Assignment2 Report

Team 10

Advanced Data Science

Nov 2018

Appliances Energy Prediction

Our team do the research on an energy data set of a house and the purpose is to analysis this data set and use the data to build a prediction model of the energy consumption. The data set is in https://github.com/LuisM78/Appliances-energy-prediction-data .And use the energydata_complete.csv for further analysis.

Part 1: Load the data and description:

This is a first look of the dataset:

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

From the df.shape we know that the data set contain 19735 rows and 29 columns. For each row, the date feature indicate the specific year, month, day, hour, minute, second. It's very precise but not a good view, so we will add 3 more features based on date and have a better look.

By adding new columns that extract the information from ‘date’ columns, those information is more useable and have better meaning.

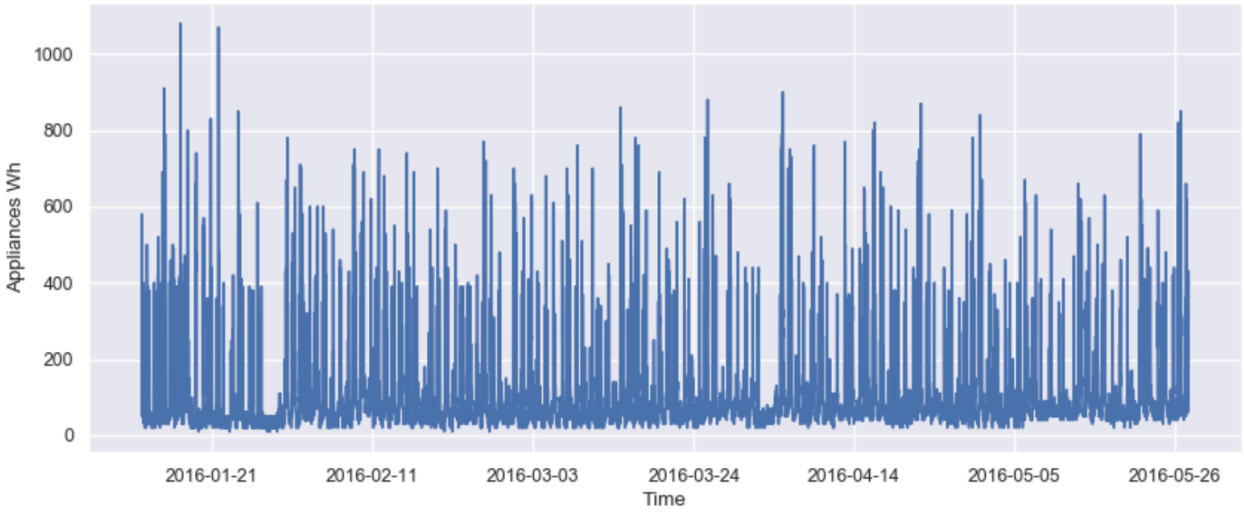
NSM	Week_Status	Day_of_Week
61200.0	Weekday	Mon
61800.0	Weekday	Mon
62400.0	Weekday	Mon
63000.0	Weekday	Mon
63600.0	Weekday	Mon

Thus, we add Number of Second from Midnight, Week_Status and Day_of_Week three new columns. These feature will be useful for grouping data and EDA.
Before analysis the dataset let's make sure that the dataset have no missing value:

```
Column wise count of null values:–
date            0
Appliances      0
lights          0
T1              0
RH_1            0
T2              0
RH_2            0
T3              0
RH_3            0
T4              0
RH_4            0
T5              0
RH_5            0
T6              0
RH_6            0
T7              0
RH_7            0
T8              0
RH_8            0
T9              0
RH_9            0
T_out           0
Press_mm_hg     0
RH_out          0
Windspeed       0
Visibility      0
Tdewpoint       0
rv1             0
rv2             0
NSM             0
Week_Status     0
Day_of_Week     0
```

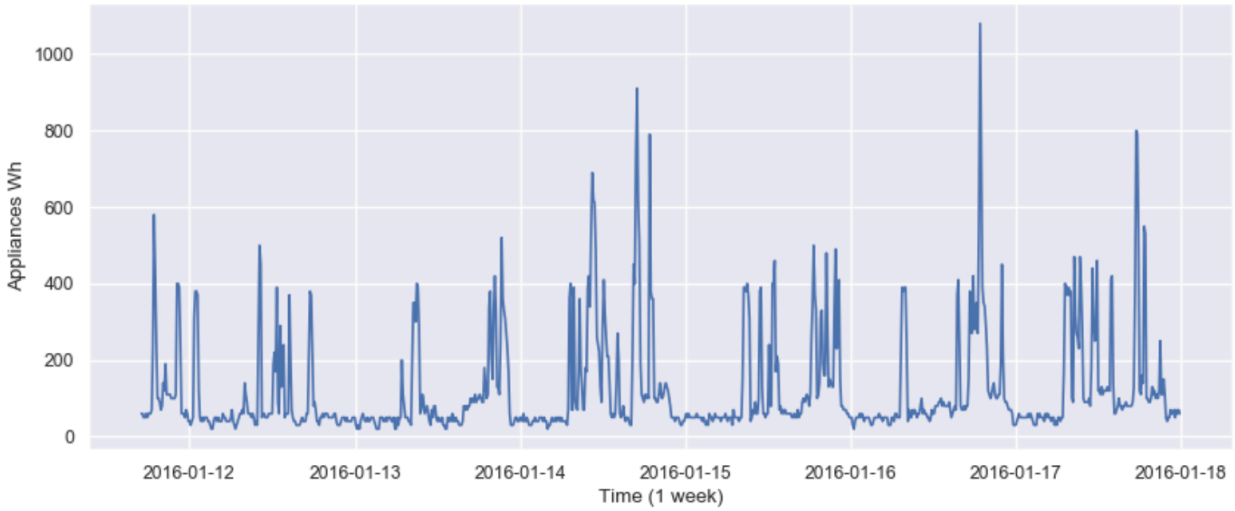
From above we can see that the dataset has no missing value, so we don't need to handle the missing value any more.

Part 2: Exploratory Data Analysis

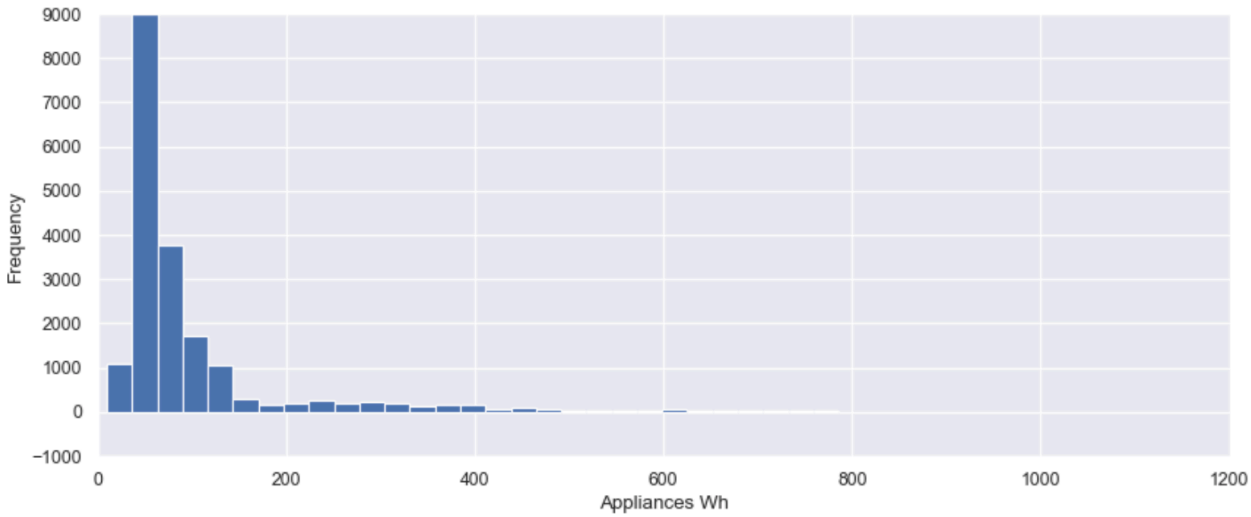


The above figure shows the energy consumption profile for the period. The energy consumption profile shows a high variability. And lets see the detail for the first week.

