Advances in Data Science and Architecture

Machine learning with Energy systems

By
Divya Priya Emmanuel

Gaurang Davda

Niranjhani Vasudevan

TABLE OF CONTENTS PAGE NO.

Abstract

AdaptiveAlgo Systems Inc. works on solutions to build algorithms and platforms to address energy modeling challenges. And require a solution for energy modelling and is interested in understanding consumer energy usage. As data scientist we build various machine learning models that could contribute to understanding energy usage by appliances and the attributes that contribute to aggregate energy usage.

Introduction

Building energy use prediction plays an important role in building energy management and conservation as it can help us to evaluate building energy efficiency, conduct building commissioning, and detect and diagnose building system faults. In this context, a proper prediction of energy demand in housing sector is very important. We are building a model by analysing different prediction models, feature engineering, and model selection processes. This would in turn aid AdaptiveAlgo Systems Inc. to build algorithms and systems to understand consumer behavior to help them make better decisions. The case study will be conduct an in-depth analysis to provide insights on feature engineering and machine learning with provided dataset.

Exploratory Data Analysis

The Dataset consists of 137 days of energy load on a single house from various source's such as house appliances, temperature and humidity values in different rooms and weather data from nearest airport weather station for every 10 min to capture quick change in the energy usage. On exploring the given dataset, lot of information was analyzed regarding the dependent and independent variables. The target variable in the dataset is 'Appliances'. Apart from target variable, the dataset contains 27 other variables which are considered as predictors before going further with any of the feature selection process. The index of the dataset is DateTime value data type. To understand the data pattern, three additional variables which are extracted from existing ones are added to the dataset. Those are 'Num_sec_midnight' that indicates number of seconds from midnight which indirectly gives you time of each observation, ‘Day_status’ gives the day of each observation and ‘week_status’ states whether a observation is week end or week day.

Table illustrating Data variables in the given Dataset:

Data variables	Units	Number of features
Appliances energy consumption	Wh	1
Light energy consumption	Wh	2
T1, Temperature in kitchen area	°C	3
RH1, Humidity in kitchen area	%	4
T2, Temperature in living room area	°C	5
RH2, Humidity in living room area	%	6
T3, Temperature in laundry room area	°C	7
RH3, Humidity in laundry room area	%	8
T4, Temperature in office room	°C	9
RH4, Humidity in office room	%	10
T5, Temperature in bathroom	°C	11
RH5, Humidity in bathroom	%	12
T6, Temperature outside the building (north side)	°C	13
RH6, Humidity outside the building (north side)	%	14
T7, Temperature in ironing room	°C	15
RH7, Humidity in ironing room	%	16
T8, Temperature in teenager room 2	°C	17
RH8, Humidity in teenager room 2	%	18
T9, Temperature in parents room	°C	19
RH9, Humidity in parents room	%	20
To, Temperature outside (from Chièvres weather station)	°C	21
Pressure (from Chièvres weather station)	mm Hg	22
RHo, Humidity outside (from Chièvres weather station)	%	23
Windspeed (from Chièvres weather station)	m/s	24
Visibility (from Chièvres weather station)	km	25
Tdewpoint (from Chièvres weather station)	°C	26
Random Variable 1 (RV_1)	Non dimensional	27
Random Variable 2 (RV_2)	Non dimensional	28

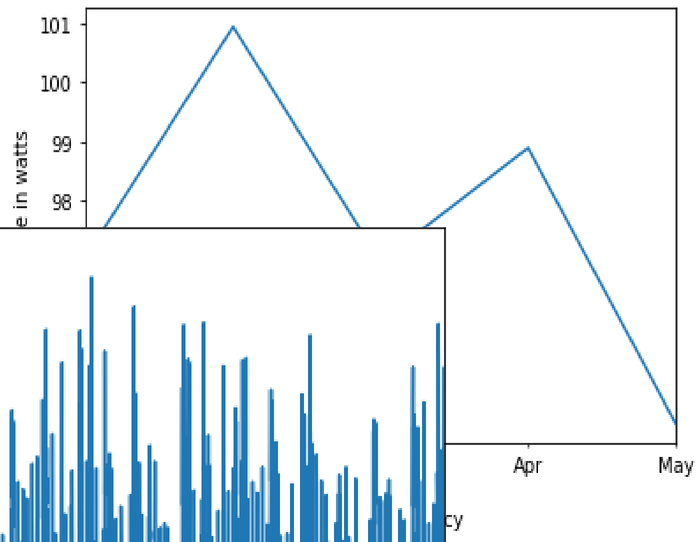
Note: Random Variable 1 & Random Variable 2 are randomly added by default to test Boruta feature selection

The below plot describes the ‘Appliances’ variable for 137 days with mean values sampled monthly. The highest energy usage was on February – 2016 compared to other months. Further study is required to get more details.

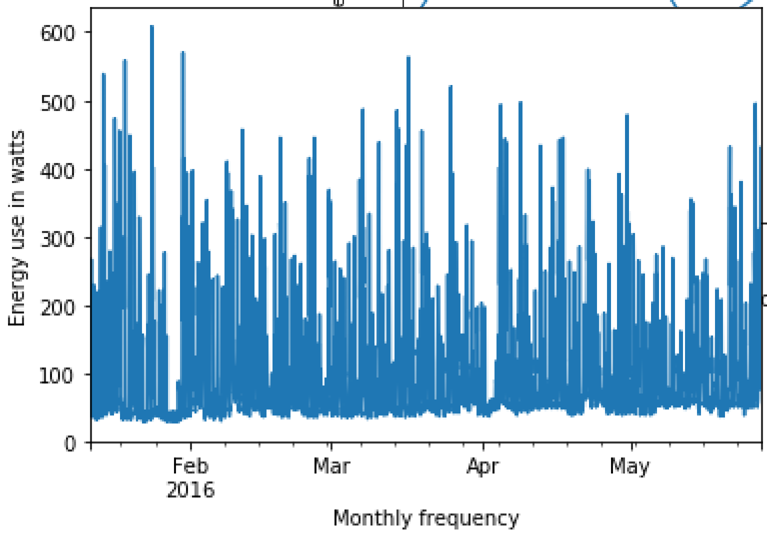
Plot 1

Date	Mean values
2016-01-31	97.026010
2016-02-29	100.945881
2016-03-31	96.953405
2016-04-30	98.888889

2016-05-31	94.199325

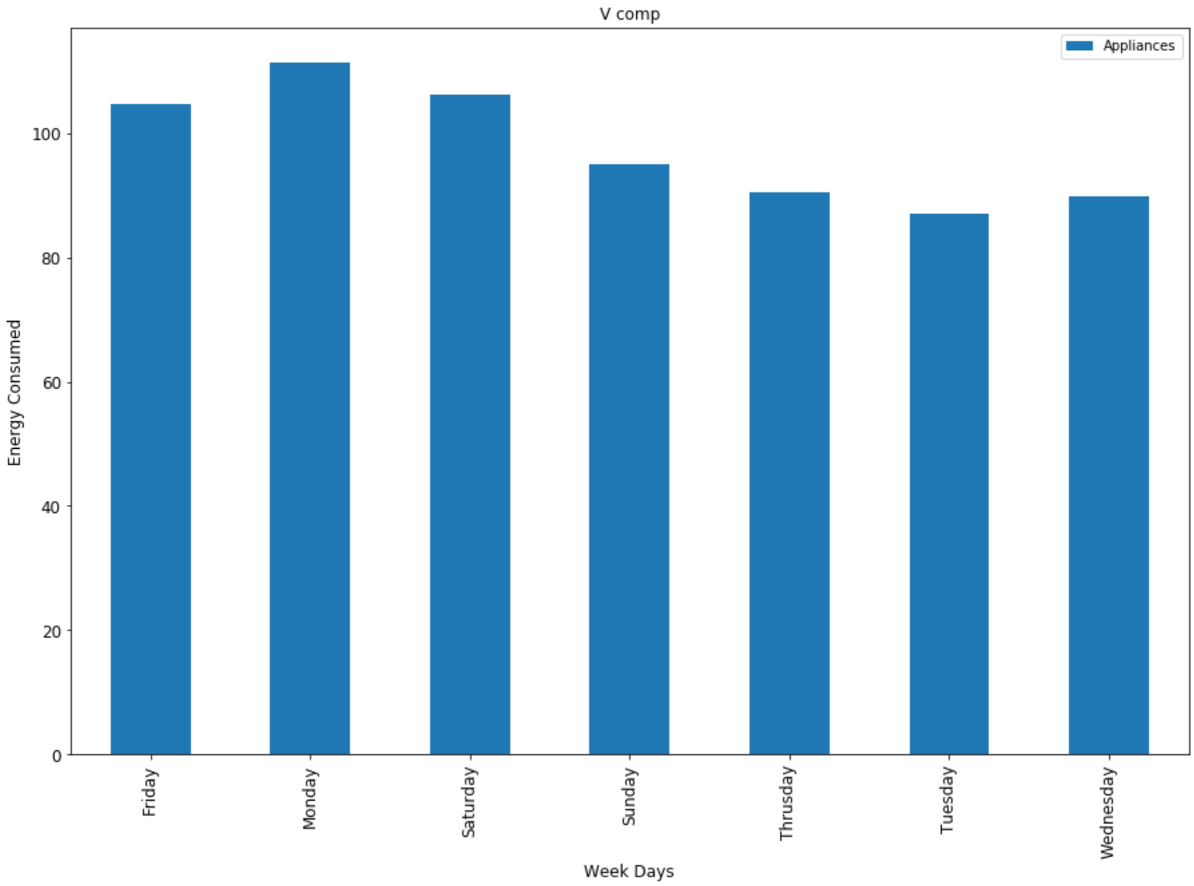


Plot 2



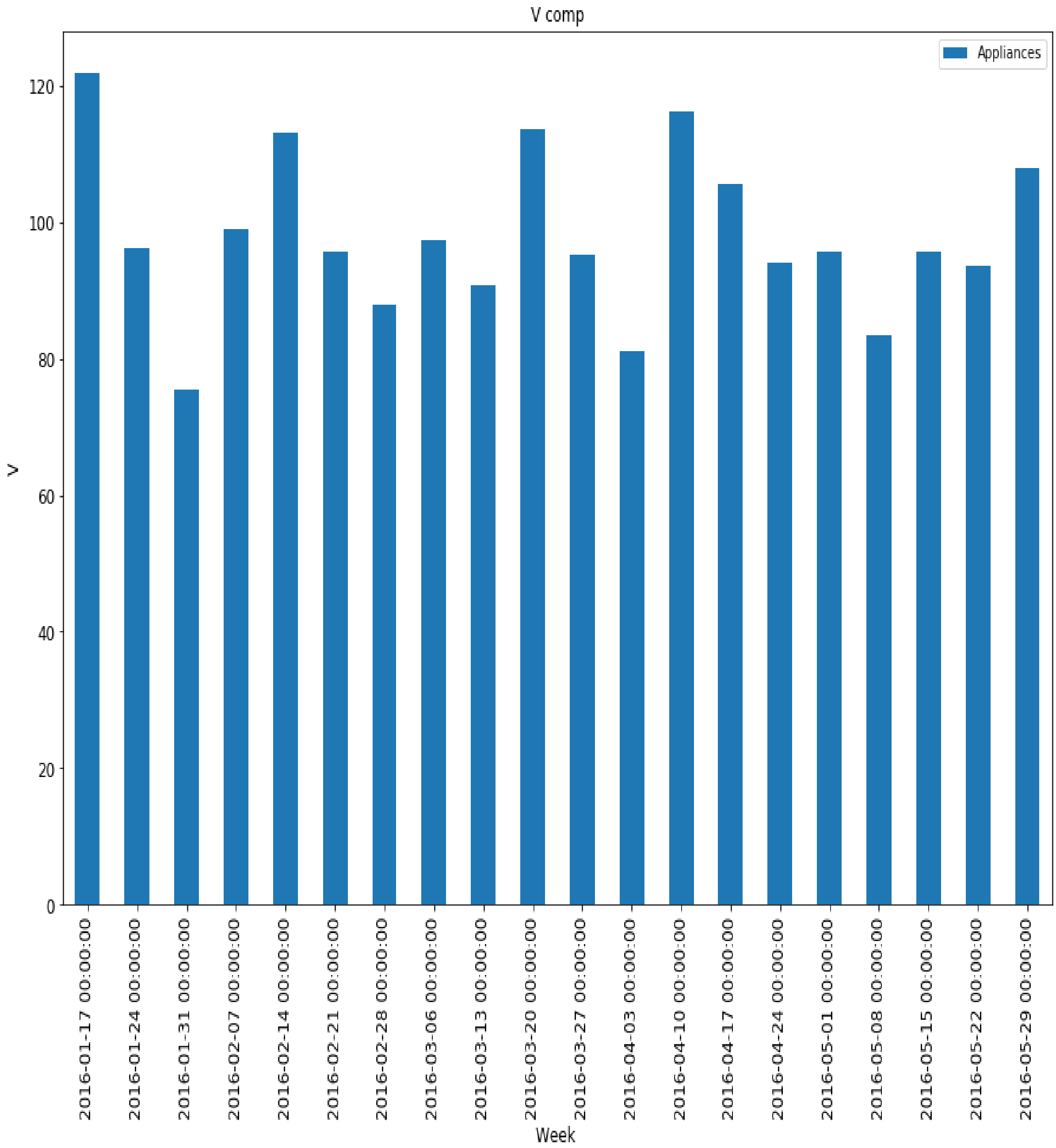
Plot 3 sums up appliances value by weekly for all 5 months and from which we can infer that more of energy was consumed on Tuesday

