# Case 2 Report

# Machine learning with Energy datasets

INFO 7390
Advanced Data Science & Architecture

Professor:
Srikanth Krishnamurthy

Students:
Team 2
Chenlian Xu
Qianli Ma

Date:
Mar 16th , 2018

# Part 1: Research

## Prediction of appliances energy use in smart homes

This paper introduces methods for prediction of energy consumption of different devices in homes. The raw data comes from a set of homes in France. The performance of the predictors is studied and two processing are proposed to improve the performance of the predictors.

Firstly, the paper discusses the need for energy prediction in housing. The basic concepts of Smart Grid are introduced in this part which is a very important topic in energy prediction field. Some traditional methods of energy load management are also introduced here. In order to obtain good prices for the traded energy, energy supplier need to predict the next day consumption more precisely. Since households and services, etc. take 37.1% of energy consumption by sector which is the biggest part, predicting the energy consumption in the housing sector is essential for the energy prediction though it is a hard one. The smart home plays an important role in smart grid which has close interactive between Power retailer about energy demands. To compute the energy plan of the house and the energy forecast more effectively, energy consumption in smart homes need to be optimized.

In the next part, energy prediction methodology for appliances in homes are presented. An assessment of predictors is set up at the first stage. The precision of the predictor is expressed by a function of hour. More weights are given to the prediction results which are close to the current hour. The prediction has different predictors such as the “will always consume” which means the appliance will consume energy permanently; the “will never consume” means the service will not consume at all in the next day; the “ARMA”(Auto Regressive Moving Average), where the current value of a time variable is assumed to be a function of its past values; and the proposed predictor which model the energy consumption as a stochastic process. The result shows that “ARMA” is not proper for predicting energy consumption in case of a singular appliance in a home; and the proposed predictor performs better than the basic predictors.

To improve the prediction precision, two processing are applied to the model ---- Segmentation and aggregation. Segmentation means choosing a specific segment of data such as the data in the same season, month or period of the day, etc. The objective of this operation is to reduce the average dispersion in order to improve the prediction. Here the paper uses a temporal segmentation that considers each day of the week as a partition. The proposed predictor is applied on each hour of the day. Aggregation means then merging similar partitions together. K-means Clustering algorithm is used to group the similar consumption days in order to make the segments more meaningful. K-means finally gives us two clusters which are weekdays data and Saturday & Sunday data. Then the initial data is divided into 2 sets according to the clusters and segments. After the applying

the proposed predictor on clustered and segmented data, the precision of predictor increases.

In the last part, the paper introduces global study of the services in the house. Here the global means predicting the energy consumption of different electrical devices in the house. The prediction precision for the refrigerator and freezer shows that a short period of time is significant for prediction if the appliances consume energy strongly depend on season. When predicting the consumption of TV or non-halogen lighting, the proposed predictor is better than the basic predictors all the time.

In conclusion, forecasting the energy consumption in homes is an important aspect in the power management of the grid. Make the prediction for energy consumption of house appliances means a lot to the grid. Since the energy consumptions in housing varies dramatically due to the inhabitant's behavior, stochastic method has been chosen in this paper and be proved as a better one compared with other basic predictors. Segmentation and aggregation are applied to improve the performance of the proposed predictor using K-means clustering algorithm. Further work focuses on the equipment level prediction for all appliances in a house which is more powerful than a house level prediction.

# Data driven prediction models of energy use of appliances in a low-energy house

This paper discusses data-driven predictive models for the energy use of appliances. The purpose of this work is to understand the relationships between appliances energy consumption and different predictors. Data filtering is used for removing non-predictive parameters and feature ranking in this paper. Multiple linear regression, SVM with radial kernel, random forest and GBM are trained with repeated cross validation and evaluated in a testing set.

In the beginning, the paper shows us the importance of the appliances energy use in buildings, such as determining adequate sizing of photovoltaics and energy storage, detecting abnormal energy use patterns and so on. Literature review part introduces some existing researches on this topic about numerical model and electricity load prediction. The paper sums up the following points for the paper review section: (1) Appliances level energy consumption is significant to residential sector and grid; (2) It is important to identify which appliance is more deciding to the energy consumption; (3) The energy consumptions of appliances may be broken down into different contributions; (4) The patterns for energy use of appliances can vary significantly; (5) Weather parameters have been proven relevant to predict the electricity energy consumption in buildings; (6) The thermal influence of appliances on internal gains is important in building energy performance for highly insulated buildings.