Assignment2 Report



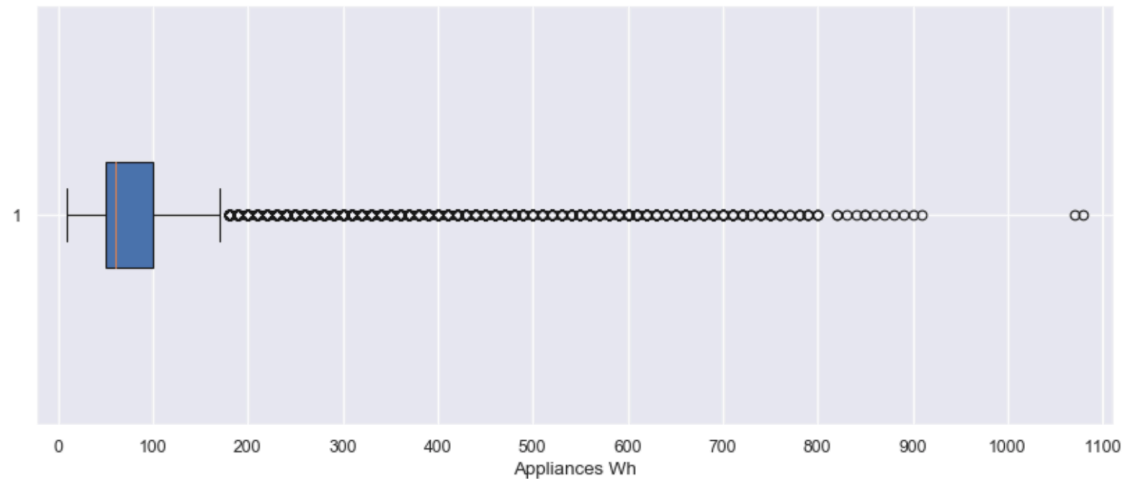
For the first week, we can see that for each day the consumption changing from the peak to the trough. The peak may seem different but the trough is basically the same for each day. So we want to know the distribution of the consumption and let's draw a histogram of the Appliances.



We can see that most values are in the range of 0-200 Wh and let's see the percentage

Percentage of dataset in range of 0–200 Wh
90.291%

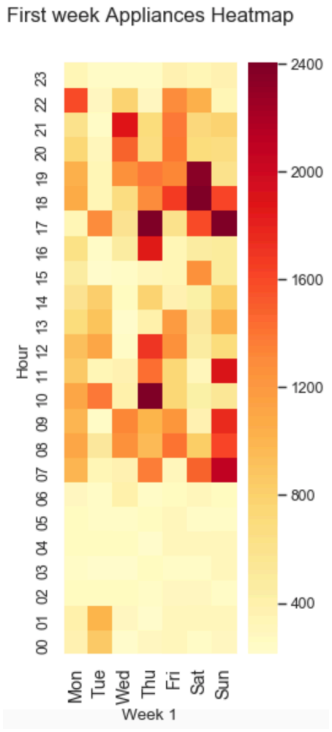
So about 90 percent of the consumption are within 0-200 and let's draw the box plot to see the outlier



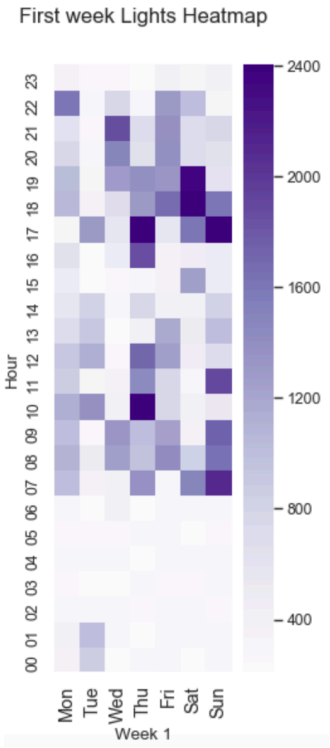
As can be seen the data distribution has a long tail. In the box plot, the median is represented with a thick black line inside the blue rectangle, and has a value of 60 Wh. The lower whisker has a value of 10 Wh and the upper whisker has a value of 170 Wh.

It also shows that the data above the median is more dispersed and that there are several outliers (marked with the round circles above the upper whisker)

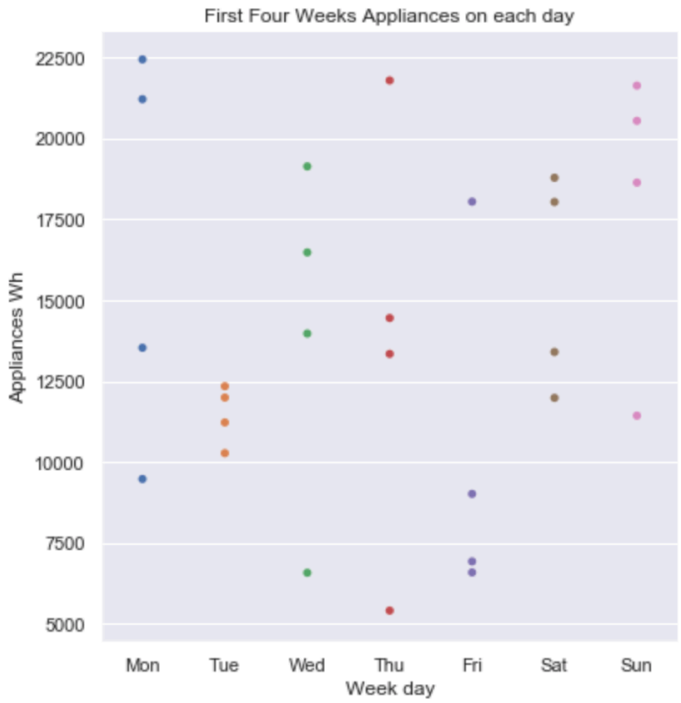
To have a better view of the Appliances, we extract the first week data and do more EDA!



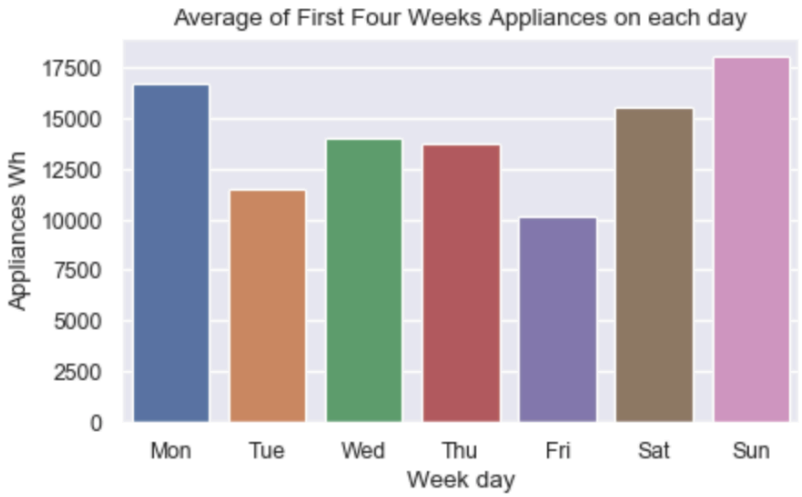
The above heat map shows the first week Appliance's heat map. We can see that the most consumption are in 7-22 o'clock which make sense. The peak consumption period is about 16 to 19, may be because the people go back home from work during that time and the consumption reach the peak in the day.