Plot 3



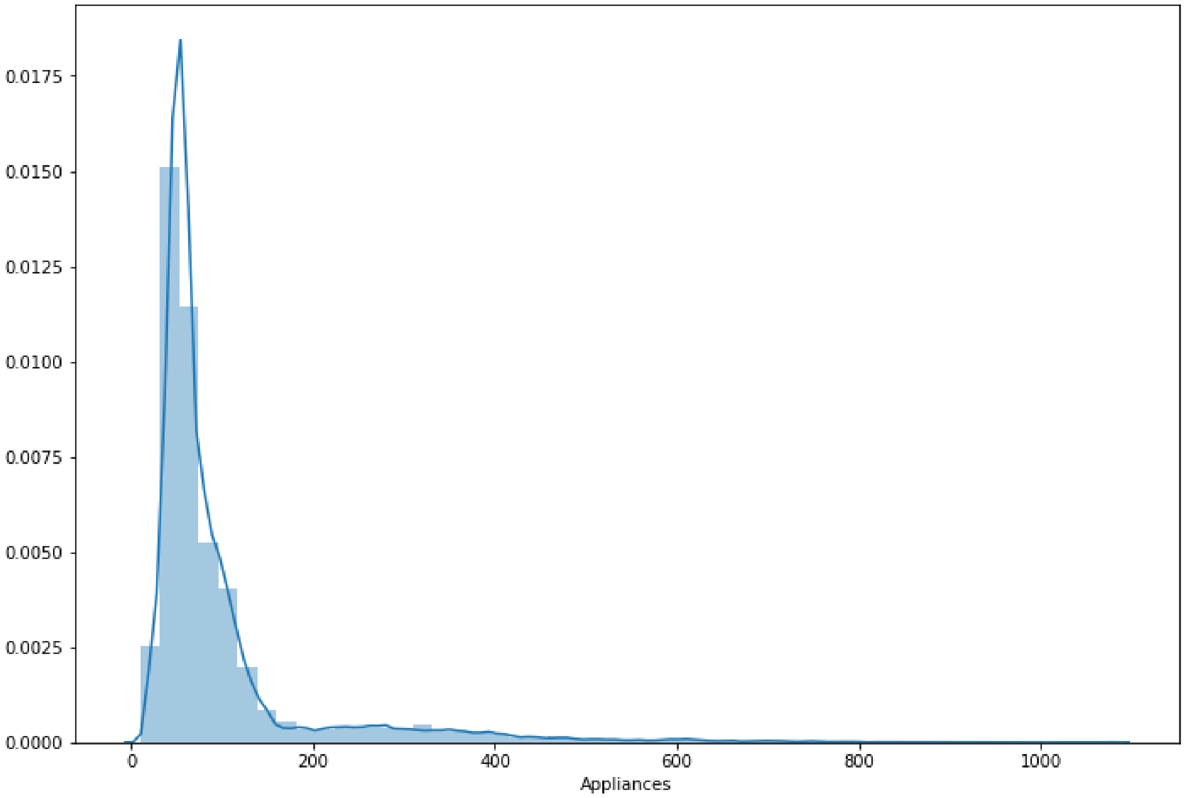
Below ***Plot 4*** gives the energy usage for the 20 weeks on the given data by its mean value.

Plot 4



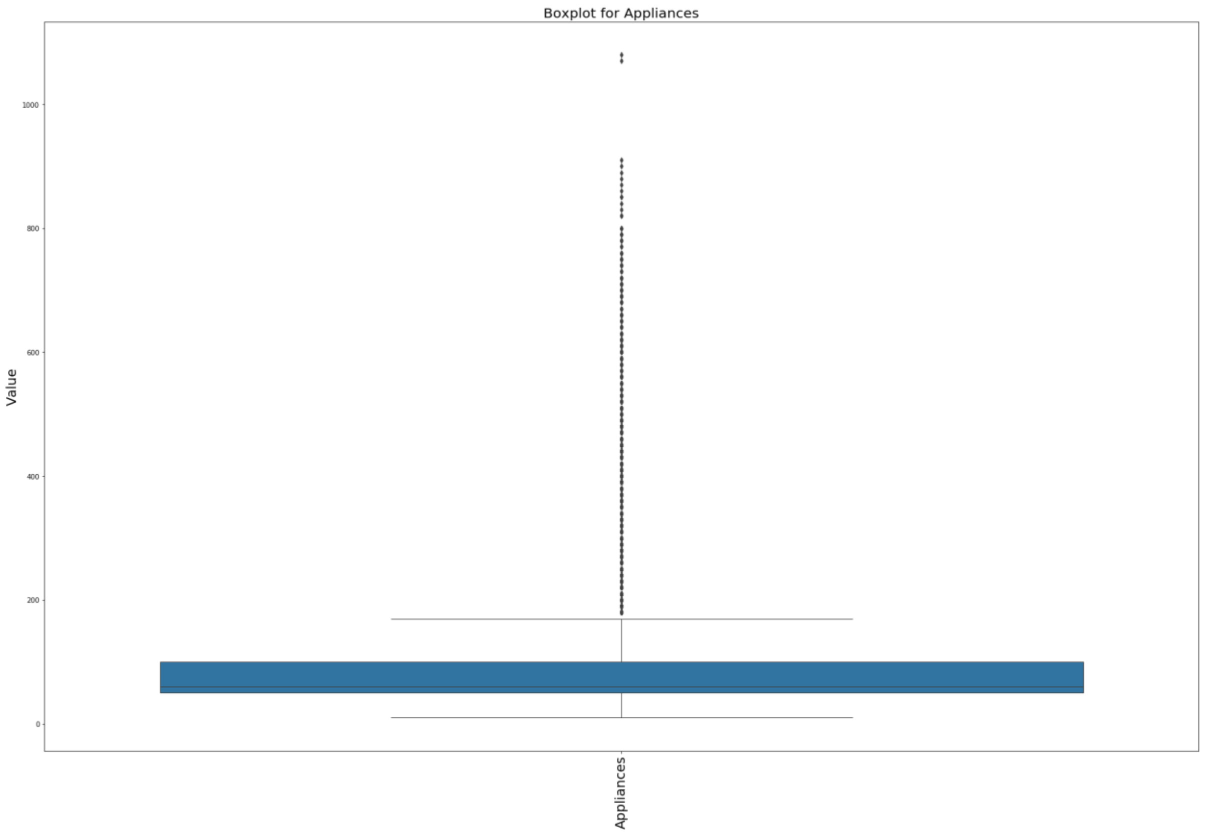
Plot 5 illustrates the data distribution in the target variable. Most of the energy values are between 10 to 180 watts. There are some extreme values in the plot which are few.

Plot 5 – Histogram of Appliance variable



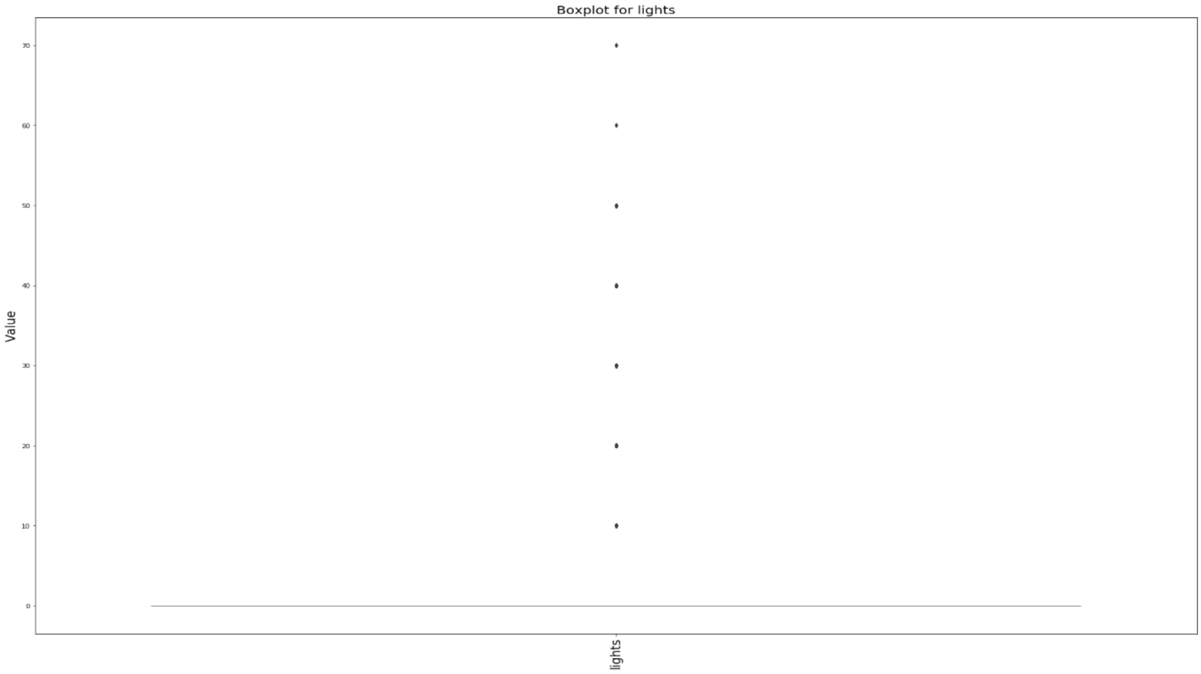
Plot 6 gives boxplot for appliances that shows the extremes values recorded in the data set

Plot 6 – Boxplot for Appliances



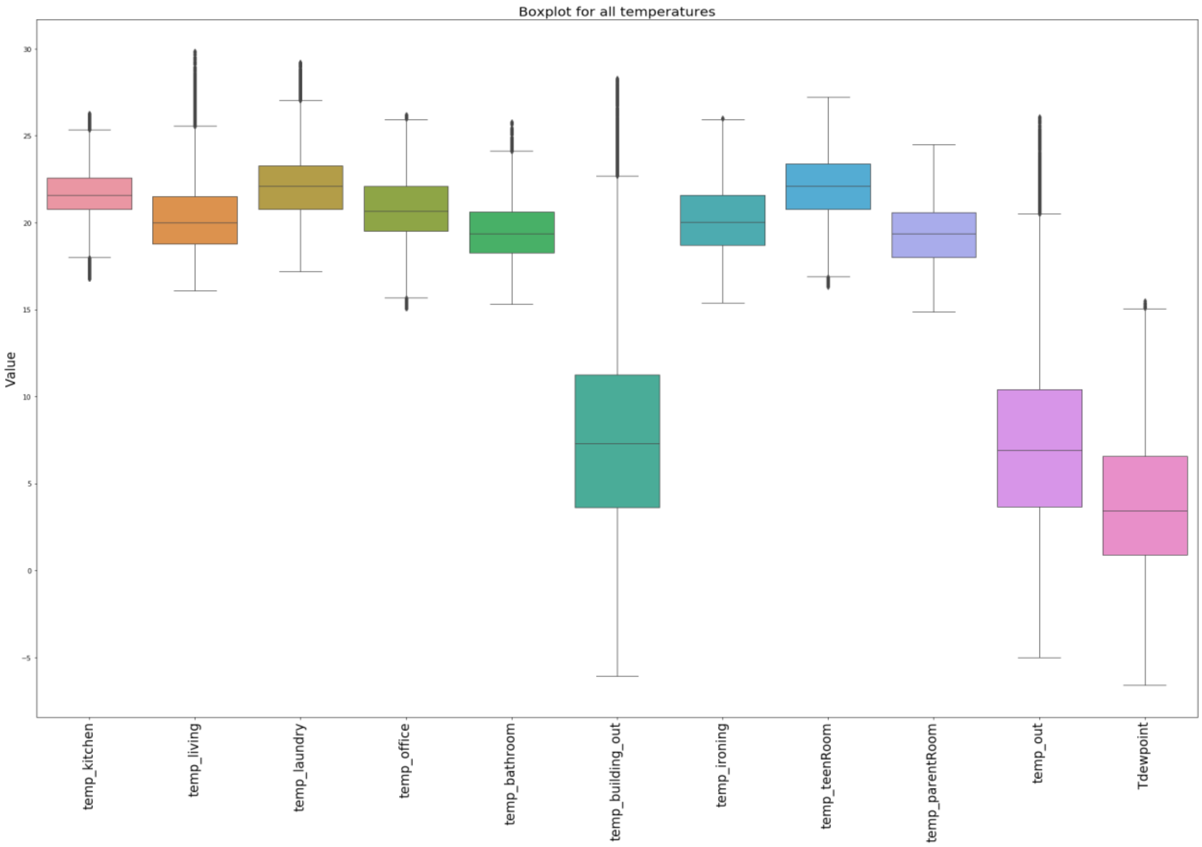
Plot 7 shows the boxplot for Light variable. This clearly shows that 75% of the values are '0'

Plot 7 – Boxplot for Lights



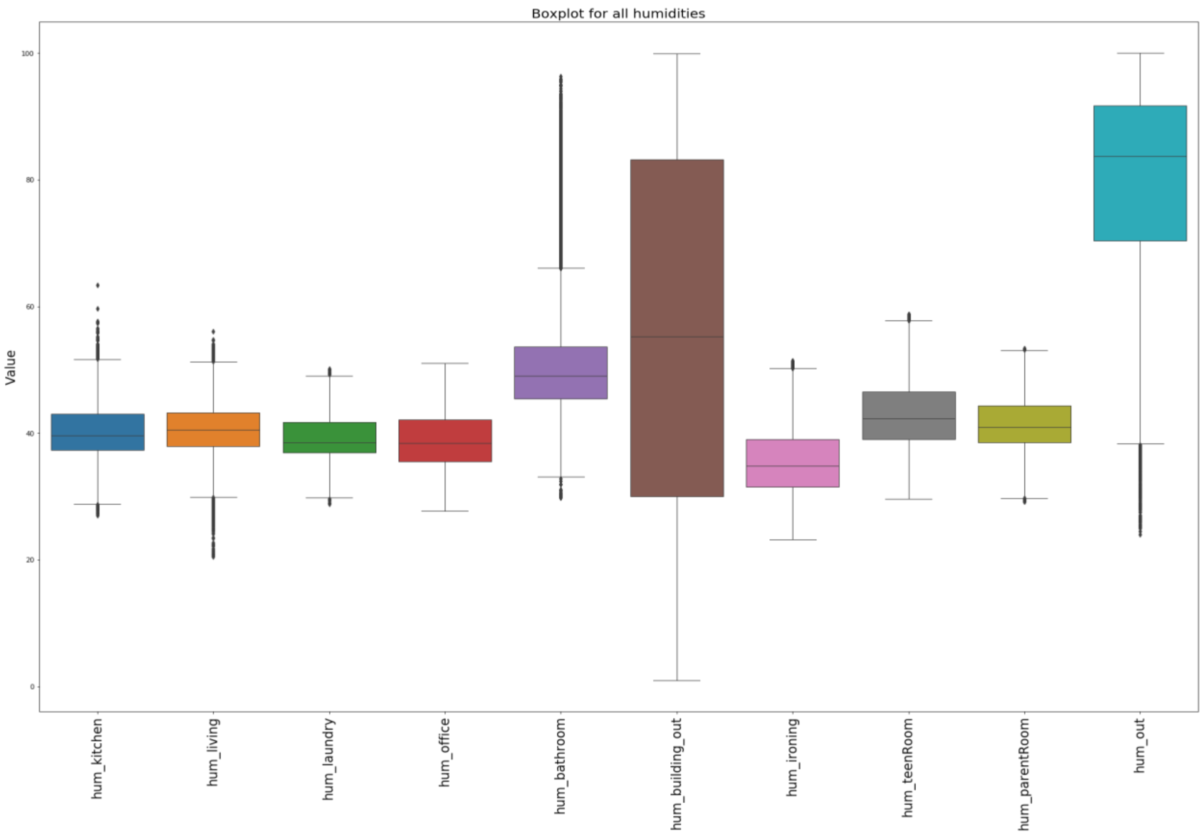
Plot 8 gives the boxplot for all the temperature variable in the Dataset.