The second section is about the house, where the data comes from. Basic information and energy usage of the house is presented in this part. The electric energy metering at the passive house was done with M-BUS energy counters. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network built with XBee radios.

The next part is about data recording and description. The energy data logged every 10 min. Lights, temperature and humidity recordings are all be averaged and recorded every 10 min periods and merged with the energy data set by date and time. The data from the nearest airport weather station is merged by date and time in this study since there is no weather station outside the house. There are totally 32 variables (31 features by dropping data time stamp).

The paper uses psych package to generate pairs plot. These plots illustrate the relationships between variables. Some significant relationships could be obtained such as positive correlation between the energy consumption of appliances and lights, temperature in living room area, outdoor temperature, NSM (the number of seconds from midnight for each day) and so on. Negative correlation between appliances and outdoor humidity, humidity outside weather station, pressure and so on. Boruta package is used for comparing importance of attributes with importance of shadow attributes that are created

by shuffling original ones. The Boruta algorithm ranks the variables in order of importance starting with the NSM variable to the Week-Status. The Classification and Regression Training package (CARET) has a RFE algorithm and is used in this study to select the optimal inputs with the r package dummies. After transforming Week status and day of the week to dummy variables, there are a total of 35 predictors used for prediction. After random forest with 10-fold cross validation, RFE algorithm shows that the optimal number of predictors is 34.

4 regression models (lm, SVM-radial, random forest and GBM) are trained with 10 fold cross validation to select the best. The doParallel package was used for parallel computation to speed up the program here. RMSE, R-squared, MAE and MAPE are used here to compare the performance of each of the regression model. Linear Model is proved to be improper for this case. The result shows that GBM is the best model according to RMSE and R-squared. RF and GBM models have very similar performance based on their RMSE and R-squared values. The result shows that GBM model without the light predictor is the most accurate. When looking at the ranking for the GBM model with no light information, we can come up with a conclusion that the information from the kitchen, living room, laundry room, bathroom, outdoors, office and bedrooms are the most important. A particular limitation of this study is that the analysis was done for only one house. Also, the sensitivity and accuracy of the sensor could also affect the result.

In conclusion, the GBM and RF models are proved to be better than SVM-radial and multiple LM models. The weather data from the nearby weather station and the humidity and temperature data from a wireless sensor can increase the prediction accuracy in the GBM models. Light consumption is ranked highly when using all the predictors, while appeared not to have a significant impact when studying different predictor subsets. Future work on considering more weather data, occupancy and occupant's activity information could be useful to improve the prediction accuracy.

# A review of artificial intelligence based building energy use prediction: Contrasting the capabilities of single and ensemble prediction models

This paper conducts a comprehensive review of articles of single AI-based methods such as multiple linear regression, artificial neural networks, and support vector regression, compared with ensemble prediction method which is combination of multiple single AI-based prediction models.

AI-based prediction method predicts building energy use according to its correlated variables such as environmental conditions, building characteristics, and occupancy status. The following literature review part explains the current research trends of AI-based building energy use and some of the frequently used features in these studies. The review shows that single prediction method is widely used for AI-based building energy use prediction, while ensemble prediction methods are very limited. This paper shows us some conclusions based on the related researches:
(1) Learning models: the mainly used learning models could be classified into four categories: regression, ANN, SVR, and all others.
(2) Building type: educational and research building takes a percentage of 42%, while 33% are commercial buildings.
(3) Energy type: the predicted energy may be classified into five categories, which are whole building energy/electricity, heating & cooling energy, heating energy, cooling energy, and all others.
(4) Prediction time scale: most of the researches choose hour as the time scale. There are also some scales like day, year, minute, 15 min, week and month.
(5) Input data type: 60% of the articles used meteorological data; Only 29% utilized occupancy information to develop their prediction models.

Then the paper introduces AI-based prediction models. Data collection, data preprocessing, model training and model testing are the four main steps of AI-based prediction method. In this part, some commonly used single prediction methods like multiple LM, artificial neural network, SVM and so on. Then the ensemble prediction methods which are implemented by the following steps: (1) Input feature identification; (2) Data monitoring and preprocessing; (3) Learning algorithm selection; (4) Base model generation; (5) Model integration. Generally, the ensemble method is consisting of many single learning methods with different weights. The weight of each base model is assigned based on its prediction accuracy, which means that the one with the least prediction errors may have the highest weight. The generalization ability of an ensemble model is stronger than that of a single model.

