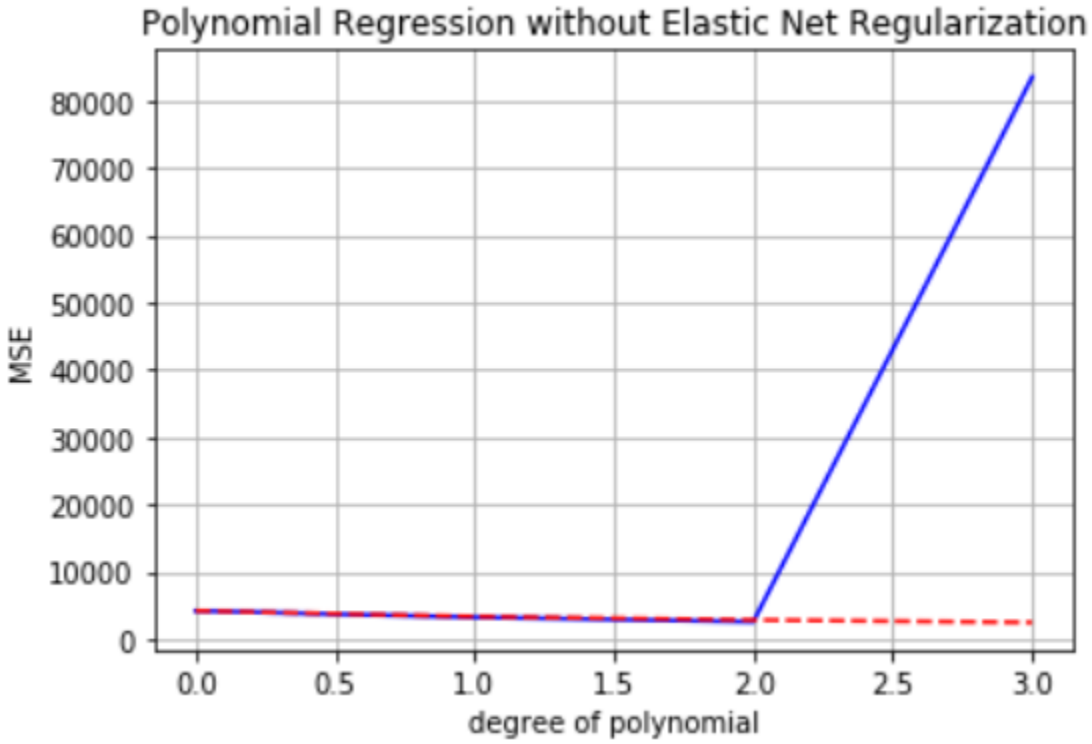
ElasticNet

```
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()
```

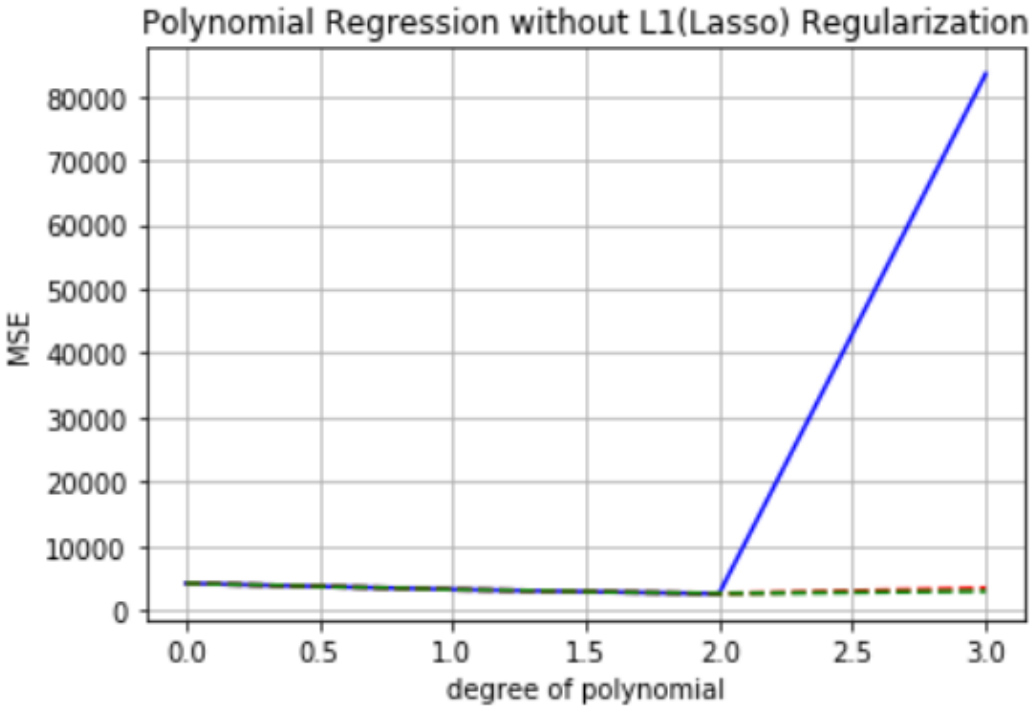
And the scores and the alpha value which we received are as follows.