Plot 8 – Boxplot for all temperature variables

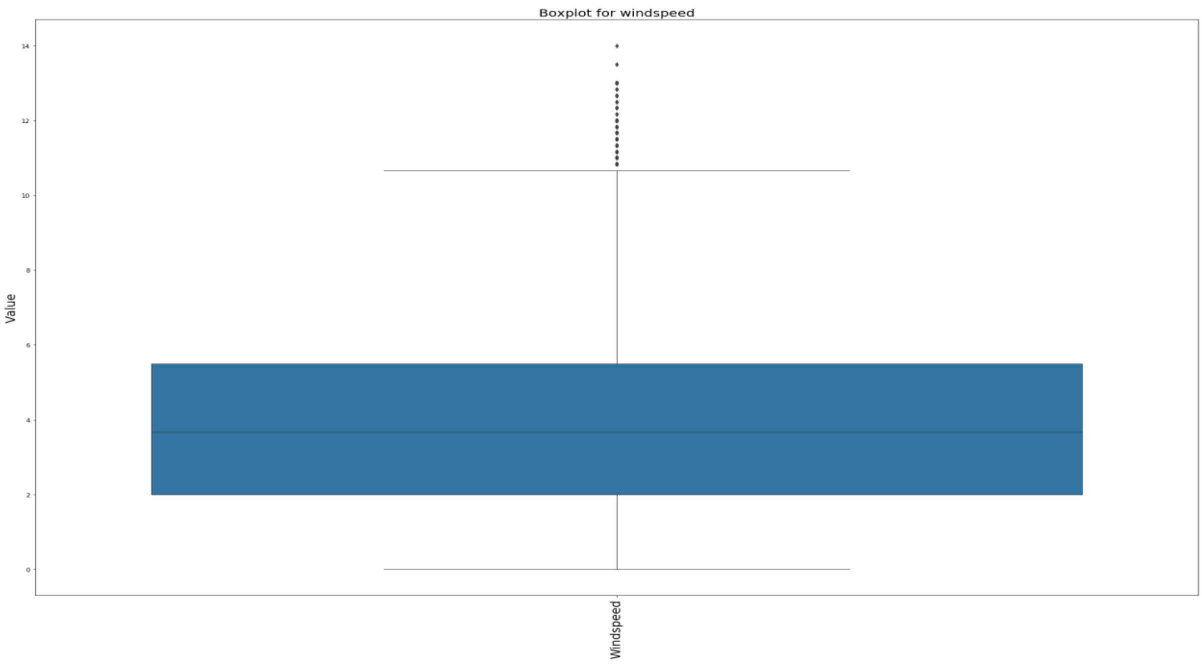


Plot 9 gives the boxplot for all humidity variable in the Dataset

Plot 9- Boxplot for all Humidity variables



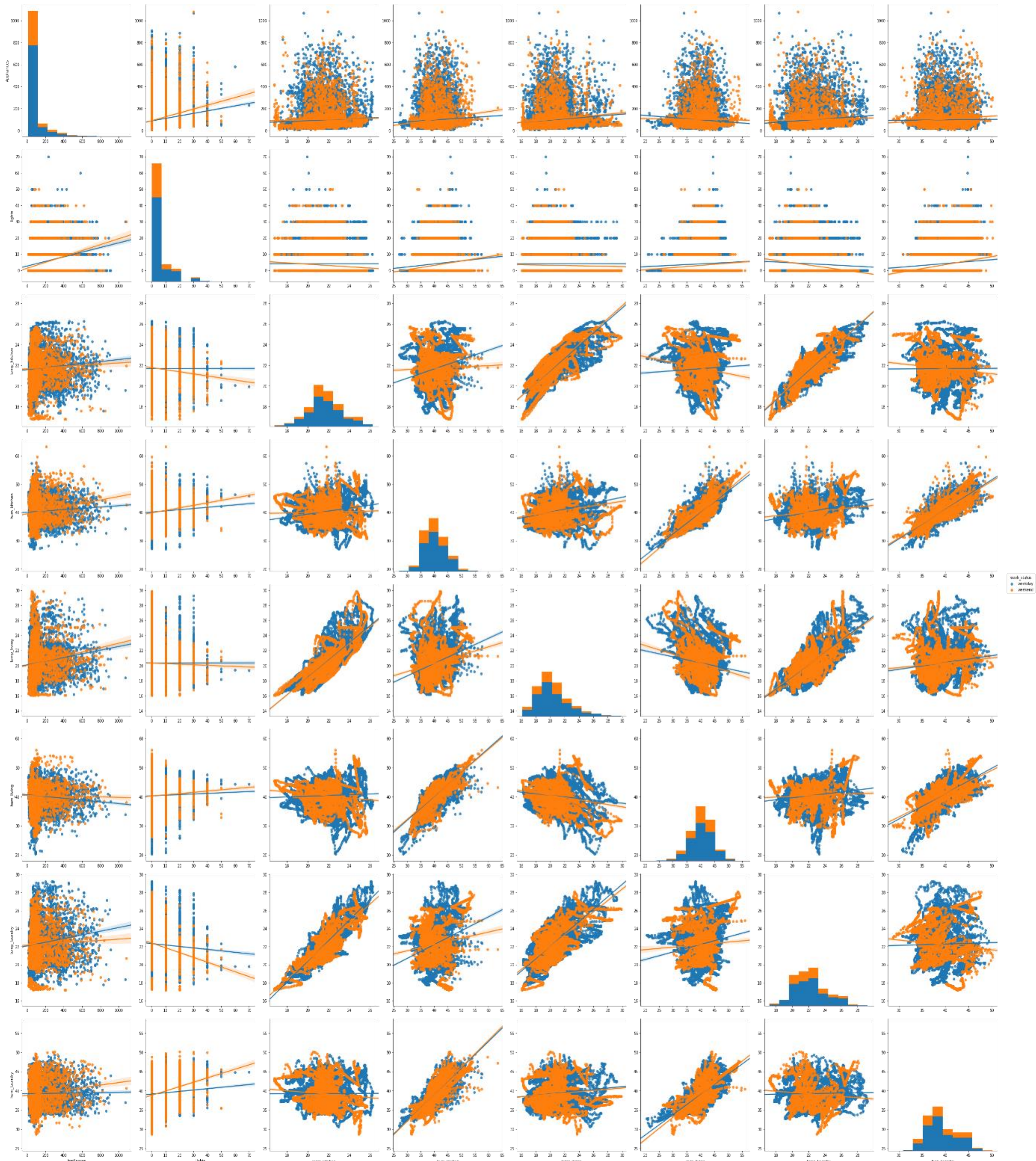
Plot 10- Boxplot for Windspeed



Inorder to find the correlation with each variable, pair plot and correlation matrix was plotted against all 31 variables (including newly added features).

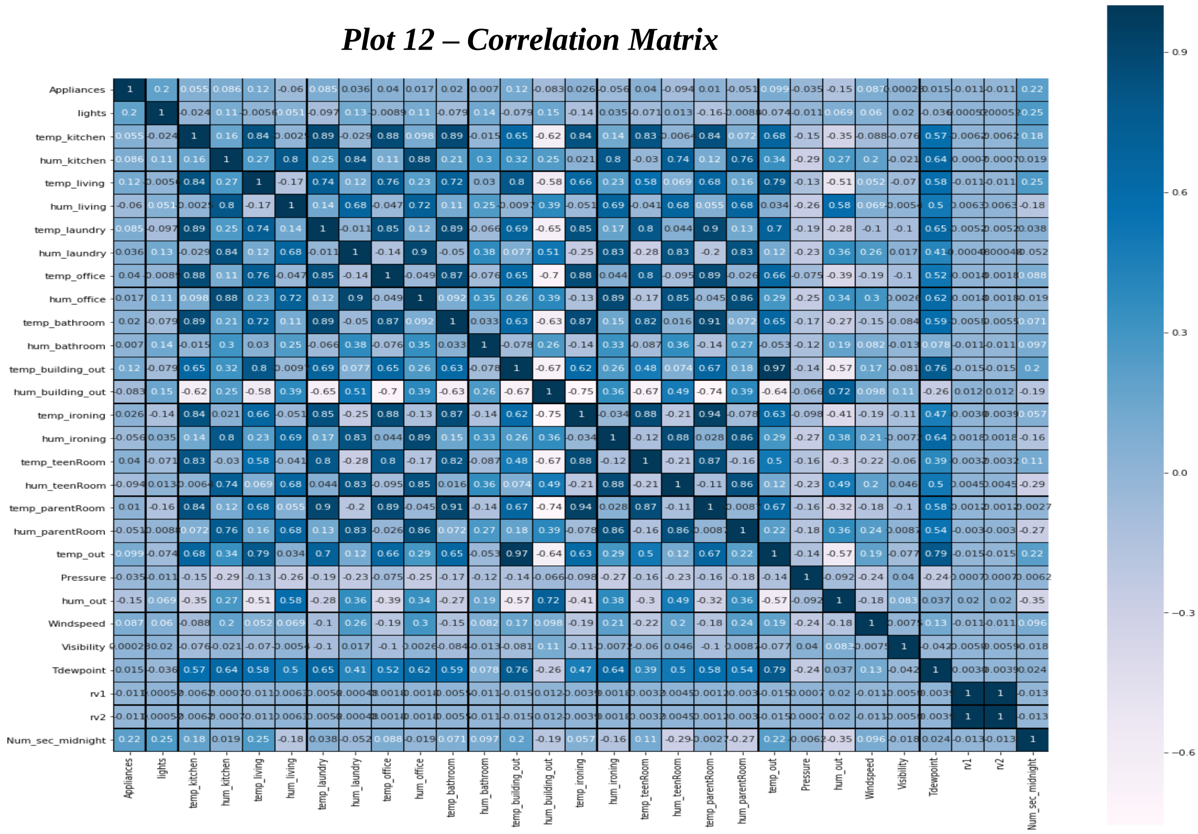
Pair plot are plotted in such a way to discriminate weekday and week-end values. Since there will always be a difference in energy usage between a week-end and weekday.

Plot 11 – One of the pair plot with 8 variables



Inorder to understand the correlation between variables on values bases refer ***Plot 12.*** The below correlation matrix clearly illustrates that there is a high correlation between 'Num_sec_midnight' and 'Appliances' with a value of 0.22. The second largest correlation with 'Appliances' is 'Lights' with a value of 0.19. For the indoor temperatures, the correlations are high as expected, since the ventilation is driven by the HRV unit and minimizes air temperature differences between rooms. For example, a positive correlation is found with 'temp_kitchen' and 'temp_laundry'. And correlation of outdoor temperature with the appliances is 0.12. There is also a negative correlation between the appliances and outdoor humidity/RH6 (−.09. There is a positive correlations between the consumption of appliances and temp_ironing, temp_teenRoom and temp_parentRoom being 0.03, 0.05 and 0.02 respectively. A positive correlation of 0.10 is seen between appliances' consumption and outdoor temperature (Tout) that is, the higher temperatures, the higher the energy use by the appliances. Also there is a positive correlation with appliances' consumption and wind speed (0.09), higher wind speeds correlate with higher energy consumption by the appliances. A negative correlation of −0.15 was found with the 'hum_out', and of −0.03 with pressure. Another important and interesting correlation is between the pressure and the wind speed. This relationship is negative (−0.23)

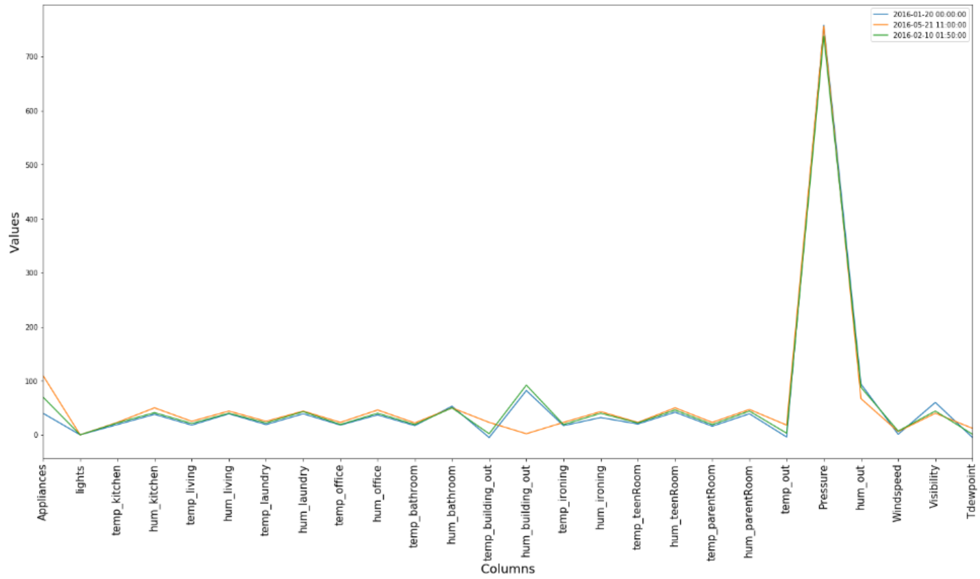
Plot 12 – Correlation Matrix



Plot 13 – provides the value of all variables for given dates. It helps us compare values on three different dates.

Plot 13

```
In [72]: def plotDatetimeWise(datetime):
             fig, ax = plt.subplots()
             count = 1
             for time in datetime:
                 newdf = Data[Data.columns[0:26]]
                 df_player = newdf.loc[time]
                 df_player = df_player.astype('int')
                 df_player.T.plot.line(figsize = (24,12),ax=ax)
                 ax.legend()
                 ax.set_xlabel('Columns',fontsize=18)
                 ax.set_ylabel('Values',fontsize=18)
                 ax.set_xticks(np.arange(len(newdf.columns)))
                 ax.set_xticklabels(labels = newdf.columns, rotation=90,fontsize=15)
                 count = count + 1
         plotDatetimeWise(['2016-01-20 00:00:00', '2016-05-21 11:00:00', '2016-02-10 01:50:00'])
```



Feature Engineering

In order move forward in our analysis, this section will help us perform feature engineering. Before we move ahead, all categorical column which was used in EDA section needs to be converted to numerical.