There are many cases where ensemble prediction methods are applied, which are discussed in the next part in the paper. Even though ensemble learning methods have been successfully applied in the area of face recognition, medical diagnosis, and gene

expression analysis, their use in the area of building energy use prediction did not commence until 2014.

Further discussion shows that reliability, ease of implementation and fast computation speed is the main advantages of single prediction method, while limited prediction accuracy and reliability is the disadvantages especially when compared to the ensemble prediction method. The particular advantage of ensemble prediction method is that it remarkably improved prediction accuracy and stability. However, ensemble prediction method requires more computation time and high level of knowledge. Also, ensemble prediction method is the fact that its prediction performance highly depends on the selection of base models. Advantages and disadvantages of AI-based prediction methods are also discussed in this part.

In conclusion, this is a comprehensive review of AI-based methods for building energy use prediction with a special focus on ensemble prediction methods. The type of studied buildings, the methods used for prediction, the type of predicted energies, the time scale of the prediction and the type of input data used for prediction is the 5 main aspects that current research works on. Two main categories of AI-based prediction methods are the single prediction methods and the ensemble prediction methods. A comparison of the single and ensemble prediction methods is conducted in the paper. A discussion of advantages and disadvantages of AI-based prediction models is provided. Finally, future works of the research on AI-based building energy use prediction are also covered in this paper.