Then let's extend the data into first 4 weeks and to see the "daily" view of the Appliances



For each Day_of_Week, We can see the daily amount of the Appliances. What we found interesting is unlike other day, Tuesday's consumption stay basically same while others varying significant.
From the above graph, despite Tuesday the daily behavior seem have nothing to do with "which day of week is that day"
So, let's see the average daily behavior to summarize the daily behavior.

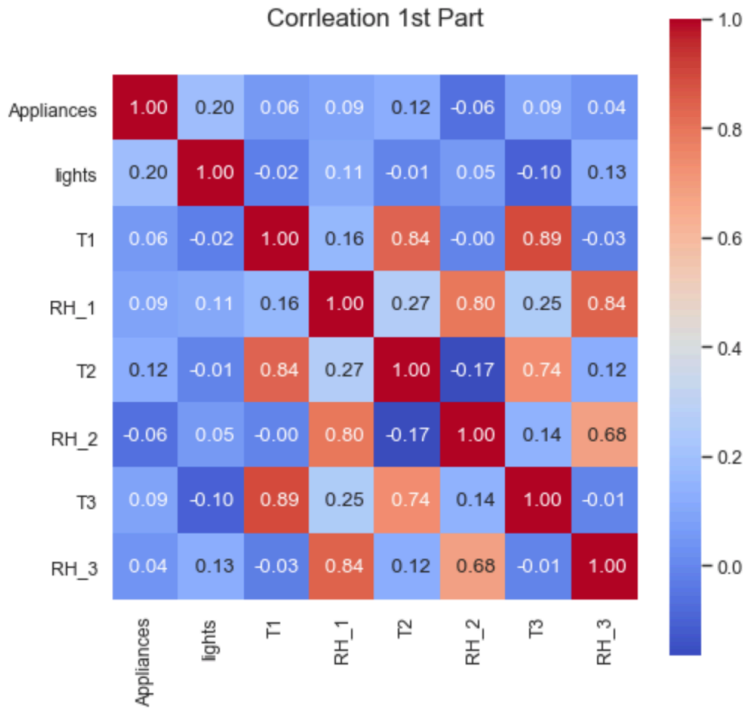


It can be seen that the top 3 consumption day is Sun, Mon and Sat which make sense because the owner will spend more time in home on the weekend and maybe on Monday the owner don't have to work either.
Combine with the cat plot we may conclude that:

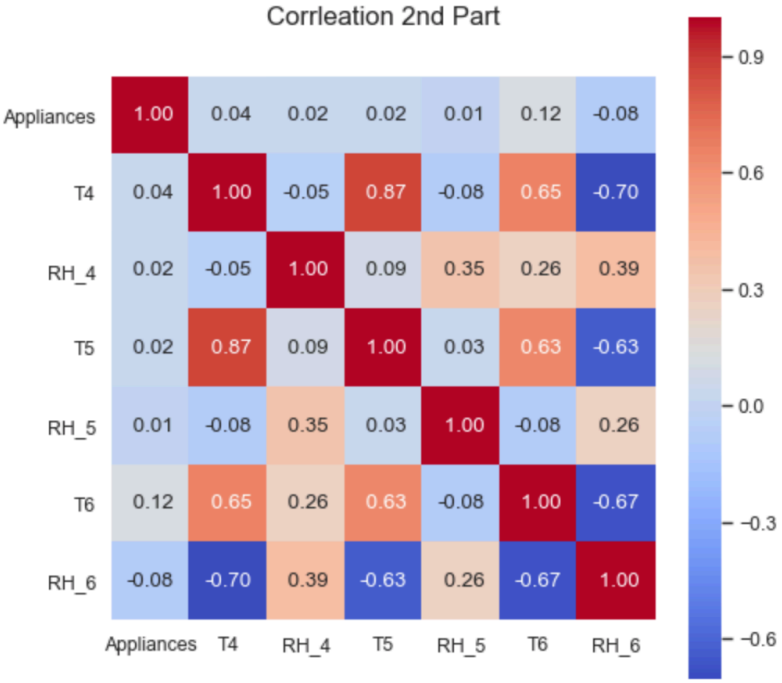
1. On each Tuesday the owner maybe have to work overtime or stay in somewhere else
2. There are 3 "Friday" data which is pretty low and maybe it is because the owner went to have a party on Friday night!

Assignment2 Report

Let's move to see the correlation of each column. Since there are many columns, we will divide it into 4 parts.

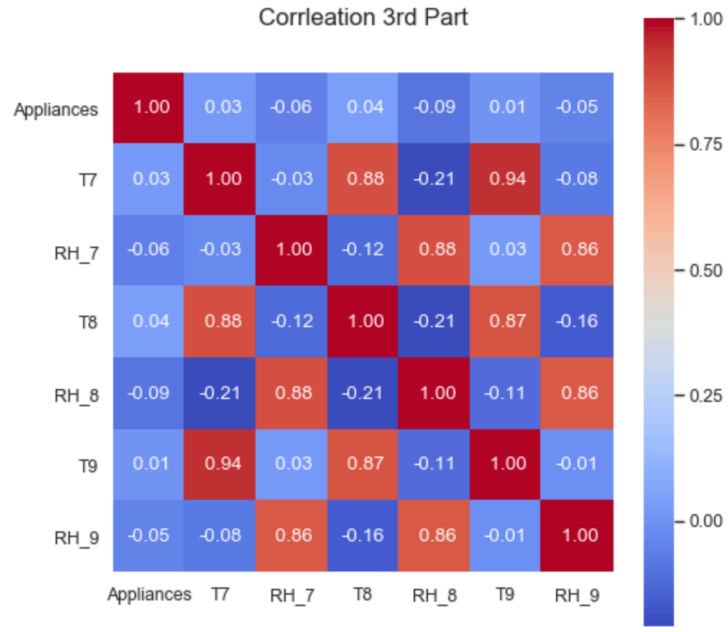


From 1st part, we can see that {"T1","T2"} and {"T1","T3"} have a strong positive relationship. T1 represents kitchen area, T2 represents living room area and T3 represents laundry room area. And when see the floor plan we can see that the kitchen are next to both laundry and living room, So it makes sense that T1 have the great positive relationship withT2 and T3.