Plot 14- Categorical Columns

```
In [162]: Data = pd.DataFrame.from_csv("modifiedData.csv")
          Data
```

ndry	hum_laundry	temp_office	hum_office	...	Pressure	hum_out	Windspeed	Visibility	Tdewpoint	rv1	rv2	Num_sec_midnight	Day_Status	week_status
0000	44.730000	19.000000	45.566667	...	733.500000	92.000000	7.000000	63.000000	5.300000	13.275433	13.275433	61200	Monday	weekday
0000	44.790000	19.000000	45.992500	...	733.600000	92.000000	6.666667	59.166667	5.200000	18.606195	18.606195	61800	Monday	weekday
0000	44.933333	18.926667	45.890000	...	733.700000	92.000000	6.333333	55.333333	5.100000	28.642668	28.642668	62400	Monday	weekday
0000	45.000000	18.890000	45.723333	...	733.800000	92.000000	6.000000	51.500000	5.000000	45.410389	45.410389	63000	Monday	weekday
0000	45.000000	18.890000	45.530000	...	733.900000	92.000000	5.666667	47.666667	4.900000	10.084097	10.084097	63600	Monday	weekday

Plot 15 – One hot encoded Numerical columns

In [163]:
```
label_encoder = LabelEncoder()
int_encoded = label_encoder.fit_transform(Data['week_status'])
int_encoded_day = label_encoder.fit_transform(Data['Day_Status'])
onehot_encoder = OneHotEncoder(sparse=False)
int_encoded = int_encoded.reshape(len(int_encoded), 1)
int_encoded_day = int_encoded_day.reshape(len(int_encoded_day), 1)
newWeek = onehot_encoder.fit_transform(int_encoded)
newDay = onehot_encoder.fit_transform(int_encoded_day)
# new2 = label_encoder.inverse_transform([argmax(new[len(new)-1, :])])
Data.drop(['week_status', 'Day_Status'], axis=1, inplace=True)
Data['Friday'] = pd.Series(newDay[:,0], index=Data.index)
Data['Monday'] = pd.Series(newDay[:,1], index=Data.index)
Data['Saturday'] = pd.Series(newDay[:,2], index=Data.index)
Data['Sunday'] = pd.Series(newDay[:,3], index=Data.index)
Data['Thursday'] = pd.Series(newDay[:,4], index=Data.index)
Data['Tuesday'] = pd.Series(newDay[:,5], index=Data.index)
Data['Wednesday'] = pd.Series(newDay[:,6], index=Data.index)
Data['WeekDay'] = pd.Series(newWeek[:,0], index=Data.index)
Data['Weekend'] = pd.Series(newWeek[:,1], index=Data.index)
Data
```

_living	temp_laundry	hum_laundry	temp_office	hum_office	...	Num_sec_midnight	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday	WeekDay	Weekend
'90000	19.790000	44.730000	19.000000	45.566667	...	61200	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
'22500	19.790000	44.790000	19.000000	45.992500	...	61800	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
326667	19.790000	44.933333	18.926667	45.890000	...	62400	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
590000	19.790000	45.000000	18.890000	45.723333	...	63000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
530000	19.790000	45.000000	18.890000	45.530000	...	63600	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

The data was converted to numerical using One hot encoding. Now those columns can be analyzed for prediction models. In general, with machine learning, you ideally want your data normalized, which means all features are on a similar scale. ***Plot 16*** shows the normalized data so that all variables are on similar scale.

Plot 16 – Normalised Data

In [164]:
```
centered_scaled_data = pd.DataFrame(data=Data)
for j in range(1, len(Data.columns)-1,1):
    centered_scaled_data.iloc[:,[j]] = (Data.iloc[:,[j]] - Data.iloc[:,[j]].mean())/Data.iloc[:,[j]].std()
centered_scaled_data
```

date	Appliances	lights	temp_kitchen	hum_kitchen	temp_living	hum_living	temp_laundry	hum_laundry	temp_office	hum_office	...	Num_sec_midnight	Frida
2016-01-11 17:00:00	60	3.301180	-1.118616	1.843774	-0.520398	1.073656	-1.235032	1.686087	-0.908194	1.506399	...	-0.007118	-0.41040
2016-01-11 17:10:00	60	3.301180	-1.118616	1.616766	-0.520398	1.057071	-1.235032	1.704523	-0.908194	1.604488	...	140.474197	-0.41040
2016-01-11 17:20:00	50	3.301180	-1.118616	1.517921	-0.520398	1.033523	-1.235032	1.748563	-0.944091	1.580878	...	-0.007118	-0.41040
2016-01-11 17:30:00	50	4.561263	-1.118616	1.459284	-0.520398	1.024514	-1.235032	1.769047	-0.962039	1.542487	...	-0.007118	-0.41040
2016-01-11 17:40:00	60	4.561263	-1.118616	1.526298	-0.520398	1.009771	-1.235032	1.769047	-0.962039	1.497953	...	-0.007118	-0.41040

We have performed feature analysis and importance with different algorithms to compare and contrast along with a summary plot of all three algorithms (***Plot 20***).

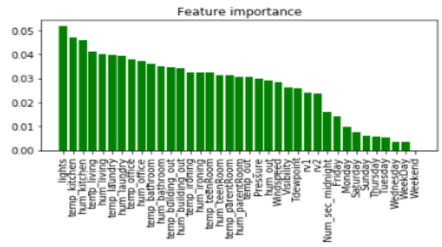
Plot 17 - Feature analysis and Importance using ExtraTreesRegresso

```
In [186]: y=centered_scaled_data['Appliances']
          df4 = centered_scaled_data.iloc[:,1:]
          X_train, X_test, y_train, y_test = train_test_split(df4, y, test_size=0.25)
          train = X_train.join(y_train)
          test = X_test.join(y_test)
          train.to_csv("train.csv")
          test.to_csv("test.csv")
```

```
In [191]: model = ExtraTreesRegressor(n_estimators=100, n_jobs=4, verbose=2)
          X_train = train.iloc[:,:len(train.columns)-1]
          y_train = train.iloc[:,len(train.columns)-1]
          model.fit(X_train, y_train)
          importance = model.feature_importances_
          importance
```

```
Out[191]: array([  5.15971693e-02,   3.22520062e-02,   4.58708757e-02,
                   3.72206213e-02,   3.98477188e-02,   4.00472759e-02,
                   4.12072995e-02,   3.07182223e-02,   3.22820970e-02,
                   3.05000488e-02,   3.42650333e-02,   3.45197452e-02,
                   3.80193657e-02,   2.85510576e-02,   3.12966900e-02,
                   3.50920844e-02,   3.91753836e-02,   2.60752617e-02,
                   3.25718757e-02,   2.91096685e-02,   3.14783271e-02,
                   4.71571169e-02,   3.61428190e-02,   2.58102670e-02,
                   2.99299287e-02,   2.36006691e-02,   2.39042982e-02,
                   9.97279914e-09,   1.42884117e-02,   1.59107098e-02,
                   9.77552601e-03,   5.69106213e-03,   6.00044742e-03,
                   7.44959524e-03,   5.53022074e-03,   3.61855161e-03,
                   3.49253892e-03])
```

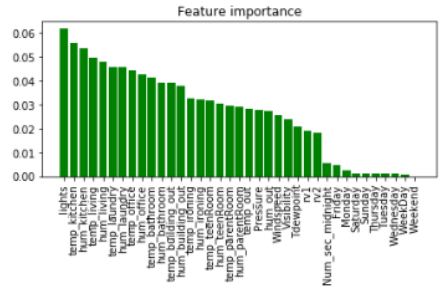
```
In [202]: def plotFImp(importance, X_train):
              indices = np.argsort(importance)[::-1]
              plt.bar(range(X_train.shape[1]), importance[indices], color='green', align = 'center')
              plt.title('Feature importance')
              plt.xticks(range(X_train.shape[1]), X_train.columns, rotation=90)
              plt.tight_layout()
              plt.show()

          plotFImp(importance, X_train)
```



Plot 18 – Feature analysis and Importance using RandomForestRegressor

```
In [204]: model = RandomForestRegressor(n_estimators=300, n_jobs=4, verbose=2)
          model.fit(X_train, y_train)
          importance_rf = model.feature_importances_
          importance_rf

          building tree 24 of 300
          building tree 25 of 300
          building tree 26 of 300
          building tree 27 of 300
          building tree 28 of 300
          building tree 29 of 300
          building tree 30 of 300
          building tree 31 of 300
          building tree 32 of 300
          building tree 33 of 300
          building tree 34 of 300
          building tree 35 of 300
          building tree 36 of 300

          [Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:    9.3s

          building tree 37 of 300
          building tree 38 of 300
          building tree 39 of 300
          building tree 40 of 300
```

```
In [205]: plotFImp(importance_rf, X_train)
```

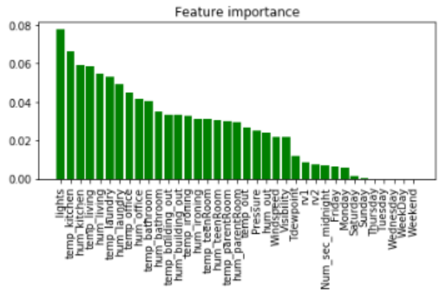


Plot 19-Feature analysis and Importance using GradientBoostingRegressor

In [206]:
```
model = GradientBoostingRegressor(n_estimators=300, verbose=2)
model.fit(X_train, y_train)
importance_gb = model.feature_importances_
importance_gb
```

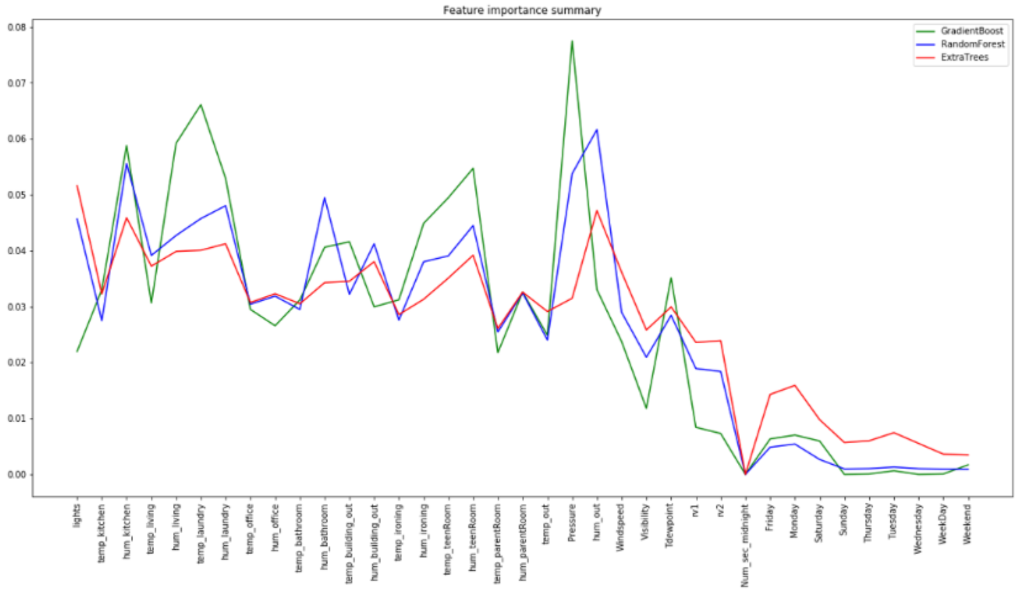
```
        66          7556.8445            17.54s
        67          7535.1385            17.48s
        68          7522.5440            17.36s
        69          7501.9980            17.20s
        70          7480.3593            17.14s
        71          7473.1660            16.96s
        72          7447.0669            16.90s
        73          7430.6539            16.85s
        74          7409.1419            16.75s
        75          7386.9083            16.71s
        76          7378.7758            16.54s
        77          7354.2229            16.43s
        78          7339.7170            16.35s
        79          7321.9759            16.18s
        80          7313.1354            16.04s
        81          7303.8624            15.93s
        82          7282.0299            15.79s
        83          7262.3837            15.81s
        84          7253.6775            15.86s
        85          7232.5311            15.76s
```

In [207]:
```
plotFImp(importance_gb, X_train)
```



Plot 20 – Feature Importance summary

In [222]:
```
rcParams['figure.figsize'] = 20,10
plt.plot(range(X_train.shape[1]), importance_gb, color='g')
plt.plot(range(X_train.shape[1]), importance_rf, color='b')
plt.plot(range(X_train.shape[1]), importance, color='r')
plt.title('Feature importance summary')
plt.xticks(range(X_train.shape[1]), X_train.columns, rotation=90)
plt.legend(('GradientBoost', 'RandomForest', 'ExtraTrees'))
plt.show()
```