# Part 2: Exploratory Data Analysis

## 2.1 Aggregated energy consumption

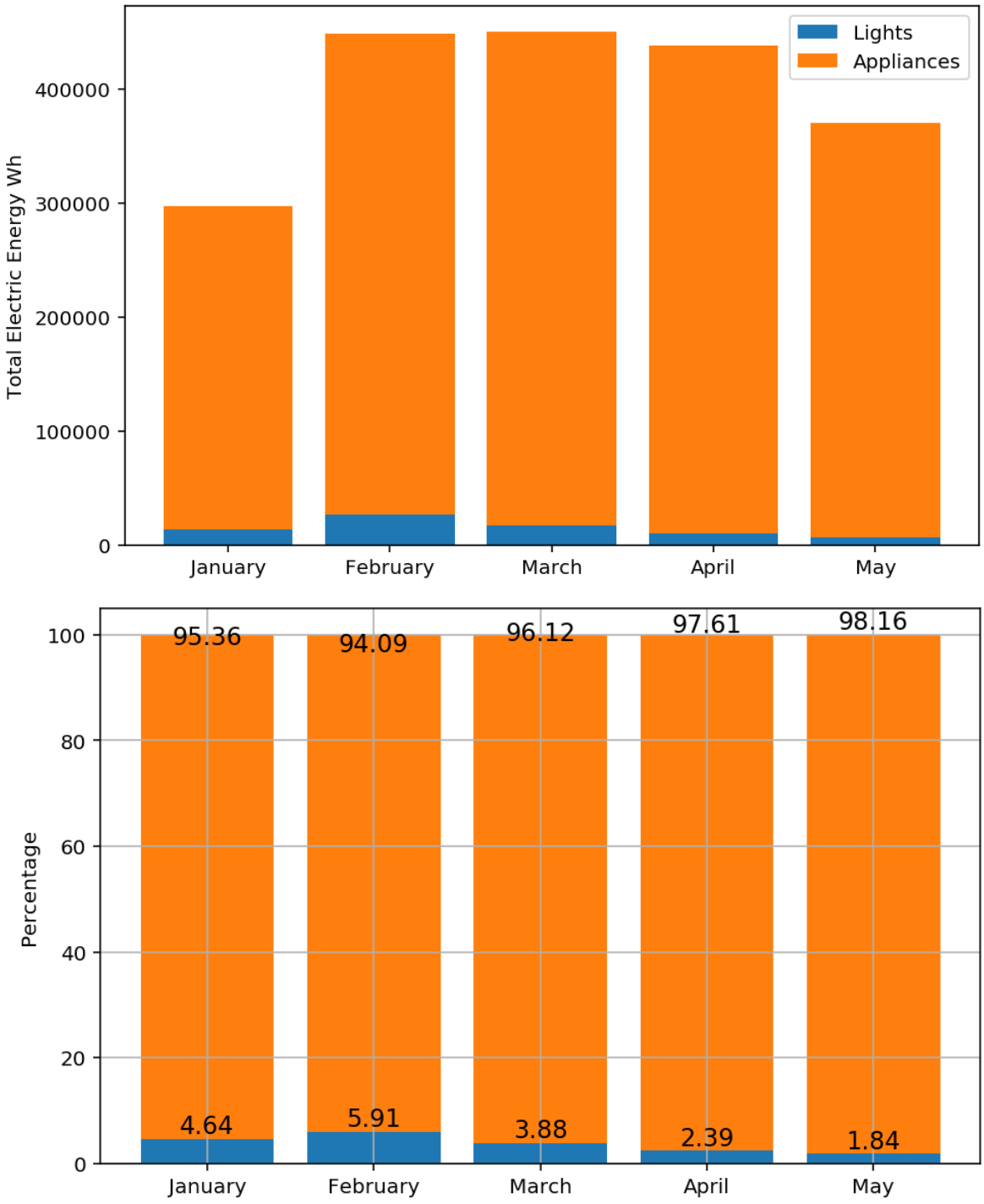



Figure 1. Aggregated energy consumption of appliances and lights per month. This graph includes all the energy consumed from January 1st until May 22nd 2016. Note that the energy for the month of May is not complete.

Figure 1 shows the aggregated energy consumption of appliances and lights from per month and the percentage of each contribution. Total energy consumption of the two contributions in Jan are much lower than other months. We can see that Lights energy consumption decreases from 5.91% to 1.84% in 4 months from Feb to May. It might because of the increasing of daylight time. Lights always take a small percentage of the total consumption since most of the lights in the house are LEDs which save energy. Appliances' consumption range from 94.09% to 98.16% which is far more energy consuming than the lights but also too general for research, which means data of appliance level energy consumption is more useful and meaningful at this point.

