Assignment 2 and Mid-Term - Documentation

INFO 7390

Advanced Data Science and Architecture

Machine learning with Energy Datasets

Presented by:

DhanishaDamodarPhadate

001859234

phadate.d@husky.neu.edu

Palak Sharma

001834478

sharma.pala@husky.neu.edu

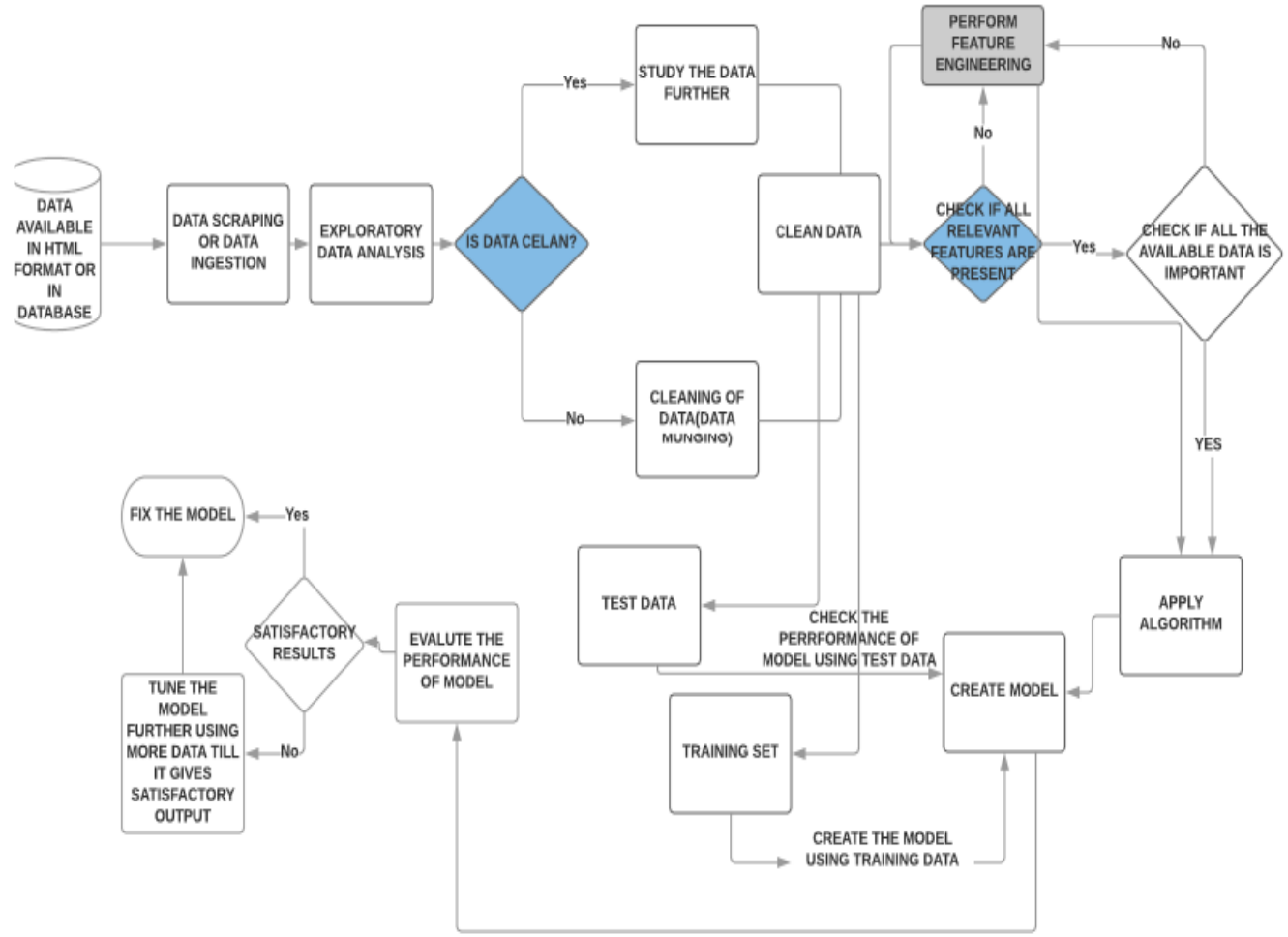
Dharani Thirumalaisamy

001887922

thirumalaisamy.d@husky.neu.edu

CONTENT

FLOW CHART


DATA AVAILABLE IN HTML FORMAT OR IN DATABASE
DATA SCRAPING OR DATA INGESTION
EXPLORATORY DATA ANALYSIS
IS DATA CELAN?
Yes
STUDY THE DATA FURTHER
No
CLEANING OF DATA(DATA MUNGING)
CLEAN DATA
PERFORM FEATURE ENGINEERING
No
CHECK IF ALL RELEVANT FEATURES ARE PRESENT
No
Yes
CHECK IF ALL THE AVAILABLE DATA IS IMPORTANT
YES
APPLY ALGORITHM
CREATE MODEL
CHECK THE PERRFORMANCE OF MODEL USING TEST DATA
TEST DATA
TRAINING SET
CREATE THE MODEL USING TRAINING DATA
EVALUTE THE PERFORMANCE OF MODEL
SATISFACTORY RESULTS
Yes
FIX THE MODEL
No
TUNE THE MODEL FURTHER USING MORE DATA TILL IT GIVES SATISFACTORY OUTPUT

PART 1 : EXPLORATORY DATA ANALYSIS

EDA which stands for Exploratory Data Analysis is used for the analysis of the dataset that is given by plotting graphs of different nature.

In this project , libraries such as matplotlib , seaborn and plotly are used to analyze the dataset.

Step 1: All the necessary modules are imported.

Step 2: Importing the dataset using pandas dataframe.

Step 3 : In order to make proper analysis of the dataset , new columns are derived from the existing columns . For example : from 'date' column , other columns such as 'month' , 'hour' , 'time' , 'day' ,'day_of_week' ,'weekStatus' and 'WeekStatus' are derived. Each column gives us a clear information on whether it is weekday or weekend , at what time the value has increased etc.

```
df['date'] = pd.to_datetime(df['date'])
df['month'] , df['time'] , df['hour'] ,df['day'] , df['day_of_week'],df['Numerical_Week'] =  df['date'].dt.month , df['date'].dt.
df['weekStatus'] = df['date'].dt.dayofweek
df['WeekStatus'] = np.where(df['weekStatus'] < 5, 'Weekday', 'Weekend')
```

Step 4 : As this is a time-series dataset , it is not a good habit to delete the 'date' column which is in the dataset but keeping the time column won't be much of a use as it is in datetime.datetime.series format. So ,in order to make it useful , the 'date' column is converted into 'NSM' which gives us the seconds for the corresponding time and after this , ' date' column is made as the index.

```
d = df.date[0:len(df.date)]
data = []
#print(data)
for i in range (len(d)):
    if (i==0):
        a = 61200
        data.append(a)
    elif (i > 0 ):
        a = a + 600
        data.append(a)
#print((data))
df1 = pd.DataFrame({'data_converted' : data})
df['data_converted'] = df1
```

New columns :

month	time	hour	day	day_of_week	Numerical_Week	weekStatus	WeekStatus	data_converted
1	17:00:00	17	11	Monday	0	0	Weekday	61200
1	17:10:00	17	11	Monday	0	0	Weekday	61800
1	17:20:00	17	11	Monday	0	0	Weekday	62400
1	17:30:00	17	11	Monday	0	0	Weekday	63000
1	17:40:00	17	11	Monday	0	0	Weekday	63600

Step 5 : To make the dataset further useful , the month column which is in integer format is converted to its corresponding month.

```
import calendar
df['month'] = df['month'].apply(lambda x: calendar.month_abbr[x])

first_month = df.loc[df['month'] == 'Jan']
second_month = df.loc[df['month'] == 'Feb']
third_month = df.loc[df['month'] == 'Mar']
forth_month = df.loc[df['month'] == 'Apr']
fifth_month = df.loc[df['month'] == 'May']
```

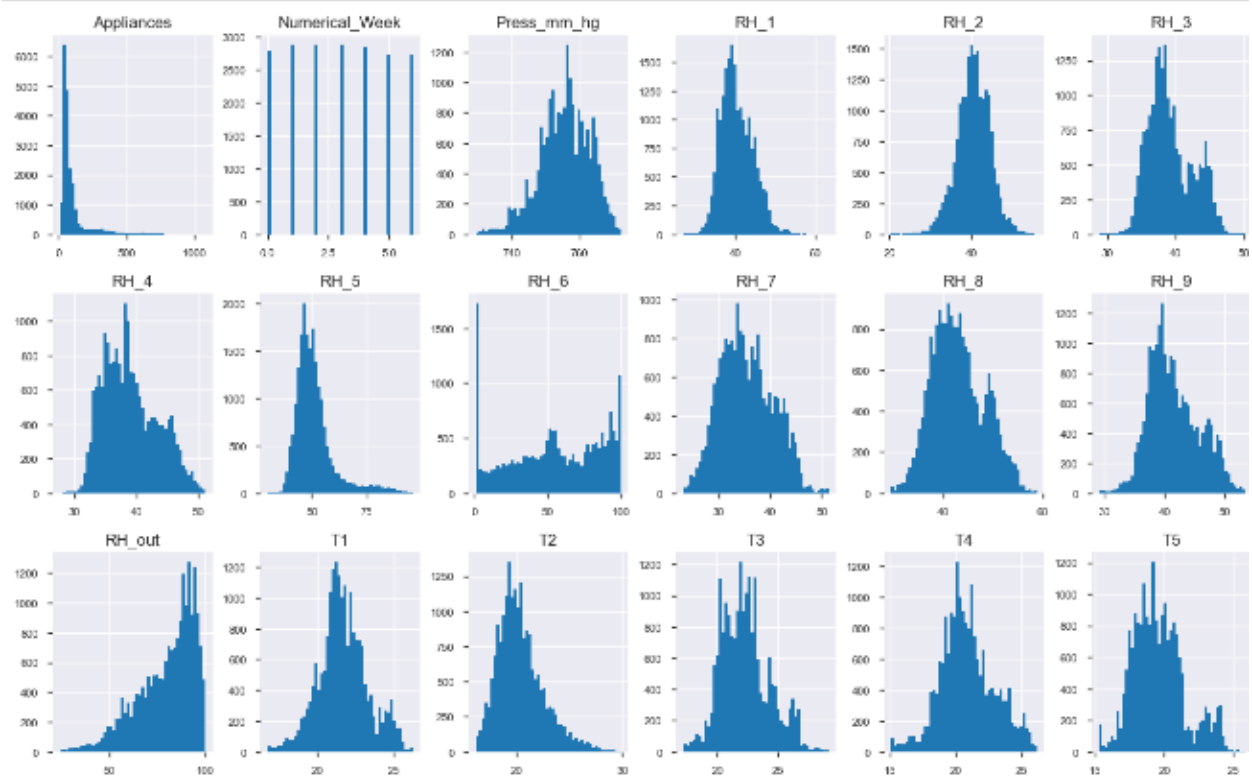
Step 6 : Before starting the analysis it is important to know if there are any missing values present in the dataset. So we do data profiling to check that and our dataset has no missing values.

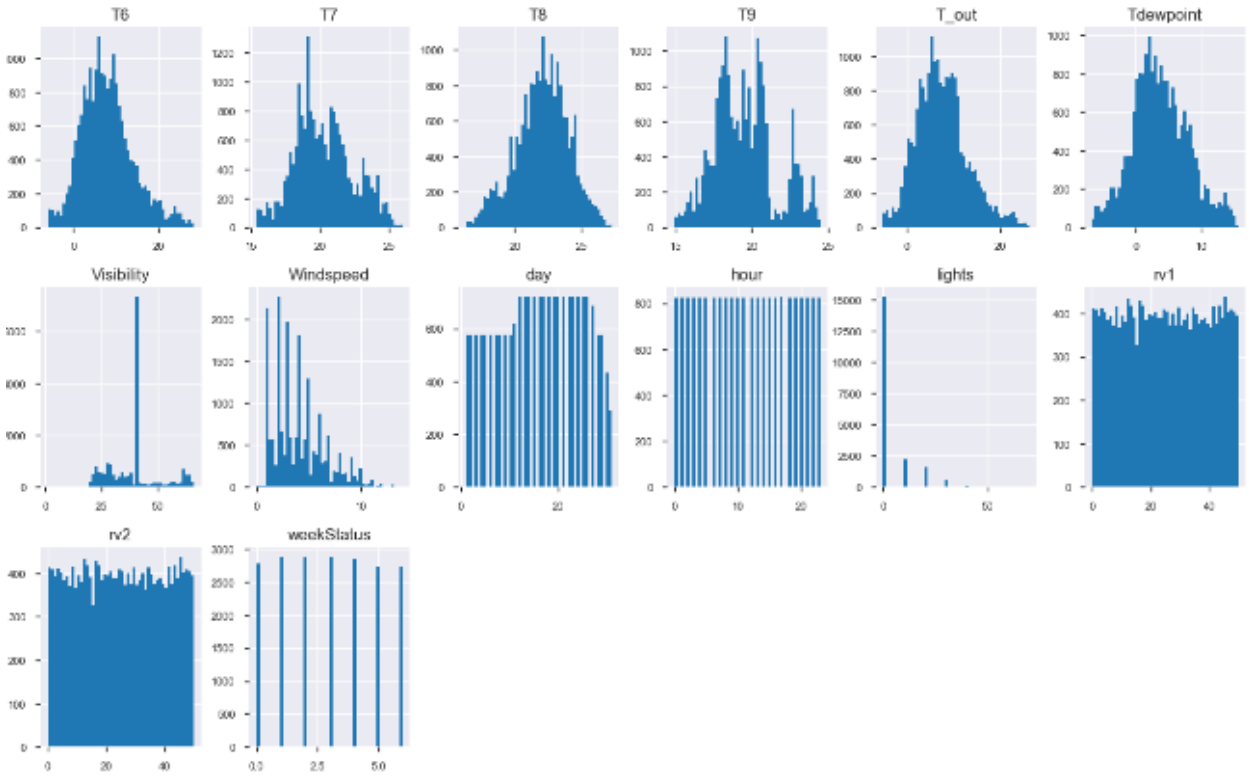
Step 7 : Now we have 37 variables and 19735 observations taken every 10 mins. We will start analyzing one by one to see how it affects other variables.

Step 8 : Analyzing the dataset.

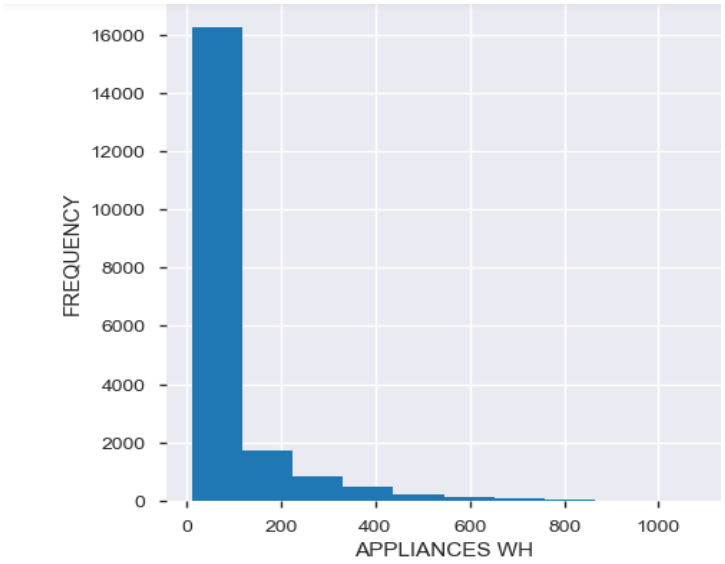
Analysis 1 : Overview of the data :

Histogram is plotted for all the variables for us to know which observation in each variable has occurred the most.



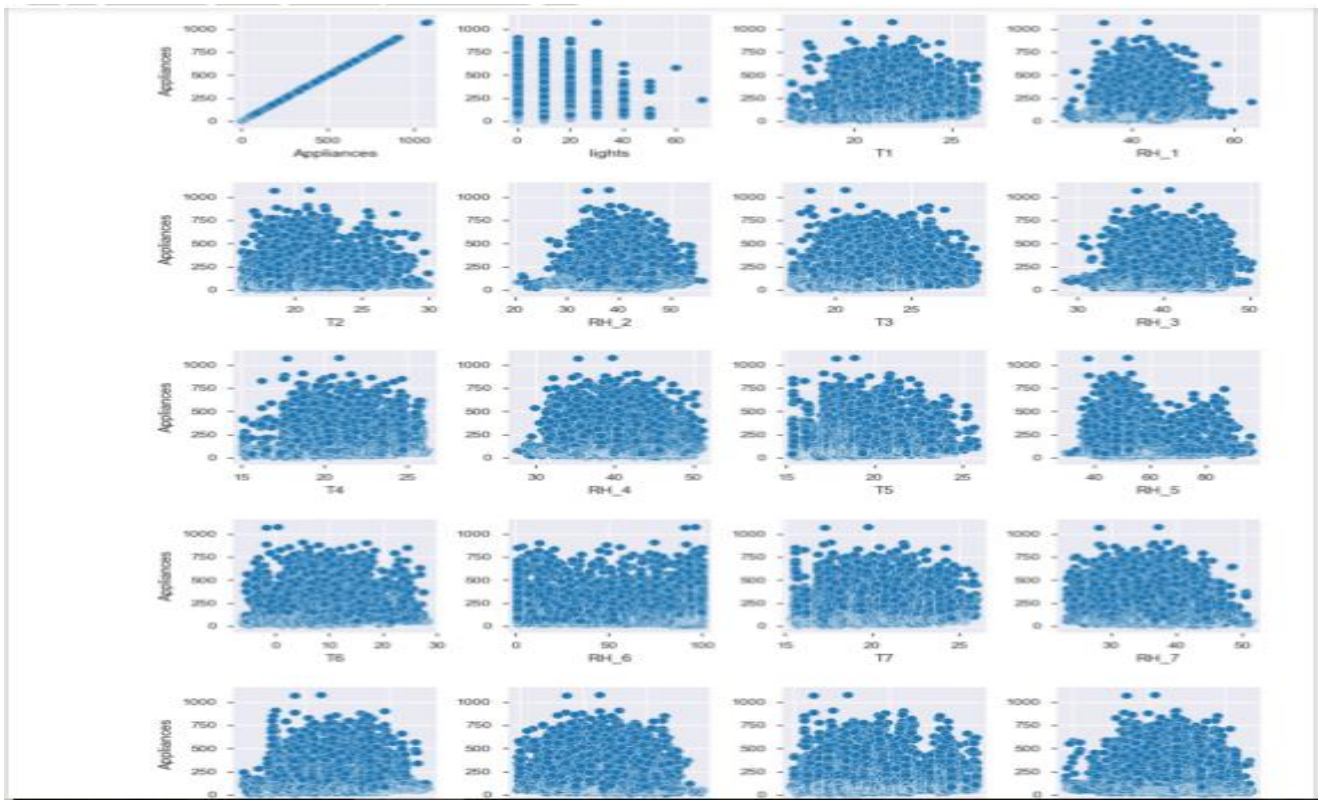


Analysis 2 : Analyzing 'APPLIANCES'

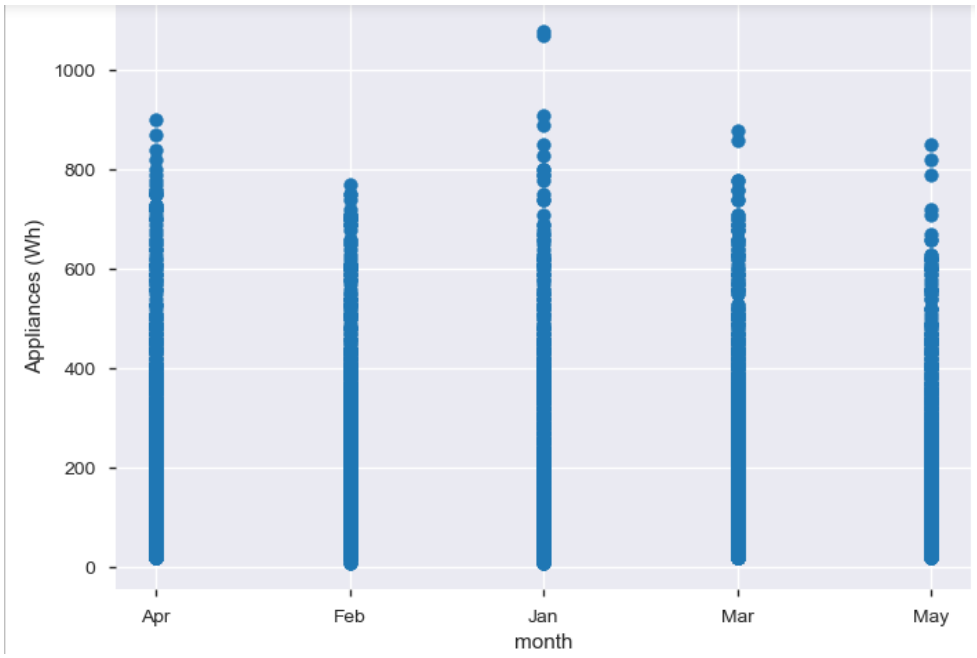


1. The maximum observation made in Appliances variables is between 0 - 100 whs.

2.There are very few observations that is greater than 500 Whs. This might be the outliers too which will be discussed later.

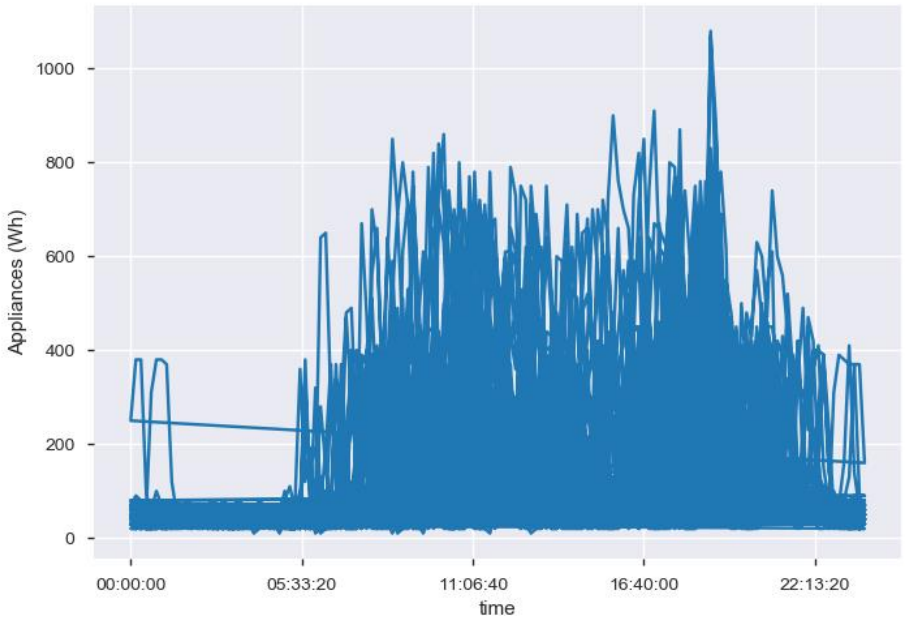


1. None of the variables seem to have a linear relationship with appliances. So it indicates that it's a combination of variables that is affecting the appliances.



1. This graph is Appliances vs. Usage of appliances each month.

2. It indicates that Jan has used the maximum energy at some point of time which is greater than 1000 Whatts.

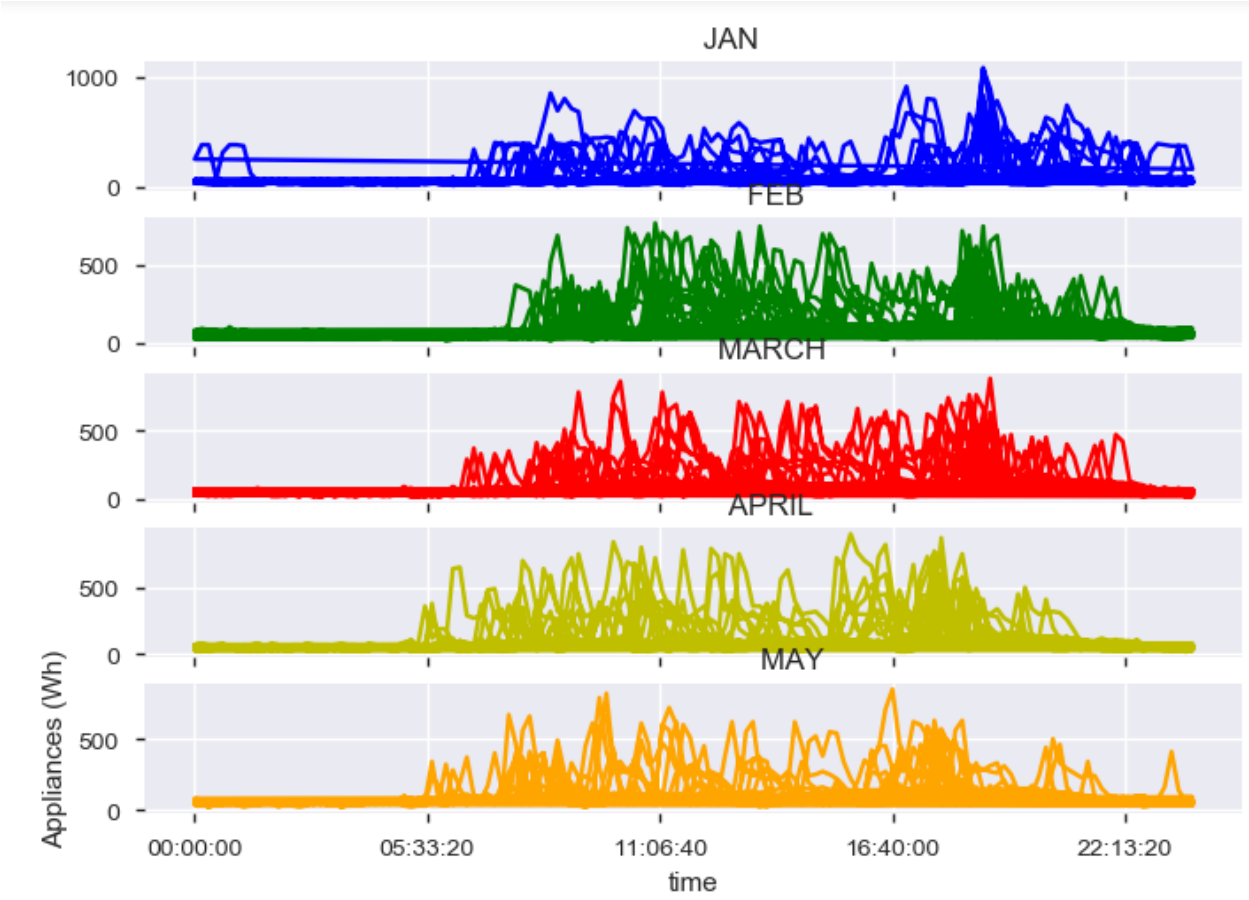


1. This graph is plotted for Appliances vs. time.

2. This tells us when the appliances was used maximum. From the graph we can see that its somewhere between 7pm to 8pm.

3. As we saw earlier there are anomalies in the dataset , this hike could be due to that too.

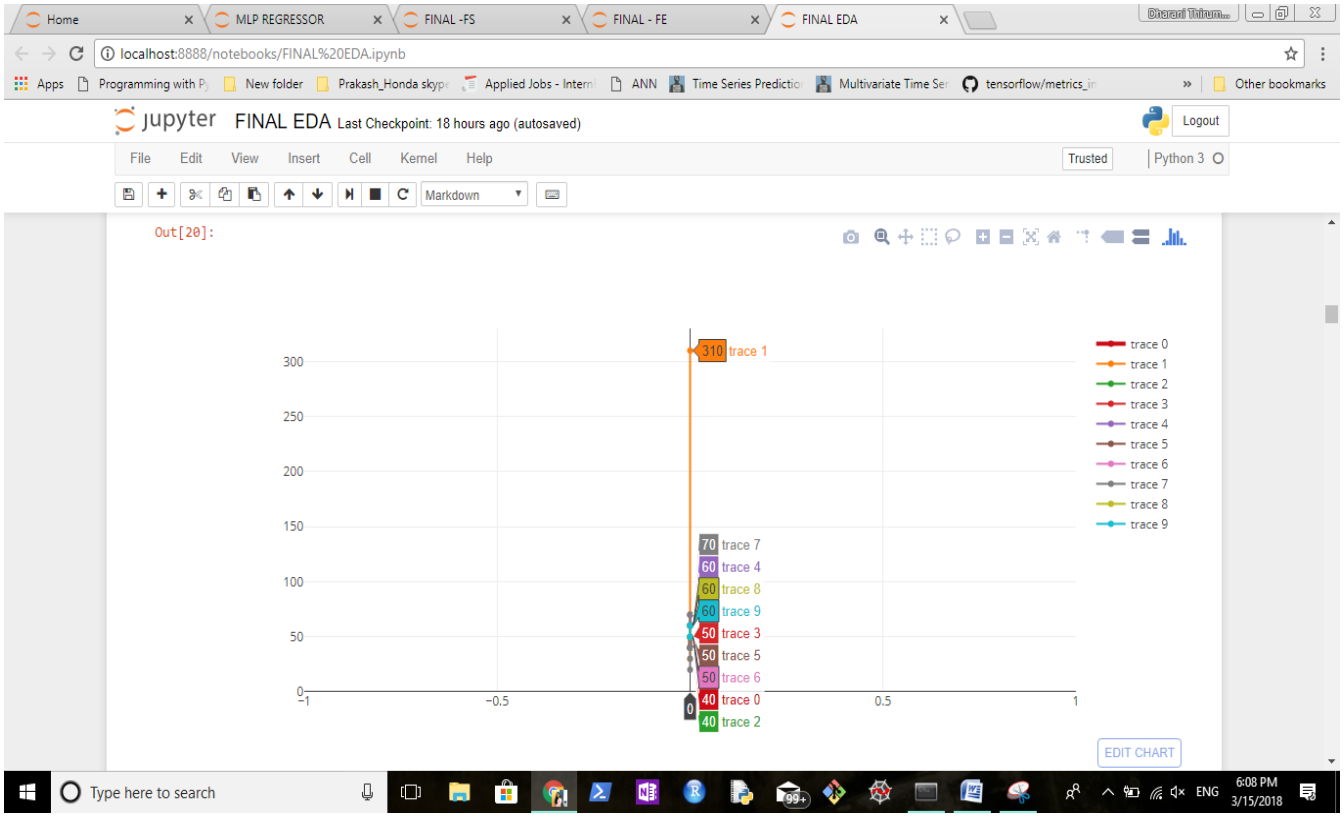
4. So , in order to analyze if it is because of that , we will plot it for each month and check.



1. This graph is plotted each month appliances usage with respect to time.

2. We can see that there is a hike in the appliances usage between 7pm to 8pm in all the months except for the month of May.

3. Between 12 midnight to 5:30 am , all the month except for Jan has constant reading.



1.This plot is plotted for appliances with respect to time(between 12 midnight to 5:30 am) for different week for all months.

trace 0 , 1 - Jan

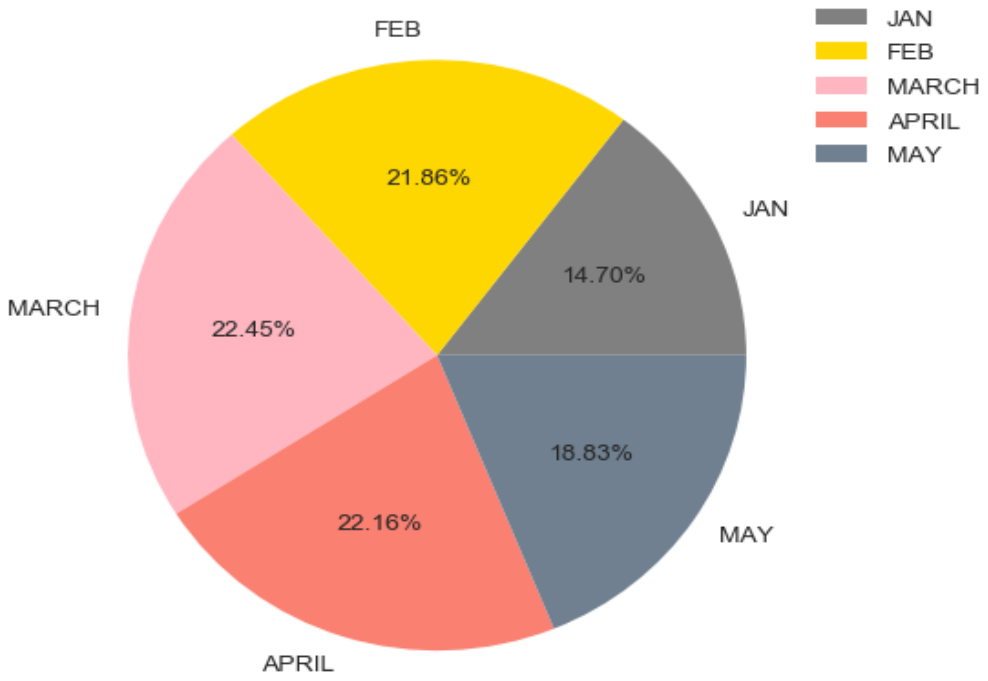
trace 2, 3 - Feb

trace 4, 5 - March

trace 6, 7 - April

trace 8, 9 - May

We can see that all the months except for Jan has values less than 100 wh. Only Jan is showing a high value which is the reason behind that trend in the previous graph.