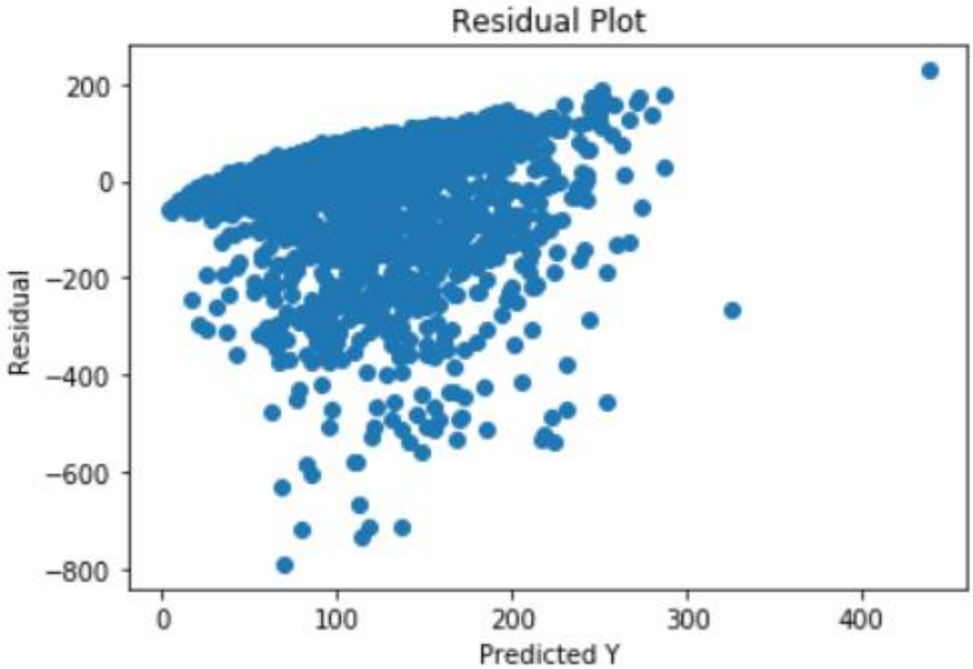


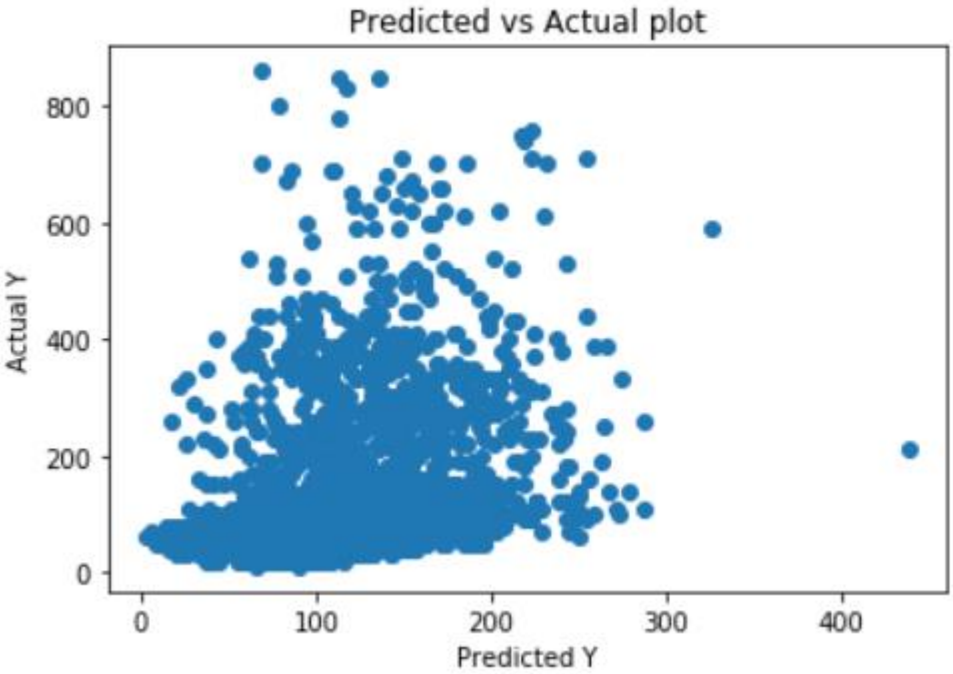
Prediction algorithms

In this section we have considered, four different algorithms - Linear Regression, Random Forests, Neural Networks and Gradient Boosting Regression and compared its values for selecting better prediction model for our regression model. Before learning the data, we have split data into 75% of train and 25% of test sets.

The residual plot and the actual plot are as follows.

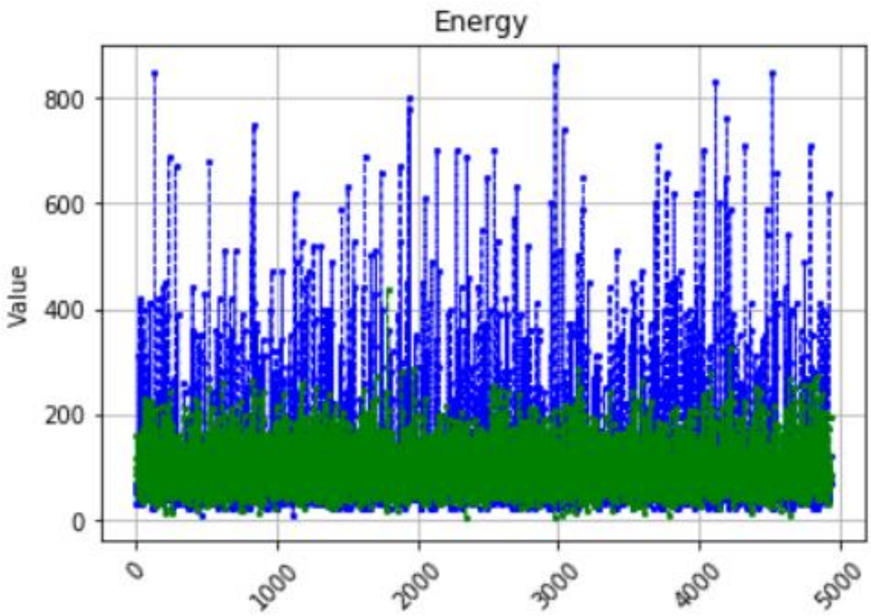
Linear Regression Plots



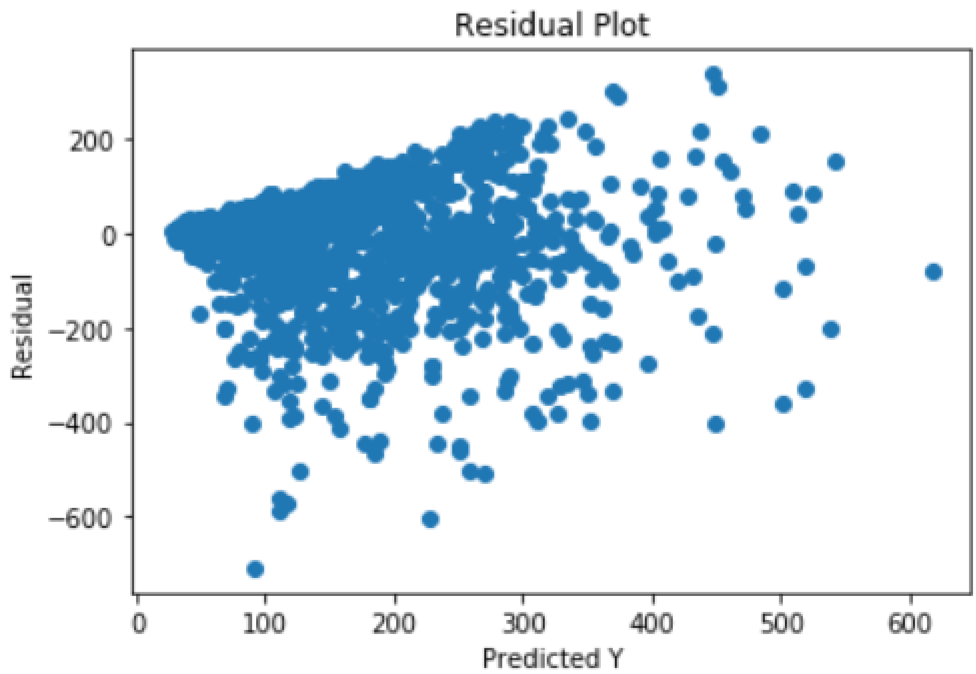


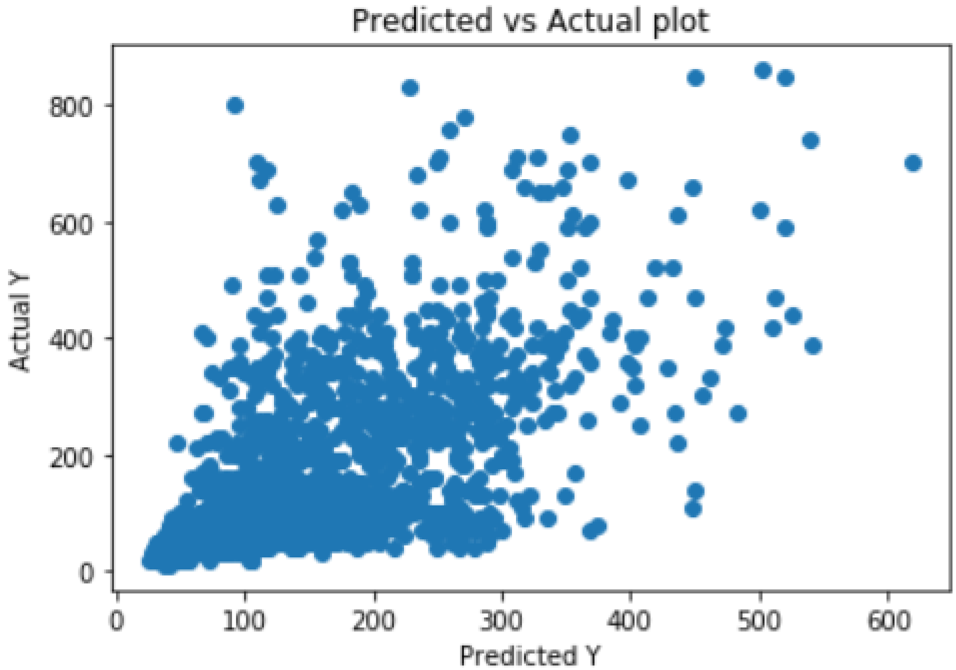
Predicted Y

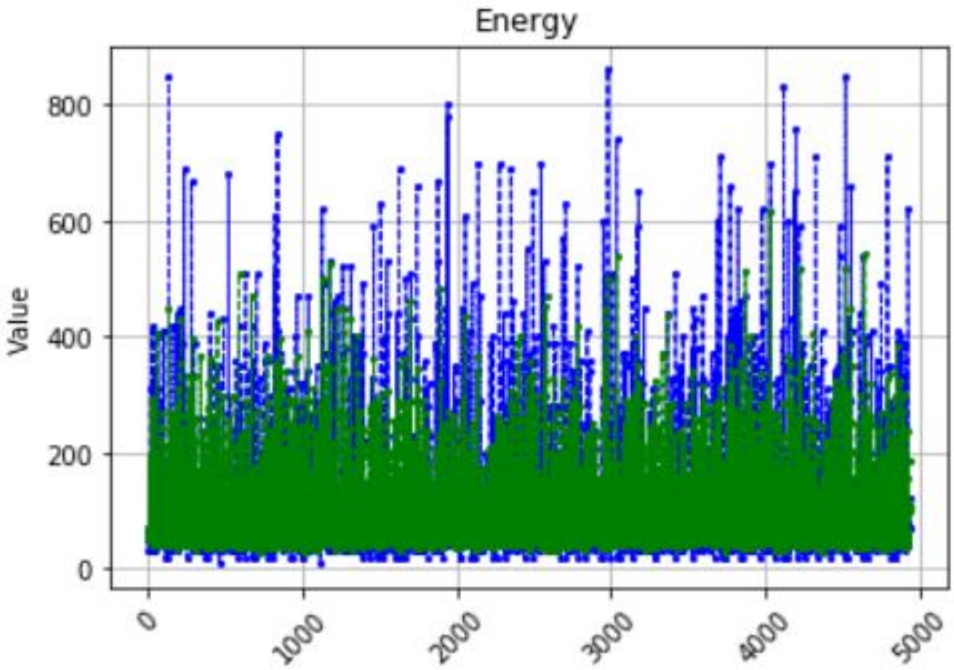
```
In [215]: plot_trend(pred,y_test,'Energy')
```



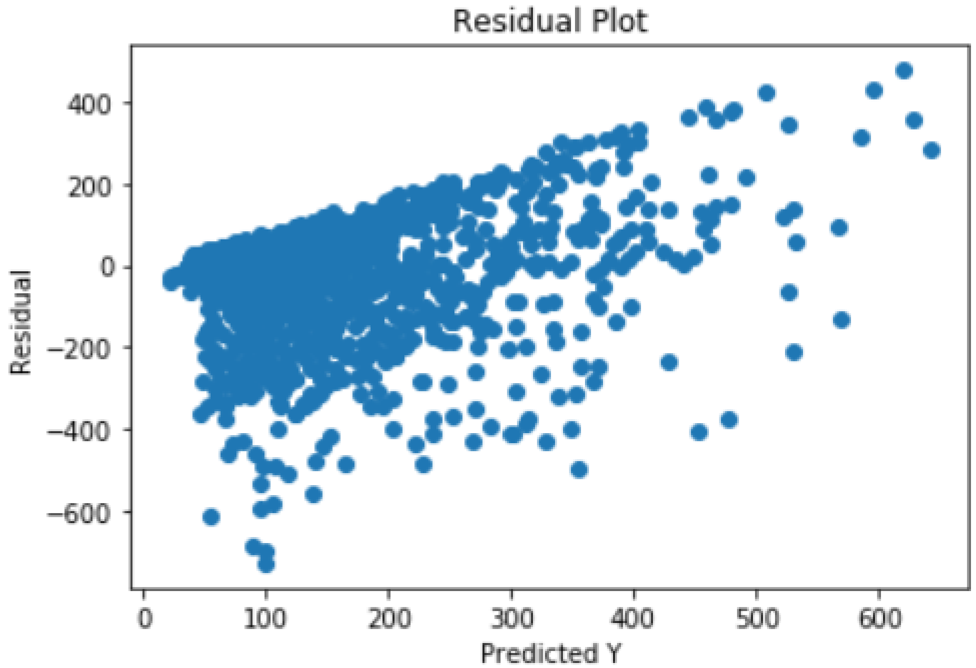
Random Forest plots

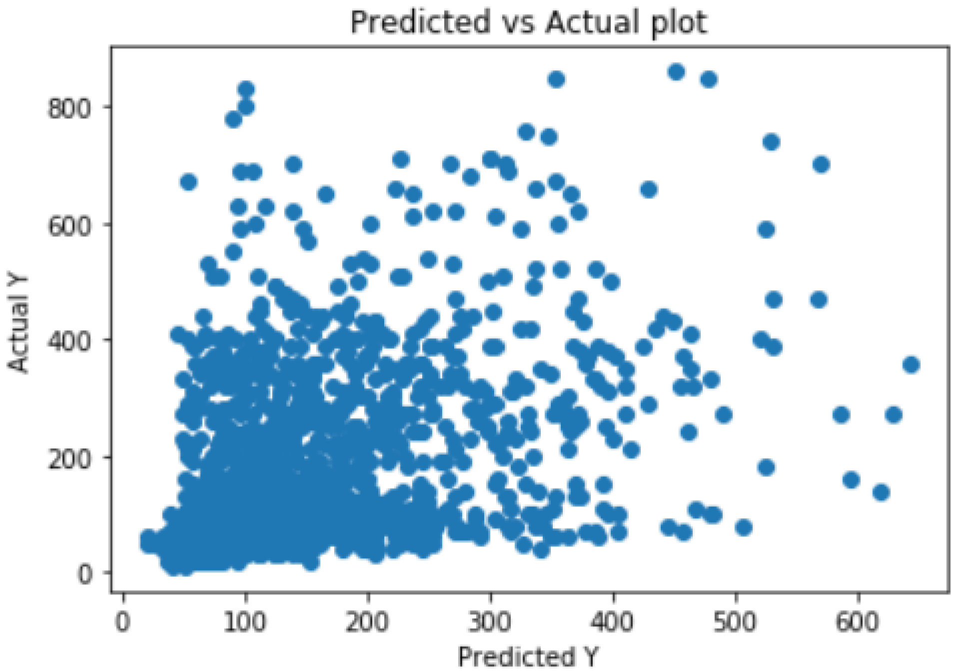

Residual Plot
200
0
−200
−400
−600
Residual
0
100
200
300
400
500
600
Predicted Y