## 2.2 Appliances Energy Consumption On Timeline

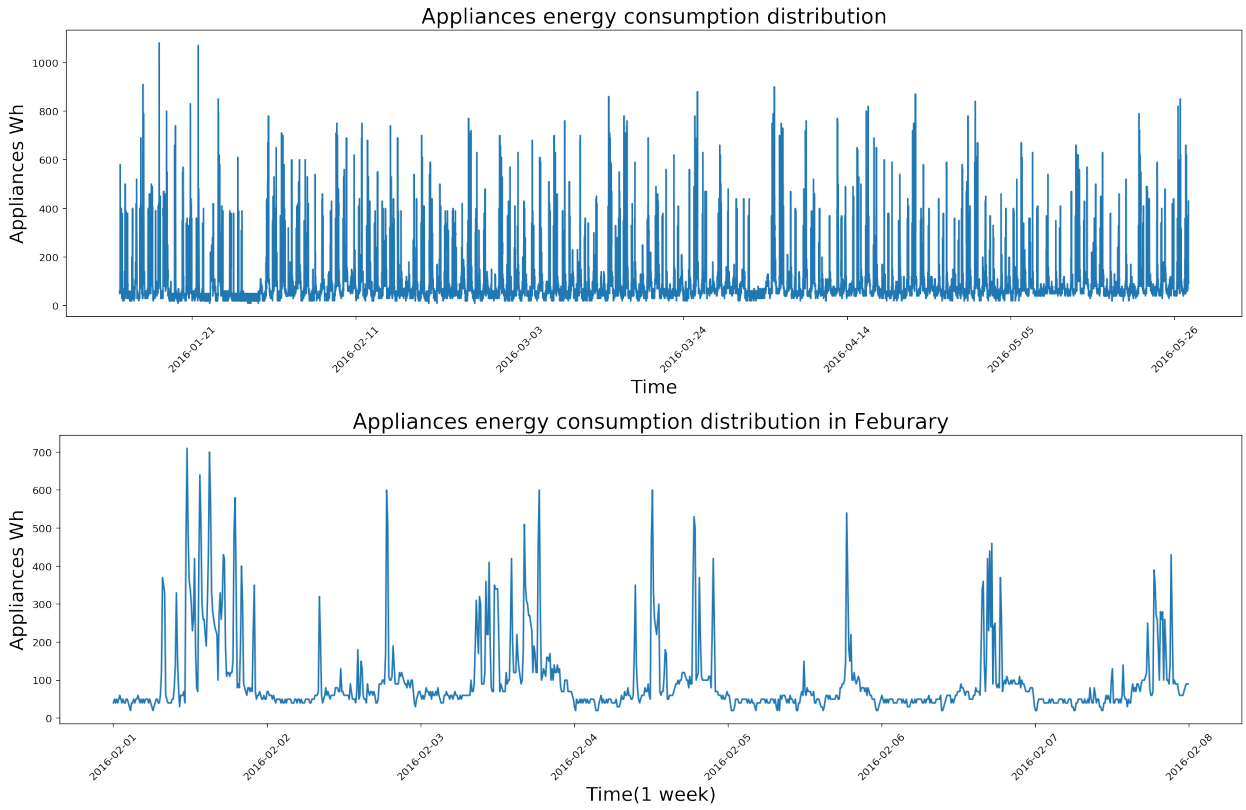



*Figure 2. Appliances energy consumption measurement for the whole period and a specific week of data.*

Figure 2 shows the energy consumption for the period and the 1st week in Feb of the recorded data. As you can see there is a high variability among the consumptions even having a closer look at a single week. The highest peak appears in January which might due to the heater and hot water. There are two main peaks in the second graph which might not be that obvious. We can see the basic pattern over there. It is in the morning

and evening when the energy consumption is comparative high. It is quite easy to understand that people might use more appliances after they waking up in the morning or getting home after work.