1.This is a pie chart which says which month has contributed how much to the energy consumption.

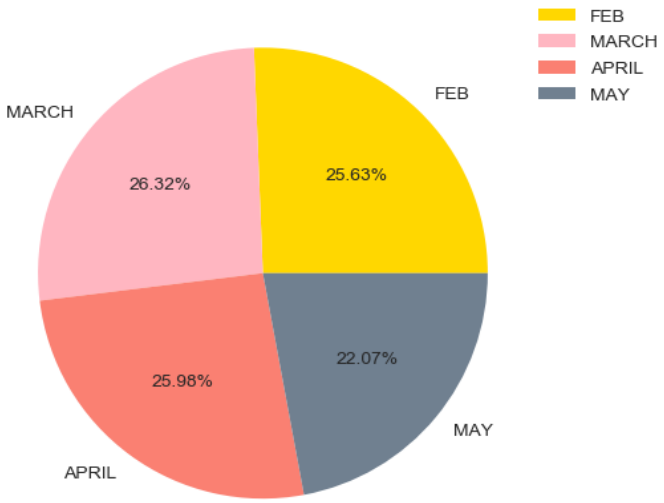
2. The total appliances energy is summed up and appliances energy of each month is summed up and the ratio is found.

3. We can see that March has consumed more compared to other months.

March >April>February>May>Jan

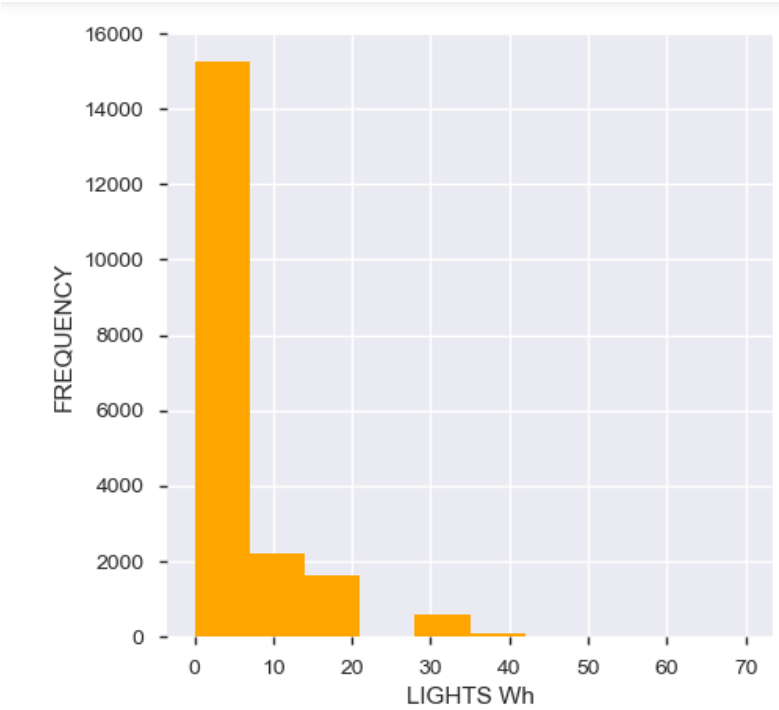
4. One disadvantage about this pie chart is that , for the month of Jan we have the data only from 11th Jan 5:00:00 pm. So this might be a reason to why the use in the month of Jan is very less.

5. Even if the missed out 1 week data is given for the month of Jan , there is a very less possibility that it be more than March. So , we can conclude that March has used maximum energy in Whatts.



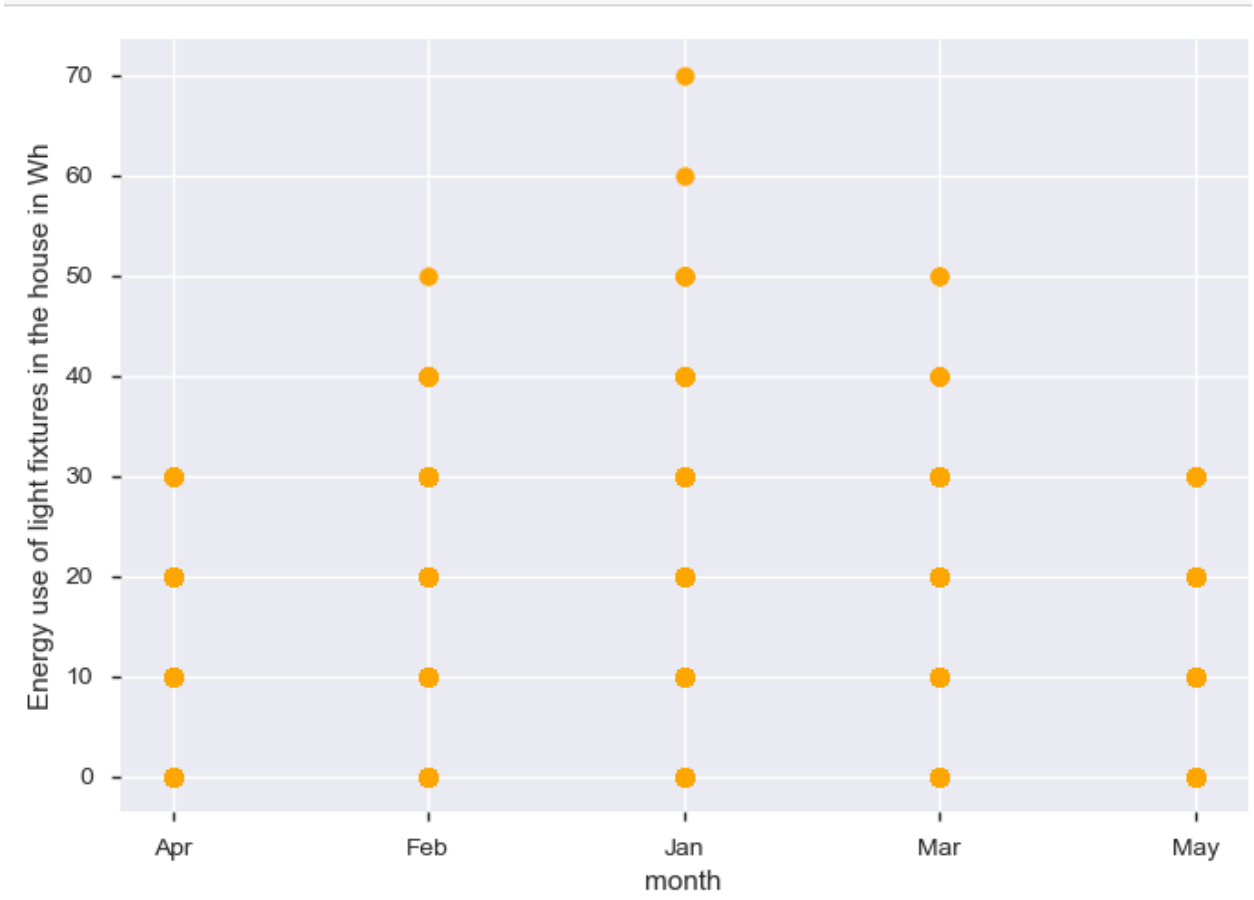
1. This pie chart is without the appliances usage in May

Analysis 3 : Analyzing 'LIGHTS'



1. We can see that most of the light fixtures have consumed 0 Whatts , which implies that they haven't used lights.

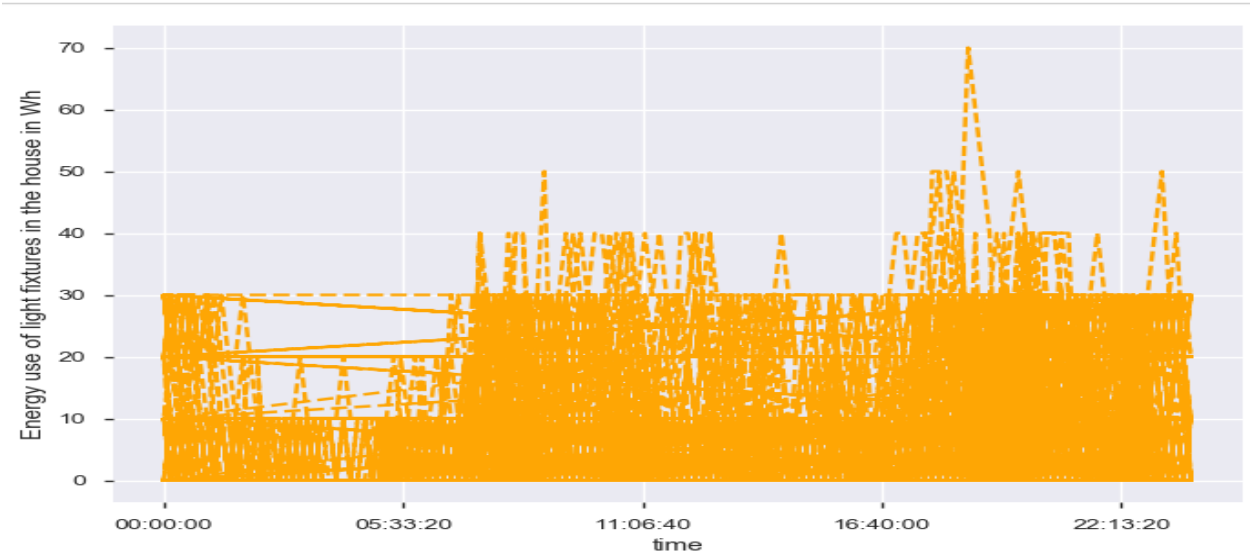
2. Very few days have light fixtures that have consumed 30 to 50 Whatts which can be treated as outliers.



1. Jan has measured the highest value of lights followed by Feb , March and April and May.

2. The hike in the month of Jan can also be due to the outliers.

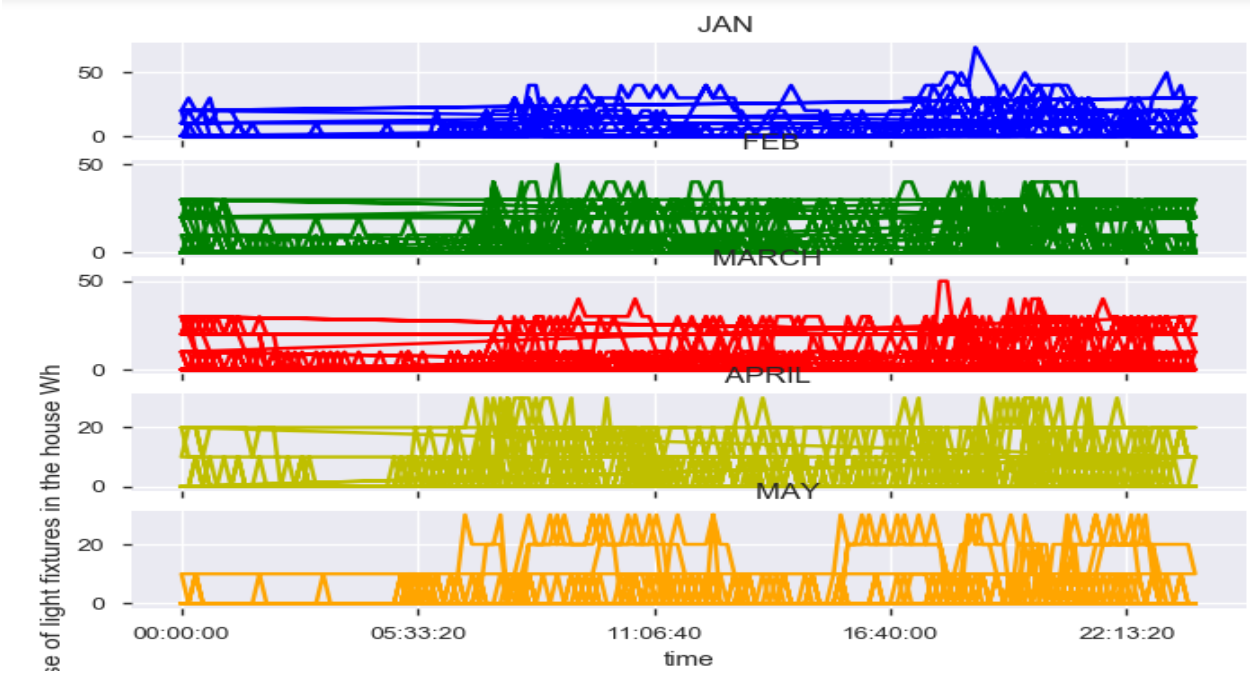
3. So ,we can see that outliers help us in concluding few important factors so it can't be removed at a later stage.



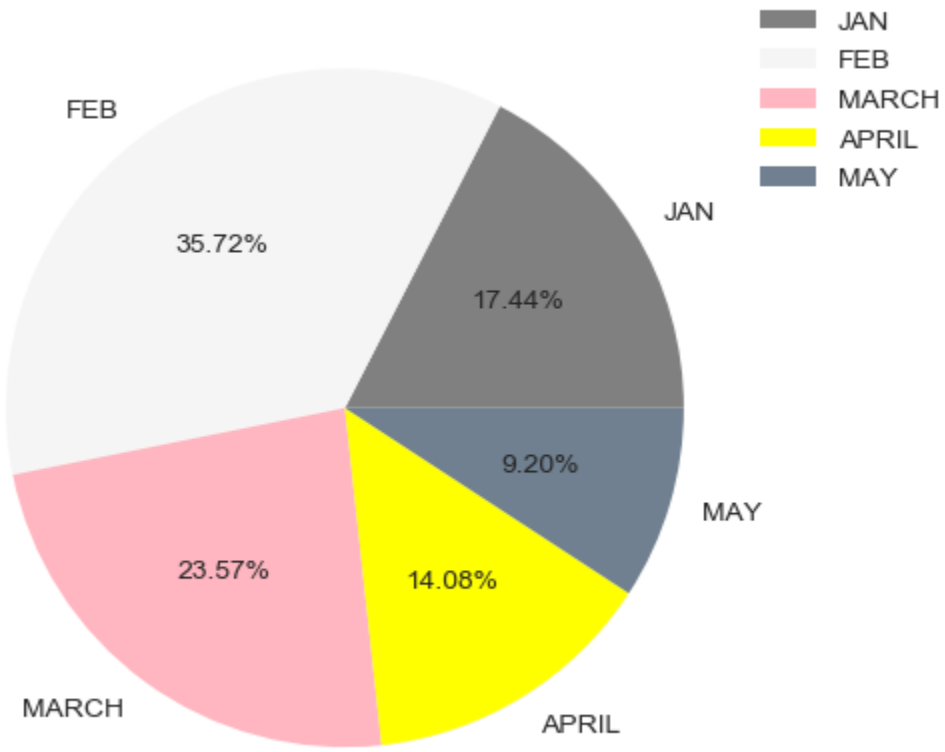
1. This is Light fixtures vs. time.

2. This tells us that there is a hike in the usage of lights between 7pm to 8pm which is similar to the hike in appliances.

3. We need to check which all months shows the same trend in the graph.



1. There is a hike in the light fixture usage between 7pm to 8pm for every month but the maximum is seen in the month of Jan which means that Jan has recorded the highest value for light fixture energy consumption.



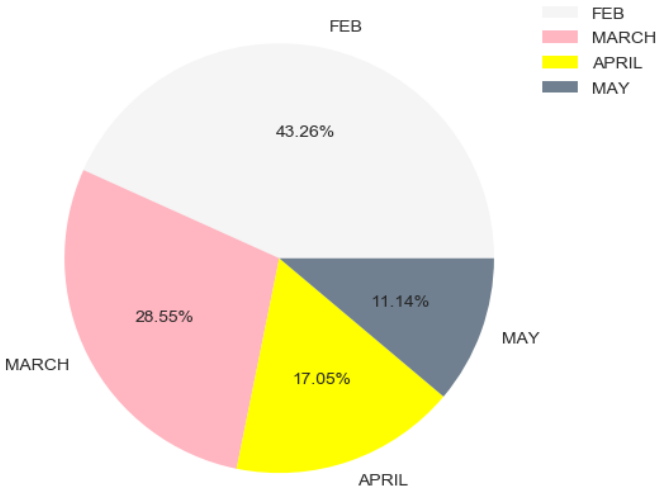
1. Similar to what we saw in appliances , this is for lights. Again , March has high contribution after February.

February > March > Jan > April > May.

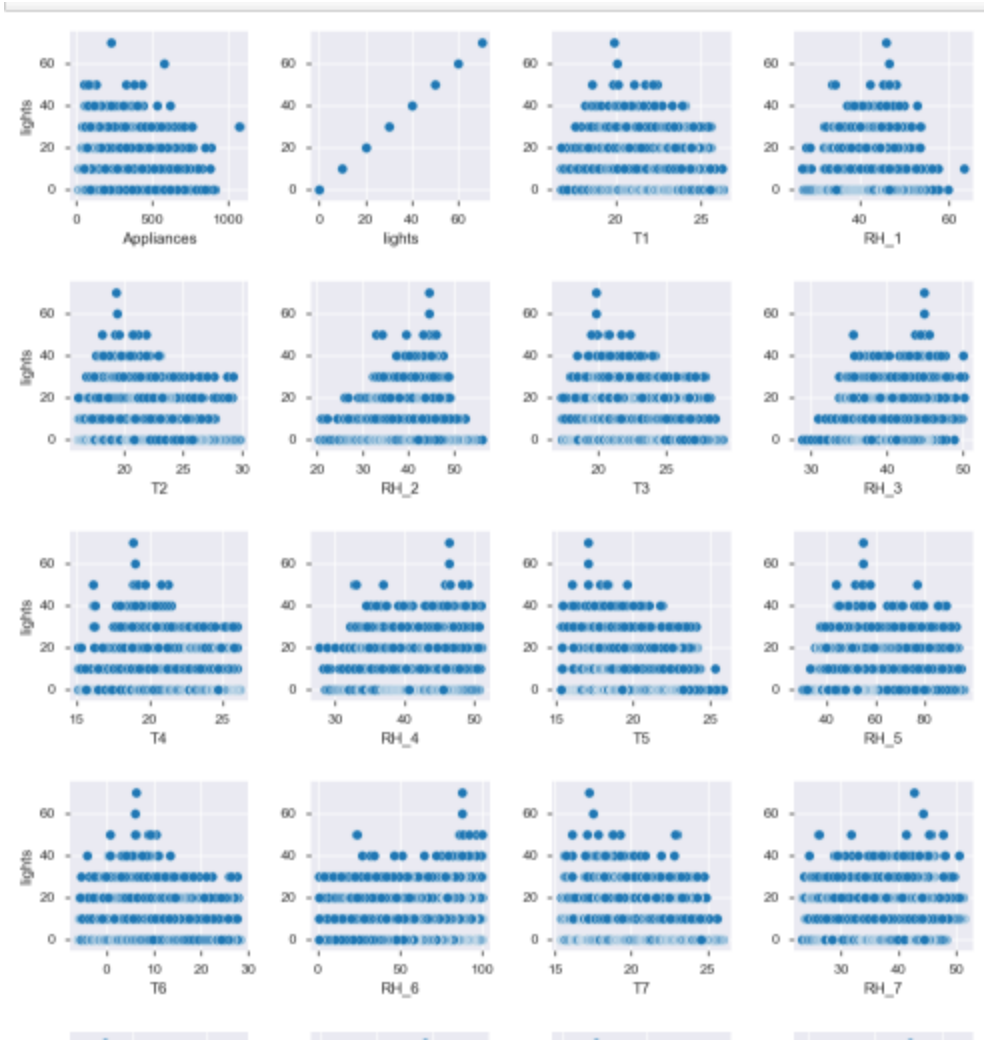
2. Here without the first week data , Jan has contributed approximately 18% to the overall light consumption. So if the first week data has been provided , there is a chance that it might have been equal to March or greater than that.

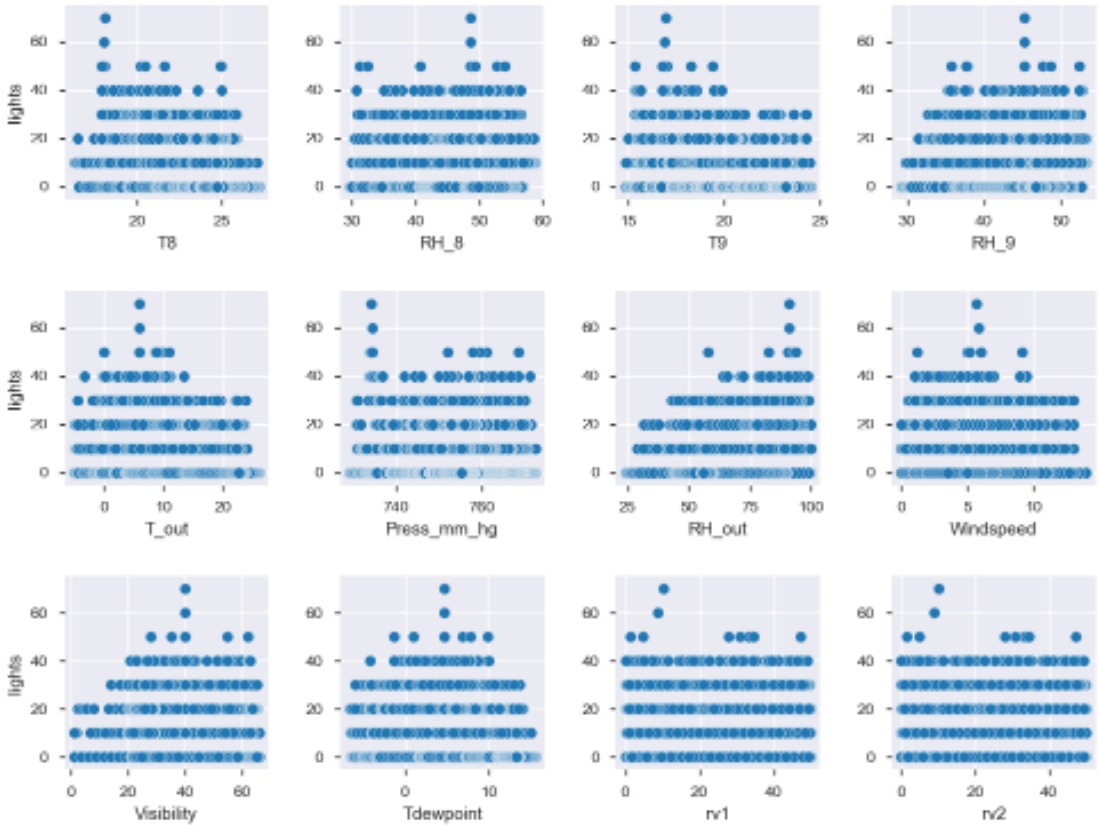
3. Comparing Appliances and lights pie chart , it looks like March has consumed a lot of energy compared to other months and May has consumed the least.

4. This could be due to extreme winter in March and Pleasant climate in May. (Just a thought)



1. This is without including the Jan month. Result remains the same.



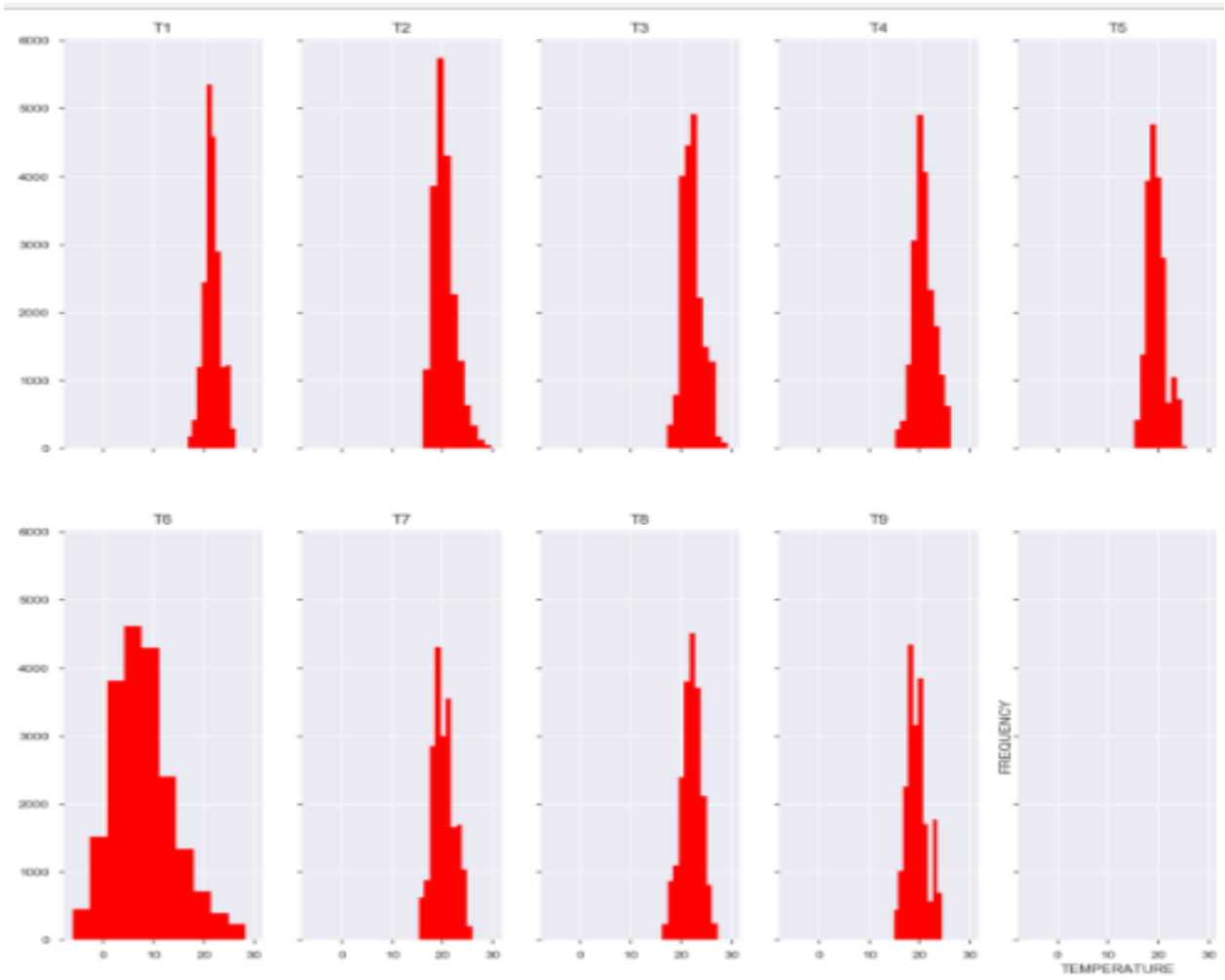


1. This is light vs. all other variables.

2. It doesn't provide any information except that no single variable individually contribute to the usage of light fixtures.

Analysis 4 : Analyzing 'TEMPERATURE' (T1 - T6)

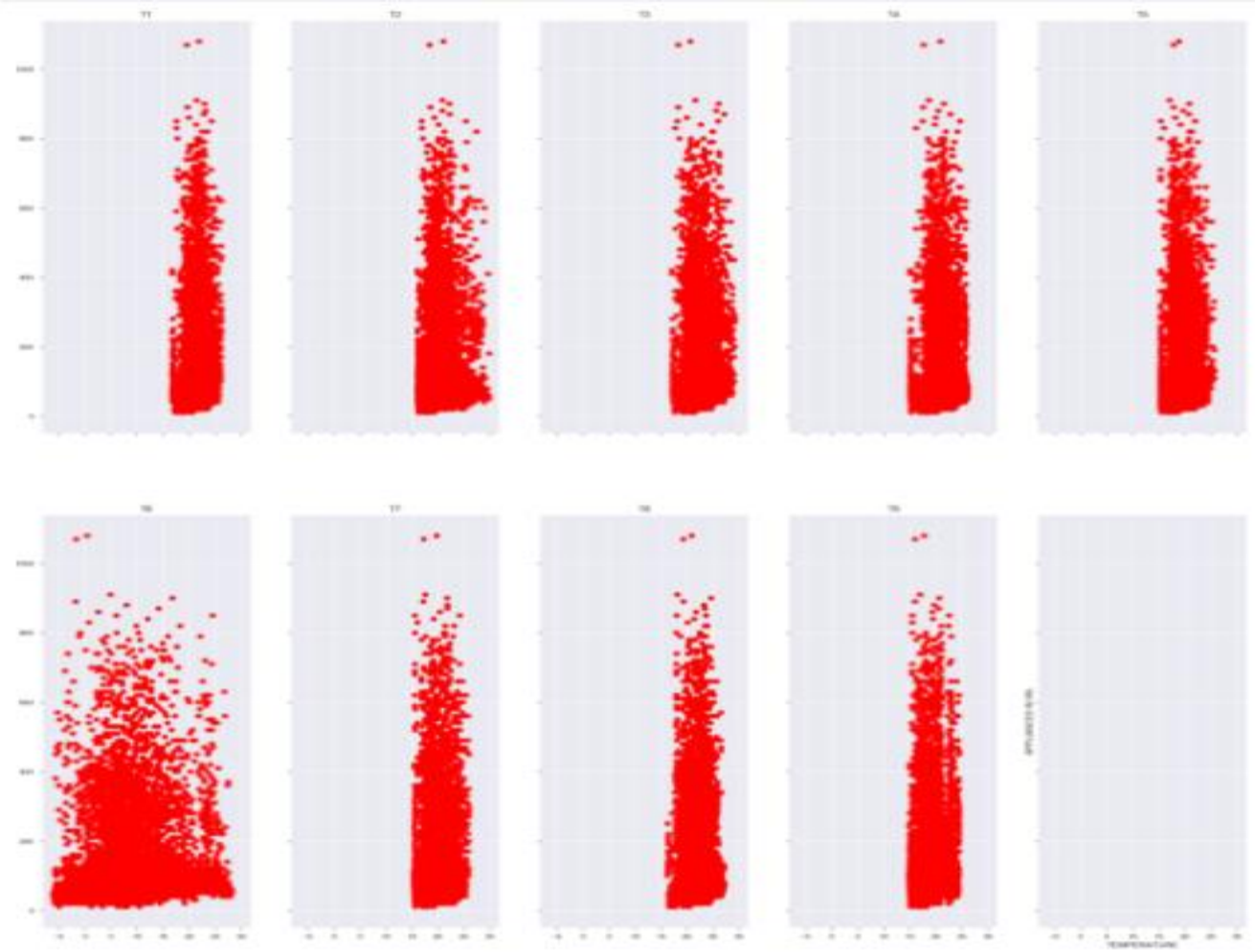
T1, Temperature in kitchen area, in Celsius
T2, Temperature in living room area, in Celsius
T3, Temperature in laundry room area
T4, Temperature in office room, in Celsius
T5, Temperature in bathroom, in Celsius
T6, Temperature outside the building (north side), in Celsius
T7, Temperature in ironing room , in Celsius
T8, Temperature in teenager room 2, in Celsius
T9, Temperature in parents room, in Celsius



1. This is a histogram plotted for all the 9 temperatures.

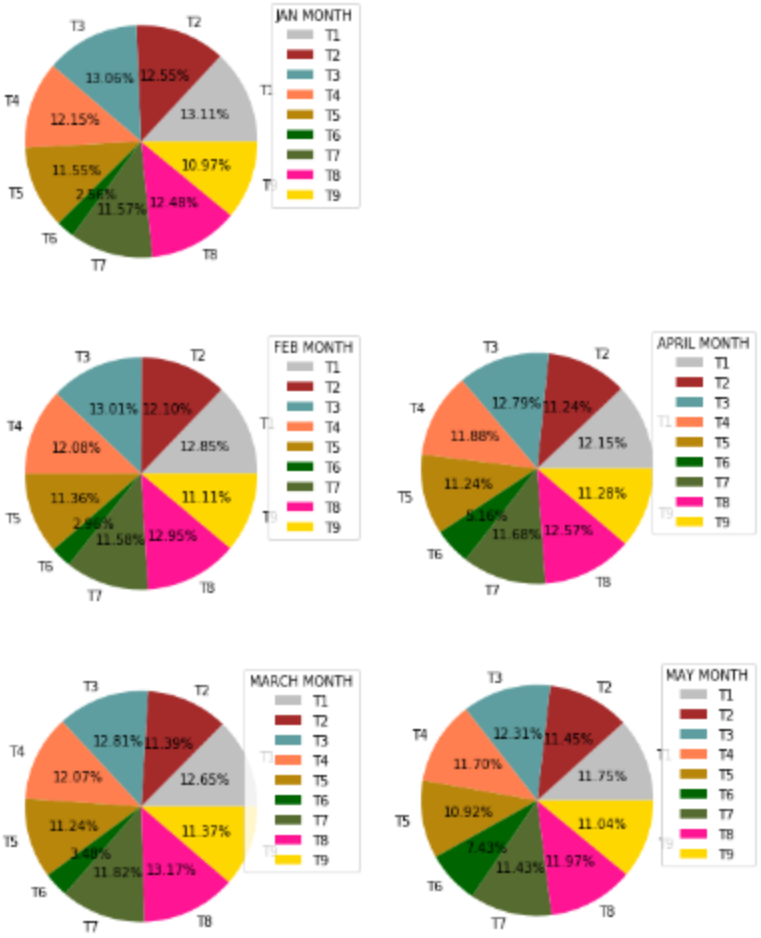
2. This shows us that all the temperature is between 15 degree Celsius to 25 degree Celsius except for T6 which has value in negative.