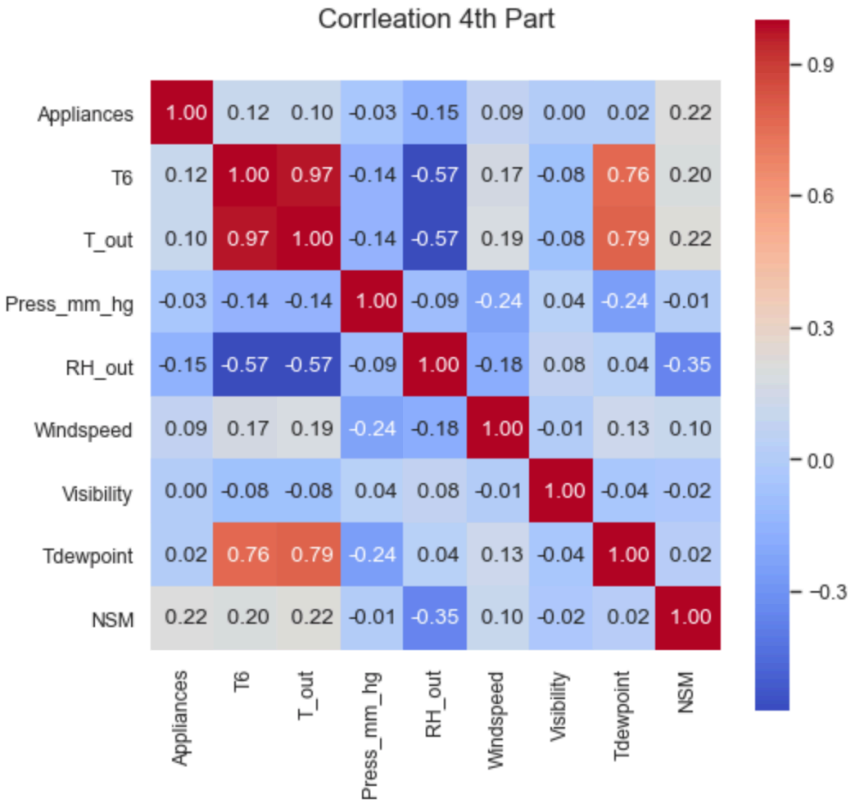


In the 2nd Part, T4 and T5 have a strong positive relationship(0.87) and it is also because of the location.

Assignment2 Report



From 3rd Part, we can see that {"T7","T8"} ,{"T7","T9"},{"T8","T9"},{"RH_7","RH_8"} , {"RH_7","RH_9"} and {"RH_8","RH_9"} have strong positive relationship and 7,8,9 represents the iron room ,teenager room and parent room. Which from the 2nd floor plan we can see these three room is close to each other.



From part 4 we can see that the T6 and T_out have a very strong relationship we is 0.97. Because T6 is the temperature from the sensor outside the building and T_out is the temperature collected from the weather station so these two features are pretty same.

Part 3: Feature Engineering

Now that we finish the EDA part, let's jump into how to prepare the data for the prediction model
First, let's divide the feature columns into different part to analysis it

```
#Feature Engineering
df['Appliances'].describe()
```

```
count    19735.000000
mean        97.694958
std        102.524891
min         10.000000
25%         50.000000
50%         60.000000
75%        100.000000
max       1080.000000
Name: Appliances, dtype: float64
```

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

```
Percentage of Appliances in range of 0–200 Wh
90.291%
```

We can see that the max Appliances is 1080 while 75% is only 100 so there is a strong outlier in the Appliances. Because from the previous box plot we know that most of the Appliances is between 0-200, and the outlier will strongly effect the prediction model so by removing the outlier we can build a better model.

```
#Feature Engineering
# Columns for temperature sensors
temp_cols = ["T1", "T2", "T3", "T4", "T5", "T6", "T7", "T8", "T9"]

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

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

#Columns for random varibles
rand_cols = ["rv1","rv2"]
```

For each type of feature we treat it like a group and to find the inner relation.

```
df[temp_cols].describe()
```

	T1	T2	T3	T4	T5	T6	T7	T8
count	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000
mean	21.687676	20.294921	22.230049	20.858577	19.607705	7.764725	20.277619	22.046567
std	1.605252	2.172435	1.971209	2.048053	1.838655	6.031990	2.102188	1.963094
min	16.790000	16.100000	17.200000	15.100000	15.330000	-6.065000	15.390000	16.306667
25%	20.760000	18.790000	20.790000	19.566667	18.290000	3.500000	18.700000	20.823333
50%	21.600000	19.926667	22.100000	20.666667	19.390000	7.160000	20.100000	22.150000
75%	22.600000	21.472333	23.290000	22.100000	20.600000	11.070714	21.600000	23.390000
max	26.200000	29.856667	29.200000	26.200000	25.795000	28.290000	25.890000	27.230000

Assignment2 Report

From all the temperature columns we can see: Temperature ranges for all home sensors is between 14.89°C to 29.86°C except for T6 for which it is -6.06°C to 28.29°C. The reason for such low readings is that the sensor is kept outside.

Let's see the humidity columns

```
df[rho_cols].describe()
```

	RH_1	RH_2	RH_3	RH_4	RH_5	RH_6	RH_7	RH_8
count	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000
mean	40.158323	40.470961	39.167393	38.991000	50.987044	54.917044	35.435410	43.019409
std	3.933742	4.062130	3.223465	4.324842	9.009473	30.746291	5.085182	5.204613
min	27.023333	20.463333	28.766667	27.660000	29.815000	1.000000	23.290000	29.600000
25%	37.260000	37.930000	36.826667	35.500000	45.400000	31.145000	31.556905	39.200000
50%	39.560000	40.560000	38.471429	38.363333	49.090000	55.290000	34.900000	42.453889
75%	42.900000	43.326667	41.590000	42.090000	53.826667	83.060000	39.051865	46.590000
max	59.633333	56.026667	49.656667	51.000000	96.321667	99.900000	51.400000	58.780000

From humidity feature columns: Similarly, humidity ranges for all home sensors is between 20.60% to 63.36%. Except for RH_5 and RH_6, whose ranges are 29.82% to 96.32% and 1% to 99.9% respectively.
The reason behind this is that RH_5 is inside the bathroom, And RH_6 is outside the building, explaining the high humidity values.

```
df[weather_cols].describe()
```

	T_out	Tdewpoint	RH_out	Press_mm_hg	Windspeed	Visibility
count	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000	17819.000000
mean	7.315671	3.762120	80.236718	755.559383	3.975014	38.306600
std	5.290522	4.186178	14.771215	7.345043	2.448213	11.951954
min	-5.000000	-6.600000	24.000000	729.366667	0.000000	1.000000
25%	3.533333	0.933333	71.166667	751.000000	2.000000	29.000000
50%	6.850000	3.433333	84.333333	756.100000	3.500000	40.000000
75%	10.333333	6.550000	91.845238	760.933333	5.333333	40.000000
max	26.100000	15.500000	100.000000	772.283333	14.000000	66.000000

From weather columns, we can see that:
T_out and Tdewpoint may have outlier because the 75% of the is pretty away from it's max. It may because some extreme weather happened.

```
df[rand_cols].describe()
```

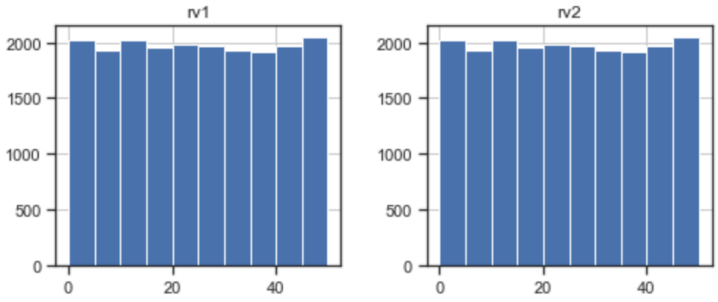
	rv1	rv2
count	17819.000000	17819.000000
mean	25.002765	25.002765
std	14.519549	14.519549
min	0.005322	0.005322
25%	12.461009	12.461009
50%	24.940753	24.940753
75%	37.660263	37.660263
max	49.996530	49.996530

```
print((df.rv1 == df.rv2).sum())
```

```
17819
```