## 2.3 Appliances Energy Consumption Distribution

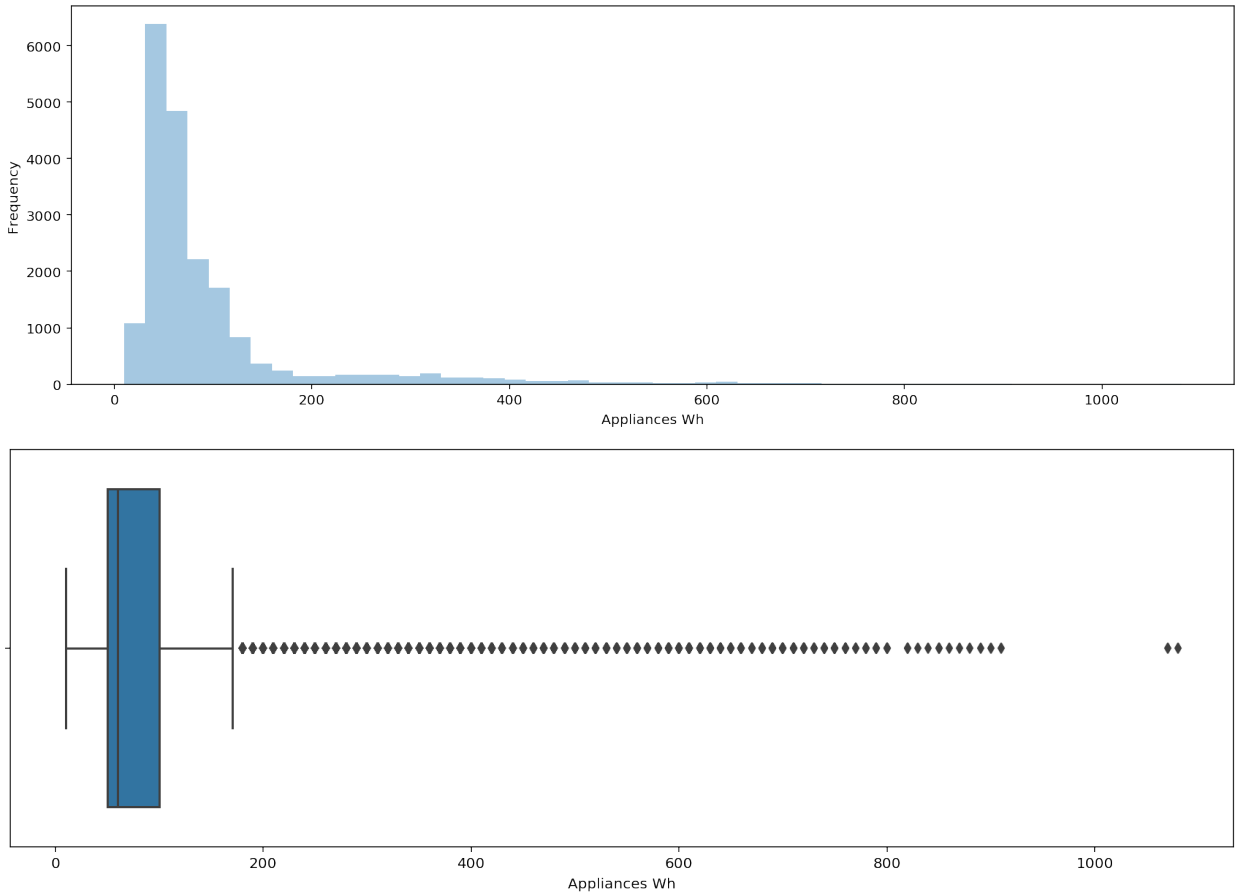



*Figure 3. Appliances energy consumption distribution. Top: histogram, bottom: boxplot.*

Figure 3 are histogram and boxplot of appliances energy consumption distribution. The histogram shows the frequency of energy consumption in a certain interval. As we can see most of the data located between 20 and 100. The long tail of the histogram means there are some large values with very low frequencies. The boxplot reveals the middle value, which is 60 Wh, with the black line inside the blue box. The lower whisker of the boxplot is 10 Wh while the upper one is 170 Wh. Both of the graphs show that the data above the median value is more dispersed. There are many outliers appears in the boxplot which are the black rhombus over the upper whisker.