3. Kitchen(T1) and Living Room(T2) is showing a hike for the same temperature compared to other rooms.



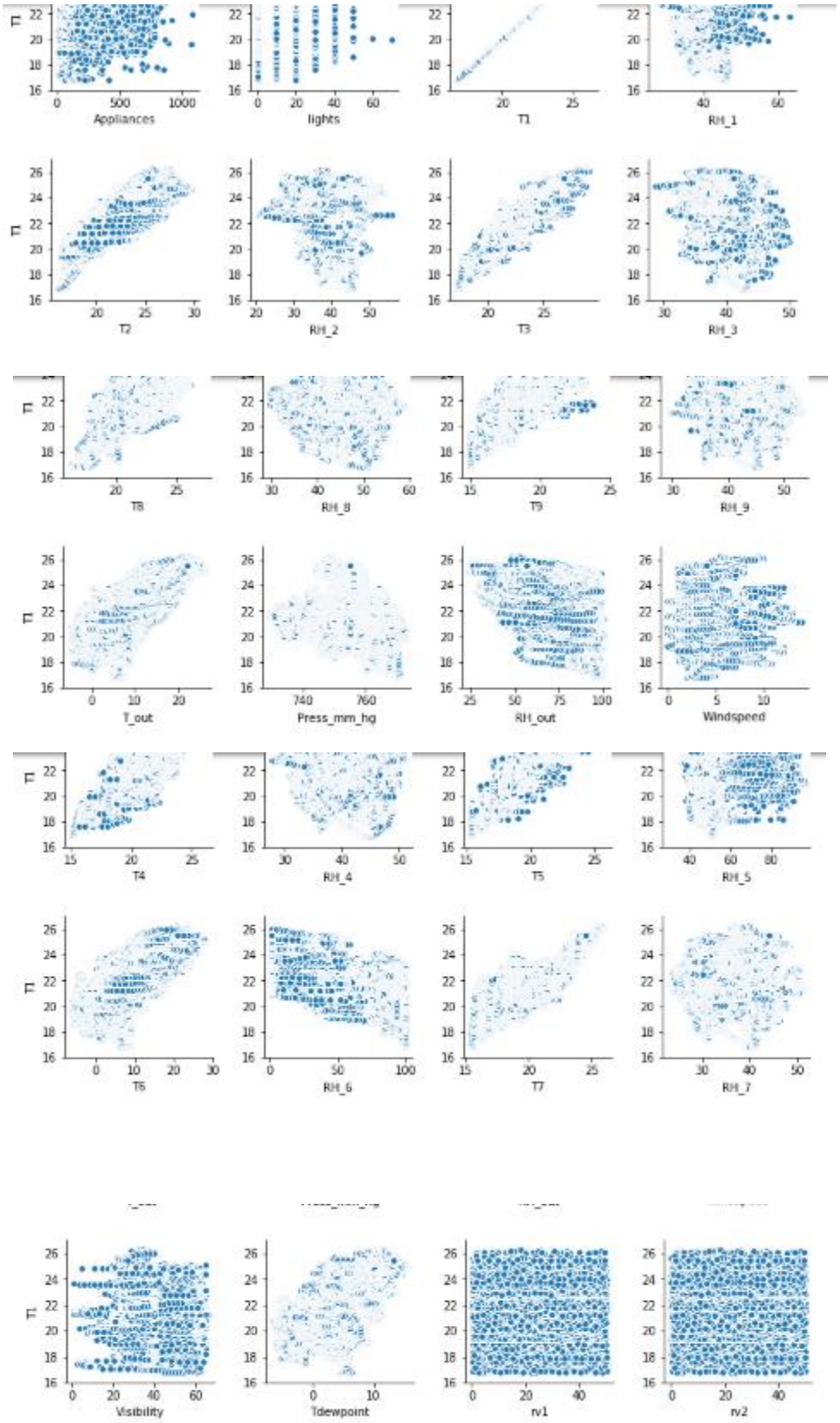
1. This is temperatures vs. Appliances.

2. Again T6 is showing a different trend form others.



1.Out of all ,T6 has contributed less to the overall temperature of the house in every month which means its correlation with appliances should be zero or close to zero.

2.Out of all the temperatures in the rooms that is present inside the house , T3 has the highest contribution followed by T1 and T8.


T1
Appliances
lights
T1
RH_1
T2
RH_2
T3
RH_3
T8
RH_8
T9
RH_9
T_out
Press_mm_hg
RH_out
Windspeed
T4
RH_4
T5
RH_5
T6
RH_6
T7
RH_7
Visibility
Tdewpoint
rv1
rv2

1. This is temperature (T1) with respect to other variables.

2. All temperatures are linear to each other.

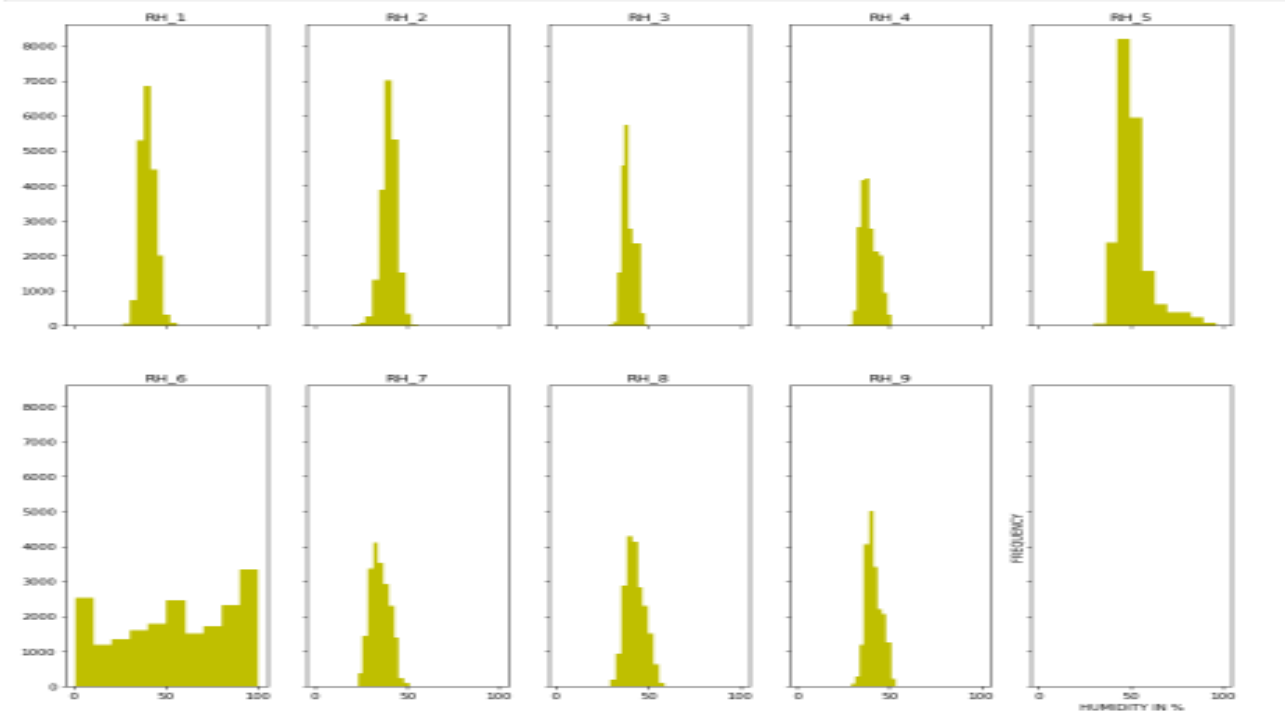
3. All humidity's are inversely proportional to temperature.

4. Tdewpoint is linear related to temperatures.

5. This trend is seen when plotted for all temperatures vs. all other variables.

Analysis 5 : Analyzing 'HUMIDITY'(RH_1 to RH_9)

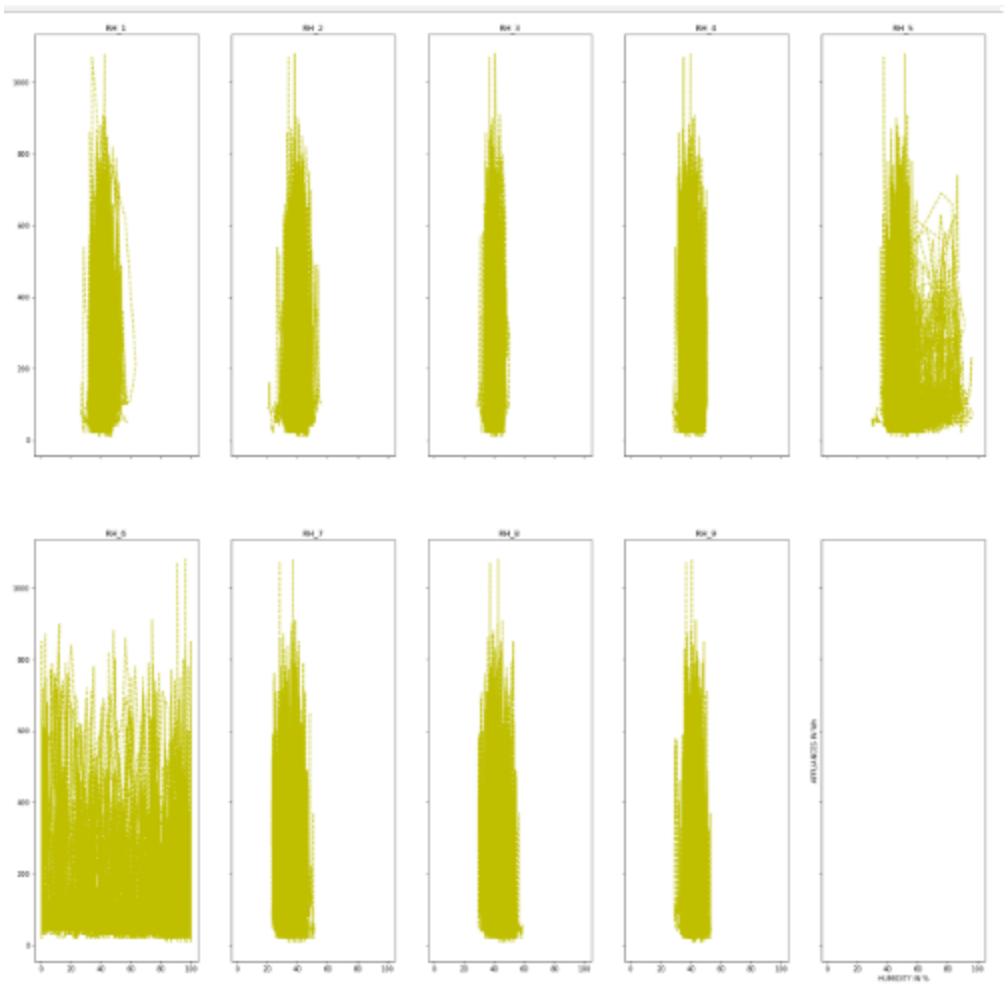
RH_1, Humidity in kitchen area, in %
RH_2, Humidity in living room area, in %
RH_3, Humidity in laundry room area, in %
RH_4, Humidity in office room, in %
RH_5, Humidity in bathroom, in %
RH_6, Humidity outside the building (north side), in %
RH_7, Humidity in ironing room, in %
RH_8, Humidity in teenager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



1. Histogram plotted for all the humidity variable.

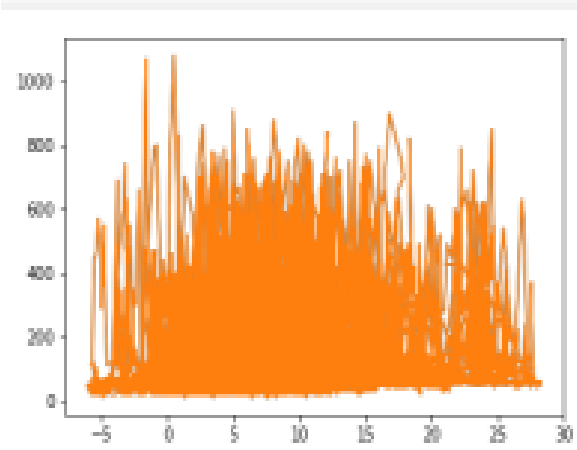
2. A dissimilar trend is seen in Rh_6 .

3. RH_5 has recorded maximum of humidity compared to other rooms.

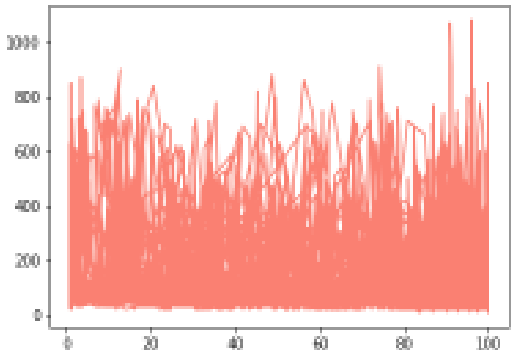


1. This plot is Appliances vs. Humidity in %.

2. Again RH_6 is the odd one out and RH_5 also has a different behavior compared to others. This is because , RH_5 the humidity has recorded more value than other rooms and moreover in RH_5 we can see that as humidity increases there is a decrease in the appliances energy consumption.



(1)



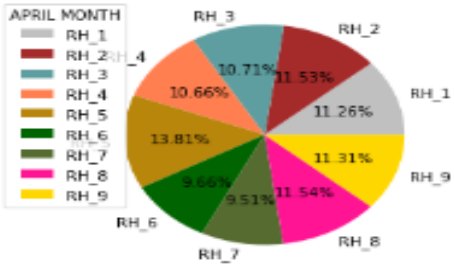
(2)

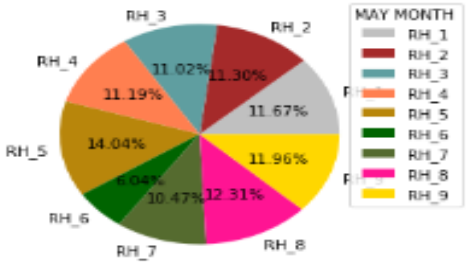
(1) - Appliances vs. T6

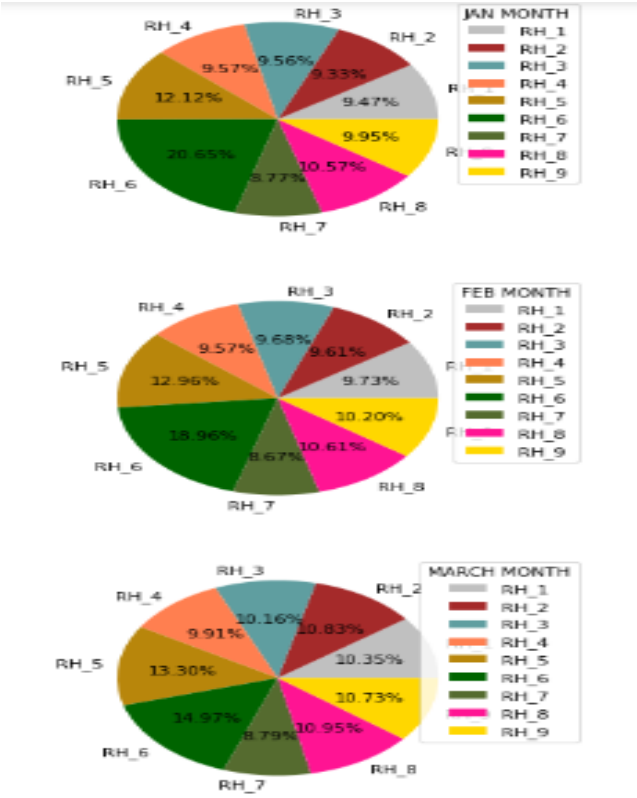
(2) - Appliances vs. RH_6

1. Comparing these two graphs , we can see that when there is a lower temperature , there is a hike in humidity and vice versa.

2. That implies , temperature is inversely proportional to temperature.

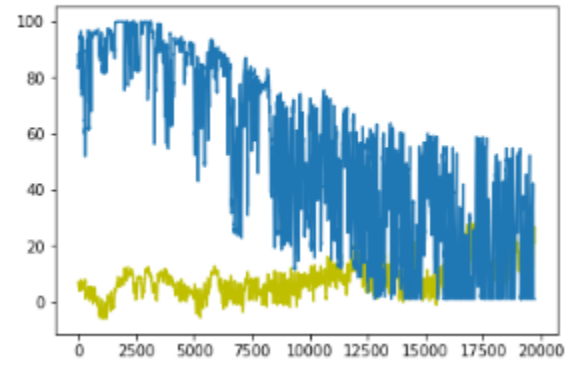


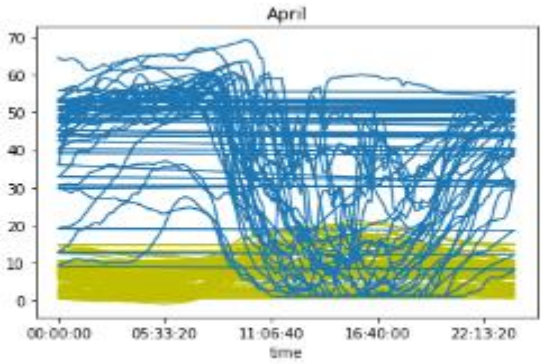




From the above pie chart , it is seen that RH_6 has the highest value in all months except for May. In all other months , the Temperature is low and the humidity is high , whereas in month of May , temperature is high so the humidity is less.

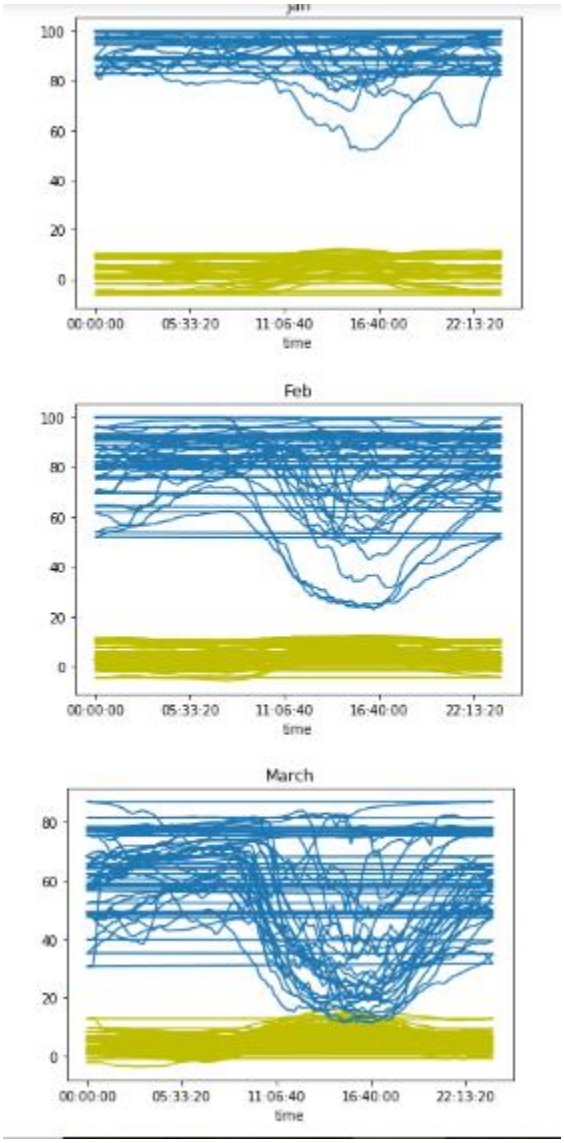
```
plt.plot(df.T6 , color = 'y')
plt.plot(df.RH_6)
plt.show()
```





1. Compared to other months , May and April has less humidity contribution , this is because there is a point where the humidity and temperature are more or less the same.

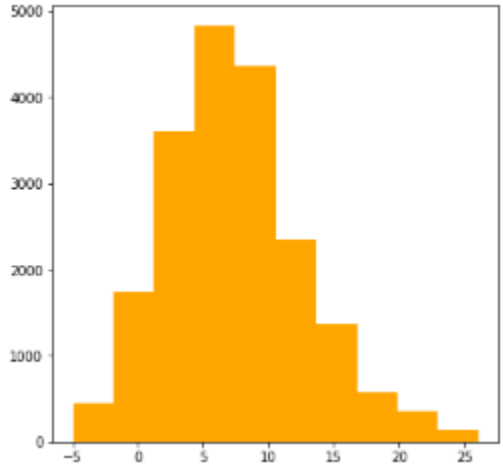
2. But in case of other months ,



When temperature is low , the humidity is very high and as the temperature increases the humidity also decreases.

When the temperature and humidity has high value difference , the graph is far apart but when the values come closer the graph collides like in the case of May and April which says that as temperature increases , there is a high decrease in the humidity unlike the other months.

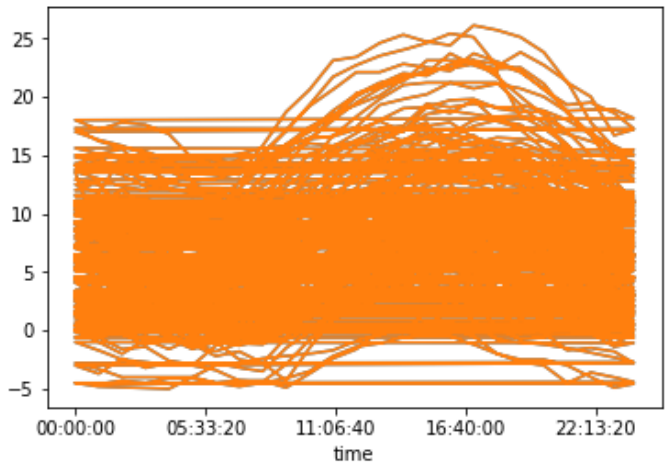
Analysis 6 : Analysis of 'T_out'-Temperature outside (from Chièvres weather station)



1. Histogram for T_out.

2. There are negative temperatures too.

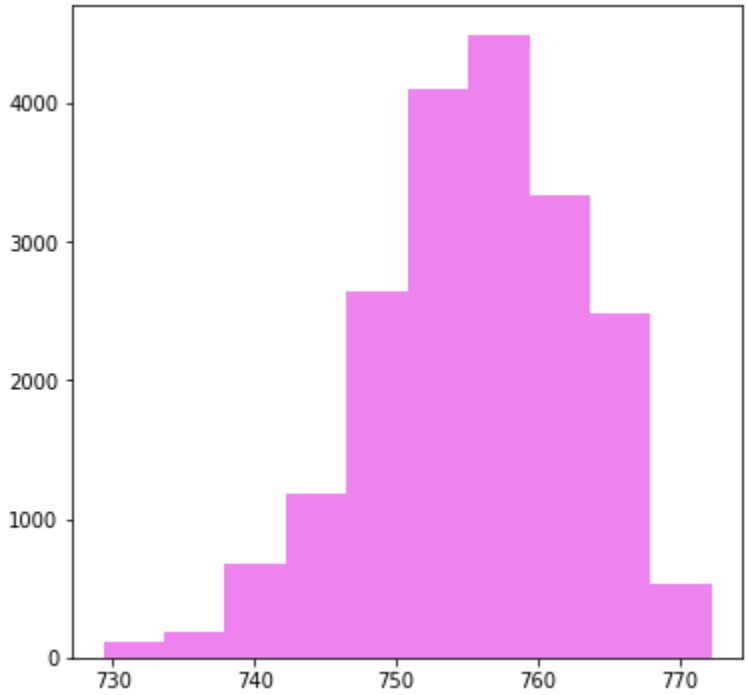
3. Maximum recorded temperature is between 5 degrees Celsius to 10 degree Celsius.



1. Time vs. T_out.

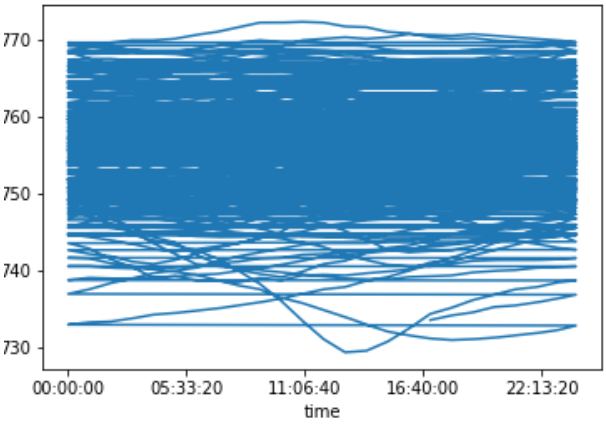
2.From the above plot , we can see that this has a similar trend like appliances . So there is a possibility for it to be positively related to it.

Analysis 7 : Analysis of 'PRESSURE' (from Chièvres weather station), in mm Hg

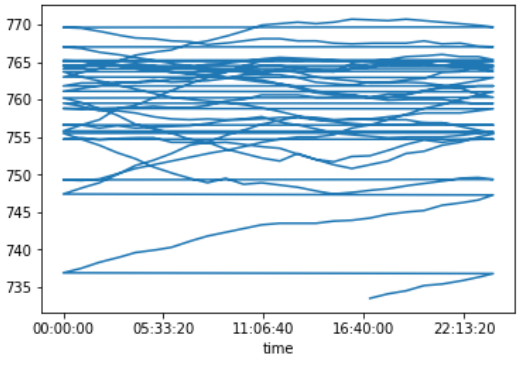


1. Histogram for Press_mm_hg.

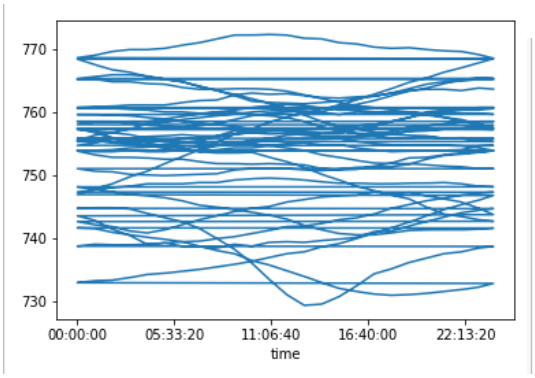
2. Maximum recorded pressure is between 755 mmHg to 760 mmHg.

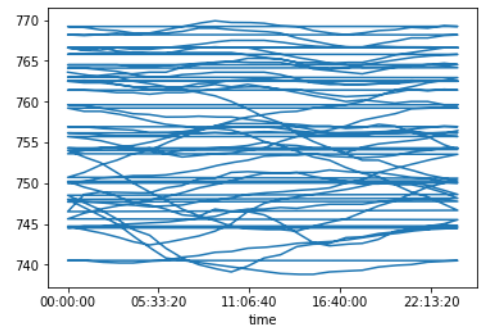


(1) - Overall T_out vs.Time

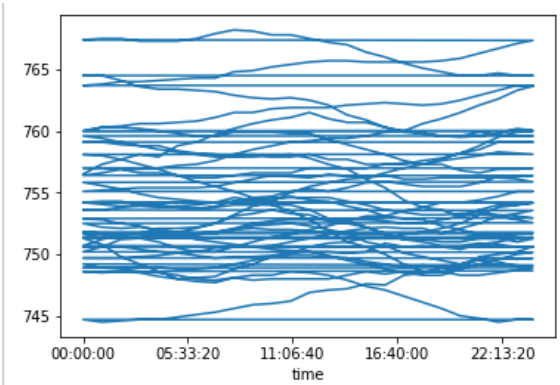


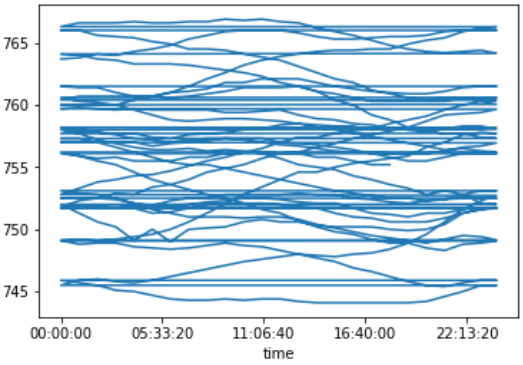
(2) Jan T_out vs. Time





(3) Feb T_out vs. Time (4) March T_out vs Time

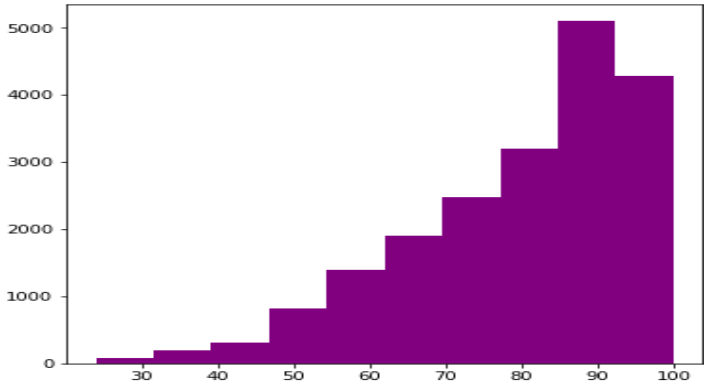




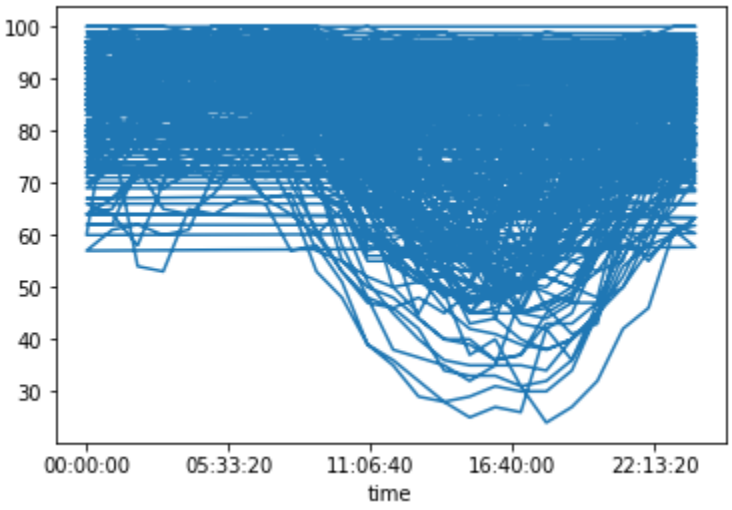
(5) April T_out vs. Time(6) May T_out vs. Time

1. There is no trend shown by this , so there is a chance that it is not correlated to appliances.

Analysis 8 : Analysis of RH_out, Humidity outside (from Chièvres weather station), in %