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

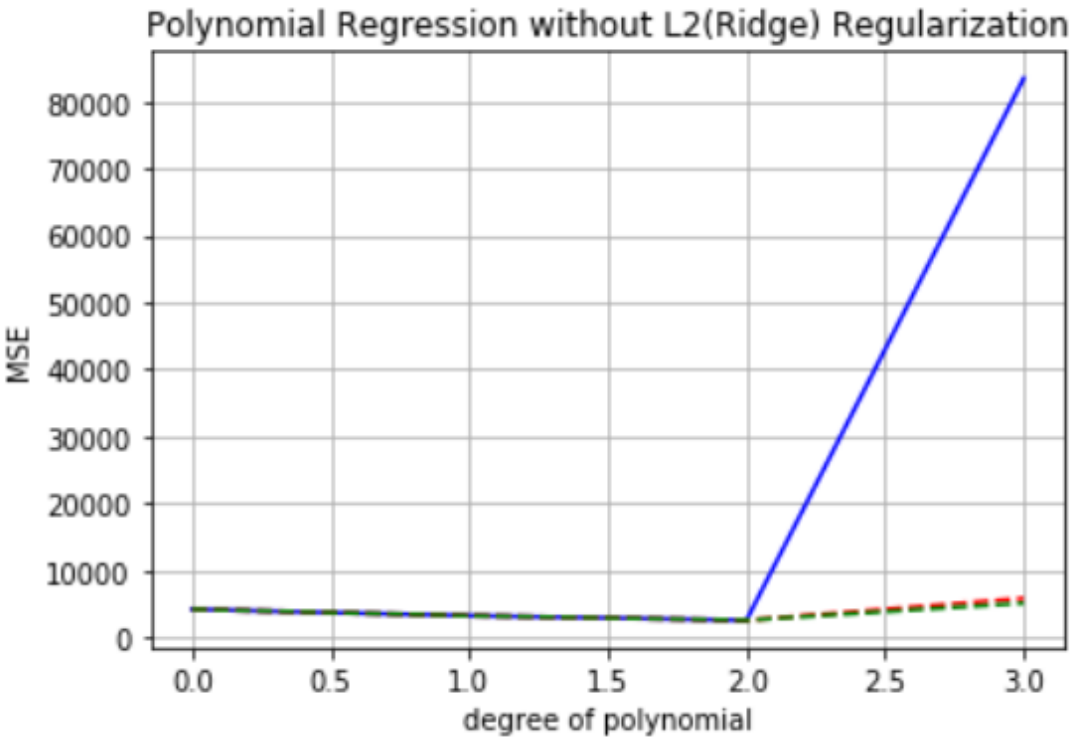
```
The calculated optimal alpha is 0.000050
The max generalization score of L1 regularized polynomial regression model is 0.377953 +- 0.000245
```