Assignment2 Report

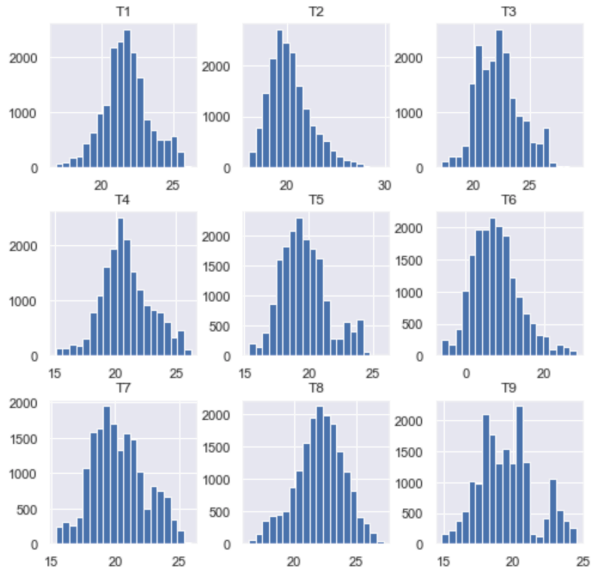
About the random variable rv1 and rv2. We can see that they are totally the same. and let's see the distribution:



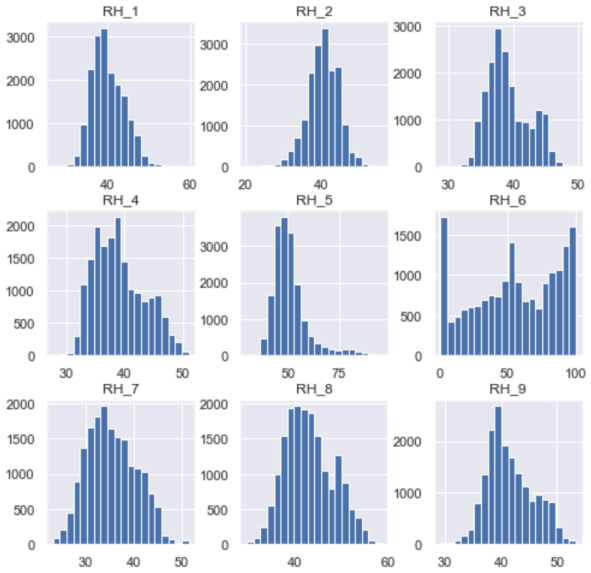
The above plot is the histogram of rv1 and rv2. it's also same and we can see from the plot that:

The distribution is mostly like "Uniform Distribution". So the rv1 and rv2 is unpredictable, because in the uniform distribution the probability is equal. So we can "predict" the outcome. Thus, the rv1 and rv2 should be removed.

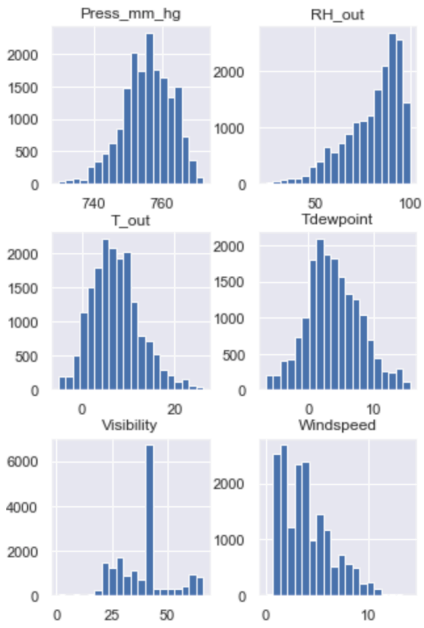
Histogram of temperature columns:



Histogram of humidity columns:



Assignment2 Report



From all the above histogram we can conclude that:

1. All temperature feature follow a Normal distribution except for T9.
2. All humidity values except RH_6 and RH_out follow a Normal distribution. That is, all the readings from sensors inside the home are from a Normal distribution.
3. Out of the remaining columns, we can see that Visibility, Windspeed are skewed.

Now we see the summary and distribution of each features and From the EDA step we know that the temperature and humidity features may have some strong inner relationship and we will find them all

```
from scipy.stats import pearsonr

# Calculate the coefficient and p-value
corr_coef, p_val = pearsonr(df["T7"], df["T9"])
print("Correlation coefficient : {}".format(corr_coef))
print("p-value : {}".format(p_val))
```

```
Correlation coefficient : 0.9462793252230384
p-value : 0.0
```

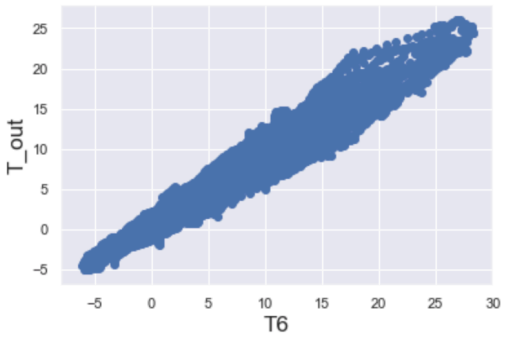
We use pearsonr to get the correlation between two variable and the p-value which indicate the probability if the two are uncorrelated.
From above we can see that the T7 and T9 are very strong correlated
Let's see all possible combination pair of the feature in the dataset (numeric data)

```
    if corr_coef > 0.9 or corr_coef < -0.9:
        # Print details for pairs with high correlation
        print("Column pair : {}, {}".format(*pair))
        print("Correlation coefficient : {}".format(corr_coef))
        print("p-value : {}".format(p_val))
```

```
Column pair : T3, T9
Correlation coefficient : 0.9114580670294231
p-value : 0.0
Column pair : RH_3, RH_4
Correlation coefficient : 0.9092959584282132
p-value : 0.0
Column pair : T5, T9
Correlation coefficient : 0.9134079358280152
p-value : 0.0
Column pair : T6, T_out
Correlation coefficient : 0.9740951648685382
p-value : 0.0
Column pair : T7, T9
Correlation coefficient : 0.9462793252230384
p-value : 0.0
```

What we can see is that 3 columns have a high degree of correlation with T9, all of which have a p-value equal to 0. Therefore, T9 can be considered as redundant.
Also, a very high correlation exists between features T6 and T_out. This shouldn't be surprising as T6 is reading from a temperature sensor kept outside the building and T_out is temperature obtained from Weather station.

Let's plot T6 and T_out to get a detailed visualization.



It's very obvious that the T6 and T_out are linear correlated and so the T6 is also can be considered as redundant.

Preparing the data:

To prepare the data for the use of prediction we must do the two things:
1. For all the numeric data --- Feature Scaling
2. For all the categorical data ----Encoding

We will divide the feature based on these two type and deal with it separately.

```
# Remove correlated features T6 and T9
data =  df.drop(["date","T6", "T9"], axis=1)
# data cleaning – remove random variables rv1 and rv2
data = data.drop('rv1', 1)
data = data.drop('rv2', 1)

# divide data into category and numerical and do preprocessing
numeric_columns = data.select_dtypes(include=['float64','int64']).columns
categorical_columns = data.select_dtypes(include=['category']).columns

# One–hot encoding for category features
data_cat = pd.get_dummies(data[categorical_columns])

#Normalize the data set for ease of calculations so that all features have values between 0 and 1
df_num = data[numeric_columns]

df_num=(df_num–df_num.min())/(df_num.max()–df_num.min())

data1 = pd.concat([df_num,data_cat],axis = 1)
```

After that the data looks like this:

T4	RH_4	T5	...	NSM	Week_Status_Weekend	Week_Status_Weekday	Day_of_Week_Mon	Day_of_Week_Tue
0.909910	0.721508	0.808409	...	0.370629	0	1	0	0
0.566667	0.348329	0.408027	...	0.370629	0	1	1	0
0.107207	0.468295	0.083134	...	0.573427	0	1	0	0
0.422523	0.280206	0.312470	...	0.062937	0	1	1	0
0.161261	0.384462	0.273610	...	0.426573	0	1	0	0

Now all the data can be used for the prediction model!

Part 4: Prediction algorithms

Based on the data well prepared, the program chooses five different algorithms against clean dataset.