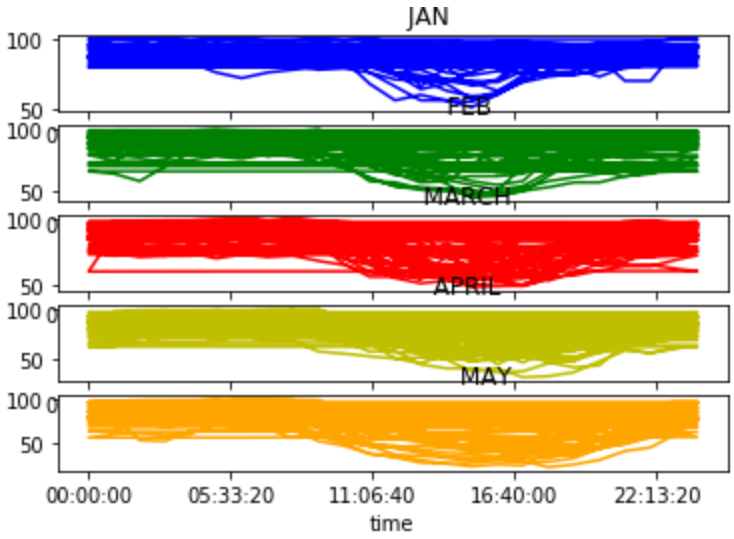


1. Maximum humidity recorded outside is 90% to 100%.

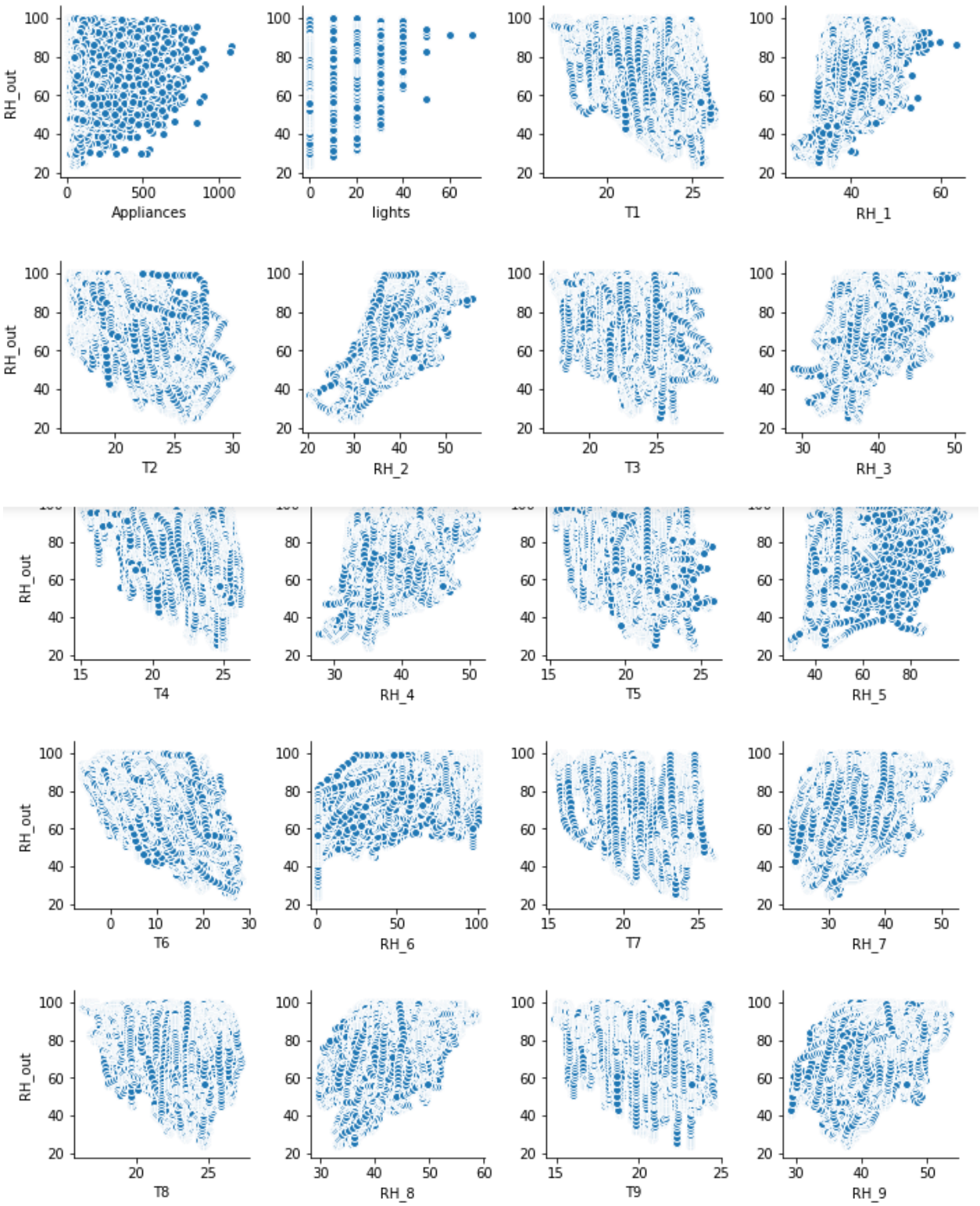


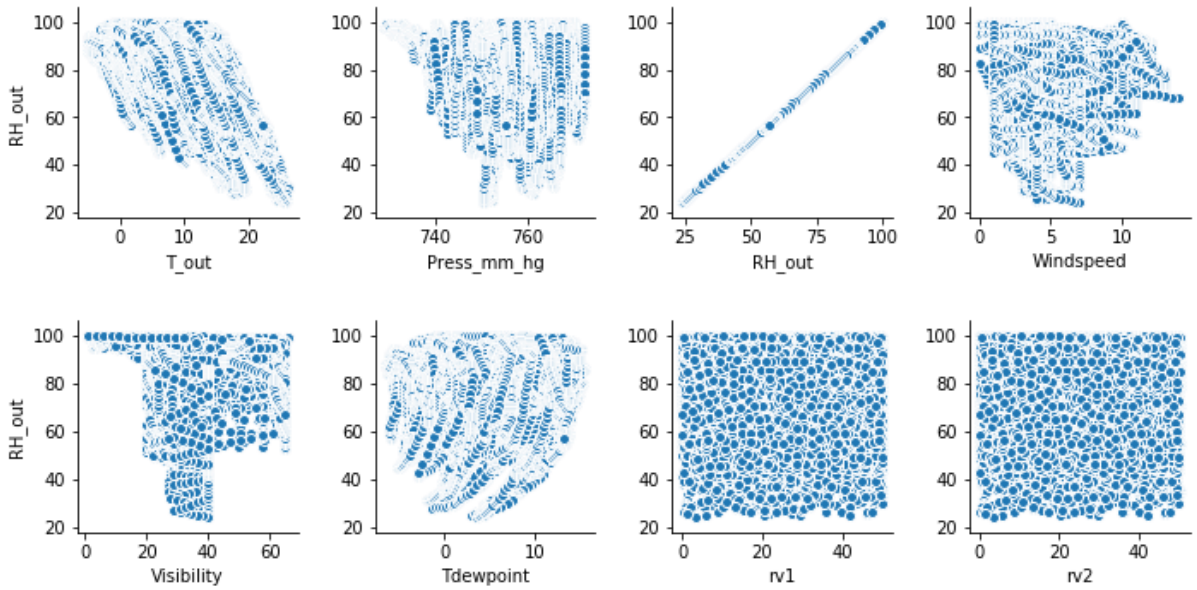
1. RH_out vs. . Time.

2. When we compare this plot to T_out vs. Time plot , we can see that there is a increase in the temperature between 6am to 11:30 pm which is seen as a decrease in the RH_out.



1.Every month shows a decrease in RH_out during that time period , so it is possible that T_out increase during that time and also appliances have maximum energy usage during this time interval.

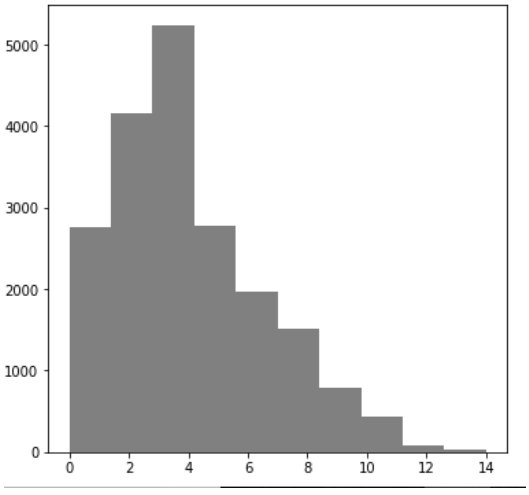

RH_out
Appliances
lights
T1
RH_1
T2
RH_2
T3
RH_3
T4
RH_4
T5
RH_5
T6
RH_6
T7
RH_7
T8
RH_8
T9
RH_9



1. This is RH_out with other variables

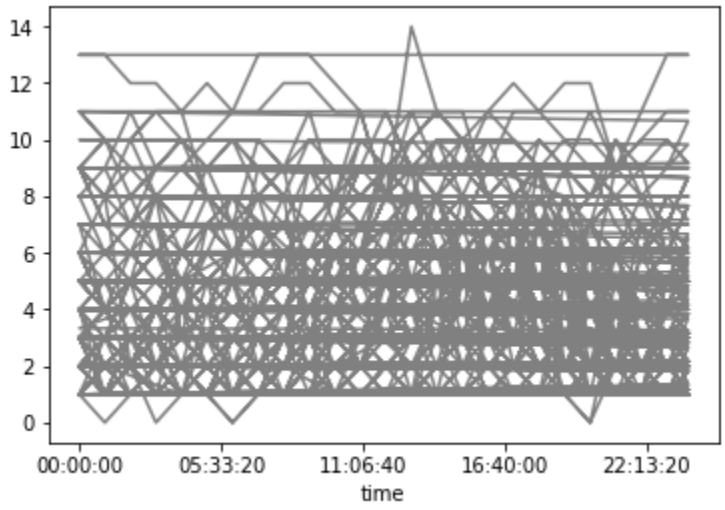
2. From this pairplot , we can say that humidity's are positively correlated to one another and negatively correlated to temperatures.

Analysis 9 : Analysis of 'WINDSPEED' in m/s



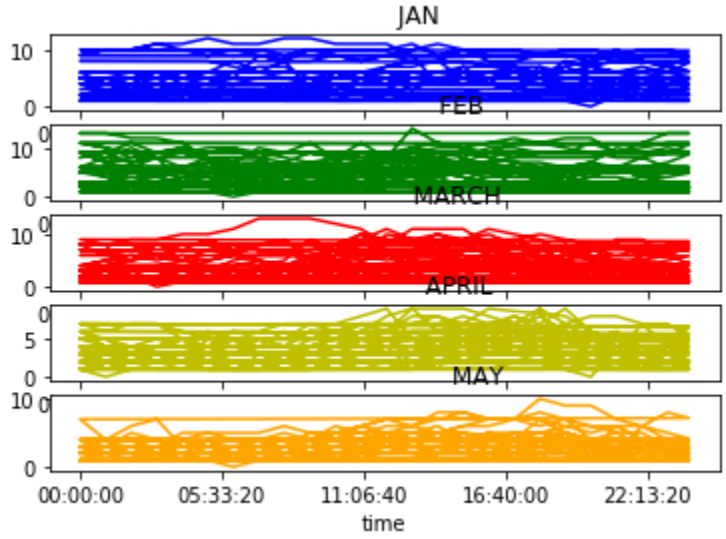
1. Histogram for wind speed.

2. Most of the wind speed in Km has been recorded between 2 m/s to 4m/s.



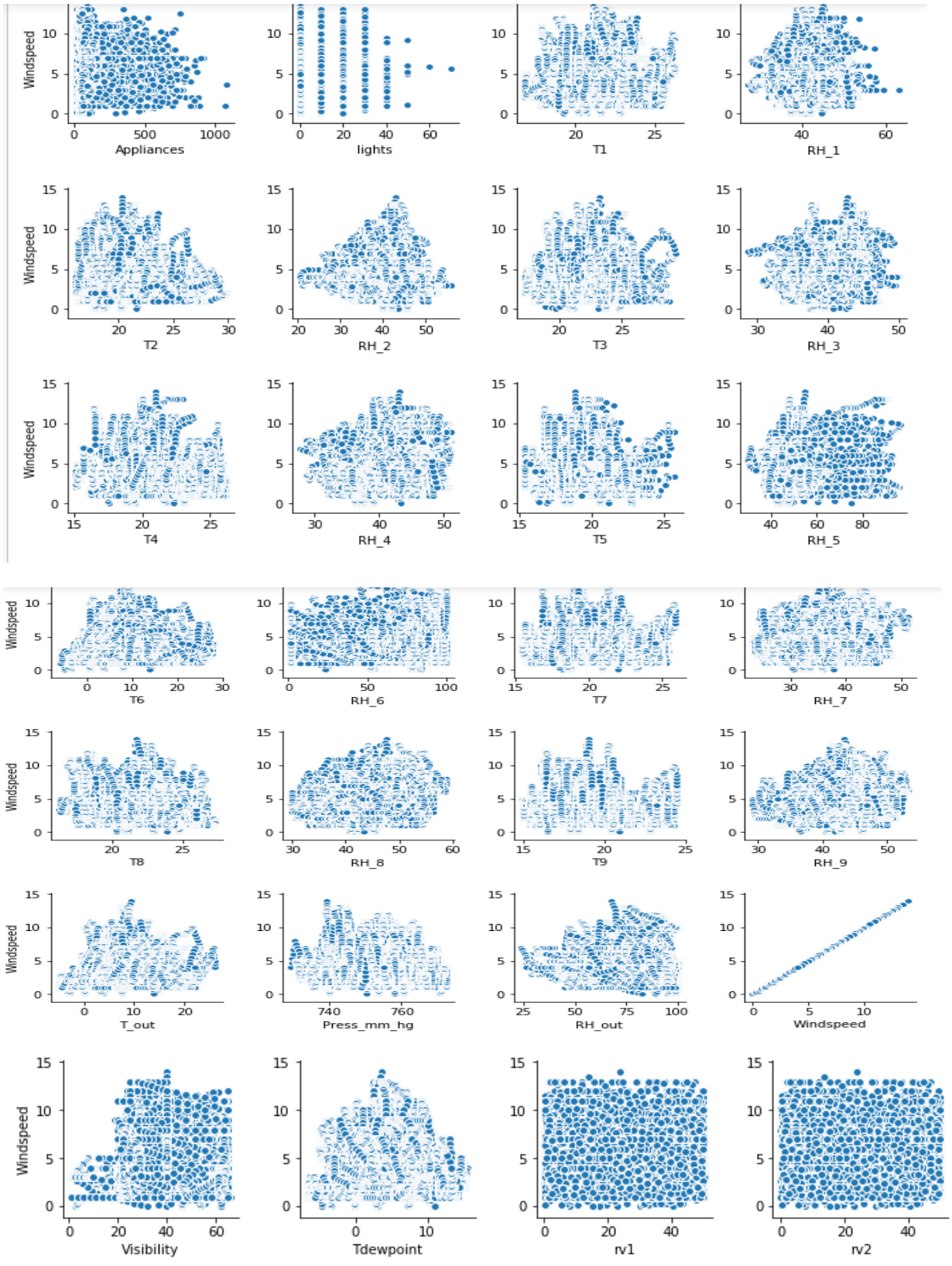
1. Windspeed vs.Time

2. No conclusion can be drawn from this as it has no trend which is similar to trend showed by any variable. There is a chance that windspeed along with someother variable influences appliances.



1. Windspeed of each month vs. Time.

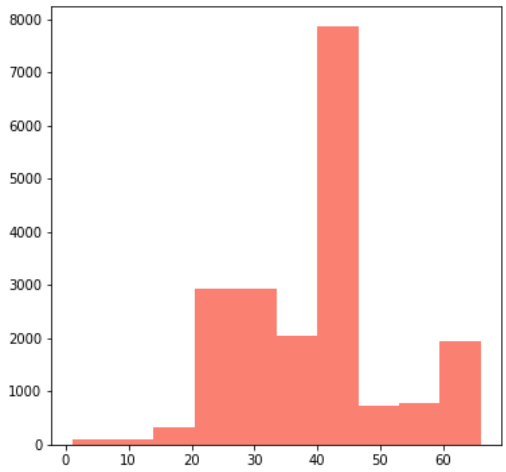
2. Constant value over time for all the months.


Windspeed
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

1. This is a plot for windspeed with respect to all other variables.

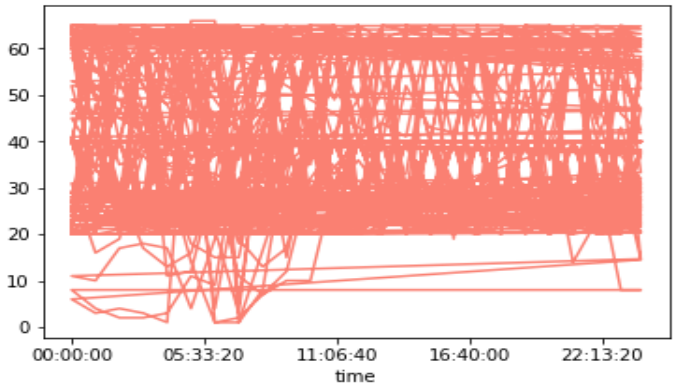
2. This says that windspeed is not related to any variable.

Analysis 10 : Analysis of 'VISIBILITY' in Km



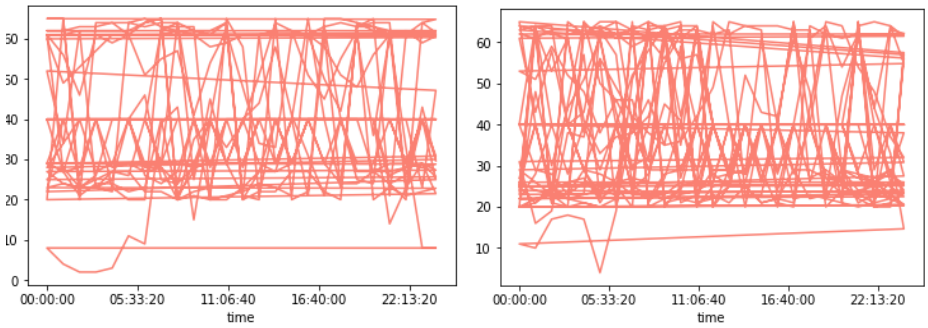
1. Histogram for visibility.

2. Shows us that maximum visibility was 40 km to 50 km.

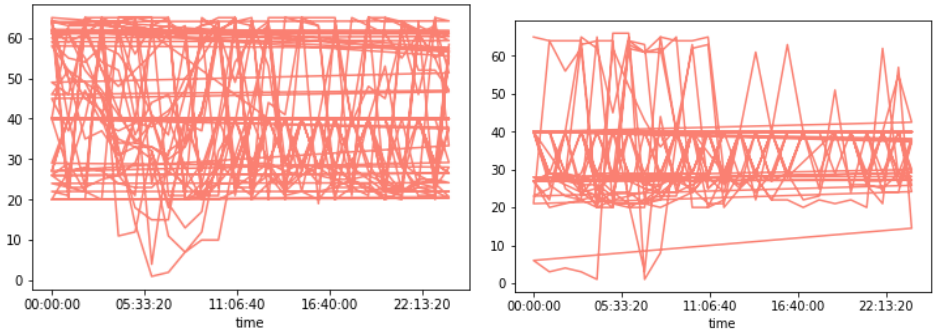


1. Visibility vs. Time.

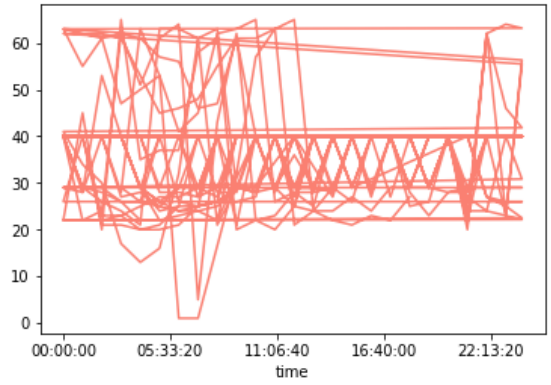
2.There is a constant trend in the visibility overall , but between 12 midnight to 5:30 am there is a decrease in few datapoints



(1) Jan visibility vs. Time(2) Feb visibility vs. Time



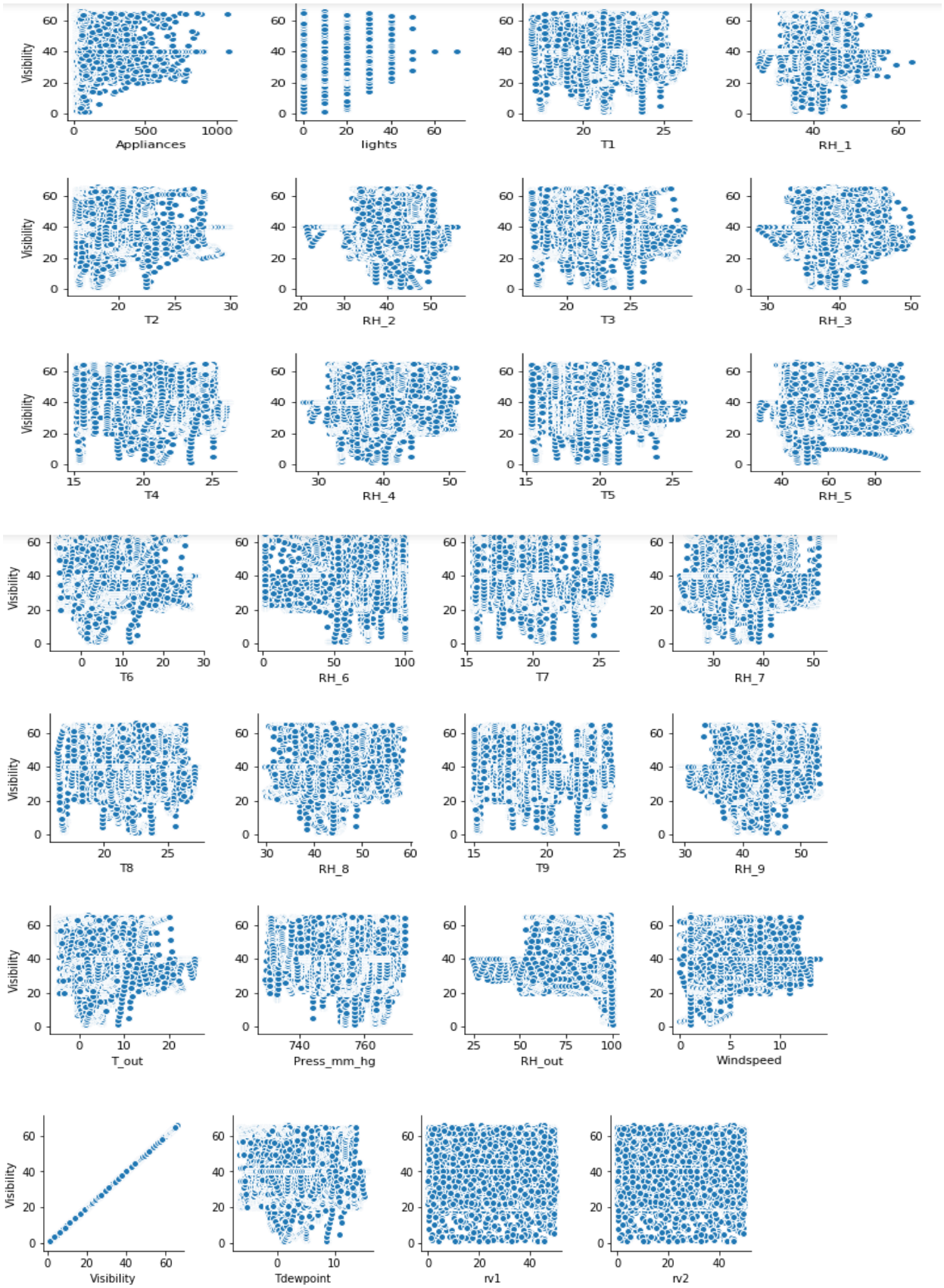
(3) March visibility vs. Time (4) April visibility vs. Time



(5) May visibility vs Time

1.From all the above plots , we can see that there is a decrease in the visibility between 12 pm midnight to 5:30 am.

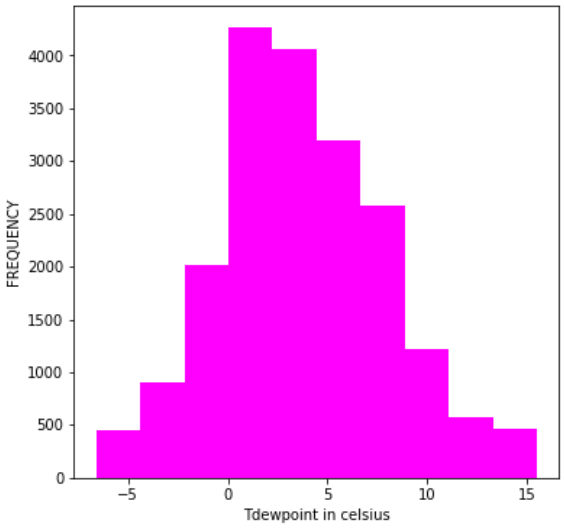
2. This decrease in visibility may be due to many reasons combined , since none of the other variables have shown any increase or decrease between this time interval.


Visibility
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

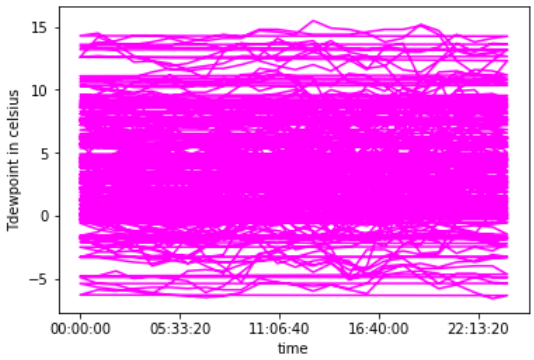
1. These plots are between visibility vs. all other variables.

2. From these plots only humidity and temperatures are somewhat related to its non-linearly. It could be due to this relation that there is a decrease in the visibility between 12pm to 5:30 am .

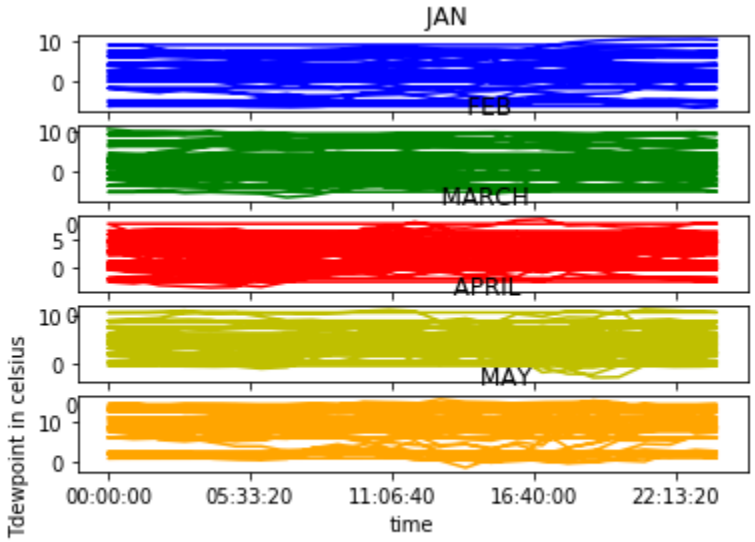
Analysis 11 : Analysis of 'Tdewpoint' (from Chièvres weather station), °C



1.Dew point is mostly positive and the maximum observation is made between 0 degrees to 5 degrees.



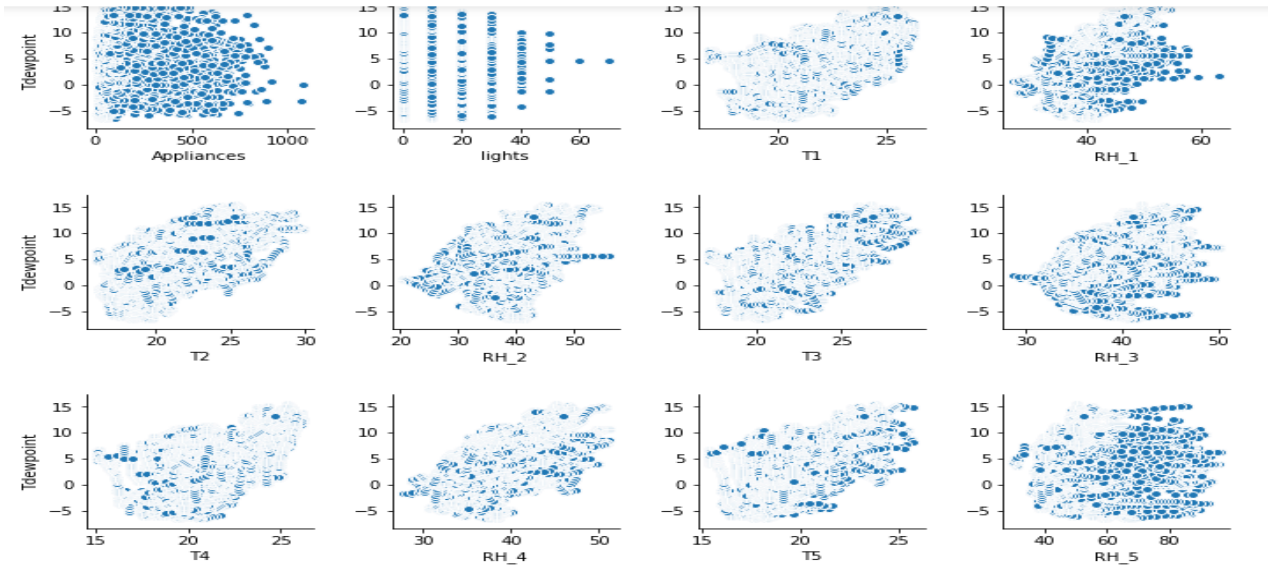
1.Looks like dew point doesn't change with time which implies that its effect on appliances energy should be less.