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

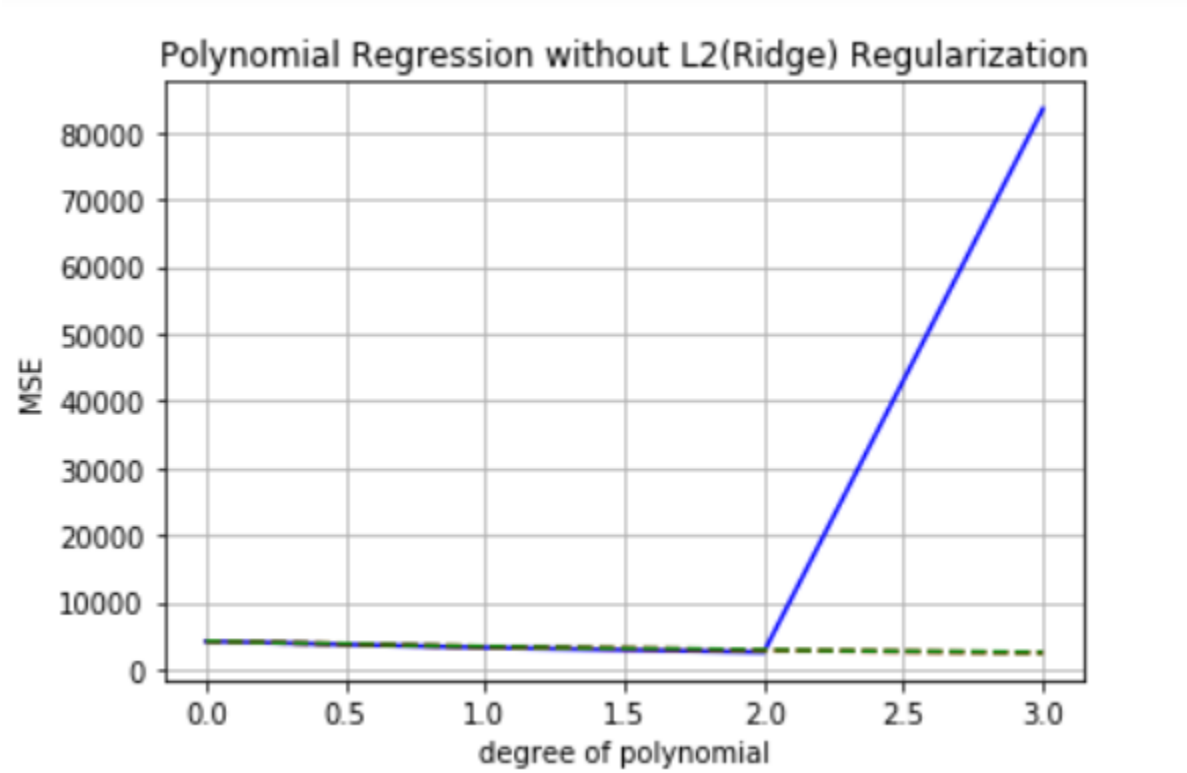
Lasso(L1)



Ridge(L2)


Polynomial Regression without L2(Ridge) Regularization
MSE
80000
70000
60000
50000
40000
30000
20000
10000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
degree of polynomial

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.

Part 7: Final Pipeline

First, we should create a model

Then we should split the data into training and testing.

```
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']
x_test=df_test.iloc[:,1:]
y_test=df_test['Appliances']
```

Then we should run the pipeline code.

```
pipe = Pipeline([('scl', StandardScaler()),
    ('pca', PCA(n_components=21)),
    ('gfdh', 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))])
pipe.fit(x_train, y_train)
```

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   : 0.999409730548
MAE  : 1.42771284441
RMSE : 2.50137745182
MAPE : 2.15155029381
```

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   : 0.473373115213
MAE  : 36.1106762811
RMSE : 73.658927189
MAPE : 36.5689136924
```

Conclusion

We noticed that ‘random Forest’ model is the best algorithm which can be used to derive a reliable prediction.