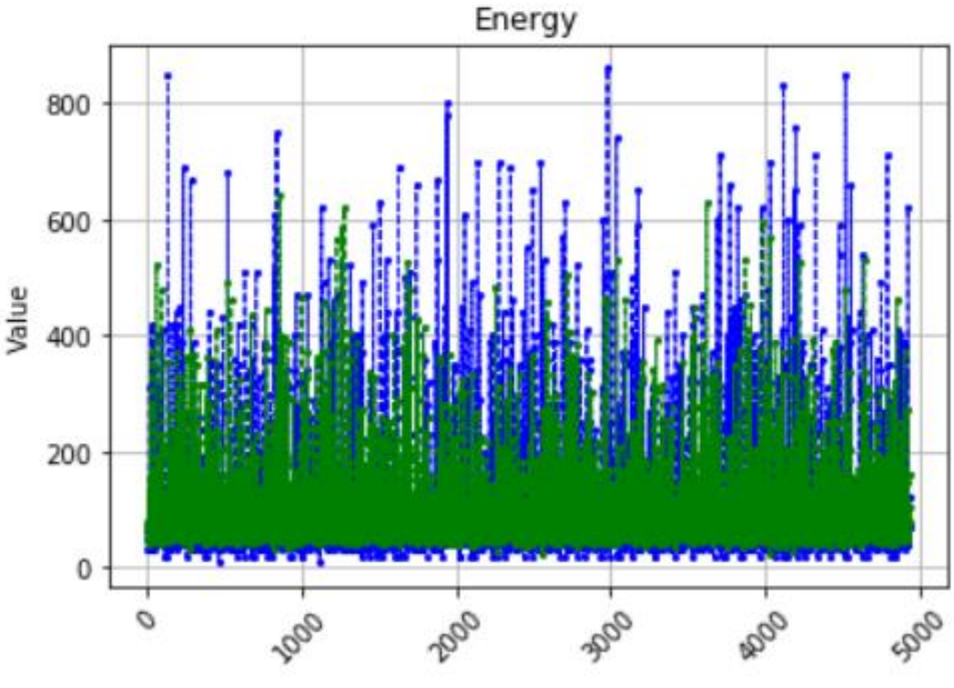

Predicted vs Actual plot
800
600
400
200
0
Actual Y
0
100
200
300
400
500
600
Predicted Y



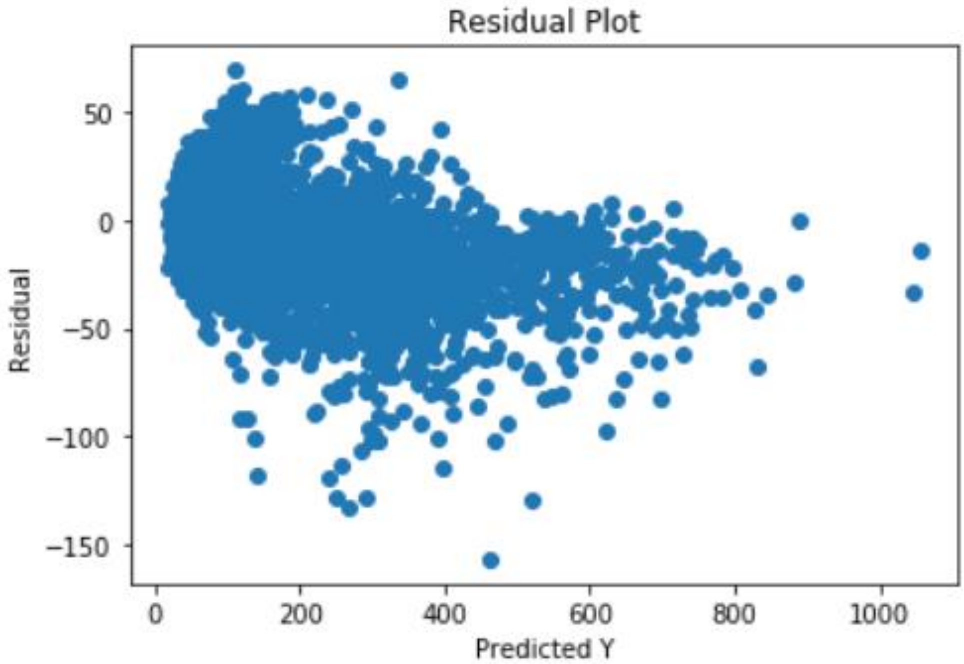
Neural Network plots

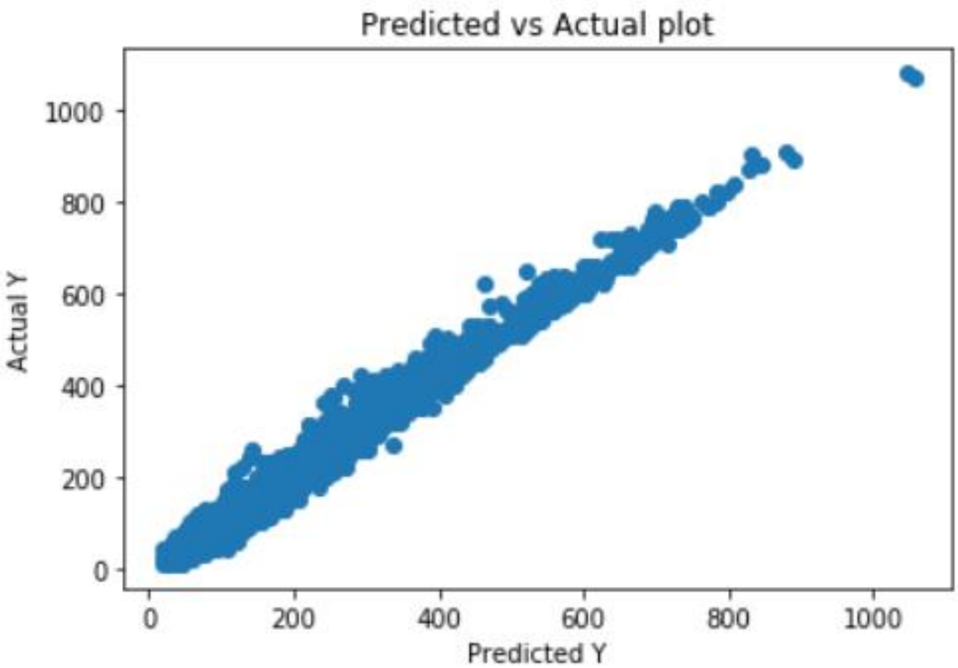

Residual Plot
Residual
400
200
0
−200
−400
−600
0
100
200
300
400
500
600
Predicted Y

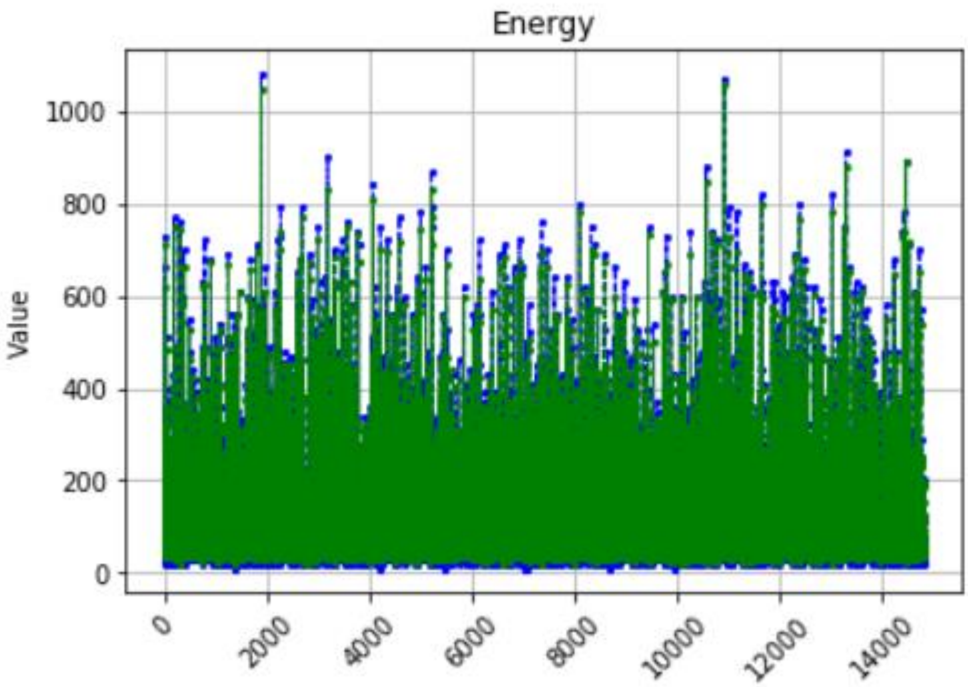

Predicted vs Actual plot
Actual Y
800
600
400
200
0
0
100
200
300
400
500
600
Predicted Y



Gradient Boosting Regression plots


Residual Plot
50
0
−50
−100
−150
Residual
0
200
400
600
800
1000
Predicted Y


Predicted vs Actual plot
1000
800
600
400
200
0
Actual Y
0
200
400
600
800
1000
Predicted Y



The summary metrics of each of these algorithms with the training sets on the left and the testing set on the right are as follows.

Plot 21- Summary of prediction models

	Linear	RandomForest	NeuralNetwork	GradientBoosting
MAE	53.204941	12.646981	37.594546	10.386794
MAPE	61.377133	12.701410	39.962768	14.621911
RMSE	94.338395	26.562544	72.393883	15.176064
R2	0.170316	0.934223	0.511416	0.978529

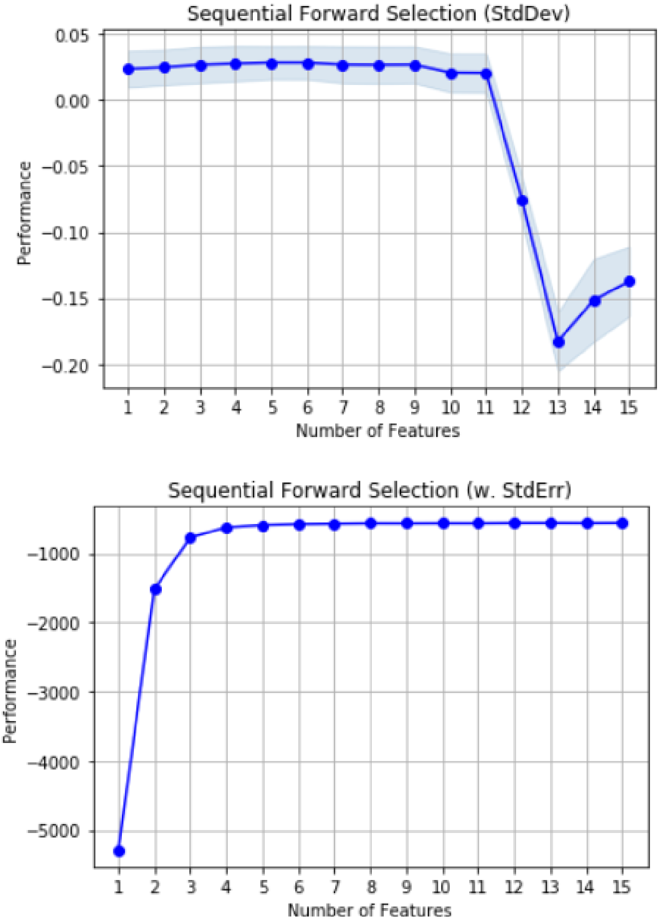
	Linear	RandomForest	NeuralNetwork	GradientBoosting
MAE	52.063485	32.304128	40.901195	34.103741
MAPE	60.313166	32.829551	41.772017	36.279194
RMSE	90.222844	67.163081	80.355282	66.313426
R2	0.174676	0.542646	0.345333	0.554144

The above figure clearly states that only Random Forest and Gradient Boost are better compared to linear and neural network. **So we have selected Random Forest as primary and Gradient Boost as secondary option and will finalize one of the 2 after cross validations and regularizations.**

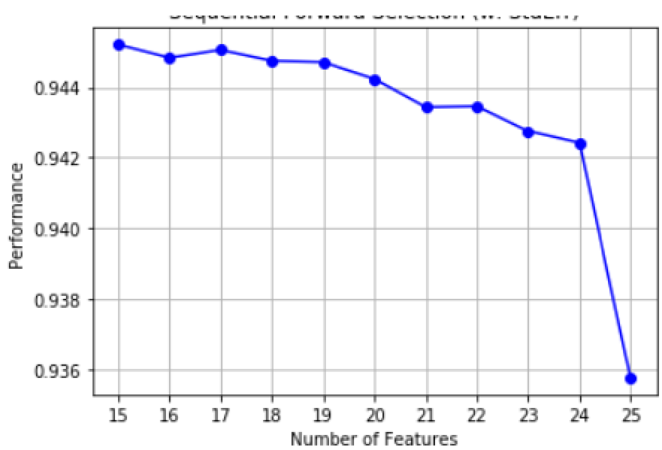
Feature Selection

Feature selection is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested. We have considered different feature selection such as Sequential Forward Selection of feature in RandomForest, Sequential backward selection of feature in RandomForest, Feature Selection using TPOT, RandomForest using TSFresh feature selection, RandomForest using Boruta feature selection, Boruta and RandomForest using Exhaustive Search.