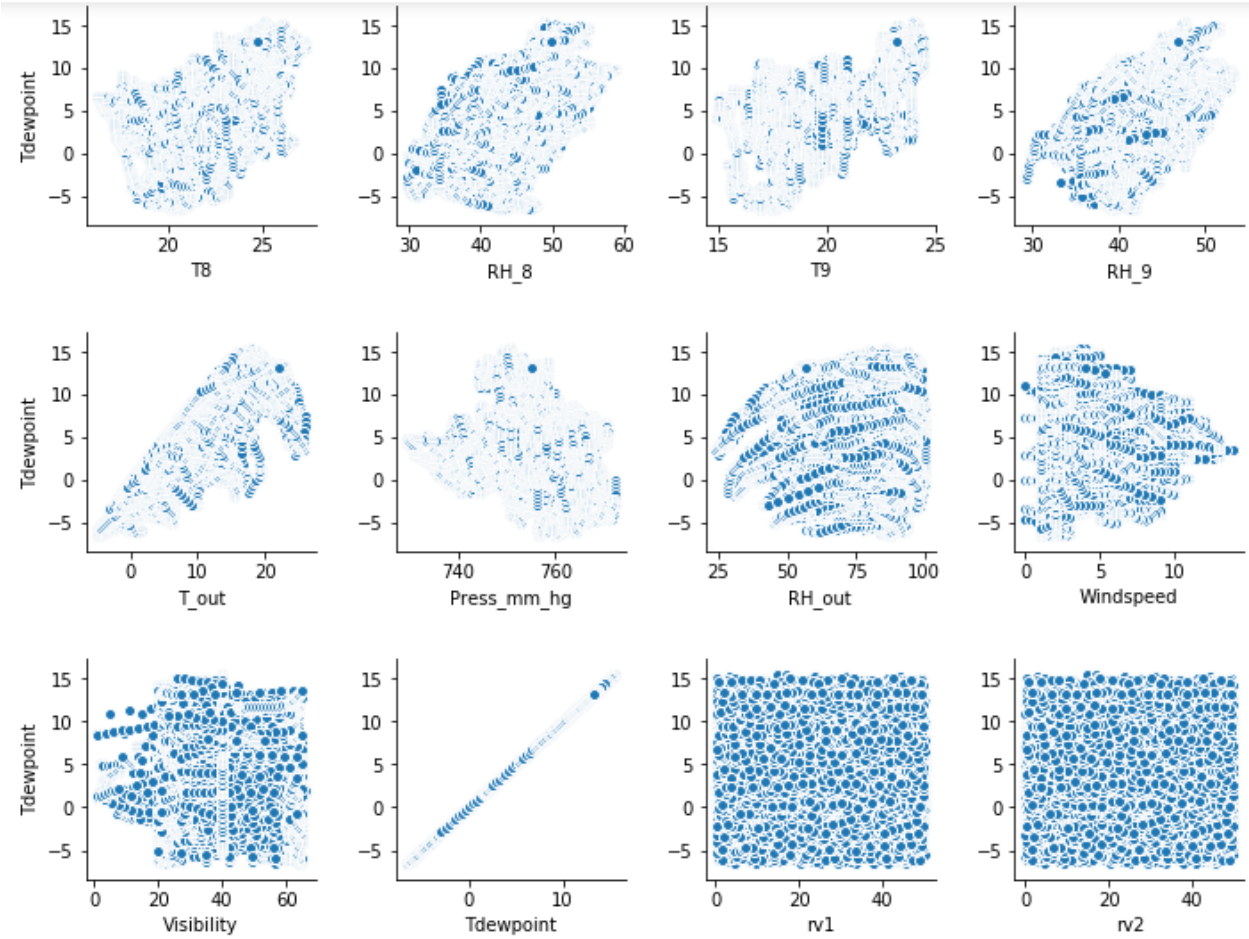


1. The plot is same for all.

2. There is no increase or decrease at any point of time.

3. Can say that they do affect any other variable at all.

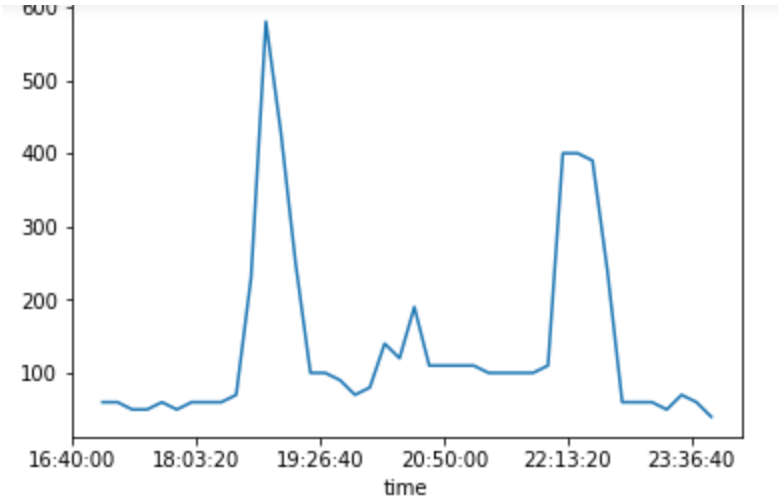


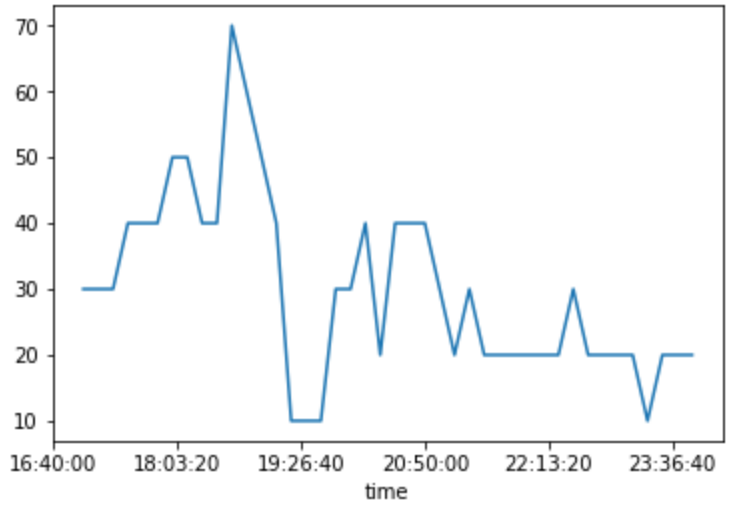


1. All the temperatures are linear to dewpoint.

2. Interesting point is , some humidity variables are linear to dewpoint whereas some are non-linear.

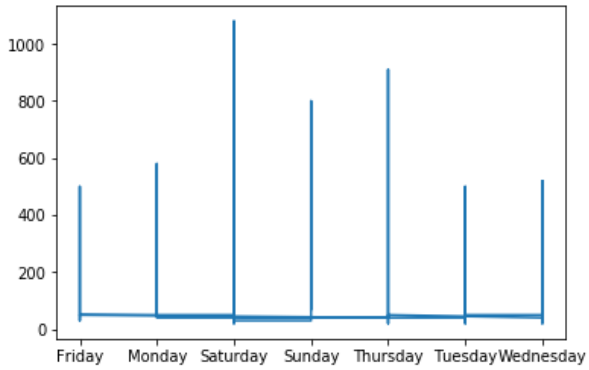
Analysis 12 : Analyzing with respect to date column





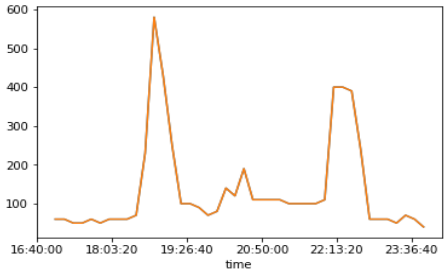
1. The first graph is Appliances vs.Time , second graph is for Lights vs. Time (both for first week of Jan).

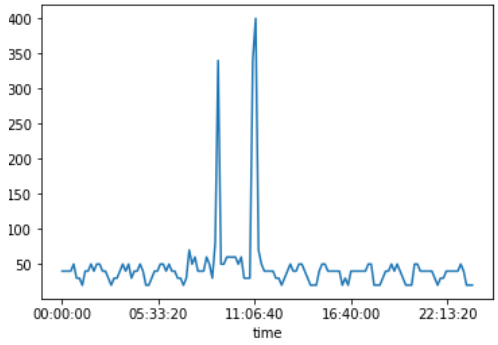
2. They both have the same trend . So we can say that lights and Appliances are positively related.

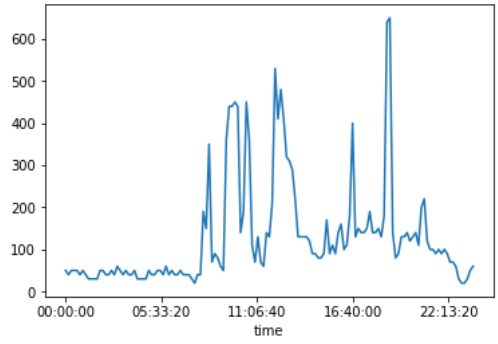


1. This is use of appliances for the first week of Jan.

2. This shows that Saturday and Thursday has used most of the energy.





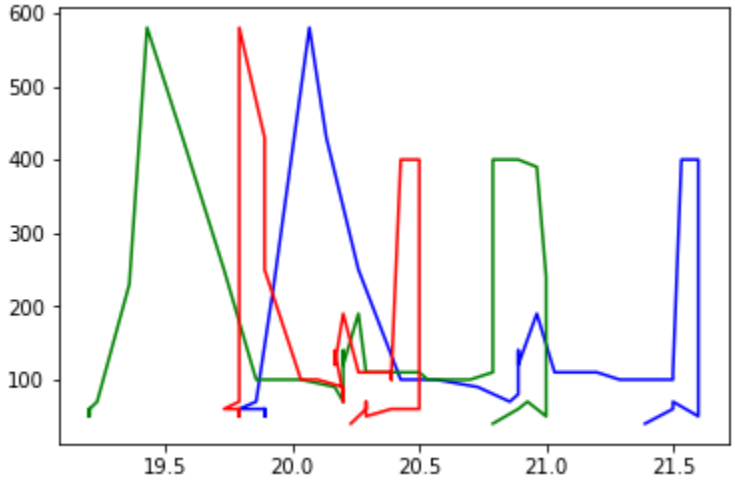


1. These graphs are plotted for first 3 weeks of January.

2. In the previous graph which was plotted for overall appliances usage with respect to time , we saw there was a peak between 7pm to 8pm but in these graphs it can be seen that for each week , the trend changes.

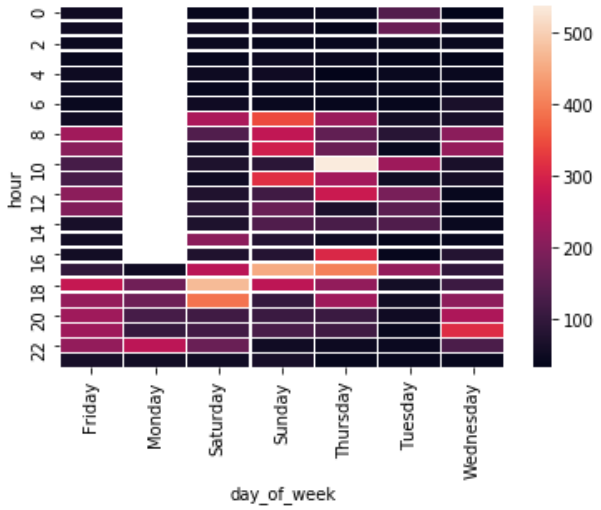
3. Two things can be concluded from this: first , we can say that though the graphs plotted for 3 weeks show peaks at different time , overall the maximum use of appliances energy was between 7pm to 8pm ; second , probably the peak is due to the anomalies that are present in the data.

Blue - T1 ; Green - T2 ; Red - T3

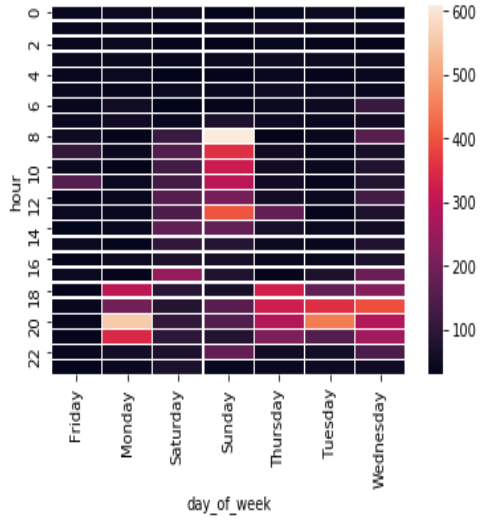


1. Three Temperatures vs. Appliances (first week of Jan)

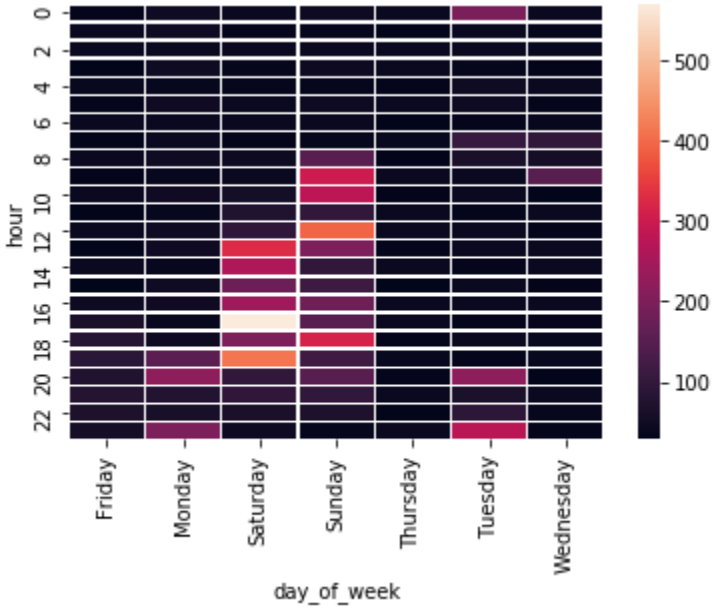
2. So when the temperature is low , the appliances have been used maximum and as temperature increases , the appliances usage has drastically decreased.



(1) - First week of Jan



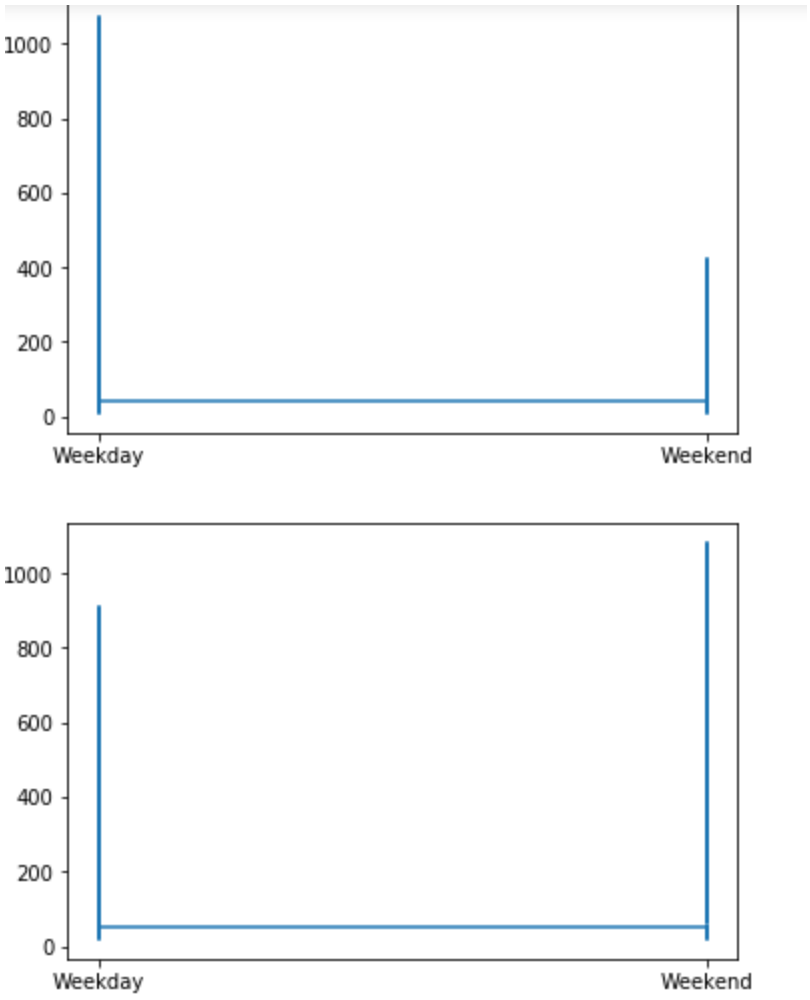
(2) Second week of Jan



(3) Third week of Jan

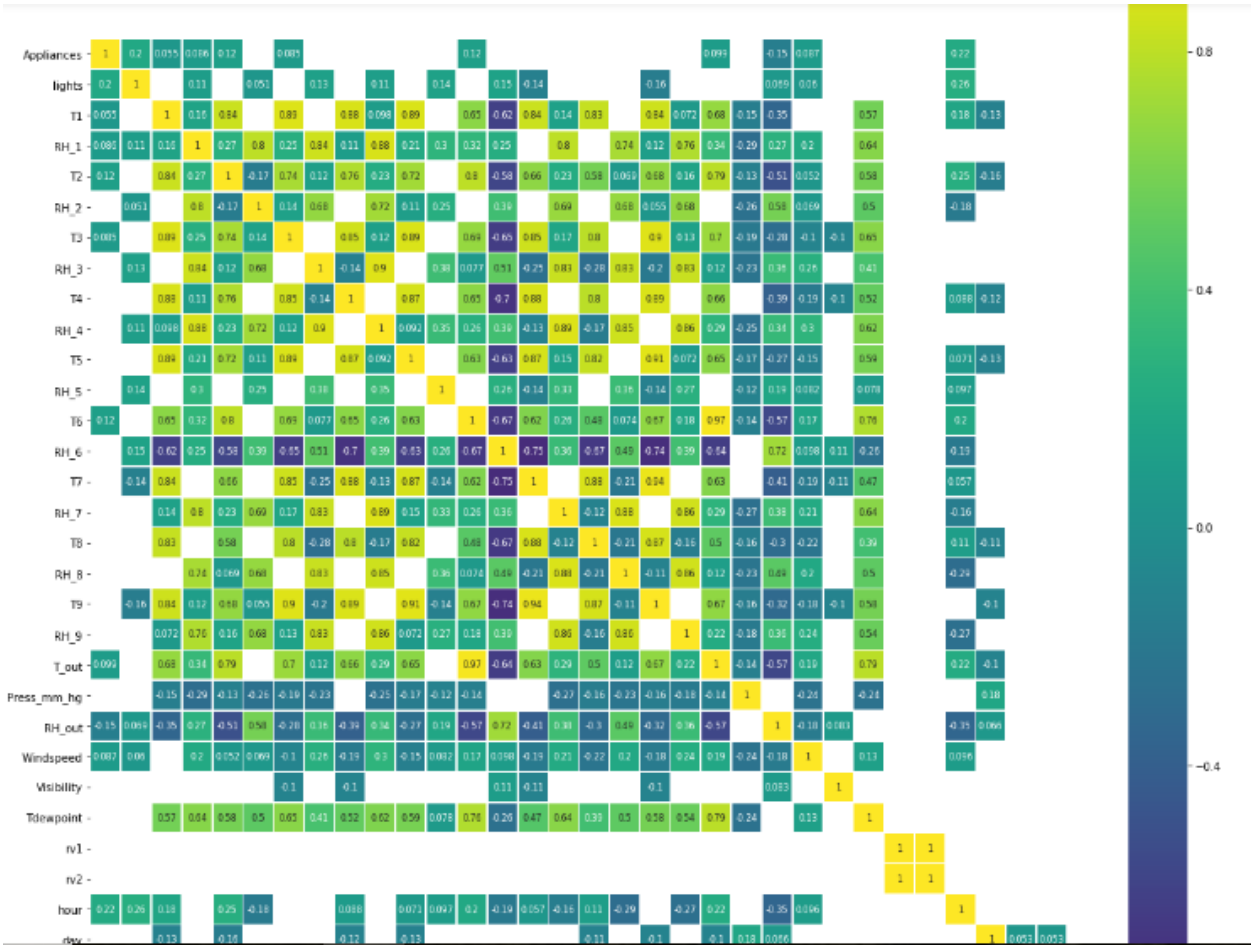
1) This is a heat map which shows on which day at what time how much appliances energy was consumed.

2) We can see that the maximum of appliances was used on Monday from 12midnight to 3 pm which was more than 600 whatts.



1. The first one use of appliances in second week of Jan and the second one is for First week of Jan.

2. So , we can see that irrespective of the nature of the day , the appliances were consumed. Usually when it is weekend , the appliances has to be consumed more , but we can see that that is not the case. So based on days we can't decide the use of appliances. This implies appliances is least affected by days.

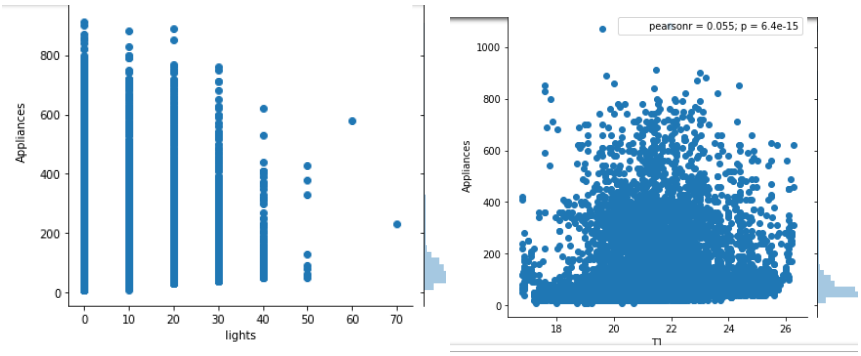


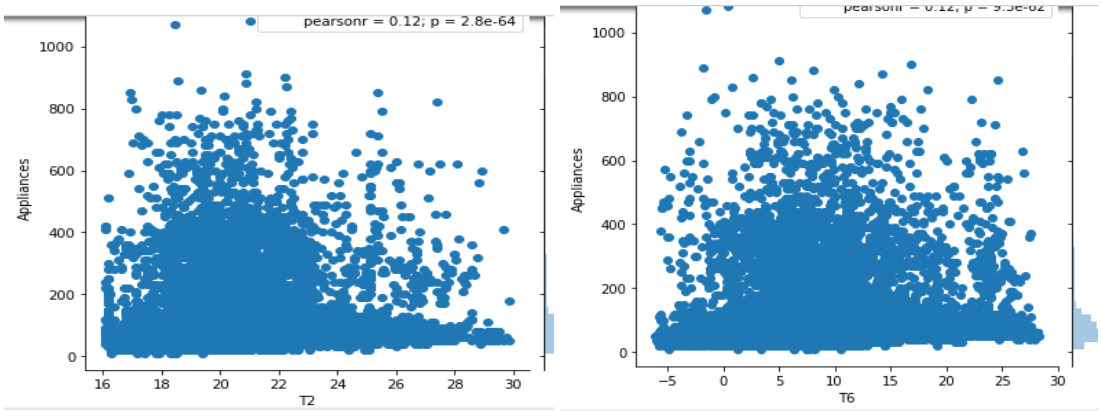
1. Overall heatmap.

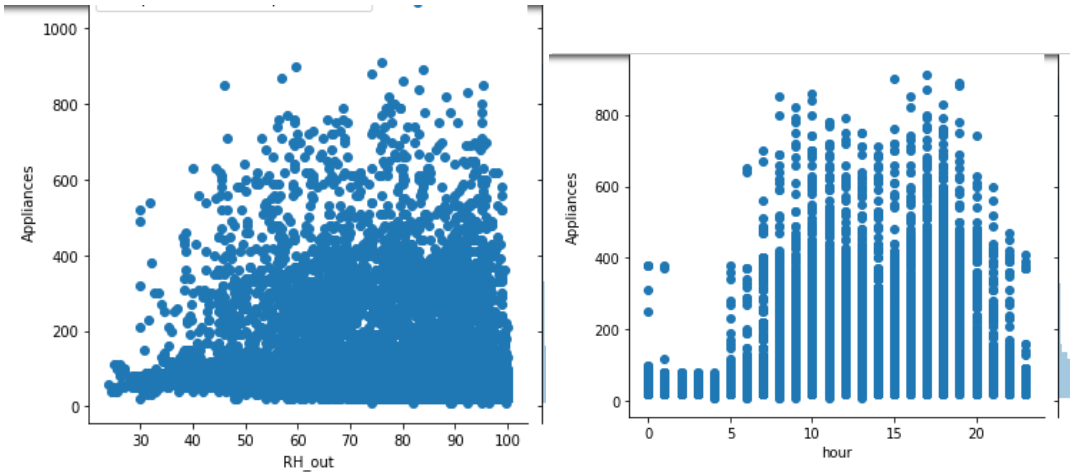
2. Gives us idea about which variable is positively affected by other variables and which are negatively correlated.

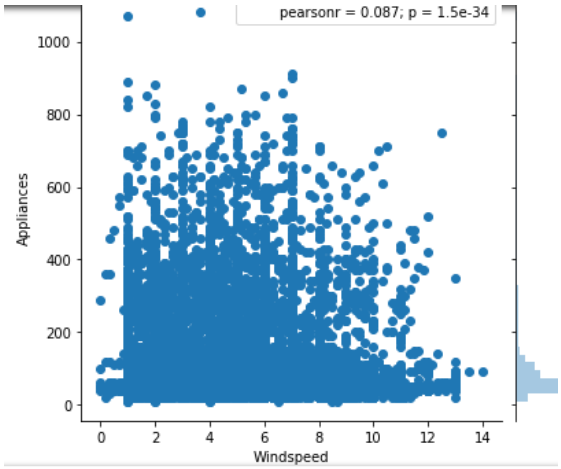
3. Blank spaces indicate that they are merely correlated or not correlated at all.

4. From the heatmap , we can see that appliances has positive correlation with lights , hour , RH_out , T1,T2,T6,Windspeed.









Appliances vs. lights - When the light fixture is 0Wh , the corresponding appliances energy is also high but when the light fixtures increases , the appliance is increasing.

Appliances vs. T1 - Maximum appliances is between 0 to 400 Whs.

Appliances vs. T2 - Maximum appliances usage is between 0 to 400Whs for temperature till 22 degree Celsius , once temperature increases above 24 degree Celsius , the maximum appliances usage decreases to 200 Whs.

Appliances vs. T6 - Majority of the data points are between 0 to 200 Whs and few are between 0 to 400 Whs.

Appliances vs. RH_out - As RH_out increases , the appliances value increases.

Appliances vs. hour - Appliances usage has increased as time goes.

Appliances vs. Windspeed - With increase in windspeed , the appliances usage is decreasing.

A highest datapoint which is >1000 Whs(for appliances) is recorded at low temperature ,low windspeed , high humidity and in the evening.

PART 2 : FEATURE ENGINEERING

After performing Exploratory Data Analysis on our dataset we have a picture about the overall data. We know what variables occur most , at what time , what variables may be important and what variables may influence the target which is the appliances.

Once that is done , it is important to check the dataset again for outliers and missing data or NaN values before we start the modeling.

So , for that feature engineering is performed on the dataset.

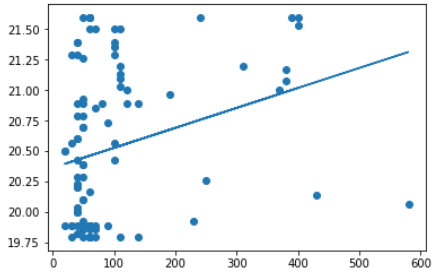
Step 1: First we have checked for missing data.

As this test was carried out previously during EDA we know there are no missing values. So we are moving forward as there is no need for imputing the data.

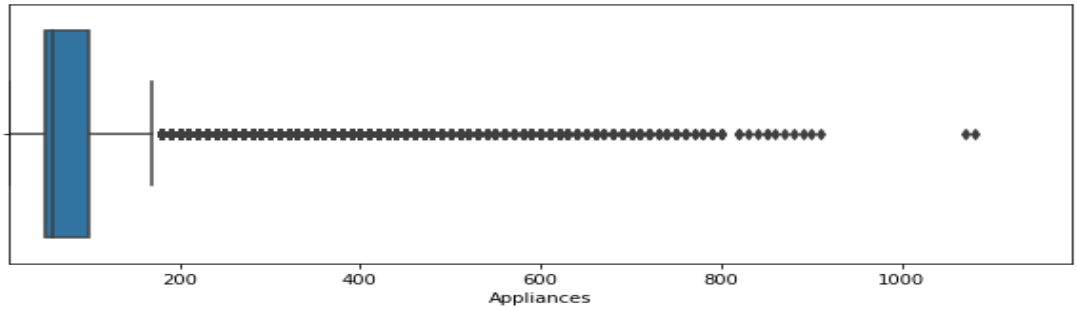
Step 2 : Detecting outliers and dealing with it.

Method 1 : Log transform :

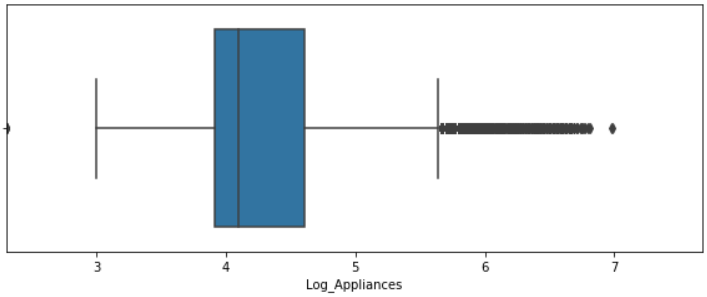
Here we first tried fitting the model in the linear regression to check for outliers.



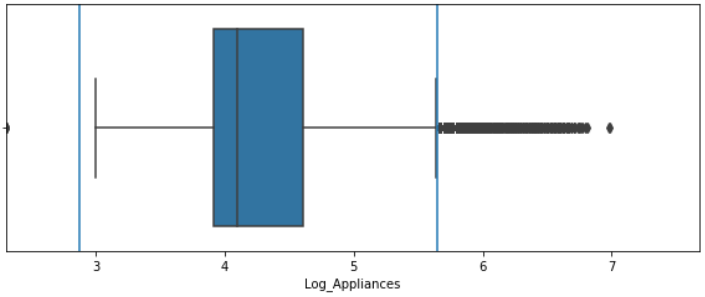
Plotted a boxplot graph for that :



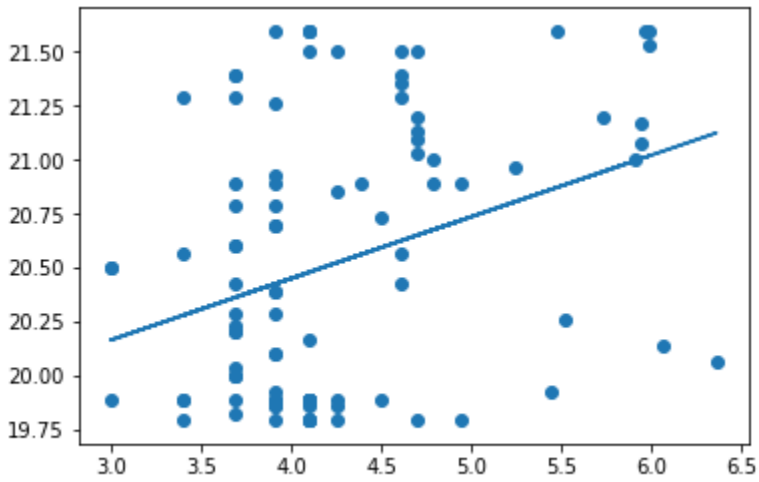
We can see that there are a lot of anomalies. So we tried applying log scale to the target and tried plotting it .



The outliers range is :



Tried fitting in the model again :



The above method , though it gives us the outliers , it doesn't improve the output. So , we go for other methods.

Method 2 : Scaling the outliers :

There are 7 methods to scale a data to fit the outliers :

1. Standard Scaler
2. MinMax Scaler
3. MaxABS Scaler
4. Robust Scaler
5. Quartile transformer(normal)
6. Quartile transformer(uniform)
7. Normalizer

STANDARD SCALER :

Here , it calculates the mean and standard deviation of a sample and when it scales the next sample it uses this stored value to scale that sample.

	Appliances	lights	T1	RH_1	T2	RH_2	T3
0	-0.367676	3.301264	-1.118645	1.843821	-0.520411	1.073683	-1.235063
1	-0.367676	3.301264	-1.118645	1.616807	-0.520411	1.057097	-1.235063
2	-0.465215	3.301264	-1.118645	1.517959	-0.520411	1.033550	-1.235063
3	-0.465215	4.561378	-1.118645	1.459321	-0.520411	1.024540	-1.235063
4	-0.367676	4.561378	-1.118645	1.526336	-0.520411	1.009797	-1.235063

	RH_3	T4	RH_4	...	RH_out	Windspeed	Visibility
0	1.686130	-0.908217	1.506438	...	0.82208	1.207694	2.091596
1	1.704566	-0.908217	1.604528	...	0.82208	1.071703	1.766584
2	1.748608	-0.944115	1.580918	...	0.82208	0.935713	1.441572
3	1.769092	-0.962063	1.542526	...	0.82208	0.799723	1.116559
4	1.769092	-0.962063	1.497991	...	0.82208	0.663733	0.791547

	Tdewpoint	rv1	rv2	hour	day	weekStatus	NSM
0	0.366975	-0.807974	-0.807974	0.794304	-0.598455	-1.499445	-1.731963
1	0.343135	-0.440240	-0.440240	0.794304	-0.598455	-1.499445	-1.731788
2	0.319294	0.252109	0.252109	0.794304	-0.598455	-1.499445	-1.731612
3	0.295454	1.408801	1.408801	0.794304	-0.598455	-1.499445	-1.731436
4	0.271613	-1.028122	-1.028122	0.794304	-0.598455	-1.499445	-1.731261

As it is giving negative value after scaling , this method is not selected.

MINMAX SCALER :

This is similar to normalized scaler which scales the data in the range [0,1] so any high value will be equal to 1 or close to 1.

	Appliances	lights	T1	RH_1	T2	RH_2	T3	\
0	0.046729	0.428571	0.32735	0.566187	0.225345	0.684038	0.215188	
1	0.046729	0.428571	0.32735	0.541326	0.225345	0.682140	0.215188	
2	0.037383	0.428571	0.32735	0.530502	0.225345	0.679445	0.215188	
3	0.037383	0.571429	0.32735	0.524080	0.225345	0.678414	0.215188	
4	0.046729	0.571429	0.32735	0.531419	0.225345	0.676727	0.215188	

	RH_3	T4	RH_4	...	RH_out	Windspeed	Visibility	\
0	0.746066	0.351351	0.764262	...	0.894737	0.500000	0.953846	
1	0.748871	0.351351	0.782437	...	0.894737	0.476190	0.894872	
2	0.755569	0.344745	0.778062	...	0.894737	0.452381	0.835897	
3	0.758685	0.341441	0.770949	...	0.894737	0.428571	0.776923	
4	0.758685	0.341441	0.762697	...	0.894737	0.404762	0.717949	

	Tdewpoint	rv1	rv2	hour	day	weekStatus	NSM
0	0.538462	0.265449	0.265449	0.73913	0.333333	0.0	0.000000
1	0.533937	0.372083	0.372083	0.73913	0.333333	0.0	0.000051
2	0.529412	0.572848	0.572848	0.73913	0.333333	0.0	0.000101
3	0.524887	0.908261	0.908261	0.73913	0.333333	0.0	0.000152

This method provides us with good result.

MaxABS scaler :

MaxAbsScaler differs from the previous scaler such that the absolute values are mapped in the range [0, 1]. On positive only data, this scaler behaves similarly to MinMaxScaler and therefore also suffers from the presence of large outliers.