## 2.4 Lights Energy Consumption On Timeline

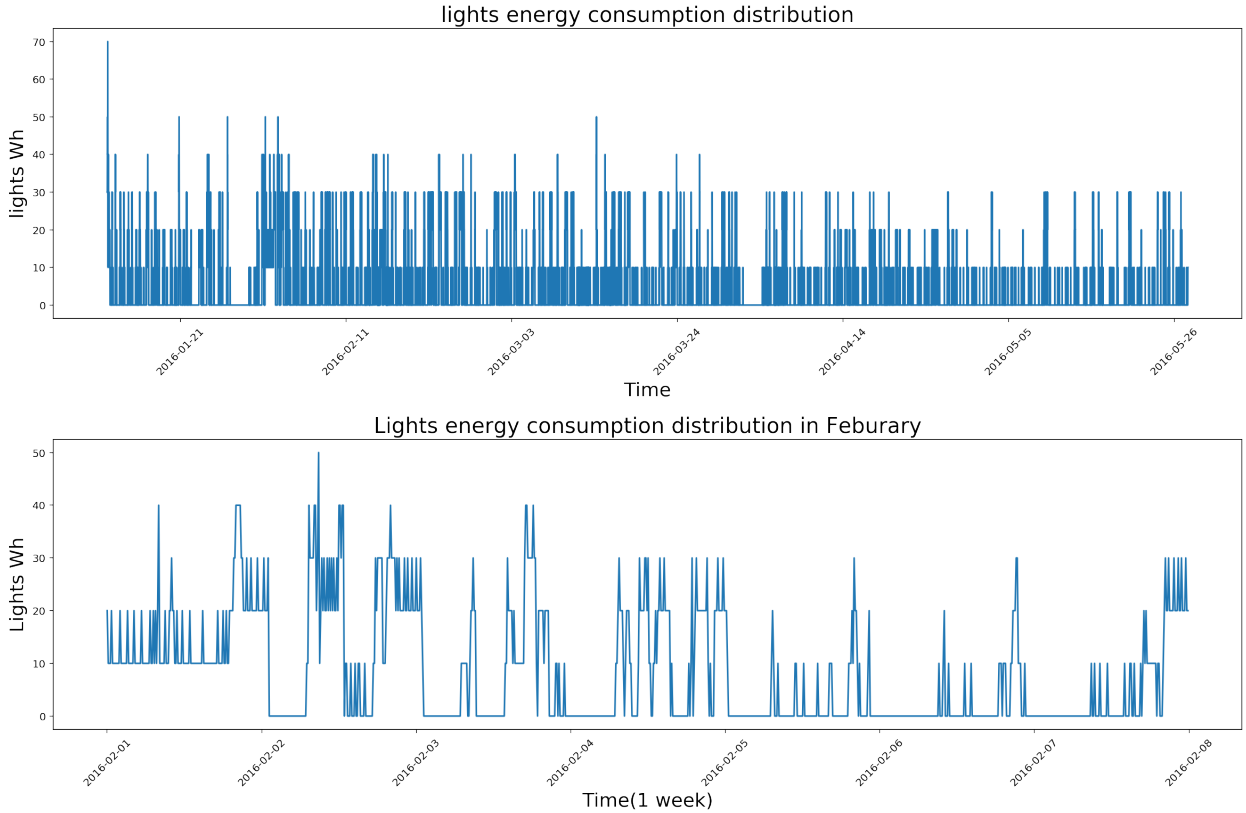



*Figure 4. Lights energy consumption measurement for the whole period and a specific week of data.*

Here some insights are also given to the light consumption. As figure 4 shows, light energy consumptions are more stable compared with appliances energy consumptions. There are only a few points which go above 30 Wh and a lot of data have a value of 0. In the second graph which looks into a certain week of data, a lot of straight line in the bottom represents the midnight of the day when occupants are sleeping and would not use light. The trend of two peaks in the morning and night of the second graph are more obvious than the one of appliance's. This might because that lights usage is more depend on the time since we would not leave the lights on when we are out of home or sleeping unless we forget to turn them off.

All the figures above are drawn by the matplotlib.pylot package.

## 2.5 Pair Plots

The figure 5 and Appendix B-D are pair plots which drawn by seaborn together with matplotlib.pylot packages. These plots could provide a better insight of the relationships between all the variables with the appliances' energy consumption in the whole data set. These Figures show the bivariate scatter plots below the diagonal, histogram plots along the diagonal. Pearson correlation also showed above the diagonal, which is a measure of

the linear dependence between two variables. A correlation of 1 is total positive correlation, –1 is total negative correlation and 0 represents no correlation. The blue solid line in each plots show the linear regression fits for each pair.