Plot 22- Sequential Forward Selection



Plot 23- Sequential Backward Selection



Plot 24- Feature selection using TPOT

In [19]:
```
from tpot import TPOTRegressor
my_tpot = TPOTRegressor(generations=50, population_size=1000,n_jobs=8, early_stop=5, offspring_size=600, verbosity=2, config_dict
result = my_tpot.fit(X_train, y_train)
```

```
Generation 1 - Current best internal CV score: -6166.952302793087

Generation 2 - Current best internal CV score: -5980.126525587899

Generation 3 - Current best internal CV score: -5980.126525587899

Generation 4 - Current best internal CV score: -5551.999825036415

Generation 5 - Current best internal CV score: -5551.999825036415

Generation 6 - Current best internal CV score: -5551.999825036415
```

In [21]:
```
print(my_tpot.score(X_test, y_test))
```

```
-3828.21616094
```

In [22]:
```
my_tpot.export('tpot_light_pipeline.py')
```

Out[22]: True

Plot 25- Feature selection using TSFRESH

RandomForest using TSFresh feature selection

In [8]:
```
df_shift, y = make_forecasting_frame(Data["Appliances"], kind="Watts", max_timeshift=10, rolling_direction=1)
X = extract_features(df_shift, column_id="id", column_sort="time", column_value="value",show_warnings=False, impute_function=impu
X
```

```
 'value__fft_coefficient__coeff_97__attr_"imag"'
 'value__fft_coefficient__coeff_97__attr_"real"'
 'value__fft_coefficient__coeff_98__attr_"abs"'
 'value__fft_coefficient__coeff_98__attr_"angle"'
 'value__fft_coefficient__coeff_98__attr_"imag"'
 'value__fft_coefficient__coeff_98__attr_"real"'
 'value__fft_coefficient__coeff_99__attr_"abs"'
 'value__fft_coefficient__coeff_99__attr_"angle"'
 'value__fft_coefficient__coeff_99__attr_"imag"'
 'value__fft_coefficient__coeff_99__attr_"real"'
 'value__fft_coefficient__coeff_9__attr_"abs"'
 'value__fft_coefficient__coeff_9__attr_"angle"'
 'value__fft_coefficient__coeff_9__attr_"imag"'
 'value__fft_coefficient__coeff_9__attr_"real"'

 'value__spkt_welch_density__coeff_8'] did not have any finite values. Filling with zeros.
```

Out[8]:

variable	value__abs_energy	value__absolute_sum_of_changes	value__agg_autocorrelation__f_agg_"mean"	value__agg_autocorrelation__f_agg_"median"	value_

In [14]:
```
X.head()
```

Out[14]:

variable	value__abs_energy	value__absolute_sum_of_changes	value__agg_autocorrelation__f_agg_"mean"	value__agg_autocorrelation__f_agg_"median"	value__ag
id					
2016-01-11 17:10:00	3600.0	0.0	0.000000	0.0000	
2016-01-11 17:20:00	7200.0	0.0	0.000000	0.0000	
2016-01-11 17:30:00	9700.0	10.0	-0.625000	-0.6250	
2016-01-11 17:40:00	12200.0	10.0	-0.555556	-1.0000	
2016-01-11 17:50:00	15800.0	20.0	-0.114583	-0.0625	

5 rows × 283 columns

Plot 26- Feature selection using BORUTA

RandomForest using Boruta feature selection

In [6]:
```
regressor = RandomForestRegressor(n_estimators=100, n_jobs=2)
feat_selector = BorutaPy(regressor, n_estimators='auto', verbose=2)
feat_selector.fit(X_train.as_matrix(),y_train.as_matrix())
```

```
BorutaPy finished running.

Iteration:      100 / 100
Confirmed:      19
Tentative:      4
Rejected:       14
```

Out[6]:
```
BorutaPy(alpha=0.05,
     estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=67, n_jobs=2,
           oob_score=False,
           random_state=<mtrand.RandomState object at 0x000001CEE1244480>,
           verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x000001CEE1244480>,
     two_step=True, verbose=2)
```

In [11]:
```
def determineAnalysis(true, pred, regressor):
    mae = mean_absolute_error(true, pred)
    rmse = sqrt(mean_squared_error(true, pred))
    r2 = r2_score(true, pred)
    true, pred = np.array(true), np.array(pred)
    mape = np.mean(np.abs((true - pred) / true)) * 100
    n =len(X_train)
    r2_adj =1- (1-r2)*(n-1)/(n-(len(regressor.estimator_params)+1))
    print('Mean absolute error is ',mae)
    print('Mean absolute percentage error is ',mape)
    print('Root mean squared error is ',rmse)
    print('RSquare is ',r2)
    print('RSquare adjusted ',r2_adj)

determineAnalysis(y_test, y_pred, regressor)
```

```
Mean absolute error is  34.6887572676
Mean absolute percentage error is  32.7674653027
Root mean squared error is  75.59053177565613
RSquare is  0.492268404548
RSquare adjusted  0.491925110704
```

We have considered Boruta as our final selection as it provided us with RSquare better than any other selections.

Model Validation and Selection

A better sense of a model's performance can be found using what's known as a *holdout set*: that is, we hold back some subset of the data from the training of the model, and then use this holdout set to check the model performance. This splitting can be done using the train_test_split utility in Scikit-Learn.

Plot 26 – Holdout sets

```
In [6]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

One disadvantage of using a holdout set for model validation is that we have lost a portion of our data to the model training. In the above case, half the dataset does not contribute to the training of the model. 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. In this section would have compared various algorithms with cross validation and performed grid_search for best model selection.

Plot 27 – K-fold cross validation

```
In [11]: #GBM
model = GradientBoostingRegressor(n_estimators=1500, learning_rate=0.1,max_depth=4, random_state=0, loss='ls').fit(X, y)

In [12]:
k_fold = KFold(len(y), n_folds=10, shuffle=True, random_state=0)
print (cross_val_score(model, X, y, cv=k_fold, n_jobs=1))

[ 0.51596998  0.52889133  0.52898749  0.53374406  0.52356115  0.5181723
  0.54088485  0.52148472  0.58003088  0.55897463]

In [68]: #RandomForest
RF= RandomForestRegressor(n_estimators=300,max_depth=4,)
model2=RF.fit(X,y)

In [22]: k_fold = KFold(len(y), n_folds=10, shuffle=True, random_state=0)
print (cross_val_score(model2, X, y, cv=k_fold, n_jobs=1))

[0.12739586 0.14106554 0.16019255 0.1566742  0.11456478 0.1316116
 0.13834619 0.15287295 0.1728952  0.14669629]

In [71]: #Linear
k_fold = KFold(len(y), n_folds=10, shuffle=True, random_state=0)
lm = linear_model.LinearRegression()
model3 = lm.fit(X, y)
print (cross_val_score(model3, X, y, cv=k_fold, n_jobs=1))

In [7]: #NeuralNetwork
mlp = MLPRegressor(hidden_layer_sizes=(50,50,50,50,50),max_iter=500)
mlp.fit(X,y)
train_predict=mlp.predict(X)
print (cross_val_score(mlp, X, y, cv=k_fold, n_jobs=1))

[0.36643616 0.36945491 0.36409526 0.35757147 0.3805682  0.3805224
 0.40229774 0.41789279 0.40343481 0.34727024]
```

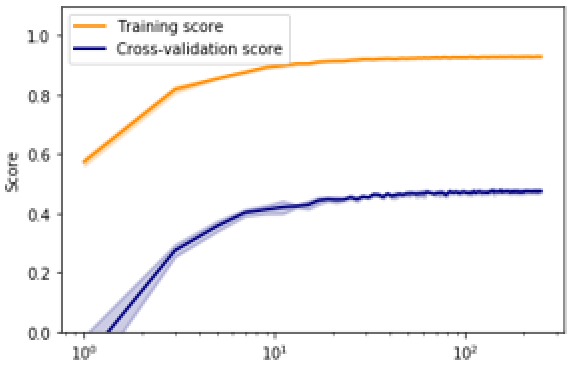
The dataset also run through Leave one out exhaustive cross validation but didn't complete its process.

Plot 28 – Leave one out cross validation

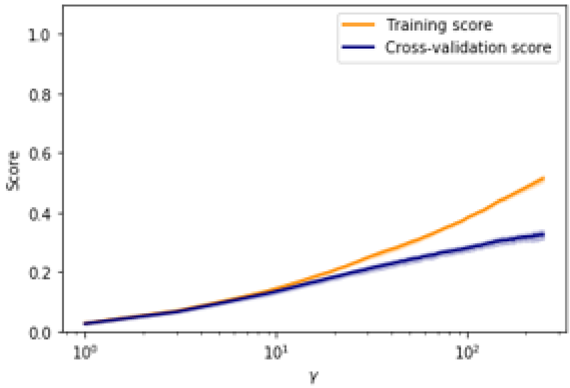
```
In [ ]: #GBM
        scores = cross_val_score(model, X, y, cv=LeaveOneOut(len(X)), n_jobs=8, verbose=2)
        scores

[Parallel(n_jobs=8)]: Done  25 tasks      | elapsed:  4.2min
[Parallel(n_jobs=8)]: Done 146 tasks      | elapsed: 21.0min
[Parallel(n_jobs=8)]: Done 349 tasks      | elapsed: 49.6min

In [ ]: #Neural Network
        scores = cross_val_score(mlp, X, y, cv=LeaveOneOut(len(X)), n_jobs=8, verbose=2)
        scores

[Parallel(n_jobs=8)]: Done  25 tasks      | elapsed:  4.3min
[Parallel(n_jobs=8)]: Done 146 tasks      | elapsed: 388.9min

In [ ]: #Random forest
        scores_rf = cross_val_score(model2, X, y, cv=LeaveOneOut(len(X)))
        scores_rf

In [25]: #LinearRegression
         scores3 = cross_val_score(model3, X, y, cv=LeaveOneOut(len(X)), n_jobs=8, verbose=2)
         scores3

[Parallel(n_jobs=8)]: Done  25 tasks      | elapsed:   27.2s
[Parallel(n_jobs=8)]: Done 372 tasks      | elapsed:   29.1s
[Parallel(n_jobs=8)]: Done 1184 tasks      | elapsed:   33.6s
[Parallel(n_jobs=8)]: Done 2316 tasks      | elapsed:   40.0s
[Parallel(n_jobs=8)]: Done 3776 tasks      | elapsed:   48.1s
[Parallel(n_jobs=8)]: Done 5556 tasks      | elapsed:   57.8s
[Parallel(n_jobs=8)]: Done 7664 tasks      | elapsed:  1.2min
[Parallel(n_jobs=8)]: Done 10092 tasks      | elapsed:  1.4min
[Parallel(n_jobs=8)]: Done 12848 tasks      | elapsed:  1.6min
[Parallel(n_jobs=8)]: Done 15924 tasks      | elapsed:  1.9min
[Parallel(n_jobs=8)]: Done 19328 tasks      | elapsed:  2.2min
[Parallel(n_jobs=8)]: Done 19720 out of 19735 | elapsed:  2.3min remaining:    0.0s
[Parallel(n_jobs=8)]: Done 19735 out of 19735 | elapsed:  2.3min finished

Out[25]: array([0., 0., 0., ..., 0., 0., 0.])
```

From the below validation curve, we can read-off that the optimal trade-off between bias and variance is found for a third-order polynomial; we can compute and display this fit over the original data as follows:

Plot -29 – Validation curve with Random Forest



Plot -30 – Validation curve with Gradient Boost



Regularization methods like L1, L2 and Elastic net were also tried but their RSquare scores were very less. The following are the metrics and plots for Lasso Regression, Ridge experimented with different Alpha values and ElasticNet

Lasso Regression

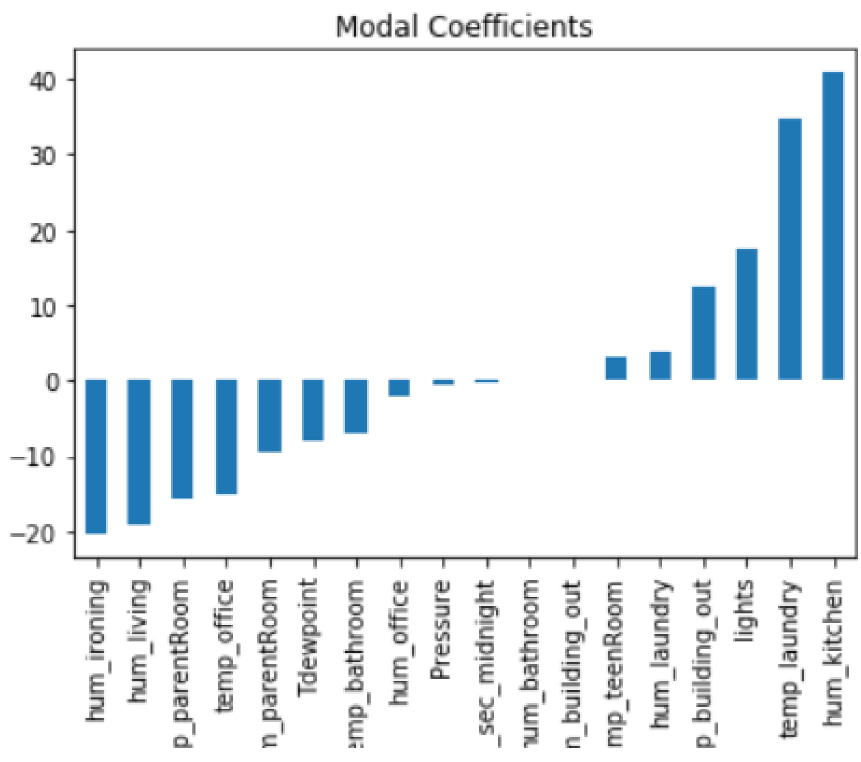
In [23]:

```
alphas = 10**np.linspace(10,-2,100)*0.5
lasso = Lasso(alpha= 0,max_iter = 10000, normalize = True)
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(scale(X_train), y_train)
    coefs.append(lasso.coef_)
pred = lasso.predict(X_test)
```

In [39]:

```
predictors = X_train.columns
coefs = pd.Series(lasso.coef_,predictors).sort_values()
coefs.plot(kind='bar', title='Modal Coefficients')
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x21e6ead8f98>