	Appliances	lights	T1	RH_1	T2	RH_2	T3	\
0	0.055556	0.428571	0.757426	0.751210	0.643072	0.799441	0.676905	
1	0.055556	0.428571	0.757426	0.736953	0.643072	0.798236	0.676905	
2	0.046296	0.428571	0.757426	0.730745	0.643072	0.796525	0.676905	
3	0.046296	0.571429	0.757426	0.727062	0.643072	0.795871	0.676905	
4	0.055556	0.571429	0.757426	0.731271	0.643072	0.794800	0.676905	

	RH_3	T4	RH_4	...	RH_out	Windspeed	Visibility	\
0	0.891687	0.725191	0.891890	...	0.92	0.500000	0.954545	
1	0.892883	0.725191	0.900225	...	0.92	0.476190	0.896465	
2	0.895741	0.722392	0.898219	...	0.92	0.452381	0.838384	
3	0.897070	0.720992	0.894957	...	0.92	0.428571	0.780303	
4	0.897070	0.720992	0.891172	...	0.92	0.404762	0.722222	

	Tdewpoint	rv1	rv2	hour	day	weekStatus	NSM
0	0.341935	0.265527	0.265527	0.73913	0.354839	0.0	0.005142
1	0.335484	0.372150	0.372150	0.73913	0.354839	0.0	0.005193
2	0.329032	0.572893	0.572893	0.73913	0.354839	0.0	0.005243
3	0.322581	0.908271	0.908271	0.73913	0.354839	0.0	0.005293

This provides results similar to that of MinMaxScaler.

ROBUST SCALER :

Here , just like standard scaler the statistical values are stored and the next samples are scaled on that basis. The difference here is the median and interquartile ranges are stored in this case. The scaling is done between 25th quantile and 75th quantile.

	Appliances	lights	T1	RH_1	T2	RH_2	T3	\
0	0.0	30.0	-0.929348	1.384884	-0.295203	0.800373	-0.924	
1	0.0	30.0	-0.929348	1.227326	-0.295203	0.787780	-0.924	
2	-0.2	30.0	-0.929348	1.158721	-0.295203	0.769900	-0.924	
3	-0.2	40.0	-0.929348	1.118023	-0.295203	0.763060	-0.924	
4	0.0	40.0	-0.929348	1.164535	-0.295203	0.751866	-0.924	

	RH_3	T4	RH_4	...	RH_out	Windspeed	Visibility
0	1.275720	-0.648508	1.081489	...	0.390625	0.952381	2.090909
1	1.288066	-0.648508	1.145749	...	0.390625	0.857143	1.742424
2	1.317558	-0.677043	1.130282	...	0.390625	0.761905	1.393939
3	1.331276	-0.691310	1.105131	...	0.390625	0.666667	1.045455
4	1.331276	-0.691310	1.075956	...	0.390625	0.571429	0.696970

	Tdewpoint	rv1	rv2	hour	day	weekStatus	NSM
0	0.329412	-0.463297	-0.463297	0.454545	-0.357143	-0.75	-1.000000
1	0.311765	-0.250797	-0.250797	0.454545	-0.357143	-0.75	-0.999899
2	0.294118	0.149288	0.149288	0.454545	-0.357143	-0.75	-0.999797
3	0.276471	0.817700	0.817700	0.454545	-0.357143	-0.75	-0.999696

This also gives negative values , so this will not selected.

QUARTILE TRANSFORMER :

This type of scaling method follows a uniform or normal distribution. So when this is applied on a feature ,this tends to spread out the most frequently occurring observation. So , it is suppose to reduce the outliers impact on the data.

There are 2 methods : Normal(Gaussian) and Uniform

Uniform :

	Appliances	lights	T1	RH_1	T2	RH_2	T3	\
0	0.460961	0.981481	0.11962	0.956898	0.326326	0.874875	0.076577	
1	0.460961	0.981481	0.11962	0.932751	0.326326	0.869834	0.076577	
2	0.267267	0.981481	0.11962	0.923217	0.326326	0.862362	0.076577	
3	0.267267	0.997497	0.11962	0.914915	0.326326	0.859359	0.076577	
4	0.460961	0.997497	0.11962	0.924182	0.326326	0.854354	0.076577	

	RH_3	T4	RH_4	...	RH_out	Windspeed	\
0	0.926927	0.165666	0.902109	...	0.766767	0.868869	
1	0.930430	0.165666	0.920702	...	0.766767	0.846346	
2	0.939613	0.157658	0.916588	...	0.766767	0.828829	
3	0.943443	0.151151	0.908967	...	0.766767	0.798298	
4	0.943443	0.151151	0.900697	...	0.766767	0.762763	

	Visibility	Tdewpoint	rv1	rv2	hour	day	\
0	0.957958	0.659159	0.267264	0.267264	0.729229	0.307307	
1	0.899900	0.652152	0.374375	0.374375	0.729229	0.307307	

Normal :

	Appliances	lights	T1	RH_1	T2	RH_2	T3	\
0	-0.098013	2.085356	-1.17689	1.715766	-0.45008	1.149742	-1.428482	
1	-0.098013	2.085356	-1.17689	1.496597	-0.45008	1.125607	-1.428482	
2	-0.621099	2.085356	-1.17689	1.427047	-0.45008	1.090995	-1.428482	
3	-0.621099	2.806711	-1.17689	1.371657	-0.45008	1.077446	-1.428482	
4	-0.098013	2.806711	-1.17689	1.433779	-0.45008	1.055293	-1.428482	

	RH_3	T4	RH_4	...	RH_out	Windspeed	Visibility	\
0	1.453280	-0.971436	1.293665	...	0.72824	1.121060	1.727466	
1	1.479004	-0.971436	1.409811	...	0.72824	1.020888	1.280981	
2	1.551535	-1.004131	1.382478	...	0.72824	0.949547	1.137677	
3	1.584354	-1.031509	1.334418	...	0.72824	0.835558	1.053106	
4	1.584354	-1.031509	1.285532	...	0.72824	0.715218	0.955471	

	Tdewpoint	rv1	rv2	hour	day	weekStatus	NSM
0	0.410169	-0.621108	-0.621108	0.610484	-0.503497	-5.199338	-5.199338
1	0.391137	-0.320288	-0.320288	0.610484	-0.503497	-5.199338	-3.887342
2	0.374936	0.186102	0.186102	0.610484	-0.503497	-5.199338	-3.715633
3	0.357496	1.314765	1.314765	0.610484	-0.503497	-5.199338	-3.611830

Out of these two , normal method is giving negative outputs so this will be not selected whereas uniform method will be selected.

NORMALIZER :

This also scales the data between [0,1].

```
[5 rows x 32 columns]
   Appliances    lights        T1      RH_1        T2      RH_2        T3  \
0    0.000980  0.000490  0.000325  0.000778  0.000314  0.000732  0.000323
1    0.000971  0.000485  0.000322  0.000755  0.000311  0.000724  0.000320
2    0.000801  0.000481  0.000319  0.000742  0.000308  0.000715  0.000317
3    0.000794  0.000635  0.000316  0.000731  0.000305  0.000708  0.000314
4    0.000943  0.000629  0.000313  0.000728  0.000302  0.000700  0.000311
5    0.000779  0.000623  0.000310  0.000717  0.000299  0.000693  0.000308
6    0.000926  0.000772  0.000307  0.000706  0.000296  0.000687  0.000305
7    0.000917  0.000764  0.000304  0.000697  0.000294  0.000680  0.000302
8    0.000909  0.000606  0.000300  0.000691  0.000291  0.000673  0.000299
9    0.001051  0.000601  0.000298  0.000692  0.000289  0.000667  0.000297

       RH_3        T4      RH_4  ...       RH_out  Windspeed  Visibility  \
0  0.000731  0.000310  0.000744  ...     0.001503   0.000114    0.001029
1  0.000725  0.000307  0.000744  ...     0.001489   0.000108    0.000957
2  0.000720  0.000303  0.000735  ...     0.001474   0.000101    0.000887
3  0.000714  0.000300  0.000726  ...     0.001460   0.000095    0.000817
4  0.000707  0.000297  0.000716  ...     0.001446   0.000089    0.000749
```

As we can see , all the values are nearly zero which is not useful.

Method 3 : Other methods to deal with outliers

Other methods to deal with outliers:
1. Standard Deviation
2. Quartile Method
3. Eliminating outliers

STANDARD DEVIATION :

In this method , we calculate the mean and standard deviation of the data frame and subtracted the mean from the data points and divided that by standard deviation.

```
0       -0.367666
1       -0.367666
2       -0.465204
3       -0.465204
4       -0.367666
5       -0.465204
6       -0.367666
7       -0.367666
8       -0.367666
9       -0.270129
10       1.290468
11       4.704273
12       3.241213
13       1.485542
14       0.022483
15       0.022483
16      -0.075055
17      -0.270129
18      -0.172592
19       0.412632
```

We are getting negative values in this because of which we are eliminating this method.

QUARTILE METHOD :

In this we calculate , the 25th quantile and 75th quantile with which we can calculate the IQR which is the difference between these two.

min = q25 - (iqr*1.5) = -25

max = q75 + (iqr*1.5) = 175

So anything between min and max will be considered as outliers. As most of our data is greater than 175 , there will be lot of anomalies. So , we are eliminating it.

ELIMINATING /DELETING OUTLIERS :

This is similar to standard deviation method , but in this what we do is that we change the number by which we want to subtract mean from the value.

```
final_list = [x for x in df.Appliances if(x > r1 -2 * r2)]
final_list = [x for x in final_list if (x < r1 +2 * r2)]
```

r1 - mean of appliances , r2 - standard deviation of appliances.

final_ list will have only values between this range. By doing this we are removing the outliers.

Since this is a time-series plot , it is not advisable to remove any data point since it might result in a huge impact on the output.

So this method is also eliminated.

Out of the 4 scaling method that we are left with , we decided to go with MinMax scaler as it provided good result.

PART 3 :PREDICTION ALGORITHMS

Overall 7 regression and classification methods are performed for the given below dataset

- Simple Linear Regression Model
- Multiple Linear Regression Model
- Gradient Boosting Machine Model
- Random Forest Regression Model(TPOT output without NSM)
- Neural Network Model
- Support Vector Machine (Classification)
- K Nearest Neighbors Regression(TPOT output with NSM)

Dataset :

date	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	T5	...	Tdewpoint	rv1
2016-01-11 17:00:00	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	17.166667	...	5.3	13.2754
2016-01-11 17:10:00	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	17.166667	...	5.2	18.6061
2016-01-11 17:20:00	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	45.890000	17.166667	...	5.1	28.6426
2016-01-11 17:30:00	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	45.723333	17.166667	...	5.0	45.4103
2016-01-11	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	45.530000	17.200000	...	4.9	10.0840

rv1	rv2	year	month	hour	day	Numerical_Week	weekStatus	NSM
13.275433	13.275433	2016	1	17	11	0	0	61200
18.606195	18.606195	2016	1	17	11	0	0	61800
28.642668	28.642668	2016	1	17	11	0	0	62400
45.410389	45.410389	2016	1	17	11	0	0	63000
10.084097	10.084097	2016	1	17	11	0	0	63600

Basic Steps in all the models:

a. Modelling using unscaled data(Anomalies are affecting the result).
b. Modelling using the scaled response.
c. Modelling using scaling the whole data.
d. Modelling using the test and train provided already.
e. Comparing the results based on Train and test division.

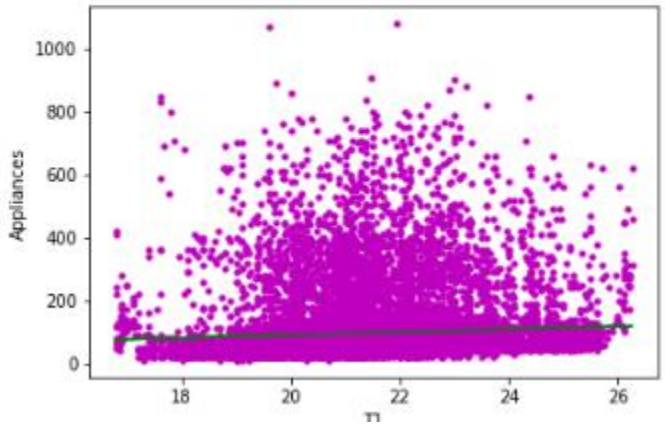
1. SIMPLE LINEAR REGRESSION: Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:One variable, denoted x, is regarded as the predictor, explanatory, or independent variable.The other variable, denoted y, is regarded as the response, outcome, or dependent variable. Because the other terms are used less frequently today, we'll use the "predictor" and "response" terms to refer to the variables encountered in this course. The other terms are mentioned only to make you aware of them should you encounter them in other arenas. Simple linear regression gets its adjective "simple," because it concerns the study of only one predictor variable. In contrast, multiple linear regression, which we study later in this course, gets its adjective "multiple," because it concerns the study of two or more predictor variables.
   Code :

```
In [9]: def estimate_coef(x,y):
            n = np.size(x) #calculating the points
            m_x,m_y = np.mean(x), np.mean(y) #mean of response and featue variable
            SS_xy = np.sum(y*x - n*m_y*m_x) #  cross-deviation
            SS_xx = np.sum(x*x - n*m_x*m_x) # deviation about x
            b_1 = SS_xy / SS_xx # Regression coeffients
            b_0 = m_y - b_1*m_x
            return(b_0, b_1)

        def plot_regression_line(x, y, b):
            # plotting the actual points as scatter plot
            plt.scatter(x, y, color = "m",
                        marker = "o", s = 10)
             # predicted response vector
            y_pred = b[0] + b[1]*x #USING LINEAR EQAUTION TO UNDERSTAND THE relations between two continous Entity
             # plotting the regression line
            plt.plot(x, y_pred, color = "g")
            # function to show plot
            plt.show()
        #Energy used on the basis of temperature T1
        def main():
            # observations
            x = data.T1
            y = data.Appliances

            # estimating coefficients
            b = estimate_coef(x, y)
            plt.xlabel('T1')
            plt.ylabel('Appliances')
            print("Estimated coefficients:\nb_0 = {}  \
                  \nb_1 = {}".format(b[0], b[1]))

            # plotting regression line
            plot_regression_line(x, y, b)


        if __name__ == "__main__":
            main()
```

Output :

1. T1 with Appliances:

```
Estimated coefficients:
b_0 = -5.817915095462922e-06
b_1 = 4.504859817246594
```

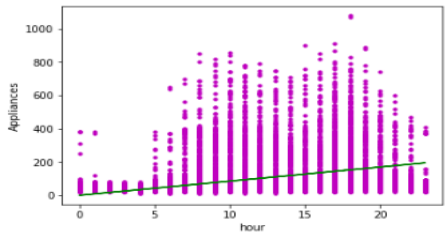


```
from statsmodels.formula.api import ols
model = ols("data.Appliances ~ data.T1 ", data).fit()#Where data.T1 is an exogenues
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        data.Appliances   R-squared:                       0.003
Model:                            OLS   Adj. R-squared:                  0.003
Method:                 Least Squares   F-statistic:                     60.85
Date:                Thu, 15 Mar 2018   Prob (F-statistic):           6.45e-15
Time:                        16:05:55   Log-Likelihood:            -1.1935e+05
No. Observations:               19735   AIC:                         2.387e+05
Df Residuals:                   19733   BIC:                         2.387e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     20.9343      9.867      2.122      0.034       1.594      40.274
data.T1        3.5395      0.454      7.801      0.000       2.650       4.429
==============================================================================
Omnibus:                    14107.462   Durbin-Watson:                   0.495
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           195656.284
Skew:                           3.414   Prob(JB):                         0.00
Kurtosis:                      16.832   Cond. No.                         295.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

2. Hour with Appliances

```
Estimated coefficients:
b_0 = -0.0011150949734002324
b_1 = 8.49383240994969
```

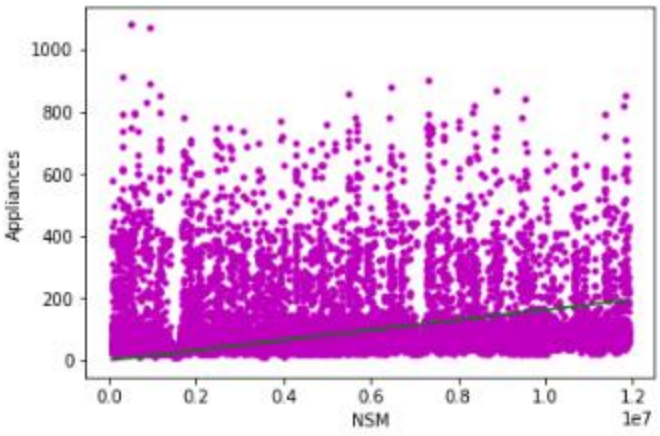



```
from statsmodels.formula.api import ols
model = ols("data.Appliances ~ data.hour ", data).fit()#Where data.hour is an exogenues
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        data.Appliances   R-squared:                       0.047
Model:                            OLS   Adj. R-squared:                  0.047
Method:                 Least Squares   F-statistic:                     973.2
Date:                Thu, 15 Mar 2018   Prob (F-statistic):          1.39e-208
Time:                        16:22:45   Log-Likelihood:            -1.1890e+05
No. Observations:               19735   AIC:                         2.378e+05
Df Residuals:                   19733   BIC:                         2.378e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     60.7618      1.382     43.974      0.000      58.053      63.470
data.hour      3.2110      0.103     31.196      0.000       3.009       3.413
==============================================================================
Omnibus:                    14153.176   Durbin-Watson:                   0.522
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           203665.822
Skew:                           3.412   Prob(JB):                         0.00
Kurtosis:                      17.181   Cond. No.                         26.1
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

3. NSM with Appliances

```
Estimated coefficients:
b_0 = -0.0016453828036304685
b_1 = 1.633340080564783e-05
```



OLS Regression Results

Dep. Variable:	data.Appliances	R-squared:	0.000
Model:	OLS	Adj. R-squared:	0.000
Method:	Least Squares	F-statistic:	1.830
Date:	Thu, 15 Mar 2018	Prob (F-statistic):	0.176
Time:	16:24:19	Log-Likelihood:	-1.1938e+05
No. Observations:	19735	AIC:	2.388e+05
Df Residuals:	19733	BIC:	2.388e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	99.4226	1.471	67.595	0.000	96.540	102.306
data.NSM	-2.888e-07	2.14e-07	-1.353	0.176	-7.07e-07	1.3e-07

Omnibus:	13994.308	Durbin-Watson:	0.494
Prob(Omnibus):	0.000	Jarque-Bera (JB):	190690.498
Skew:	3.382	Prob(JB):	0.00
Kurtosis:	16.644	Cond. No.	1.39e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.39e+07. This might indicate that there are strong multicollinearity or other numerical problems.

Deductions

1. Single variables are not making any huge impact on the Appliances Value
2. However, with the value of the Condition number in the last regression, we now know that the NSM (the number of seconds from midnight for eachday (NSM)) is making a great impact on the Appiances value(As discussed in Paper 1 as well)

2. MULTIPLE LINEAR REGRESSION:

In the multiple regression setting, because of the potentially large number of predictors, it is more efficient to use matrices to define the regression model and the subsequent analyses. This lesson considers some of the more important multiple regression formulas in matrix form. If you're unsure about any of this, it may be a good time to take a look at this Matrix Algebra Review.

The models have similar "LINE" assumptions. The only real difference is that whereas in simple linear regression we think of the distribution of errors at a fixed value of the single predictor, with multiple linear regression we have to think of the distribution of errors at a fixed set of values for all the predictors. All of the model checking procedures we learned earlier are useful in the multiple linear regression framework, although the process becomes more involved since we now have multiple predictors. We'll explore this issue further in Lesson 7.

The use and interpretation of r2 (which we'll denote R2 in the context of multiple linear regression) remains the same. However, with multiple linear regression we can also make use of an "adjusted" R2 value, which is useful for model building purposes. We'll explore this measure further in Lesson 10.

With a minor generalization of the degrees of freedom, we use t-tests and t-intervals for the regression slope coefficients to assess whether a predictor is significantly linearly related to the response, after controlling for the effects of all the opther predictors in the model.

With a minor generalization of the degrees of freedom, we use prediction intervals for predicting an individual response and confidence intervals for estimating the mean response.

Code:

```
# defining feature matrix(X) and response vector(y)
X = data2
y = data1.appliances_scaler


# splitting X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=1)

scaler = preprocessing.MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)


#Doing predictions on training and test data
predicted_train=reg.predict(X_train)
predicted_test=reg.predict(X_test)
print("predicted_train : " + str(predicted_train))
print("predicted_test : " + str(predicted_test))
```

For Scaling:

```
a1 = data.Appliances
a2 = a1.values
a3 = a2.reshape((len(a2), 1))
min_max_scaler = preprocessing.MinMaxScaler()
appliances_scaler = min_max_scaler.fit_transform(a3)
data1['appliances_scaler']  = appliances_scaler
data1.head(5)
```

```
scaler = preprocessing.MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Computational Methods:

```
#### MAE Calculation of model
test_mae=mae(y_test,predicted_test)
train_mae=mae(y_train,predicted_train)
print("test_mae : " + str(test_mae))
print("train_mae : " + str(train_mae))


#### RMSE Calculation of model
test_rmse = rmse(y_test,predicted_test)
train_rmse=rmse(y_train,predicted_train)
print("test_rmse : " + str(test_rmse))
print("train_rmse : " + str(train_rmse))

#### R Squared error calculation
test_r2=r2_score(y_test,predicted_test)
train_r2=reg.score(X_train,y_train)
print("test_r2 : " + str(test_r2))
print("train_r2 : " + str(train_r2))



#### Calculating MAPE
from scipy import stats
def mean_absolute_percentage_error1(y_test,x_predict):
    #np.seterr(divide='ignore',invalid='ignore')
    y_test,x_predict=np.array(y_test),np.array(x_predict)
    return np.abs((y_test - x_predict)/y_test)
test_mape = mean_absolute_percentage_error1(y_test, predicted_test)
train_mape = mean_absolute_percentage_error1(y_train, predicted_train)
print("test_mape : " + str(np.median(test_mape)))
print("train_mape : " + str(np.median(train_mape)))


### Getting coefficients for features
coefficients=reg.coef_
print('Coefficients: \n', reg.coef_)
```

For using the already given training and testing data

```
df_train = pd.read_csv("training.csv")
X_train = df_train.drop("Appliances",1)
y_train = pd.DataFrame(df_train["Appliances"])
df_test = pd.read_csv("testing.csv")
X_test = df_test.drop("Appliances",1)
y_test = pd.DataFrame(df_test["Appliances"])
X_train = X_train.set_index('date')
X_train = X_train.drop('Day_of_week',1)
X_train = X_train.drop('WeekStatus',1)
X_test = X_test.set_index('date')
X_test = X_test.drop('Day_of_week',1)
X_test = X_test.drop('WeekStatus',1)
X_test.head(2)
```

```
y_train = pd.DataFrame(df_train["Appliances"])
a1 = y_train.Appliances
a2 = a1.values
a3 = a2.reshape((len(a2), 1))
min_max_scaler = preprocessing.MinMaxScaler()
appliances_scaler = min_max_scaler.fit_transform(a3)
y_train['appliances_scaler']  = appliances_scaler
y_train = y_train.drop("Appliances",1)


y_test = pd.DataFrame(df_test["Appliances"])

a4 = y_test.Appliances
a5 = a4.values
a6 = a5.reshape((len(a5), 1))
min_max_scaler = preprocessing.MinMaxScaler()
appliances_scaler = min_max_scaler.fit_transform(a6)
y_test['appliances_scaler']  = appliances_scaler
y_test = y_test.drop("Appliances",1)
```

3. <u>RANDOM FOREST REGRESSOR:</u>

   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.

   Code:

```
# defining feature matrix(X) and response vector(y)
X = data1
y = data.Appliances

# splitting X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
                                                    random_state=1)
```

```
reg = RandomForestRegressor(n_estimators =100, random_state = 1,n_jobs=-1)
reg.fit(X_train, y_train)
```

The above Steps are performed for the same, below are the results for each section for Random Forest:

HIT and TRIAL for the Train and Test Division, to get the best results

Division : Train = 80 Test = 20

RMSE : 75.93 R2 : 0.475

Division : Train = 75 Test = 25

RMSE : 75.02 R2 : 0.485

Division : Train = 70 Test = 30

RMSE : 74.70 R2 : 0.471

Division : Train = 65 Test = 35

RMSE : 74.02 R2 : 0.476

Division : Train = 60 Test = 40

RMSE : 75.47 R2 : 0.45

Division 75, 25 and 70,30 were giving almost similar results so we selected 70,30 to maintain linearity

HIT and TRIAL for the number of trees(estimators), to get the best results

n_estimators = 5

RMSE : 77.54 R2 : 0.43

n_estimators = 10

RMSE : 74.63 R2 : 0.472

n_estimators = 20

RMSE : 72.66 R2 : 0.50

n_estimators = 25

RMSE : 71.76 R2 : 0.50

n_estimators = 30

RMSE : 71.76 R2 : 0.51

n_estimators = 50

RMSE : 71.01 R2 : 0.521

n_estimators = 100

RMSE : 70.75 R2 : 0.526

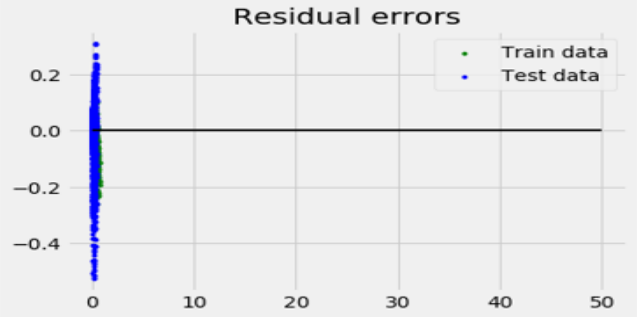
Selected estimator value : 100

For unscaled data:

```
predicted_train : [ 54.5  31.4  70.3 ...,  98.   61.4  38.6]
predicted_test : [  39.7   63.1  116.2 ...,  154.9  335.9  100.2]
test_mae : 33.7195237291
train_mae : 12.516070653
test_rmse : 70.754915396
train_rmse : 26.8473651391
test_r2 : 0.526223578378
train_r2 : 0.931262111263
test_mape : 32.9071757993
train_mape : 12.2806935685
Variance score: 0.5262235783783045
```

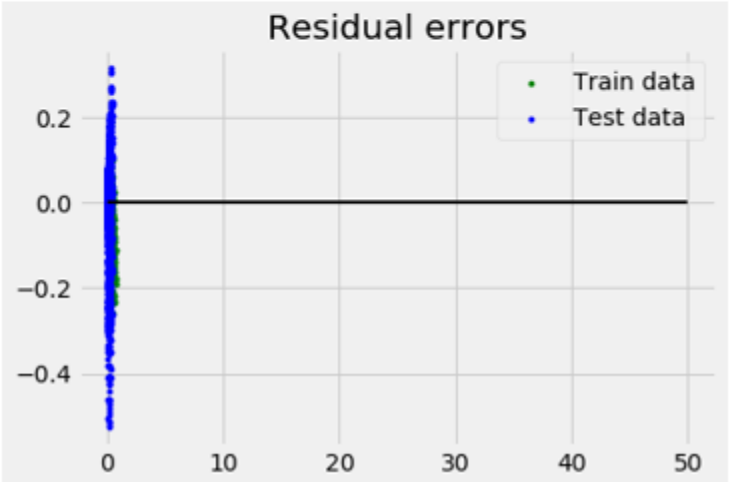
After Scaling only the appliances:

```
predicted_train : [ 0.0411215   0.02009346  0.05663551 ...,  0.07766355  0.0482243
  0.02588785]
predicted_test : [ 0.02775701  0.05        0.09953271 ...,  0.13392523  0.30934579
  0.07785047]
test_mae : 0.0316425458569
train_mae : 0.0117429087922
test_rmse : 0.0663916418841
train_rmse : 0.0251871246934
test_r2 : 0.522410703736
train_r2 : 0.930734401744
test_mape : 0.198
train_mape : 0.0748076923077
Variance score: 0.5224107037358212
```



After Scaling the whole data:

```
predicted_train : [ 0.04130841  0.02009346  0.05663551 ...,  0.07766355  0.0482243
  0.02588785]
predicted_test : [ 0.02803738  0.04971963  0.09906542 ...,  0.13392523  0.30934579
  0.07757009]
test_mae : 0.0316596716581
train_mae : 0.0117375573203
test_rmse : 0.0664123804005
train_rmse : 0.0251772069935
test_r2 : 0.522112291464
train_r2 : 0.930788939145
test_mape : 0.1975
train_mape : 0.0744444444444
Variance score: 0.5221122914637413
```



4. Gradient Boosting Machine:

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
Gradient boosting involves three elements:
A loss function to be optimized.
A weak learner to make predictions.
An additive model to add weak learners to minimize the loss function.

1. Loss Function
The loss function used depends on the type of problem being solved.It must be differentiable, but many standard loss functions are supported and you can define your own.For example, regression may use a squared error and classification may use logarithmic loss.A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.

2. Weak Learner
Decision trees are used as the weak learner in gradient boosting.Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in the predictions.Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels.It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes.This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

3. Additive Model

Trees are added one at a time, and existing trees in the model are not changed.A gradient descent procedure is used to minimize the loss when adding trees.Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

Code:

```
# defining feature matrix(X) and response vector(y)
X = data1
y = data.Appliances

# splitting X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,
                                                    random_state=1)
gbrt=GradientBoostingRegressor(n_estimators=100)
gbrt.fit(X_train, y_train)
y_pred=gbrt.predict(X_test)
print (gbrt.feature_importances_)
predicted_train=gbrt.predict(X_train)
predicted_test=gbrt.predict(X_test)
print("predicted_train : " + str(predicted_train))
print("predicted_test : " + str(predicted_test))
```

RESULTS:

```
predicted_train : [ 0.05390085  0.06264337  0.02504849 ...,  0.10882598  0.07219341
  0.05458832]
predicted_test : [ 0.03734032  0.07111162  0.09807275 ...,  0.0863589   0.04026446
  0.13241992]
test_mae : 0.0426145176887
train_mae : 0.0405134971274
test_rmse : 0.0796748367245
train_rmse : 0.075525274829
test_r2 : 0.305377904301
train_r2 : 0.380175779234
```

5. SUPPORT VECTOR MACHINE (For Classification):

In machine learning, support vector machines (SVMs, also support vector networks are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Code:

```
[5]: y = data1.weekStatus
     y1 = y.as_matrix()
     # fit a SVM model to the data
     model = SVC()
     model.fit(X, y1)
     print(model)
     # make predictions
     expected = y1
     predicted = model.predict(X)
     # summarize the fit of the model
     print(metrics.classification_report(expected, predicted))
     print(metrics.confusion_matrix(expected, predicted))
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     14263
          1       1.00      1.00      1.00      5472

avg / total       1.00      1.00      1.00     19735

[[14263     0]
 [    0  5472]]
```

We have done Classification based on weekStatus
0 : Weekday , 1: Weekend

6. NEURAL NETWORKS

Multi Layer Perceptron (MLP) :

This is a supervised learning algorithm in which we give n dimension input and n dimension output. The model will train itself accordingly, depending on the weights that is given to each input and give us the proper output. The number of hidden layers between the input and the output can be tuned by us.

In this project , MLP regressor with 'sgd' solver is applied .

Input is stored in X which contains all features except for Appliances that are scaled using MinMaxscaler .

Target which is Appliances is stored in Y after scaling.

Test and Train is divided into 30 ,70 ratio respectively.

Since the optimizer used is 'sgd' we need to provide the learning rate.

The hidden_layer_sizes , which is the number of neurons which doesn't include the input and output sizes.