1). Simple linear regression.

2). SVM regression with a kernel of rbf.
A support vector machine is aimed to find the largest margin between certain data, along with a constant coefficient parameter that weights the error when training. With a kernel trick of radial basis functions, the algorithm takes squared euclidean distance as its criterion of specifying the margin and regressing to a curve.

3). Neural network regression.
A neural network algorithm introduces hidden layouts between the inputs and outputs. Each layer has several nodes whom can be interpreted as simple functions. And the node in neighbor layers are all cascaded to each other.

4). Random forest regression.

A tree algorithm that divide the dataset into different cells, and with randomly selected trees, the algorithm takes the majority of the trees' results as its result in conclusion.

5). GBM regression.

Gradian boosting machine algorithm is another modification of tree algorithm. During the training, this algorithm fit the tree model to predict certain direction and length, thus a vector, for weighting different region of the tree function. And during each iteration, this algorithm always find the better weight compared with the current estimation.

Part 5: Feature Selection

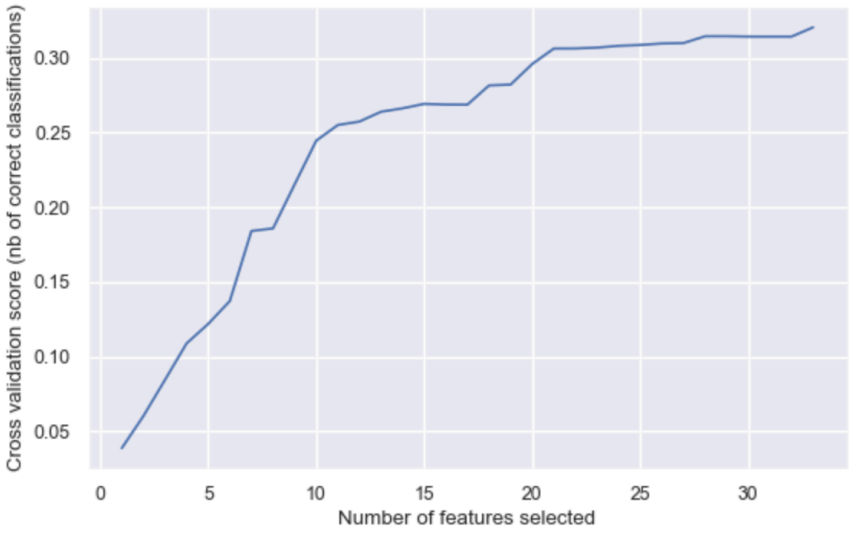
Now the data is ready to be modeled. What we want to know is that which feature is better for prediction. We use some feature selection techonology to find that if we could select some features among all the columns to produce a better prediction.
We use two selection model

1. Recursive Feature Elimination Cross-Validation
2. SelectFromeModel

RFECV result:

['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'RH_6', 'T7', 'RH_7', 'T8', 'RH_8', 'RH_9', 'T_out', '
Press_mm_hg', 'RH_out', 'Windspeed', 'Visibility', 'Tdewpoint', 'NSM', 'Week_Status_Weekend', 'Week_Status_Weekday', '
Day_of_Week_Mon', 'Day_of_Week_Tue', 'Day_of_Week_Wed', 'Day_of_Week_Thu', 'Day_of_Week_Fri',
'Day_of_Week_Sat', 'Day_of_Week_Sun']
Optimal number of features : 33

Assignment2 Report



The set up is like this:
rfecv = RFECV(estimator=lr, step=1, scoring='r2')
We use linear regression and r2 score to build the prediction model and measure the performance.
From above plot we can see that the Optimal number of features is still 33 after RFECV.

SelectFromModel result:

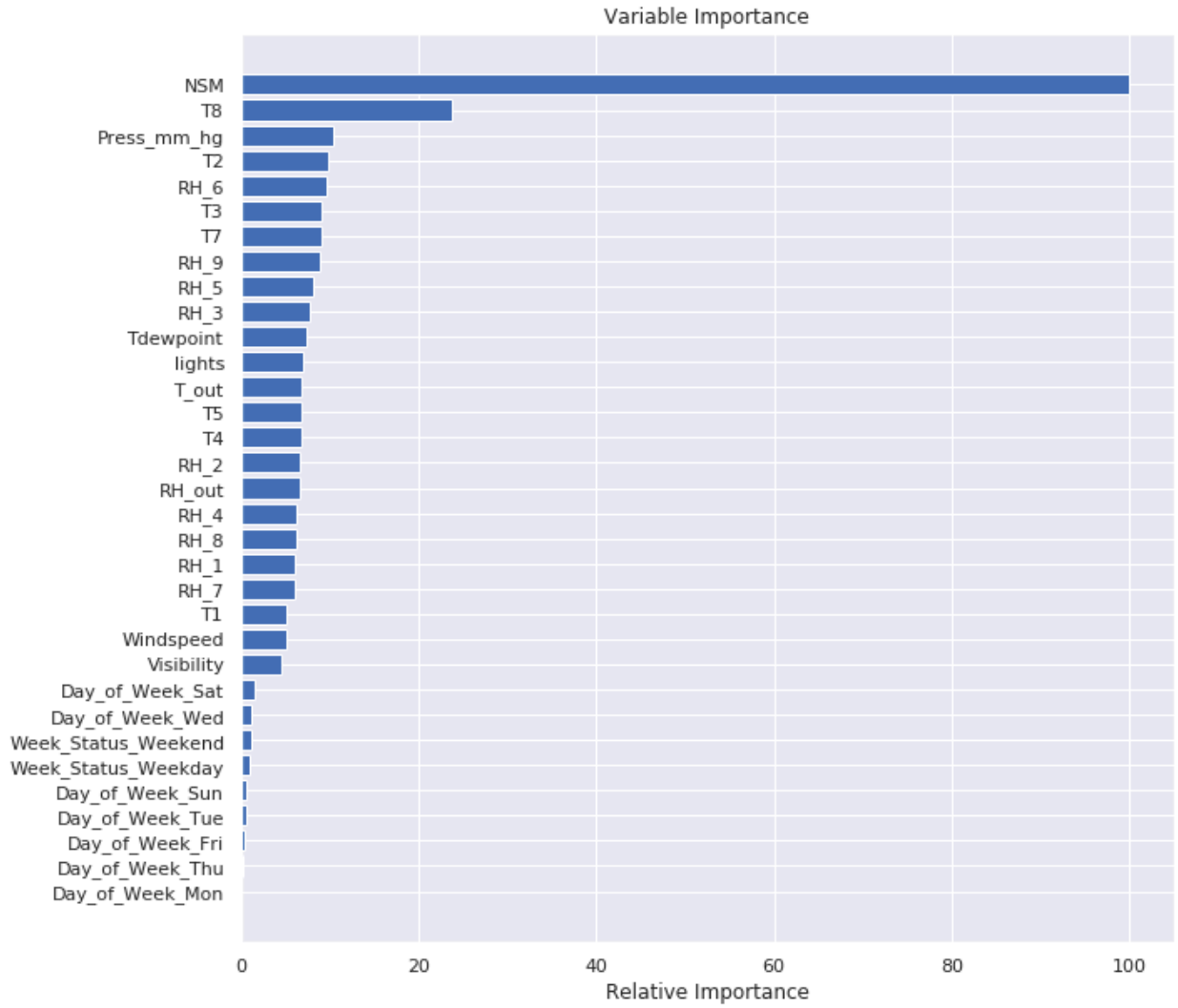
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel

rf = RandomForestRegressor(max_depth=120, random_state=15, n_estimators=100)

rf.fit(X_train, y_train)
feat_labels = list(X_train)