LassoCV

n [49]:
```
lassocv = LassoCV(alphas = alphas, cv = 10, max_iter = 100000, normalize = True)
lassocv.fit(X_train, y_train)

pred_test=lasso.predict(X_test)
mean_squared_error(y_test, lassocv.predict(X_test))
```

ut[49]: 10650.179025823614

n [50]:
```
r2_score(y_test,pred_test)
```

ut[50]: 0.020680150424673638

Ridge

Alpha Value=4

52]:
```
ridge2 = Ridge(alpha = 4, normalize = True)
ridge2.fit(X_train, y_train)             # Fit a ridge regression on the training data
pred2 = ridge2.predict(X_test)           # Use this model to predict the test data
print(pd.Series(ridge2.coef_, index = X_train.columns)) # Print coefficients
print("Mean_Square",mean_squared_error(y_test, pred2))          # Calculate the test MSE
```

```
lights              3.586090e+01
hum_kitchen         1.715271e+01
hum_living         -1.146545e+01
temp_laundry        8.530177e+00
hum_laundry         6.239689e+00
temp_office         1.329891e+00
hum_office          1.640862e+00
temp_bathroom      -8.420544e-01
hum_bathroom        9.319668e-01
temp_building_out   1.157327e+01
hum_building_out   -4.257555e+00
hum_ironing        -7.858957e+00
temp_teenRoom       2.388946e+00
temp_parentRoom    -1.804204e+00
hum_parentRoom     -7.324282e+00
Pressure           -3.936877e+00
Tdewpoint          -9.087726e-01
Num_sec_midnight   -1.480205e-12
dtype: float64
Mean_Square 10339.8893777
```

Alpha = 10**10

```
ridge3 = Ridge(alpha = 10**10, normalize = True)
ridge3.fit(X_train, y_train)             # Fit a ridge regression on the training data
pred3 = ridge3.predict(X_test)           # Use this model to predict the test data
print(pd.Series(ridge3.coef_, index = X_train.columns)) # Print coefficients
print(mean_squared_error(y_test, pred3))          # Calculate the test MSE
```

```
lights               1.776388e-08
hum_kitchen          7.739085e-09
hum_living          -5.363827e-09
temp_laundry         5.406176e-09
hum_laundry          2.307141e-09
temp_office          2.498462e-09
hum_office           7.486585e-10
temp_bathroom        1.354518e-09
hum_bathroom         3.868356e-10
temp_building_out    6.727257e-09
hum_building_out    -2.796849e-09
hum_ironing         -3.360450e-09
temp_teenRoom        2.658894e-09
temp_parentRoom      7.774290e-10
hum_parentRoom      -3.455197e-09
Pressure            -2.216885e-09
Tdewpoint            7.411781e-10
Num_sec_midnight    -5.561036e-22
dtype: float64
10650.1790256
```

Alpha = 0

```
ridge2 = Ridge(alpha = 0, normalize = True)
ridge2.fit(X_train, y_train)             # Fit a ridge regression on the training data
pred = ridge2.predict(X_test)            # Use this model to predict the test data
print(pd.Series(ridge2.coef_, index = X_train.columns)) # Print coefficients
print(mean_squared_error(y_test, pred))           # Calculate the test MSE
```

```
lights               1.600402e+02
hum_kitchen          4.234929e+02
hum_living          -1.757733e+02
temp_laundry         2.563175e+02
hum_laundry          5.397860e+01
temp_office         -1.146901e+02
hum_office          -7.953566e+01
temp_bathroom       -4.764717e+01
hum_bathroom         2.861358e+00
temp_building_out    1.166675e+02
hum_building_out     1.701818e+01
hum_ironing         -9.863231e+01
temp_teenRoom        5.066898e+01
temp_parentRoom     -1.041847e+02
hum_parentRoom      -6.367303e+01
Pressure             1.568353e+00
Tdewpoint           -6.048874e+01
Num_sec_midnight    -1.372573e-11
dtype: float64
9112.28211998
```

```
ridgecv.alpha_
```

t[55]: 0.0050000000000000001

[56]:
```
ridge4 = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge4.fit(X_train, y_train)
mean_squared_error(y_test, ridge4.predict(X_test))
```

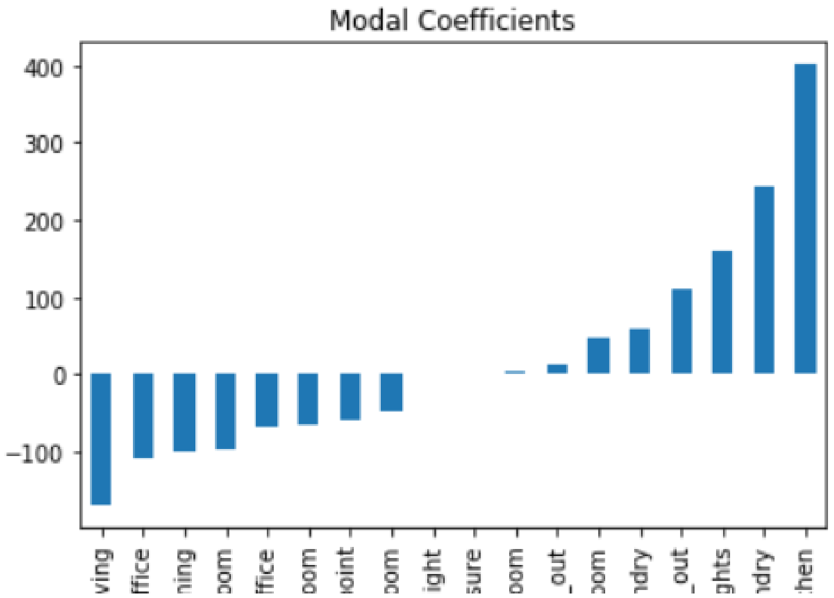
t[56]: 9115.837225415733

[57]:
```
ridge4.fit(X_train, y_train)
pd.Series(ridge4.coef_, index = X_train.columns)
ridge4.score(X_test,y_test)
```

t[57]: 0.14399724271994974

[58]:
```
predictors = X_train.columns
coefs = pd.Series(ridge4.coef_,predictors).sort_values()
coefs.plot(kind='bar', title='Modal Coefficients')
```

t[58]: <matplotlib.axes._subplots.AxesSubplot at 0x21e709a6550>



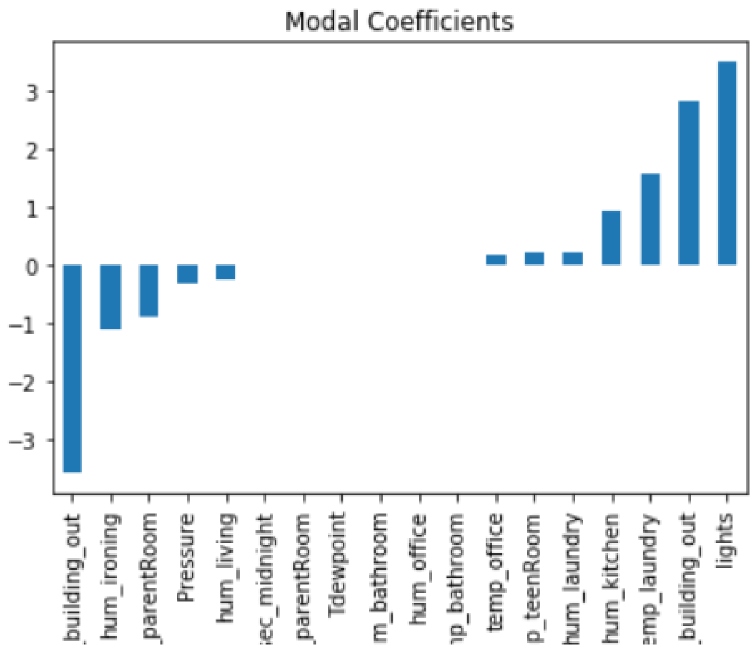
ElasticNet

59]:
```
ENreg = ElasticNet(alpha=1, l1_ratio=0.5, normalize=False)
ENreg.fit(X_train,y_train)
pred_cv = ENreg.predict(X_test)
#ENreg.score(X_test,y_test)
r2_score(y_test,pred_cv)
```

59]:
```
0.004783941134229515
```

50]:
```
predictors = X_train.columns
coefs = pd.Series(ENreg.coef_,predictors).sort_values()
coefs.plot(kind='bar', title='Modal Coefficients')
```

50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21e7098e860>
```



The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Plot -31 – Grid Search for Random forest Regressor

```
print(grid.best_score_)
print(grid.best_params_)
```

```
0.5179801145867352
{'n_estimators': 200}
```

In [102]:
```
print(grid.best_estimator_)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

And the Gradient Boosting Regressor also provide the same equivalent score.

Plot -32 -Grid Search for Gradient Boosting Regressor

```
print(grid.best_score_)
print(grid.best_params_)

0.5023166046215263
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 1500}

In [8]: print(grid.best_estimator_)

GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=5, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=1500, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
```

Since the **Random Forest and Gradient Boosting Regressor have similar outcomes, any one of them can be considered for our final pipeline. Hence we will proceed with Random Forest algorithm with Boruta feature selection.**

Final pipeline

The final pipeline is implemented using sklearn pipeline. The following operations are piped together and automated- Data cleaning, Normalizing, Splitting the Data, Extracting features from Boruta using RandomForest estimator and then finally apply the transformed data to final estimator Random Forest Regressor.

```
In [4]: dataset = pd.DataFrame.from_csv("https://raw.githubusercontent.com/LuisM78/Appliances-energy-prediction-dat

        class Cleaner(BaseEstimator, TransformerMixin):
            """Takes in dataframe, performs cleaning if needed and returns cleaned dataframe"""

            def __init__(self):
                pass

            def seconds(self, x):
                sec = x.hour*3600+x.minute*60+x.second
                return sec

            def day_week(self, z):
                a=[]
                for y in z:
                    if y == 0:
                        a.append('Monday')
                    elif y == 1:
                        a.append('Tuesday')
                    elif y == 2:
                        a.append('Wednesday')
                    elif y == 3:
                        a.append('Thrusday')
                    elif y == 4:
                        a.append('Friday')
                    elif y == 5:
                        a.append('Saturday')
                    elif y == 6:
                        a.append('Sunday')
                return a

            def week(self, x):
                a=[]
                for y in x:
                    if y == 'Saturday' or y == 'Sunday':
                        a.append('weekend')
                    else:
```

```
    def one_hot_encode(self, Data):
        label_encoder = LabelEncoder()
        int_encoded = label_encoder.fit_transform(Data['week_status'])
        int_encoded_day = label_encoder.fit_transform(Data['Day_Status'])
        onehot_encoder = OneHotEncoder(sparse=False)
        int_encoded = int_encoded.reshape(len(int_encoded), 1)
        int_encoded_day = int_encoded_day.reshape(len(int_encoded_day), 1)
        newWeek = onehot_encoder.fit_transform(int_encoded)
        newDay = onehot_encoder.fit_transform(int_encoded_day)
        # new2 = label_encoder.inverse_transform([argmax(new[len(new)-1, :])])
        Data.drop(['week_status', 'Day_Status'], axis=1, inplace=True)
        Data['Friday'] = pd.Series(newDay[:,0], index=Data.index)
        Data['Monday'] = pd.Series(newDay[:,1], index=Data.index)
        Data['Saturday'] = pd.Series(newDay[:,2], index=Data.index)
        Data['Sunday'] = pd.Series(newDay[:,3], index=Data.index)
        Data['Thursday'] = pd.Series(newDay[:,4], index=Data.index)
        Data['Tuesday'] = pd.Series(newDay[:,5], index=Data.index)
        Data['Wednesday'] = pd.Series(newDay[:,6], index=Data.index)
        Data['WeekDay'] = pd.Series(newWeek[:,0], index=Data.index)
        Data['Weekend'] = pd.Series(newWeek[:,1], index=Data.index)
        return Data

    def transform(self, df, y=None):
        """Adding the columns Day_Status, week_status and Num_sec_midnight"""

        df['Num_sec_midnight']=self.seconds(df.index)
        z = df.index.dayofweek
        df['Day_Status'] = z
        df['Day_Status'] = self.day_week(df.Day_Status)
        df['week_status'] = self.week(df.Day_Status)

        """Performing one hot encoding on week_status and day_status columns"""
```

```python
    def fit(self, df, y=None):
        """Returns `self` unless something different happens in train and test"""
        return self


class Normalizer(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def transform(self, df, y=None):
        """Performs Normalization on all the columns except for Appliances"""
        for j in range(1, len(df.columns)-1,1):
            df.iloc[:,[j]] = (df.iloc[:,[j]] - df.iloc[:,[j]].mean())/df.iloc[:,[j]].std()
        df.to_csv("normalized.csv")
        return df

    def fit(self, df, y=None):
        """Returns `self` unless something different happens in train and test"""
        return self

class SplitData(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def transform(self, df, y=None):
        y = df['Appliances']
        df4 = df.iloc[:,1:]
        X_train, X_test, y_train, y_test = train_test_split(df4, y, test_size=0.25)
        train = X_train.join(y_train)
        test = X_test.join(y_test)
        train.to_csv("train.csv")
        test.to_csv("test.csv")
        return X_train, X_test, y_train, y_test

    def fit(self, df, y=None):
```

```python
class transformData(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def transform(self, df, y=None):
        return df

    def fit(self, df, y=None):
        return self

pipeline = Pipeline([("cleaner", Cleaner()),
                     ("normalizer", Normalizer()),
                     ("train_test_split", SplitData()),
                     ("features", BorutaPy(RandomForestRegressor())),
                     ("transofrm_data", transformData()),
                     ("estimator", RandomForestRegressor())
                    ])
pipeline
```

```
]: Pipeline(memory=None,
       steps=[('cleaner', Cleaner()), ('normalizer', Normalizer()), ('train_test_split', SplitData()), ('features', BorutaPy(alp
   a=0.05,
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
              max_features='auto', max_leaf_nodes=None,
              min_impurity_decreas...timators=10, n_jobs=1,
              oob_score=False, random_state=None, verbose=0, warm_start=False))])
```

```
In [5]: pipeline.steps

Out[5]: [('cleaner', Cleaner()),
         ('normalizer', Normalizer()),
         ('train_test_split', SplitData()),
         ('features', BorutaPy(alpha=0.05,
               estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0, warm_start=False),
               max_iter=100, n_estimators=1000, perc=100, random_state=None,
               two_step=True, verbose=0)),
         ('transofrm_data', transformData()),
         ('estimator',
          RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0, warm_start=False))]
```

```
In [ ]: param_grid = [{'Boruta_rf__n_estimators': [100,200,300]}]
        grid = GridSearchCV(pipeline, cv = 10, param_grid=param_grid, n_jobs=2,verbose=2)
        grid.fit(X_train, y_train)
        pred = grid.predict(X_test)

        Fitting 10 folds for each of 3 candidates, totalling 30 fits
```

Conclusion

The report concludes by choosing Random Forest algorithm for energy usage prediction. The pipelining was based on Random Forest with Boruta feature selection. RF would not be the only best algorithm for the energy dataset since Gradient Boost machines also provide equivalent scores. May be with larger dataset, the analysis might give accurate algorithm suitable for the data. Data should be of atleast all the months so that you can predict the outliers properly and provide proper prediction of Energy consumption.