```
rmse_test = sqrt(mean_squared_error(test_Y,pred1))
print('Test RMSE: %.3f' % rmse_test)
```

```
Test RMSE: 0.082
```

```
r2 = r2_score(test_Y,pred1)
print('Test R2: %.3f' % r2)
```

```
Test R2: 0.072
```

Since the result that we get from this for r2 is very less ,we won't select this model.

7. K Nearest Neighbor Regression (TPOT output with NSM column)
   Import data into feature matrix and target

Script:

```
from sklearn import preprocessing
from tpot import TPOTRegressor
from sklearn.datasets import load_digits
from sklearn.cross_validation import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline


url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['c
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
#data['DateStr'] = data['date'].apply(lambda x: x.strftime('%Y%m%d%H%M'))
d = data.date[0:len(data.date)]
data_final = []
for i in range (len(d)):
    if(i==0):
        a= 61200
        data_final.append(a)
    elif(i>0):
        a=a+600
        data_final.append(a)

data["NSM"] = pd.DataFrame({'NSM':data_final})


```

```
data1 = data.drop('date', 1)
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('weekStatus',1)
data1 = data1.drop('time',1)
data1 = data1.drop('T1',1)
data1 = data1.drop('T4',1)
data1 = data1.drop('T6',1)
data1 = data1.drop('T7',1)
data1 = data1.drop('T_out',1)
data1 = data1.drop('RH_1',1)
data1 = data1.drop('RH_6',1)
data1 = data1.drop('RH_8',1)
data1 = data1.drop('RH_9',1)
data1 = data1.drop('rv1',1)
data1 = data1.drop('rv2',1)
data1 = data1.drop('year',1)
data1 = data1.drop('month',1)
data1 = data1.drop('day',1)
data1 = data1.drop('Numerical_Week',1)
data1 = data1.drop('Windspeed',1)
data1 = data1.drop('Visibility',1)
data1 = data1.drop('WeekStatus',1)


X = data1
y1 = data.Appliances
data1.head(5)

```

```
# Score on the training set was:-54.09793812726714
exported_pipeline = KNeighborsRegressor(n_neighbors=11, p=2, weights="distance")

exported_pipeline.fit(X_train, y_train)
#results = exported_pipeline.predict(testing_features)
```

18]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=11, p=2,
          weights='distance')
```

19]:
```
results = exported_pipeline.predict(X_test)
```

20]:
```
#### RMSE Calculation of model
def rmse(actual,prediction):
    return np.sqrt(mean_squared_error(actual,prediction))
test_rmse = rmse(y_test,results)
#train_rmse=rmse(y_train,predicted_train)
print("test_rmse : " + str(test_rmse))
#print("train_rmse : " + str(train_rmse))

#### R Squared error calculation
test_r2=r2_score(y_test,results)
#train_r2=clf.score(X_train,y_train)
print("test_r2 : " + str(test_r2))
#print("train_r2 : " + str(train_r2))
```

```
test_rmse : 68.9901491875
test_r2 : 0.549562665735
```

COMPARISION TABLE for all the methods:

Models	Unscaled Data	Whole Scaled	Already provided train n test csv
Multiple Linear Regression	RMSE: 92.94	RMSE: 0.086	RMSE: 0.088
	MAE: 52.58	MAE: 0.049	MAE: 0.048
	MAPE: 59.55	MAPE: 0.50	MAPE: 0.51
	R*2: 0.180	R*2: 0.182	R*2: 0.15
Random Forest Regressor	RMSE: 70.75	RMSE: 0.066	RMSE:0.066
	MAE: 33.71	MAE: 0.031	MAE: 0.03
	MAPE: 32.90	MAPE: 0.19	MAPE: 0.18
	R*2: 0.526	R*2: 0.52	R*2: 0.51
Gradient Boosting Machine	RMSE: 85.25	RMSE: 0.079	RMSE: 0.080
	MAE: 45.59	MAE: 0.042	MAE: 0.041
	MAPE: 49.43	MAPE: inf	MAPE: inf
	R*2: 0.301	R*2: 0.305	R*2: 0.29
Neural Network	RMSE: NA	RMSE: 0.10	RMSE: NA
	MAE: NA	MAE: NA	MAE: NA
	MAPE: NA	MAPE: NA	MAPE: NA
	R*2: NA	R*2: NA	R*2: NA

Support Vector Machine	

On the basis of above calculations, we selected Random Forest.

PART 4 : FEATURE SELECTION

Once we are done with EDA and feature engineering , we need to now select which all features are important to be considered for modeling.

Performing Various Feature Selection on the below DataSet:

```
url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['date'
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
#data['DateStr'] = data['date'].apply(lambda x: x.strftime('%Y%m%d%H%M'))
#data = data.loc[(data['Appliances'] < 900)]

d = data.date[0:len(data.date)]
data_final = []
for i in range (len(d)):
    if(i==0):
        a= 61200
        data_final.append(a)
    elif(i>0):
        a=a+600
        data_final.append(a)

data["NSM"] = pd.DataFrame({'NSM':data_final})
data1 = data.drop('date', 1)
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('WeekStatus',1)
data1 = data1.drop('time',1)
X = data1
y = data.Appliances
X.head(1)
# splitting X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=1)
```

Here are few methods that we have tried to come out with features that will provide with the best model output.

1. Recursive Feature Elimination :

First , the model takes few features into iteration and in that iteration , it provides weights to each feature and in the next iteration it takes some other features and compare the weight of the first iteration with the next iteration and eliminate the variables based on these weights. The one with maximum weights will be given as output depending on the number of features we want.

a) With Linear Regression model :

```
estimator = LinearRegression()
selector = RFE(estimator , 22 , step=1 )
selector = selector.fit(X,Y)
#selector.support_
ranking = selector.ranking_
#coeff = selector.coef_
#fi = selector.feature_importances_
print (sorted(zip(map(float, ranking), features)))
#print(sorted(zip(map(float, coeff), features)))
#print("          ")
#print(sorted(zip(map(float, fi), features)))
#print('ranking')
prediction = selector.predict(X)
scores = r2_score(Y , prediction)
print("          ")
print(scores)
```

```
[(1.0, 'Numerical_Week'), (1.0, 'RH_1'), (1.0, 'RH_2'), (1.0, 'RH_3'), (1.0, 'RH_4'), (1.0, 'RH_7'), (1.0, 'RH_8'), (1.0, 'T
1'), (1.0, 'T2'), (1.0, 'T3'), (1.0, 'T4'), (1.0, 'T6'), (1.0, 'T7'), (1.0, 'T8'), (1.0, 'T9'), (1.0, 'T_out'), (1.0, 'Tdewpoin
t'), (1.0, 'Windspeed'), (1.0, 'hour'), (1.0, 'lights'), (1.0, 'month'), (1.0, 'weekStatus'), (2.0, 'T5'), (3.0, 'RH_9'), (4.0,
'RH_out'), (5.0, 'Visibility'), (6.0, 'day'), (7.0, 'Press_mm_hg'), (8.0, 'RH_6'), (9.0, 'rv1'), (10.0, 'rv2'), (11.0, 'RH_5'),
(12.0, 'NSM')]

0.169190645744
```

In this , I am fitting the X and Y in the model(which is linearRegresion()) .

X has all my variables except the target that is Appliances.

Y has the target.

I am asking it to output 22 features.

Output - the 22 features with its rank which is based on how much it has impacted the appliances.

b) with RandomForestRegressor() :

```
estimator = RandomForestRegressor()
selector = RFE(estimator ,20, step=1 )
selector = selector.fit(X,Y)
#selector.support_
ranking = selector.ranking_
print (sorted(zip(map(float, ranking), features)))
print('          ')
prediction = selector.predict(X)
scores = r2_score(Y , prediction)
print(scores)
```

```
[(1.0, 'NSM'), (1.0, 'Press_mm_hg'), (1.0, 'RH_1'), (1.0, 'RH_2'), (1.0, 'RH_3'), (1.0, 'RH_4'), (1.0, 'RH_5'), (1.0, 'RH_6'),
(1.0, 'RH_7'), (1.0, 'RH_8'), (1.0, 'RH_out'), (1.0, 'T2'), (1.0, 'T3'), (1.0, 'T5'), (1.0, 'T8'), (1.0, 'T_out'), (1.0, 'Tdewp
oint'), (1.0, 'hour'), (1.0, 'lights'), (1.0, 'rv2'), (2.0, 'RH_9'), (3.0, 'Windspeed'), (4.0, 'T7'), (5.0, 'T4'), (6.0, 'T6'),
(7.0, 'Visibility'), (8.0, 'T1'), (9.0, 'rv1'), (10.0, 'day'), (11.0, 'T9'), (12.0, 'weekStatus'), (13.0, 'Numerical_Week'), (1
4.0, 'month')]

0.918020954333
```

I am doing the same thing as before. The only difference is i am using RandomForestRegressor() model in this case.

X , Y are same as before.

Output - I am asking for 20 variables and this is the output that it has given .

2.Feature Importance Method:

```
trees = ExtraTreesRegressor(n_estimators=250,
                            random_state=0)

trees = trees.fit(X, Y)
importances = trees.feature_importances_
print(sorted(zip(map(float ,importances),features)))
print('            ')
prediction = trees.predict(X)
#scores = oob_score(Y ,prediction)
#print(scores)
```

```
[(0.016246955592418733, 'month'), (0.018709158983822062, 'T9'), (0.0193705098715039, 'Numerical_Week'), (0.019643934750393395,
'weekStatus'), (0.021117532265162893, 'day'), (0.0219977219606682, 'rv2'), (0.02228638689572151, 'T7'), (0.022349244119737842,
'rv1'), (0.023915764574716714, 'Visibility'), (0.024266931987183212, 'T5'), (0.02448226921996136, 'NSM'), (0.02477360889404656
5, 'RH_7'), (0.02533353404456651, 'Tdewpoint'), (0.02550190523041148, 'T_out'), (0.02558887317395682, 'RH_4'), (0.0255945064691
01864, 'Press_mm_hg'), (0.025913936815239253, 'T4'), (0.02606494850108655, 'RH_6'), (0.02670825251714159, 'RH_9'), (0.026874839
200705807, 'T1'), (0.027411203014314364, 'RH_8'), (0.027567196824910916, 'T2'), (0.028712534411888056, 'T6'), (0.02953861266635
3773, 'RH_5'), (0.029806073535463943, 'T8'), (0.03122967901481279, 'RH_1'), (0.031706556443908705, 'Windspeed'), (0.03288195100
8522654, 'RH_2'), (0.03353832680553693, 'RH_3'), (0.034845374865078445, 'RH_out'), (0.037251048968137954, 'lights'), (0.0389458
77335933264, 'T3'), (0.149824750037592, 'hour')]
```

This is completely randomized tree which splits the X and the Y randomly .

n_estimators = number of trees in the model.I am fitting the variables in the model and they provide me the important features which are based on weights.

3. SelectKBest :

```
best_variables = SelectKBest(f_regression,k=20)
best_variables.fit(X,Y)
best_variables = best_variables.fit_transform(X,Y)
#print(zip(map(float ,best_variables) ,features))
best_variables
```

```
array([[ 30.        ,  19.89      ,  47.59666667, ...,  92.        ,
          7.        ,  17.        ],
       [ 30.        ,  19.89      ,  46.69333333, ...,  92.        ,
          6.66666667,  17.        ],
       [ 30.        ,  19.89      ,  46.3       , ...,  92.        ,
          6.33333333,  17.        ],
       ...,
       [ 10.        ,  25.5       ,  46.59666667, ...,  56.33333333,
          3.66666667,  17.        ],
       [ 10.        ,  25.5       ,  46.99      , ...,  56.66666667,
          3.83333333,  17.        ],
       [ 10.        ,  25.5       ,  46.6       , ...,  57.        ,
          4.        ,  18.        ]])
```

In this model , we fit the input and the target in the model which calculates the p_value for us. As this method doesn't tell us which feature values more , this is not cosidered.

We are using f_regression here. There are other estimators that can be used such as chi2 , f_classif , selectpercentile , selectfdr , selectfpr,selectfwe etc. but only f_regression suited our dataset. chi2 can also be used if there were no negative values. As T6 has negative values in it , chi2 method can't be used.

4. SelectFromModel :

```
lasso = Lasso(alpha=.05)
lasso.fit(X, Y)
Lasso = (np.abs(lasso.coef_), features)
print (Lasso)
```

```
(array([  1.92202561e+00,   3.75236632e+00,   1.45430896e+01,
         1.80860820e+01,   1.31094332e+01,   2.71830133e+01,
         4.12038495e+00,   2.20257987e+00,   1.01738977e+00,
         4.34037649e-02,   4.29283515e-02,   6.86899286e+00,
         4.34249006e-02,   6.29268266e-01,   1.50454528e+00,
         7.08519289e+00,   3.73542439e+00,   9.06159302e+00,
         6.37288879e-01,   7.31765826e+00,   1.42342007e-01,
         2.43972724e-01,   1.48410757e+00,   1.55961968e-01,
         2.40097934e+00,   4.27670800e-02,   2.10533855e-17,
         8.86260781e+00,   9.84385321e-01,   5.58805364e-01,
         1.59417650e+00,   2.04845823e-13,   8.27707964e-06]), Index(['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4',
'RH_4', 'T5',
       'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8', 'RH_8', 'T9', 'RH_9', 'T_out',
       'Press_mm_hg', 'RH_out', 'Windspeed', 'Visibility', 'Tdewpoint', 'rv1',
       'rv2', 'month', 'hour', 'day', 'Numerical_Week', 'weekStatus', 'NSM'],
      dtype='object'))
```

This method is based on Lasso regularization.Most of the time their coefficients will be zero due to which they are used in case of dimensionality reduction. Alpha is the deciding factor here. More the alpha value , less is the feature selected.Mainly used on linear models.

5. Low Variance(based on score) :

```
desired_features = VarianceThreshold(threshold=(.2 * (1 - 0.0)))
desired_features.fit_transform(X)
```

```
array([[  3.00000000e+01,   1.98900000e+01,   4.75966667e+01, ...,
          0.00000000e+00,   0.00000000e+00,   6.12000000e+04],
       [  3.00000000e+01,   1.98900000e+01,   4.66933333e+01, ...,
          0.00000000e+00,   0.00000000e+00,   6.18000000e+04],
       [  3.00000000e+01,   1.98900000e+01,   4.63000000e+01, ...,
          0.00000000e+00,   0.00000000e+00,   6.24000000e+04],
       ...,
       [  1.00000000e+01,   2.55000000e+01,   4.65966667e+01, ...,
          4.00000000e+00,   4.00000000e+00,   1.19004000e+07],
       [  1.00000000e+01,   2.55000000e+01,   4.69900000e+01, ...,
          4.00000000e+00,   4.00000000e+00,   1.19010000e+07],
       [  1.00000000e+01,   2.55000000e+01,   4.66000000e+01, ...,
          4.00000000e+00,   4.00000000e+00,   1.19016000e+07]])
```

Here it calculates the variance of each column and compares it with the variance that is provided in the input.

Any variance less that the variance of the given input is removed.

The biggest disadvantage about this technique is , it considers only one data frame. It either takes X or Y . It doesn't take both. So this method is mainly used for unsupervised learning and not for our dataset.

6. Forward Selection :

he simplest data-driven model building approach is called forward selection. In this approach, one adds variables to the model one at a time. At each step, each variable that is not already in the model is tested for inclusion in the model. The most significant of these variables is added to the model, so long as it's P-value is below some pre-set level. It is customary to set this value above the conventional .05 level at say .10 or .15, because of the exploratory nature of this method (see below).Thus we begin with a model including the variable that is most significant in the initial analysis, and continue adding variables until none of remaining variables are "significant" when added to the model. Note that this multiple use of hypothesis testing means that the real type I error rate for a variable (i.e. the chance of including it in the model given it isn't really necessary), does not equal the

critical level we choose. In fact, because of the complexity that arises from the complex nature of the procedure, it is essentially impossible to control error rates and this procedure must be viewed as exploratory.

In [13]:
```
def Forward_Selection(X, y,
                      initial_list=[],
                      threshold_in=0.01,
                      threshold_out = 0.05,
                      verbose=True):
    """ Perform a forward feature selection
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))
        if not changed:
            break
    return included

result = Forward_Selection(X.astype(float), y)

print('resulting features:')
print(result)
```

Output:

```
Add  hour                           with p-value 1.39406e-208
Add  lights                         with p-value 3.89728e-100
Add  RH_out                         with p-value 3.48902e-57
Add  RH_1                           with p-value 3.15155e-51
Add  RH_7                           with p-value 6.77888e-78
Add  RH_2                           with p-value 8.79214e-56
Add  RH_8                           with p-value 9.01866e-16
Add  Windspeed                      with p-value 3.26658e-14
Add  T3                             with p-value 4.21052e-19
Add  T2                             with p-value 1.03917e-107
Add  month                          with p-value 2.06426e-37
Add  T6                             with p-value 2.65568e-24
Add  T_out                          with p-value 5.09808e-11
Add  RH_3                           with p-value 2.44468e-10
Add  T8                             with p-value 2.3273e-09
Add  T9                             with p-value 1.32621e-16
Add  Numerical_Week                 with p-value 2.71363e-06
Add  weekStatus                     with p-value 2.71363e-06
Add  Visibility                     with p-value 0.00455741
resulting features:
['year', 'hour', 'lights', 'RH_out', 'RH_1', 'RH_7', 'RH_2', 'RH_8', 'Windspeed', 'T3', 'T2', 'month', 'T6', 'T_out', 'RH_3',
'T8', 'T9', 'Numerical_Week', 'weekStatus', 'Visibility']
```

7. Backward Selection :

Forward selection has drawbacks, including the fact that each addition of a new variable may render one or more of the already included variables non-significant. An alternate approach which avoids this is *backward selection*. Under this approach, one starts with fitting a model with all the variables of interest (following the initial screen). Then the least significant variable is dropped, so long as it is not significant at our chosen critical level. We continue by successively re-fitting reduced models and applying the same rule until all remaining variables are statistically significant.

```
def Backward_Elimination(X, y,
                         initial_list=[],
                         threshold_in=0.01,
                         threshold_out = 0.05,
                         verbose=True):

    included = list(initial_list)
    while True:
        changed=False
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included

result = Backward_Elimination(X.astype(float), y)

print('resulting features:')
print(result)
```

9 . Stepwise Regression : Performing both the above models together

It combines both Forward and Backward eliminations and drops/adds variables based on their statistical.

In [3]:
```
def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
             model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
             new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included

result = stepwise_selection(X.astype(float), y)

print('resulting features:')
print(result)
```

Output :

```
Add  hour                           with p-value 1.39406e-208
Add  lights                         with p-value 3.89728e-100
Add  RH_out                         with p-value 3.48902e-57
Add  RH_1                           with p-value 3.15155e-51
Add  RH_7                           with p-value 6.77888e-78
Add  RH_2                           with p-value 8.79214e-56
Add  RH_8                           with p-value 9.01866e-16
Drop RH_out                         with p-value 0.19489
Add  Windspeed                      with p-value 1.81232e-14
Add  T3                             with p-value 3.86665e-15
Add  T2                             with p-value 3.17576e-111
Add  month                          with p-value 7.46441e-38
Add  T6                             with p-value 1.59976e-16
Add  T_out                          with p-value 5.93633e-18
Add  RH_3                           with p-value 1.23206e-09
Add  weekStatus                     with p-value 1.29493e-09
Add  Numerical_Week                 with p-value 1.29493e-09
Add  T8                             with p-value 1.36905e-07
Add  T9                             with p-value 1.20799e-15
Add  Visibility                     with p-value 0.00257517
resulting features:
['year', 'hour', 'lights', 'RH_1', 'RH_7', 'RH_2', 'RH_8', 'Windspeed', 'T3', 'T2', 'month', 'T6', 'T_out', 'RH_3', 'weekStatu
s', 'Numerical_Week', 'T8', 'T9', 'Visibility']
```

Using TPOT on Energy Datasets:

- TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. TPOT will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for your data.
- Once TPOT is finished searching (or you get tired of waiting), it provides you with the Python code for the best pipeline it found so you can tinker with the pipeline from there.
- AutoML algorithms aren't as simple as fitting one model on the dataset; they are considering multiple machine learning algorithms (random forests, linear models, SVMs, etc.) in a pipeline with multiple preprocessing steps (missing value imputation, scaling, PCA, feature selection, etc.), the hyperparameters for all of the models and preprocessing steps, as well as multiple ways to ensemble or stack the algorithms within the pipeline.
- TPOT is meant to be an assistant that gives you ideas on how to solve a particular machine learning problem by exploring pipeline configurations that you might have never considered, then leaves the fine-tuning to more constrained parameter tuning techniques such as grid search.

Scripts:

In [2]:

```
url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['c
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
#data['DateStr'] = data['date'].apply(lambda x: x.strftime('%Y%m%d%H%M'))
d = data.date[0:len(data.date)]
data_final = []
for i in range (len(d)):
    if(i==0):
        a= 61200
        data_final.append(a)
    elif(i>0):
        a=a+600
        data_final.append(a)

data["NSM"] = pd.DataFrame({'NSM':data_final})

data1 = data.drop('date', 1)
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('WeekStatus',1)
data1 = data1.drop('time',1)

X = data1
y1 = data.Appliances
data1.head(5)
```

Out[2]:

	RH_1	T2	RH_2	T3	RH_3	T4	...	year	month	time	hour	day	day_of_week	Numerical_Week	weekStatus	WeekStatus	NSM
	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...	2016	1	17:00:00	17	11	Monday	0	0	Weekday	61200
	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	2016	1	17:10:00	17	11	Monday	0	0	Weekday	61800
	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	2016	1	17:20:00	17	11	Monday	0	0	Weekday	62400

```
In [4]:
lab_enc = preprocessing.LabelEncoder()
Y = lab_enc.fit_transform(y1)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.5, test_size=0.5)

tpot = TPOTRegressor(generations=5, verbosity=2)
tpot.fit(X_train, Y_train)
print(tpot.score(X_test, Y_test))

C:\Users\Dhanisha\AppData\Local\Continuum\anaconda3\lib\importlib\_bootstrap.py:219: ImportWarning: can't resolve package from
__spec__ or __package__, falling back on __name__ and __path__
  return f(*args, **kwds)

Generation 1 - Current best internal CV score: -54.09793812726714

Generation 2 - Current best internal CV score: -54.09793812726714

Generation 3 - Current best internal CV score: -54.09793812726714

Generation 4 - Current best internal CV score: -54.09793812726714

Generation 5 - Current best internal CV score: -54.09793812726714

Best pipeline: KNeighborsRegressor(input_matrix, n_neighbors=11, p=2, weights=distance)
-50.8263914892

In [5]:
tpot.export('tpot_exported_pipeline.py')

Out[5]:
True
```

Tpot gives KNeighbourRegressor as the best pipeline for the given dataset, when NSM is included.

Using TSFresh on Energy Datasets:

- Tsfresh is used to extract characteristics from time series. Time series often contain noise, redundancies or irrelevant information. As a result most of the extracted features will not be useful for the machine learning task at hand.
- To avoid extracting irrelevant features, the TSFRESH package has a built-in filtering procedure. This filtering procedure evaluates the explaining power and importance of each characteristic for the regression or classification tasks at hand.

Advantages of tsfresh:

- It is field tested

- The filtering process is statistically/mathematically correct
- It has a comprehensive documentation
- It is compatible with sklearn, pandas and numpy
- It allows anyone to easily add their favorite features
- It both runs on your local machine or even on a cluster

Script:

```
from sklearn import datasets, linear_model, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from subprocess import check_output
from datetime import time

url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['d
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
d = data.date[0:len(data.date)]
data_final = []
for i in range (len(d)):
    if(i==0):
        a= 61200
        data_final.append(a)
    elif(i>0):
        a=a+600
        data_final.append(a)

data["id"] = pd.DataFrame({'id':data_final})

data1 = data.drop('Appliances', 1)
data1 = data1.drop('Numerical_Week', 1)
data1 = data1.drop('WeekStatus', 1)
data1 = data1.drop('weekStatus',1)
data1 = data1.drop('time', 1)
data1 = data1.drop('hour', 1)
data1 = data1.drop('day_of_week',1)
data1.head(5)
```

Below screenshot shows the number of extra features extracted from already present labels.

In [29]:
```
#extracting features
from tsfresh import extract_features
from tsfresh.utilities.dataframe_functions import make_forecasting_frame
from sklearn.ensemble import AdaBoostRegressor
from tsfresh.utilities.dataframe_functions import impute
from tsfresh.feature_extraction import MinimalFCParameters
extracted_features = extract_features(data1, column_id="id", column_sort="date",show_warnings=False, default_fc_parameters=Mi
extracted_features
```

```
Feature Extraction: 100%|██████████████████████████████████████████████████| 10/10 [23:08<00:00, 138.86s/it]
```

Out[29]:

variable	Press_mm_hg__length	Press_mm_hg__maximum	Press_mm_hg__mean	Press_mm_hg__median	Press_mm_hg__minimum	Press_mm_hg__standard_de
id						
61200	1.0	733.500000	733.500000	733.500000	733.500000	
61800	1.0	733.600000	733.600000	733.600000	733.600000	
62400	1.0	733.700000	733.700000	733.700000	733.700000	
63000	1.0	733.800000	733.800000	733.800000	733.800000	
63600	1.0	733.900000	733.900000	733.900000	733.900000	
64200	1.0	734.000000	734.000000	734.000000	734.000000	
64800	1.0	734.100000	734.100000	734.100000	734.100000	
65400	1.0	734.166667	734.166667	734.166667	734.166667	
66000	1.0	734.233333	734.233333	734.233333	734.233333	
66600	1.0	734.300000	734.300000	734.300000	734.300000	

Set the target to Appliances

In [30]:
```
#creating target vector
y= data["Appliances"]
y.index = data1["id"]
y
```

Out[30]:
```
id
61200    60
61800    60
62400    50
63000    50
```

Relevant features have been filtered using Select_features package.

In [31]:
```
#selecting relavant features
from tsfresh import select_features
from tsfresh.utilities.dataframe_functions import impute

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

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

In [32]:
```
#spliting the feature and target into training and testing data
X_train, X_test, y_train, y_test = train_test_split(features_filtered, y, test_size=0.3, random_state=42)
```

The train , test set of feature and target is given to random forest model to validate the selected the feature

```
reg = RandomForestRegressor(n_estimators =100, random_state = 1,n_jobs=-1)
reg.fit(X_train, y_train)

#Doing predictions on training and test data
predicted_train=reg.predict(X_train)
predicted_test=reg.predict(X_test)
#data_test_predicted=reg.predict(data_test1)

#datapredict_test=reg.predict(data_test1)
print("predicted_train : " + str(predicted_train))
print("predicted_test : " + str(predicted_test))
#print("data_test_predicted : " + str(data_test_predicted))
#print("predicted_test : " + str(datapredict_test))


#### MAE Calculation of model
def mae(actual,prediction):
    return mean_absolute_error(actual,prediction)
test_mae=mae(y_test,predicted_test)
train_mae=mae(y_train,predicted_train)
print("test_mae : " + str(test_mae))
print("train_mae : " + str(train_mae))


#### RMSE Calculation of model
def rmse(actual,prediction):
    return np.sqrt(mean_squared_error(actual,prediction))
test_rmse = rmse(y_test,predicted_test)
train_rmse=rmse(y_train,predicted_train)
print("test_rmse : " + str(test_rmse))
print("train_rmse : " + str(train_rmse))
```

```
#### R Squared error calculation
test_r2=r2_score(y_test,predicted_test)
train_r2=reg.score(X_train,y_train)
print("test_r2 : " + str(test_r2))
print("train_r2 : " + str(train_r2))



#### Calculating MAPE
def mean_absolute_percentage_error(y_test,x_predict):
    np.seterr(divide='ignore',invalid='ignore')
    y_test,x_predict=np.array(y_test),np.array(x_predict)
    return np.mean(np.abs((y_test - x_predict)/y_test))*100
test_mape = mean_absolute_percentage_error(y_test, predicted_test)
train_mape = mean_absolute_percentage_error(y_train, predicted_train)
print("test_mape : " + str(test_mape))
print("train_mape : " + str(train_mape))



# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))
```

```
predicted_train : [  53.2   31.7   41.7 ...,   34.1   87.7  229.6]
predicted_test : [  52.   114.9   50.  ...,   51.5  108.8   58.6]
test_mae : 33.2334571863
train_mae : 12.7734327494
test_rmse : 70.1113645106
train_rmse : 27.1568841607
test_r2 : 0.522878390422
train_r2 : 0.930425238519
test_mape : 33.5516465259
```

Using Feature tools on Energy Datasets:

It is automated feature tools Transforms Transactional and relational datasets into feature matrices for machine learning. Deep Feature Synthesis (DFS) to performautomated feature engineering. DFS is used to create the “Data Science Machine” to automatically build predictive models for complex, multi-table datasets

Each table is called an entity in Featuretools. When 2 two entities have a one-to-many relationship, then “one” enitity, is called the “parent entity”. A relationship between a parent and child is defined like this:

(parent_entity, parent_variable, child_entity, child_variable)

A minimal input to DFS is a set of entities, a list of relationships, and the “target_entity” to calculate features for. The ouput of DFS is a feature matrix and the corresponding list of feature defintions.

Example:

feature_matrix_customers, features_defs = ft.dfs(entities=entities,

relationships=relationships,

target_entity="customers"

We can change target entity and get feature matrix for any entities of our choice.

Script:

In [5]:

```
import pandas as pd
import urllib
from urllib import request
import featuretools as ft
import numpy as np

url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-predictio
data = pd.read_csv(url)
data.head(5)
```

Out[5]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...
	2016-										

Converting dataset into Entity

```
25]:
1 entities = {
2     "energy" : (data, "id")
3 }
```

```
39]:
1 es=ft.EntitySet("my-entity-set", entities)
2 es
```

```
2018-03-14 19:11:11,554 featuretools.entityset - WARNING    index id not found
in dataframe, creating new integer column
```

```
39]: Entityset: my-entity-set
  Entities:
    energy (shape = [19735, 30])
  Relationships:
    No relationships
```

```
In [45]:
1 es["energy"].variables
```

```
Out[45]: [<Variable: date (dtype: datetime, format: None)>,
          <Variable: Appliances (dtype = numeric, count = 19735)>,
          <Variable: lights (dtype = numeric, count = 19735)>,
          <Variable: T1 (dtype = numeric, count = 19735)>,
          <Variable: RH_1 (dtype = numeric, count = 19735)>,
          <Variable: T2 (dtype = numeric, count = 19735)>,
          <Variable: RH_2 (dtype = numeric, count = 19735)>,
          <Variable: T3 (dtype = numeric, count = 19735)>,
          <Variable: RH_3 (dtype = numeric, count = 19735)>,
          <Variable: T4 (dtype = numeric, count = 19735)>
```

Creating new entity “lights” from original entity.