# Create a selector object that will use the random forest classifier to identify
# features that have an importance of more than 0.15
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

for f in range(X_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 50, feat_labels[indices[f]], importances[indices[f]]))
```

```
 1) NSM                                            0.344700
 2) T8                                             0.082412
 3) Press_mm_hg                                    0.039784
 4) T7                                             0.031466
 5) RH_9                                           0.030751
 6) RH_6                                           0.030669
 7) RH_5                                           0.028254
 8) T2                                             0.027027
 9) RH_3                                           0.026148
10) T3                                             0.025981
11) T_out                                          0.025349
12) T4                                             0.025265
13) Tdewpoint                                      0.024674
14) RH_2                                           0.024167
15) lights                                         0.024024
16) RH_8                                           0.023586
17) RH_out                                         0.023428
18) T5                                             0.023091
19) RH_4                                           0.021904
20) RH_1                                           0.021237
21) RH_7                                           0.020196
22) T1                                             0.017803
23) Windspeed                                      0.016870
24) Visibility                                     0.016486
25) Day_of_Week_Sat                                0.004478
26) Week_Status_Weekday                            0.003965
27) Day_of_Week_Wed                                0.003854
28) Day_of_Week_Tue                                0.003582
29) Week_Status_Weekend                            0.002926
30) Day_of_Week_Sun                                0.002054
31) Day_of_Week_Fri                                0.001878
32) Day_of_Week_Thu                                0.001188
33) Day_of_Week_Mon                                0.000804
```

Assignment2 Report

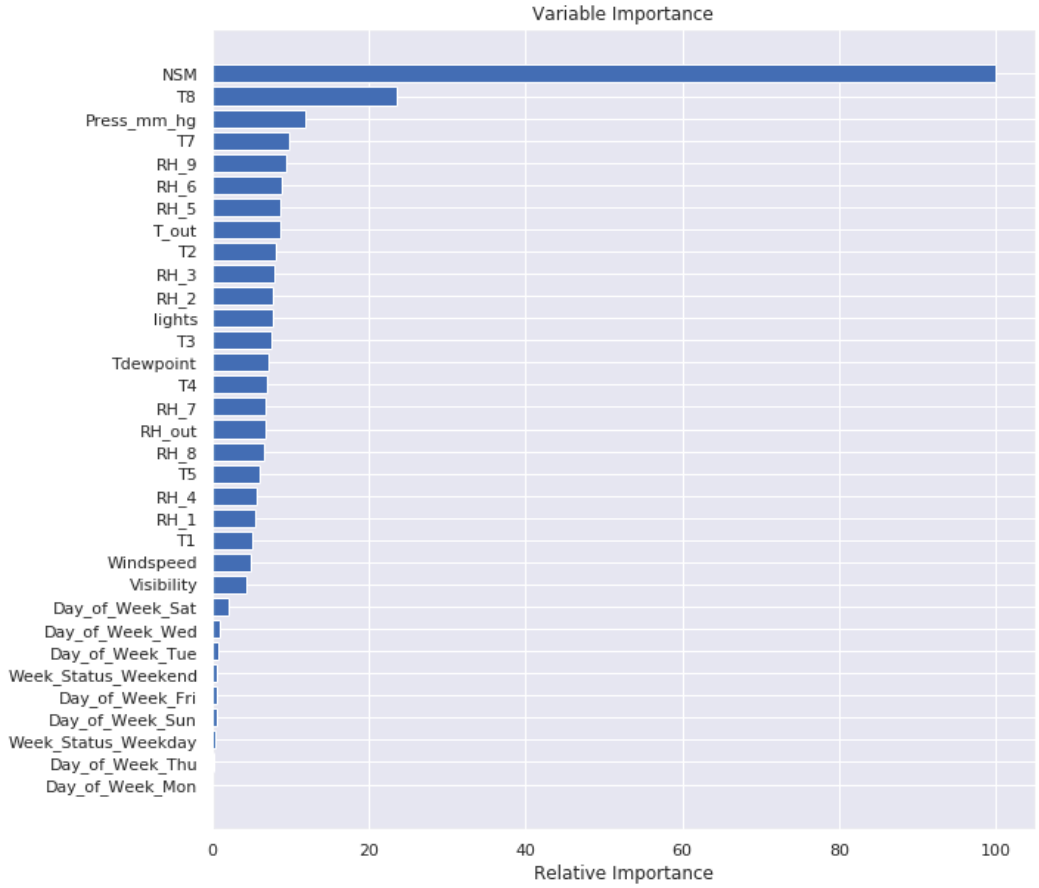
Above is the Random Forest Regressor set up and the importance of each feature.

And to make it even more visualized:



In comparison with that features who are important to gbm search, the result shows a similarity.

Assignment2 Report



Then use SelectFromModel we can decide how many feature we can keep and we will do it like backward selection:

we use a for loop to do it n (number of features) times and find the prediction performance by the score r2.