```
In [56]:
1 es
```

```
Out[56]: Entityset: my-entity-set
  Entities:
    energy (shape = [19735, 24])
    lights (shape = [92, 4])
  Relationships:
    energy.Appliances -> lights.Appliances
```

```
In [61]:
1 es["lights"].head()
```

Out[61]:

	Appliances	T3	T4	T5
Appliances				
10	10	19.700000	18.600000	17.100000
20	20	20.290000	20.533333	18.700000

Extracting feature matrix of target entity “lights”

```
In [ ]:  1 new_relationship = ft.Relationship(es["lights"]["Appliances"],
         2                                   es["energy"]["Appliances"])

In [63]: 1 feature_matrix, feature_defs = ft.dfs(entityset=es,
         2                                       target_entity="lights")
         3

In [64]: 1 feature_matrix
```

Out[64]:

Appliances	T3	T4	T5	SUM (energy.lights)	SUM (energy.RH_1)	SUM (energy.RH_2)	(energy
10	19.700000	18.600000	17.100000	10	388.211667	392.656667	377.(
20	20.290000	20.533333	18.700000	170	14075.958929	14260.389544	14008.:

Using Boruta on Energy Datasets

Boruta is an all relevant feature selection method. It is a wrapper built around the random forest classification algorithm. It tries to capture all the important, interesting features you might have in your dataset with respect to an outcome variable.

First, it duplicates the dataset, and shuffle the values in each column. These values are called shadow features. * Then, it trains a classifier, such as a Random Forest Classifier, on the dataset. By doing this, you ensure that you can an idea of the importance -via the Mean Decrease Accuracy or Mean Decrease Impurity- for each of the features of your data set. The higher the score, the better or more important.

Then, the algorithm checks for each of your real features if they have higher importance. That is, whether the feature has a higher Z-score than the maximum Z-score of its shadow features than the best of the shadow features. If they do, it records this in a vector. These are called a hits. Next,it will continue with another iteration. After a predefined set of iterations, you will end up with a table of these hits

At every iteration, the algorithm compares the Z-scores of the shuffled copies of the features and the original features to see if the latter performed better than the former.

Script:

Import data

Selecting features from unscaled data

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.feature_selection import RFE, f_regression
from sklearn.linear_model import (LinearRegression, Ridge, Lasso, RandomizedLasso)
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
import sklearn
from sklearn import preprocessing
```

In [3]:
```
url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['d
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
#data['DateStr'] = data['date'].apply(lambda x: x.strftime('%Y%m%d%H%M'))
d = data.date[0:len(data.date)]
data_final = []
#print(data)
for i in range (len(d)):
    if (i==0):
        a = 61200
        data_final.append(a)
    elif (i > 0 ):
        data_final.append(a)
#print((data))
#df['data_converted'] = data
data['NSM'] = pd.DataFrame({'NSM' : data_final})

data.head(5)
```

Out[3]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	year	month	time	hour	day	day_of_week	Numerical_W
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...	2016	1	17:00:00	17	11	Monday	
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	2016	1	17:10:00	17	11	Monday	
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	2016	1	17:20:00	17	11	Monday	
	2016-																	

In [4]:
```
data1 = data.set_index('date')
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('WeekStatus',1)
data1 = data1.drop('time',1)
data1.head(5)

# Load X and y
X = data1.values
y = data.Appliances.values
```

In [5]:
```
from boruta import BorutaPy


rf = RandomForestRegressor(n_jobs=-1, max_depth=5)

# define Boruta feature selection method
feat_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=1)

# find all relevant features - 5 features should be selected
feat_selector.fit(X, y)

# check selected features - first 5 features are selected
feat_selector.support_

# check ranking of features
feat_selector.ranking_

# call transform() on X to filter it down to selected features
X_filtered = feat_selector.transform(X)
print(X_filtered)
```

```
Iteration:      1 / 100
Confirmed:      0
Tentative:      34
Rejected:       0
Iteration:      2 / 100
Confirmed:      0
Tentative:      34
```

In [6]:
```
print ('\n Initial features: ', data1.columns.tolist() )
# number of selected features
print ('\n Number of selected features:')
print (feat_selector.n_features_)
```

```
 Initial features:  ['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_
7', 'T8', 'RH_8', 'T9', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed', 'Visibility', 'Tdewpoint', 'rv1', 'rv2', 'year',
'month', 'hour', 'day', 'Numerical_Week', 'weekStatus', 'DateStr']

 Number of selected features:
16
```

List of feature selected

In [8]:
```
feature_df = pd.DataFrame(data1.columns.tolist(), columns=['features'])
feature_df
feature_df['rank']=feat_selector.ranking_
feature_df = feature_df.sort_values('rank', ascending=True).reset_index(drop=True)
print ('\n Top %d features:' % feat_selector.n_features_)
print (feature_df.head(feat_selector.n_features_))

#print ('\n Feature ranking:')
#print (feat_selector.ranking_)
```

```
 Top 16 features:
       features  rank
0        lights     1
1          hour     1
2     Tdewpoint     1
3        RH_out     1
4   Press_mm_hg     1
5            T9     1
6            T8     1
7          RH_7     1
8          RH_5     1
9       DateStr     1
10         RH_4     1
11           T5     1
12         RH_2     1
13           T3     1
14         RH_3     1
15           T2     1
```

In [9]:
```
# check selected features
print ('\n Selected features:')
print (feat_selector.support_)
```

```
 Selected features:
[ True False False  True  True  True  True False  True  True  True False
 False False  True  True False  True False False  True  True False False
  True False False False False  True False False False  True]
```

In [10]:
```
#check weak features
print ('\n Support for weak features:')
print (feat_selector.support_weak_)
```

```
 Support for weak features:
[False False False False False False False False False False False False
  True False False False False False  True False False False False False
 False False False False False False False False False False]
```

Total 16 features are selected using scaled data

Selecting features by scaling complete data

ı [5]:
```
a1 = data.Appliances
a2 = a1.values
a3 = a2.reshape((len(a2), 1))
min_max_scaler = preprocessing.MinMaxScaler()
appliances_scaler = min_max_scaler.fit_transform(a3)
data['appliances_scaler']  = appliances_scaler
data.head(5)

data1 = data.drop('date', 1)
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('WeekStatus',1)
data1 = data1.drop('appliances_scaler',1)
data1 = data1.drop('time',1)
data1.head(5)


#scaling data using MinMAX scaler
X_scale = data1.values
X= min_max_scaler.fit_transform(X_scale)
y = data.appliances_scaler.values
```

In [14]:
```
from boruta import BorutaPy

rf = RandomForestRegressor(n_jobs=-1, max_depth=5)

# define Boruta feature selection method
feat_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=1)

# find all relevant features - 5 features should be selected
feat_selector.fit(X, y)

# check selected features - first 5 features are selected
feat_selector.support_

# check ranking of features
feat_selector.ranking_

# call transform() on X to filter it down to selected features
X_filtered = feat_selector.transform(X)
print(X_filtered)
```

```
Iteration:      1 / 100
Confirmed:      0
Tentative:      34
Rejected:       0
Iteration:      2 / 100
```

In [15]:
```
print ('\n Initial features: ', data1.columns.tolist() )
# number of selected features
print ('\n Number of selected features:')
print (feat_selector.n_features_)
```

```
 Initial features:  ['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_
7', 'T8', 'RH_8', 'T9', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed', 'Visibility', 'Tdewpoint', 'rv1', 'rv2', 'year',
'month', 'hour', 'day', 'Numerical_Week', 'weekStatus', 'DateStr']

 Number of selected features:
13
```

In [16]:
```
feature_df = pd.DataFrame(data1.columns.tolist(), columns=['features'])
feature_df
feature_df['rank']=feat_selector.ranking_
feature_df = feature_df.sort_values('rank', ascending=True).reset_index(drop=True)
print ('\n Top %d features:' % feat_selector.n_features_)
print (feature_df.head(feat_selector.n_features_))

```

```
 Top 13 features:
     features  rank
0      lights     1
1        hour     1
2   Tdewpoint     1
3      RH_out     1
4          T8     1
5        RH_7     1
6        RH_5     1
7        RH_4     1
8     DateStr     1
9          T2     1
10       RH_3     1
11         T3     1
12       RH_2     1
```

In [ ]:
```
1
```

Total 13 features are selected using scaled data

PART 5 : Model Validation and Selection

Hyperparameters and Model Validation

Basic recipe for applying a supervised machine learning model:

1. Choose a class of model

2. Choose model hyperparameters

3. Fit the model to the training data

4. Use the model to predict labels for new data

we need a way to validate that our model and our hyperparameters are a good fit to the data

Model validation via cross-validation

- One disadvantage of using a holdout set for model validation is that we have lost a portion of our data to the model training. This is not optimal, and can cause problems – especially if the initial set of training data is small.
- One way to address this is to use cross-validation; that is, to do a sequence of fits where each subset of the data is used both as a training set and as a validation set.
- Form of cross-validation is a two-fold cross-validation—that is, one in which we have split the data into two sets and used each in turn as a validation set. We could expand on this idea to use even more trials, and more folds in the data. Split the data into five groups, and use each of them in turn to evaluate the model fit on the other 4/5 of the data. This would be rather tedious to do by hand, and so we can use Scikit-Learn's cross_val_score convenience routine to do it succinctly.
- Scikit-Learn implements a number of useful cross-validation schemes that are useful in particular situations; these are implemented via iterators in the cross_validation module.

Random Forest Model validation using hyper parameter, pipeline and Grid search cross validation

Script:

In [14]:

```
import pandas as pd
import urllib
from urllib import request
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import roc_auc_score
from sklearn import preprocessing
from sklearn.preprocessing import label_binarize
import numpy as np
import sklearn
from random import seed
from random import randrange
from math import sqrt
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model, metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.cross_validation import train_test_split
from sklearn.metrics import r2_score
from sklearn import datasets, linear_model, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from subprocess import check_output
from datetime import time
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn import feature_selection


url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['d
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
```

```
url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['d
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')

d = data.date[0:len(data.date)]
data_final = []
for i in range (len(d)):
    if(i==0):
        a= 61200
        data_final.append(a)
    elif(i>0):
        a=a+600
        data_final.append(a)

data["NSM"] = pd.DataFrame({'NSM':data_final})
data1 = data.set_index('date', 1)
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('WeekStatus',1)
data1 = data1.drop('time',1)
data1.head(5)
```

4]:

date	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	T5	...	Tdewpoint	rv1	rv2	year	month	hour
2016-01-11 17:00:00	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	17.166667	...	5.3	13.275433	13.275433	2016	1	17
2016-01-11 17:10:00	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	17.166667	...	5.2	18.606195	18.606195	2016	1	17

Defining feature matrix(X) and response vector(y)

In [16]:
```
X = data1
y = data.Appliances

# splitting X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
                                                    random_state=1)
```

Creating Pipeline with feature selection , prepocessing scaler and model estimator

In [4]:
```
est= RandomForestRegressor()
minmax_scaler = preprocessing.MinMaxScaler()
selector = feature_selection.RFE(est)
pipe_params = [('feat_selection',selector),('std_scaler', minmax_scaler), ('clf', est)]
pipe = Pipeline(pipe_params)
```

Performing Grid search cross validation with hyperparameter tuning

In [6]:
```
# param_grid = {'clf__n_estimators' : [100, 50, 200],
#                   'clf__max_features' : ['auto', 'sqrt', 'log2'],
#                  'clf__max_depth': [None, 5, 3, 1] }

# Declare hyperparameters to tune
param_grid = {'clf__n_estimators' : [100, 50, 200] }
clf1 = GridSearchCV(pipe, param_grid=param_grid, cv=10)
clf1.fit(X_train, y_train)
```

Out[6]:
```
GridSearchCV(cv=10, error_score='raise',
```

Predicting the target and calculating R sqaure, MAE, RMSE and MAPE

20]:
```
#Doing predictions on training and test data
predicted_train=clf1.predict(X_train)
predicted_test=clf1.predict(X_test)
#data_test_predicted=reg.predict(data_test1)

#datapredict_test=reg.predict(data_test1)
print("predicted_train : " + str(predicted_train))
print("predicted_test : " + str(predicted_test))
#print("data_test_predicted : " + str(data_test_predicted))
#print("predicted_test : " + str(datapredict_test))


#### MAE Calculation of model
def mae(actual,prediction):
    return mean_absolute_error(actual,prediction)
test_mae=mae(y_test,predicted_test)
train_mae=mae(y_train,predicted_train)
print("test_mae : " + str(test_mae))
print("train_mae : " + str(train_mae))


#### RMSE Calculation of model
def rmse(actual,prediction):
    return np.sqrt(mean_squared_error(actual,prediction))
test_rmse = rmse(y_test,predicted_test)
train_rmse=rmse(y_train,predicted_train)
print("test_rmse : " + str(test_rmse))
print("train_rmse : " + str(train_rmse))

#### R Squared error calculation
test_r2=r2_score(y_test,predicted_test)
train_r2=clf1.score(X_train,y_train)
print("test_r2 : " + str(test_r2))
print("train_r2 : " + str(train_r2))
```

```

#### Calculating MAPE
def mean_absolute_percentage_error(y_test,x_predict):
    np.seterr(divide='ignore',invalid='ignore')
    y_test,x_predict=np.array(y_test),np.array(x_predict)
    return np.mean(np.abs((y_test - x_predict)/y_test))*100
test_mape = mean_absolute_percentage_error(y_test, predicted_test)
train_mape = mean_absolute_percentage_error(y_train, predicted_train)
print("test_mape : " + str(test_mape))
print("train_mape : " + str(train_mape))



# variance score: 1 means perfect prediction
print('Variance score: {}'.format(clf1.score(X_test, y_test)))
```

```
predicted_train : [  51.85   33.65   76.   ...,  125.6    61.5    96.05]
predicted_test : [  39.8    60.35  114.4  ...,   95.9   356.05  110.65]
test_mae : 31.1613409897
train_mae : 11.5746054727
test_rmse : 67.8133787658
train_rmse : 25.3815825741
test_r2 : 0.564797904068
train_r2 : 0.938562965494
test_mape : 29.5965124311
train_mape : 11.2258828782
Variance score: 0.56479790406795
```

Random Forest with Recursive feature elimination (RFE) feature selection

Script:

In [5]:

```
import pandas as pd
import urllib
from urllib import request
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import roc_auc_score
from sklearn import preprocessing
from sklearn.preprocessing import label_binarize
import sklearn
from random import seed
from random import randrange
from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.cross_validation import train_test_split
from sklearn.metrics import r2_score
from sklearn import datasets, linear_model, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from subprocess import check_output
from datetime import time
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV

url = "https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-data/master/energydata_complete.csv"
data = pd.read_csv(url)
data['date'] = pd.to_datetime(data['date'])
data['year'], data['month'] , data['time'] , data['hour'] ,data['day'] , data['day_of_week'],data['Numerical_Week'] = data['c
data['weekStatus'] = data['date'].dt.dayofweek
data['WeekStatus'] = np.where(data['weekStatus'] < 5, 'Weekday', 'Weekend')
data['DateStr'] = data['date'].apply(lambda x: x.strftime('%Y%m%d%H%M'))
data.head(5)
```

Dropping irrelevant columns

n [3]:

```
data1 = data.drop('date', 1)
data1 = data1.drop('Appliances', 1)
data1 = data1.drop('day_of_week',1)
data1 = data1.drop('weekStatus',1)
data1 = data1.drop('time',1)
data1 = data1.drop('T1',1)
data1 = data1.drop('T4',1)
data1 = data1.drop('T6',1)
data1 = data1.drop('T7',1)
data1 = data1.drop('T_out',1)
data1 = data1.drop('RH_1',1)
data1 = data1.drop('RH_6',1)
data1 = data1.drop('RH_8',1)
data1 = data1.drop('RH_9',1)
data1 = data1.drop('rv1',1)
data1 = data1.drop('rv2',1)
data1 = data1.drop('year',1)
data1 = data1.drop('month',1)
data1 = data1.drop('day',1)
data1 = data1.drop('Numerical_Week',1)
data1 = data1.drop('Windspeed',1)
data1 = data1.drop('Visibility',1)
data1 = data1.drop('WeekStatus',1)

data1.head(5)
```

ut[3]:

	lights	T2	RH_2	T3	RH_3	RH_4	T5	RH_5	RH_7	T8	T9	Press_mm_hg	RH_out	Tdewpoint	hour	DateStr
0	30	19.2	44.790000	19.79	44.730000	45.566667	17.166667	55.20	41.626667	18.2	17.033333	733.5	92.0	5.3	17	201601111700
1	30	19.2	44.722500	19.79	44.790000	45.992500	17.166667	55.20	41.560000	18.2	17.066667	733.6	92.0	5.2	17	201601111710
2	30	19.2	44.626667	19.79	44.933333	45.890000	17.166667	55.09	41.433333	18.2	17.000000	733.7	92.0	5.1	17	201601111720

In [6]:

```
# defining feature matrix(X) and response vector(y)
X = data1
y = data.Appliances

# splitting X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
                                                    random_state=1)

pipeline = make_pipeline(preprocessing.MinMaxScaler(),
                         RandomForestRegressor(n_estimators=100))
# Declare hyperparameters to tune
hyperparameters = { 'randomforestregressor__max_features' : ['auto', 'sqrt', 'log2'],
                   'randomforestregressor__max_depth': [None, 5, 3, 1]}
clf = GridSearchCV(pipeline, hyperparameters, cv=10)

clf.fit(X_train, y_train)

# Evaluate model pipeline on test data
pred = clf.predict(X_test)
```

Predicting the target and calculating R sqaure, MAE, RMSE and MAPE

In [ ]:

```
#Doing predictions on training and test data
predicted_train=clf1.predict(X_train)
predicted_test=clf1.predict(X_test)
#data_test_predicted=reg.predict(data_test1)

#datapredict_test=reg.predict(data_test1)
print("predicted_train : " + str(predicted_train))
print("predicted_test : " + str(predicted_test))
#print("data_test_predicted : " + str(data_test_predicted))
#print("predicted_test : " + str(datapredict_test))


#### MAE Calculation of model
def mae(actual,prediction):
    return mean_absolute_error(actual,prediction)
test_mae=mae(y_test,predicted_test)
train_mae=mae(y_train,predicted_train)
print("test_mae : " + str(test_mae))
print("train_mae : " + str(train_mae))


#### RMSE Calculation of model
def rmse(actual,prediction):
    return np.sqrt(mean_squared_error(actual,prediction))
test_rmse = rmse(y_test,predicted_test)
train_rmse=rmse(y_train,predicted_train)
print("test_rmse : " + str(test_rmse))
print("train_rmse : " + str(train_rmse))

#### R Squared error calculation
test_r2=r2_score(y_test,predicted_test)
train_r2=clf1.score(X_train,y_train)
print("test_r2 : " + str(test_r2))
print("train_r2 : " + str(train_r2))
```

Cross validation techniques

Cross Validation is a very useful technique for assessing the effectiveness of your model, particularly in cases where you need to mitigate overfitting. It is also of use in determining the hyper parameters of your model, in the sense that which parameters will result in lowest test error.

An error estimation for the model is made after training, better known as evaluation of residuals.

In this process, a numerical estimate of the difference in predicted and original responses is done, also called the training error. However, this only gives us an idea about how well our model does on data used to train it. o, the problem with this evaluation technique is that it does not give an indication of how well the learner will generalize to an independent/ unseen data set. Getting this idea about our model is known as Cross Validation.

1.Holdout Method

The error estimation then tells how our model is doing on unseen data or the validation set. This is a simple kind of cross validation technique, also known as the holdout method. It still suffers from issues of high variance. This is because it is not certain which data points will end up in the validation set and the result might be entirely different for different sets.

2.K-Fold Cross Validation

By reducing the training data, we risk losing important patterns/ trends in data set, which in turn increases error induced by bias. In K Fold cross validation, the data is divided into k subsets.The holdout method is repeated k times, such that each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set. This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set. As a general rule and empirical evidence, K = 5 or 10 is generally preferred

3.Stratified K-Fold Cross Validation

A slight variation in the K Fold cross validation technique is made, such that each fold contains approximately the same percentage of samples of each target class as the complete set, or in case of prediction problems, the mean response value is approximately equal in all the folds. This variation is also known as Stratified K Fold.

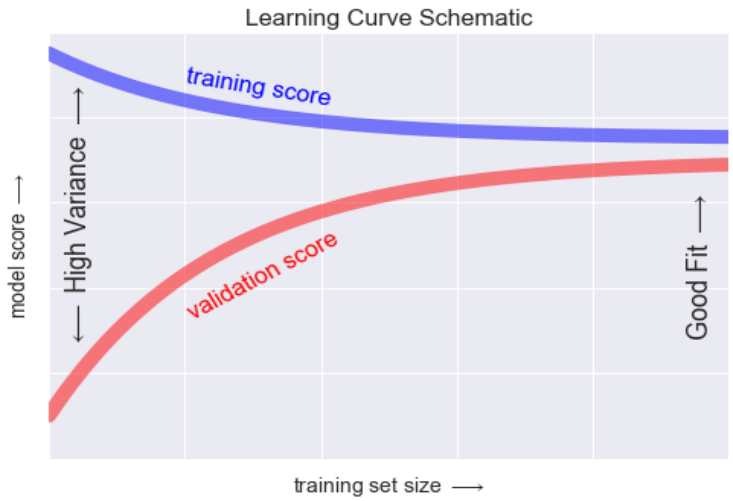
4.Leave-P-Out Cross Validation

This approach leaves p data points out of training data, i.e. if there are n data points in the original sample then, n-p samples are used to train the model and p points are used as the validation set. This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness. A particular case of this method is when p = 1. This is known as Leave one out cross validation.

Bias -variance tradeoff

- The score here is the R2 score, or coefficient of determination, which measures how well a model performs relative to a simple mean of the target values. R2=1 indicates a perfect match, R2=0 indicates the model does no better than simply taking the mean of the data, and negative values mean even worse models. From the scores associated with these two models, we can make an observation that holds more generally:
- For high-bias models, the performance of the model on the validation set is similar to the performance on the training set
- For high-variance models, the performance of the model on the validation set is far worse than the performance on the training set
- The training score is everywhere higher than the validation score. This is generally the case: the model will be a better fit to data it has seen than to data it has not seen
- For very low model complexity (a high-bias model), the training data is under-fit, which means that the model is a poor predictor both for the training data and for any previously unseen data
- For very high model complexity (a high-variance model), the training data is over-fit, which means that the model predicts the training data very well, but fails for any previously unseen data.

- For some intermediate value, the validation curve has a maximum. This level of complexity indicates a suitable trade-off between bias and variance

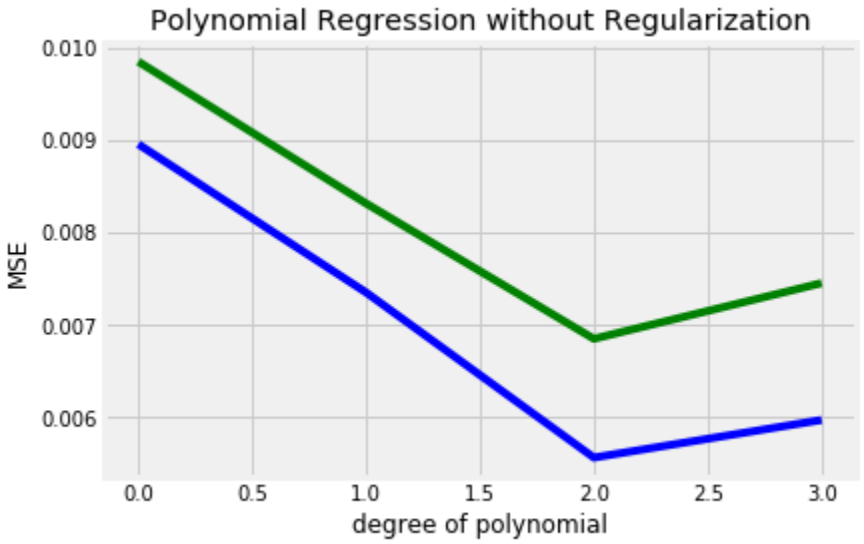


Regularization (L1, L2, Elastic net)[Performed on Multiple Linear Regression]

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 6, 4
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import *

X = data2
y = data1.appliances_scaler

X_train, X_test, y_train, y_test = train_test_split(X, y, randc

train_mse_list = []
test_mse_list = []
degree_of_polynomial = []

for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    # Transfer the X to a polynomial form by using fit_transfor
    X_train_ = model.fit_transform(X_train)
    X_test_ = model.fit_transform(X_test)
    lm = linear_model.LinearRegression()
    lm.fit(X_train_, y_train)
    train_pred = lm.predict(X_train_)
    train_mse_list.append(mean_squared_error(y_train, train_pre
    test_pred = lm.predict(X_test_)
    test_mse_list.append(mean_squared_error(y_test, test_pred))
    degree_of_polynomial.append(i)
```

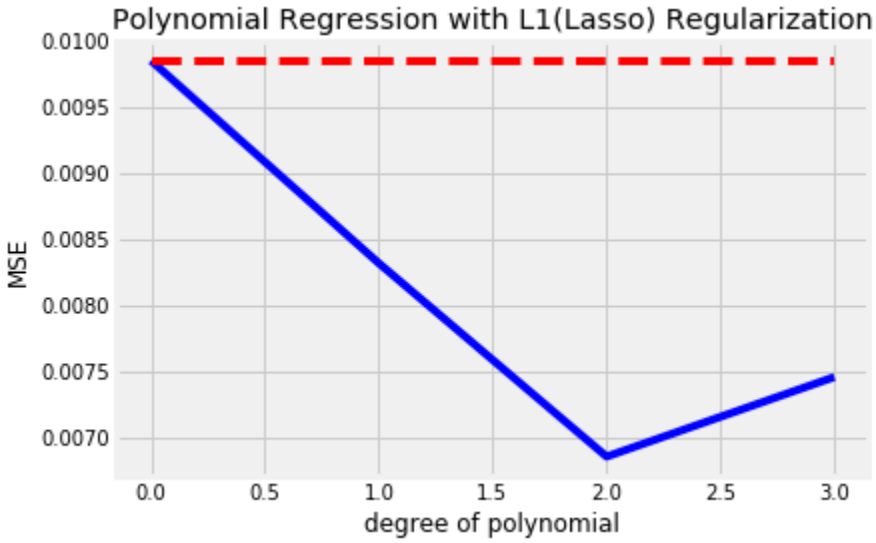
Output:



Using Lasso(L1)

In [36]:
```
# Use Lasso(L1) regularization to fix overfitting

from sklearn.linear_model import Lasso

# Randomly pick a alpha value for regularization
l1reg = Lasso(alpha=0.005, normalize=True)

l1reg_test_mse_list = []

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

    l1reg.fit(X_train_, y_train)
    test_pred_l1 = l1reg.predict(X_test_)
    l1reg_test_mse_list.append(mean_squared_error(y_test, test_pred_l1))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression with L1(Lasso) Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, l1reg_test_mse_list, '--r')
plt.show()
```

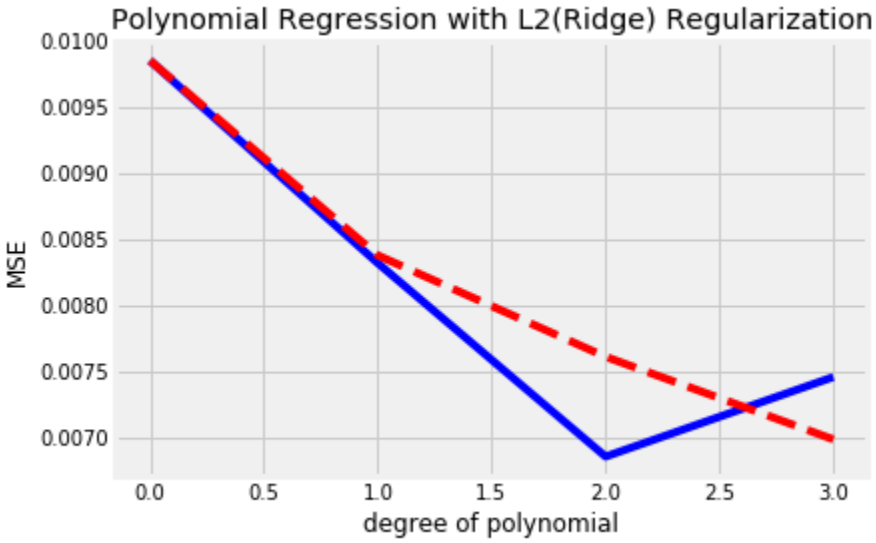
Output:



Using L2(Ridge)

In [37]:
```
# Use Ridge(L2) regularization to fix overfitting

from sklearn.linear_model import Ridge

l2reg = Ridge(alpha=0.005, normalize=True)

l2reg_test_mse_list = []

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

    l2reg.fit(X_train_, y_train)
    test_pred_l2 = l2reg.predict(X_test_)
    l2reg_test_mse_list.append(mean_squared_error(y_test, test_pred_l2))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression with 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
```

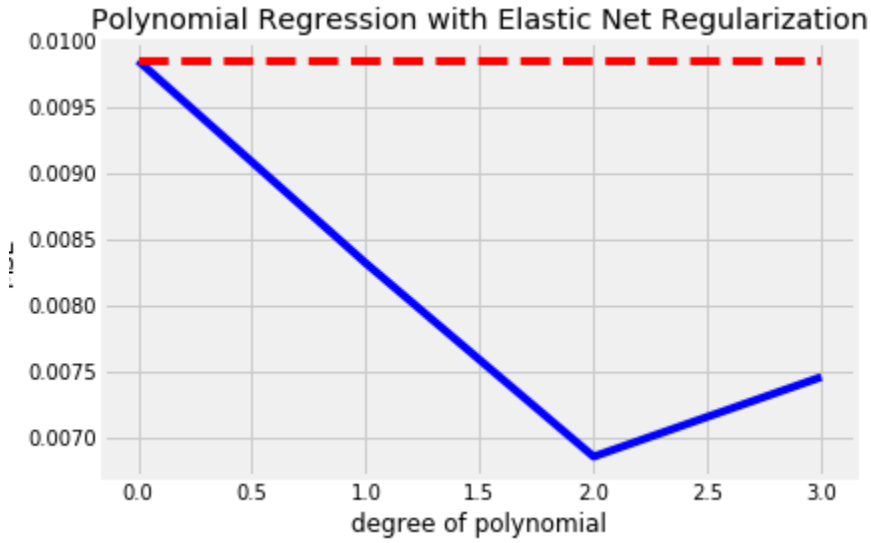
Output:



Using Elastic Net

In [38]:
```
# Use Elastic Net regularization to fix overfitting

from sklearn.linear_model import ElasticNet

enreg = ElasticNet(alpha=0.005, normalize=True)

enreg_test_mse_list = []

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

    enreg.fit(X_train_, y_train)
    test_pred_en = enreg.predict(X_test_)
    enreg_test_mse_list.append(mean_squared_error(y_test, test_pred_en))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression with 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
```

Output:



Calculating Better Alpha

```
from sklearn import model_selection

# Use cross_val_score to measure the generalization performance of the model. We want to get the optimal alpha
# which maximums the cross_val_score
# Set the fold to 5 by "cv=5" since we does not want a high computation
lm_score = np.mean(model_selection.cross_val_score(lm, X_train, y_train, n_jobs=1, cv=5))
print('The generalization score of linear regression model is %f' % np.mean(lm_score))

# Since alpha=0.005 is not that good, we set it as a lower bound to find a better value. You can also set it as a
# higher bound which does not matter.
alphas = np.arange(5e-5, 1e-3, 5e-5)

scores = []
scores_std = []

# Let's just set the degree of polynomial as 2 to simplify the calculation

model = PolynomialFeatures(degree=2)
X_train_ = model.fit_transform(X_train)

pm_score = np.mean(model_selection.cross_val_score(lm, X_train_, y_train, n_jobs=1, cv=5))
print('The generalization score of quadratic regression model is %f' % np.mean(pm_score))

# 5-fold CV will train the same alpha 5 times on 5 different train sets and return 5 different models.
# Then it will test these 5 models on corresponding test sets to get the cross validation scores.
# Average the scores as the final score of the given alpha.

for alpha in alphas:
    l1reg.alpha = alpha
    this_scores = model_selection.cross_val_score(l1reg, X_train_, y_train, n_jobs=1, cv=5)
    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')
```

```
]: # Set the alpha as the optimal value we got from the last step

   optimal_l1reg = Lasso(alpha=0.04, normalize=True)

   opt_l1reg_test_mse_list = []

   for i in range(0, 4):
       model = PolynomialFeatures(degree=i)
       X_train_ = model.fit_transform(X_train)
       X_test_ = model.fit_transform(X_test)
       optimal_l1reg.fit(X_train_, y_train)
       test_opt_pred_l1 = optimal_l1reg.predict(X_test_)
       opt_l1reg_test_mse_list.append(mean_squared_error(y_test, test_opt_pred_l1))

   print('MSE of linear regression model is %f' % test_mse_list[1])
   print('MSE of quadratic regression model is %f' % test_mse_list[2])
   print('MSE of optimal L1 regularized quadratic regression model is %f' % opt_l1reg_test_mse_list[2])


   plt.xlabel('degree of polynomial')
   plt.ylabel('MSE')
   plt.ylim((0.35, 0.65))
   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',
            degree_of_polynomial, opt_l1reg_test_mse_list, '--g')
   plt.show()

   # Green dash line is the testing error after L1 regularization with optimal alpha. We can see that it is lower
```

```
# L1 encourages sparsity

model = PolynomialFeatures(degree=2)
X_train_ = model.fit_transform(X_train)
X_test_ = model.fit_transform(X_test)
optimal_l1reg.fit(X_train_, y_train)
lm.fit(X_train_, y_train)

quadratic_features = model.get_feature_names(data1.columns)
total_coef = len(model.get_feature_names(data1.columns))
nz_lm_coef = 0
l1_coef_dict = {}
for i in range(total_coef):
    # if the coefficient is not zero, add 1 to the count.
    if lm.coef_.T[i] != 0.0:
        nz_lm_coef += 1
    if optimal_l1reg.coef_[i] != 0.0:
        l1_coef_dict[quadratic_features[i]] = optimal_l1reg.coef_[i].round(10)

# Let's see how many features could be used in a quadratic model.
print(quadratic_features)

# And let's see how many non-zero coefficients of features (which means the features the model actually uses)
# in a quadratic model without L1 regularization.
print('Before Lasso regularization, there are %i nonzero coefficients in quadratic regression model.' % nz_lm_coef)

print ('After Lasso regularization, there are only %i nonzero coefficients in quadratic regression model.' % len(l1_coef_dict))
```

CONCLUSION:

The pipeline is created successfully with the help of Random Forest Regressor

References :

http://epistasislab.github.io/tpot/

http://www.randalolson.com/2016/05/08/tpot-a-python-tool-for-automating-data-science/

https://github.com/EpistasisLab/tpot

https://github.com/scikit-learn-contrib/boruta_py

https://www.kaggle.com/tilii7/boruta-feature-elimination

https://pypi.python.org/pypi/Boruta/0.1.5

http://tsfresh.readthedocs.io/en/latest/text/introduction.html

https://github.com/blue-yonder/tsfresh

https://github.com/EpistasisLab/scikit-mdr

https://epistasislab.github.io/scikit-rebate/

https://github.com/EpistasisLab/scikit-rebate

http://scikit-learn.org/stable/

https://elitedatascience.com/overfitting-in-machine-learning

https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f

http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/

https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991

https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674

https://chrisalbon.com/machine_learning/trees_and_forests/random_forest_classifier_example/

https://spark.apache.org/docs/latest/ml-classification-regression.html

http://bigdata-madesimple.com/dealing-with-unbalanced-class-svm-random-forest-and-decision-tree-in-python/

http://benalexkeen.com/feature-scaling-with-scikit-learn/

http://sebastianraschka.com/Articles/2014_about_feature_scaling.html

https://machinelearningmastery.com/rescaling-data-for-machine-learning-in-python-with-scikit-learn/