```
y_rf_before = rf.predict(X_test)
r2_1 = r2_score(y_test,y_rf_before)
m_1 = median_absolute_error(y_test,y_rf_before)
print("R2 score: ",r2_1)
print("Median Absolute Error: ", m_1)
```

R2 score: 0.6298207609303796
Median Absolute Error: 7.650000000000002

The above is the R2 and MAE before selection and the n times selection is like:

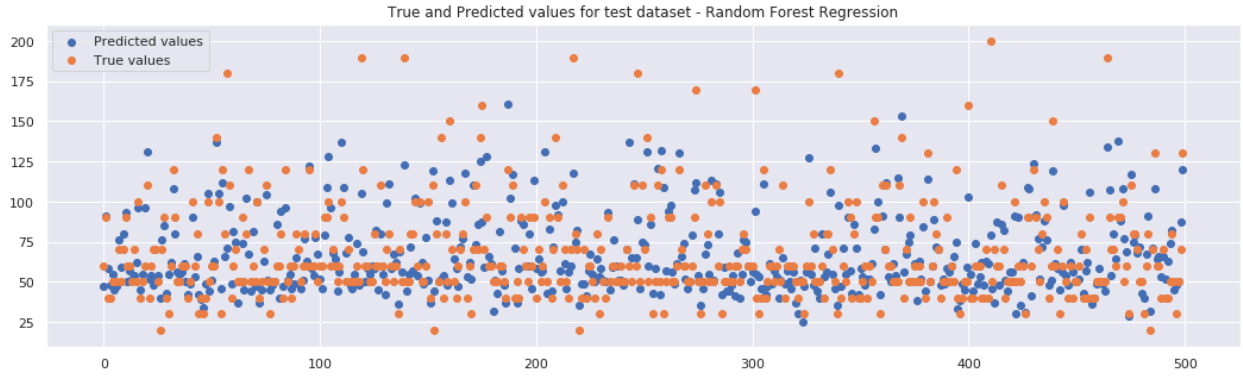
```
RFR r2: 0.629
Thresh=0.00080, n=33, R2: 0.6295
Thresh=0.00119, n=32, R2: 0.6297
Thresh=0.00188, n=31, R2: 0.6327
Thresh=0.00205, n=30, R2: 0.6287
Thresh=0.00293, n=29, R2: 0.6298
Thresh=0.00358, n=28, R2: 0.6304
Thresh=0.00385, n=27, R2: 0.6306
Thresh=0.00397, n=26, R2: 0.6297
Thresh=0.00448, n=25, R2: 0.6298
Thresh=0.01649, n=24, R2: 0.6288
Thresh=0.01687, n=23, R2: 0.6280
Thresh=0.01780, n=22, R2: 0.6281
Thresh=0.02020, n=21, R2: 0.6284
Thresh=0.02124, n=20, R2: 0.6272
Thresh=0.02190, n=19, R2: 0.6282
Thresh=0.02309, n=18, R2: 0.6269
Thresh=0.02343, n=17, R2: 0.6287
Thresh=0.02359, n=16, R2: 0.6280
Thresh=0.02402, n=15, R2: 0.6273
Thresh=0.02417, n=14, R2: 0.6300
Thresh=0.02467, n=13, R2: 0.6300
Thresh=0.02527, n=12, R2: 0.6285
Thresh=0.02535, n=11, R2: 0.6329
Thresh=0.02598, n=10, R2: 0.6281
Thresh=0.02615, n=9, R2: 0.6247
Thresh=0.02703, n=8, R2: 0.6225
Thresh=0.02825, n=7, R2: 0.6207
Thresh=0.03067, n=6, R2: 0.6202
Thresh=0.03075, n=5, R2: 0.6153
Thresh=0.03147, n=4, R2: 0.6071
Thresh=0.03978, n=3, R2: 0.5811
Thresh=0.08241, n=2, R2: 0.2775
Thresh=0.34470, n=1, R2: 0.3008
```

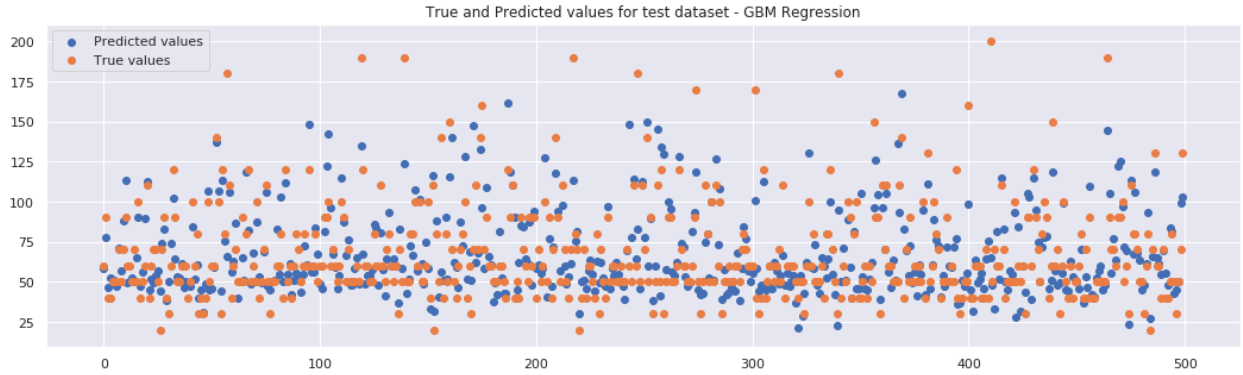
From above we can see that the r2 score didn't improve significantly and maybe the random forest regressor isn't best set up So from above result we still can't remove some features.

Also, we can use the same process against gbm predictions, who results in:

Part 6: Model-Validation and Selection

The program generate an overview of the two best-performing algorithms: random forest and gbm.



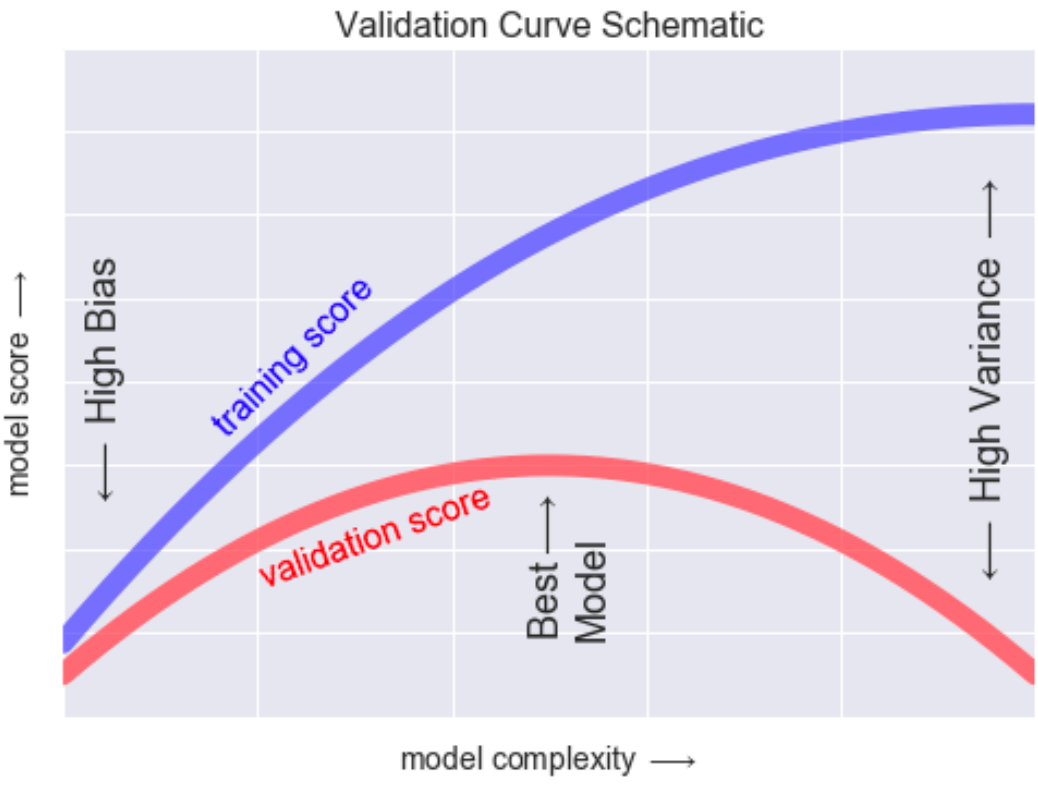


The program demonstrate the cross-validation performance of each model respectively.

And by doing bias-varience trade-off against the models, the program finds the best tuning status for each algorithm according to the output as the following:

```
j=600.000000
i=12.000000 score=0.622383
i=12.000000 validation=-0.485632
i=14.000000 score=0.600762
i=14.000000 validation=-0.357640
i=16.000000 score=0.582152
i=16.000000 validation=-0.464915
j=700.000000
i=12.000000 score=0.622010
```

By this pattern of data, we observe a learning curve and validation curve like:



So with the shown pattern the team found each algorithm with its best tuning, along with their performance metrics respectively.

```
GBM Regression:
RMSE: 19.94
MAPE: 18.24%
R2: 0.60
```

```
MAE: 12.00

Random Forest Regression:
RMSE: 19.84
MAPE: 18.91%
R2: 0.61
MAE: 12.35

Neural Network Regression:
RMSE: 26.84
MAPE: 25.94%
R2: 0.28
MAE: 17.76

Linear Regression:
RMSE: 26.43
MAPE: 28.97%
R2: 0.30
MAE: 18.20

SVM RBF Regression
RMSE: 26.78
MAPE: 26.78%
R2: 0.29
MAE: 17.31
```

And the final model can be selected as gbm so far.

Part 7: Final Pipeline

As for the final pipeline, the program implements the following steps:

1). Feature selection

2). Regression

The code is shown as below.

```
#Pipeline
from sklearn.pipeline import Pipeline

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
# ANOVA Regressor
lr = LinearRegression()
anova_filter = RFECV(estimator=lr)
gbm = ensemble.GradientBoostingRegressor()

anova_gbm = Pipeline([('anova',anova_filter),('regression', gbm)])

anova_gbm.set_params(anova__step = 1, anova__scoring = 'r2', regression__n_estimators
= 700,regression__max_depth = 14).fit(X_train, y_train)
prediction = anova_gbm.predict(X_test)

anova_gbm.score(X, y)
```

And the result is as good as:
0.918930071117557