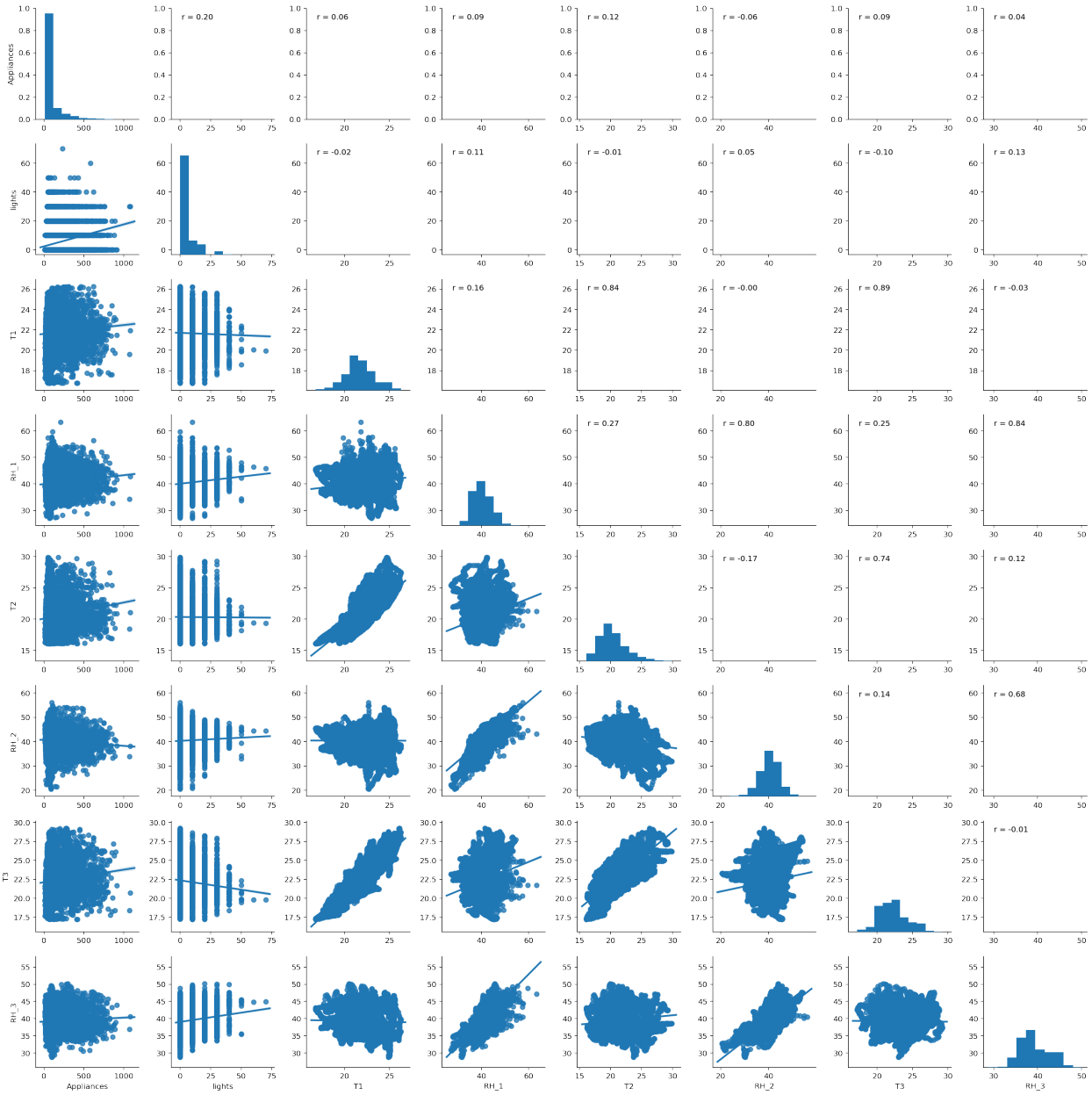



*Figure 5. Pairs plot. Relationship between the energy consumption of appliances with: lights, T1, RH1, T2, RH2, T3, RH3.*

Table 1 shows the rank of Pearson Correlation between energy consumption of appliances and all the variables from all the pair plots. Number of seconds from midnight (NSM), light energy consumption, Temperature outside the building and Temperature outside the

weather station present stronger positive correlations with appliance energy consumption compared with other variables. Their Pearson Correlation value are 0.22, 0.20, 0.12, 0.12 and 0.10. Humidity in kitchen area, temperature in the laundry room area and wind speed (from weather station) all have a Pearson Correlation value of 0.09, which also could be considered as significant positive correlation. There is only one variable has a negative Pearson Correlation more negative than -0.10, which is Humidity outside the weather station with -0.15. Humidity in teenager room2 and humidity outside the building also show significant negative correlation as their Pearson Correlation values are -0.09 and -0.08. Other variables are considered as not having significant correlations with appliance energy consumption since their Pearson Correlation are closer to 0 compared to the mentioned ones. Also, it seems that visibility has no correlation because its Pearson Correlation is 0.

| ***Variables*** | ***Pearson Correlation*** |
|---|---|
| *NSM* | 0.22 |
| *lights* | 0.20 |
| *T2* | 0.12 |
| *T6* | 0.12 |
| *T out* | 0.10 |
| *RH1* | 0.09 |
| *T3* | 0.09 |
| *Windspeed* | 0.09 |
| *T1* | 0.06 |
| *TH3* | 0.04 |
| *T4* | 0.04 |
| *T8* | 0.04 |
| *T7* | 0.03 |
| *RH4* | 0.02 |
| *T5* | 0.02 |
| *Tdewpoint* | 0.02 |
| *RH5* | 0.01 |
| *T9* | 0.01 |
| *Visibility* | 0.00 |
| *Pressure* | -0.03 |
| *RH9* | -0.05 |
| *TH2* | -0.06 |
| *RH7* | -0.06 |
| *RH6* | -0.08 |
| *RH8* | -0.09 |
| *RH out* | -0.15 |

*Table 1. Rank of Pearson Correlation between energy consumption of appliances and all the variables*

## 2.6 Heat Map of Hourly Appliances Energy Consumption

Figure 6 is the heat map of hourly appliance energy consumption. These figures are built with matplotlib.pyplot and seaborn. The energy consumption has a strong pattern in time. The heat rise significantly in the morning around 7. The energy consumption in the noon keeps a comparatively high level. There is hottest spot can be clear seen around 18 to 19, which means the energy demand of this period in a day is very high. The heat in the Wednesday is lower than other days in the week as can be revealed by the heat map except the first one. However, the pattern regarding the day of the week is still not very significant.

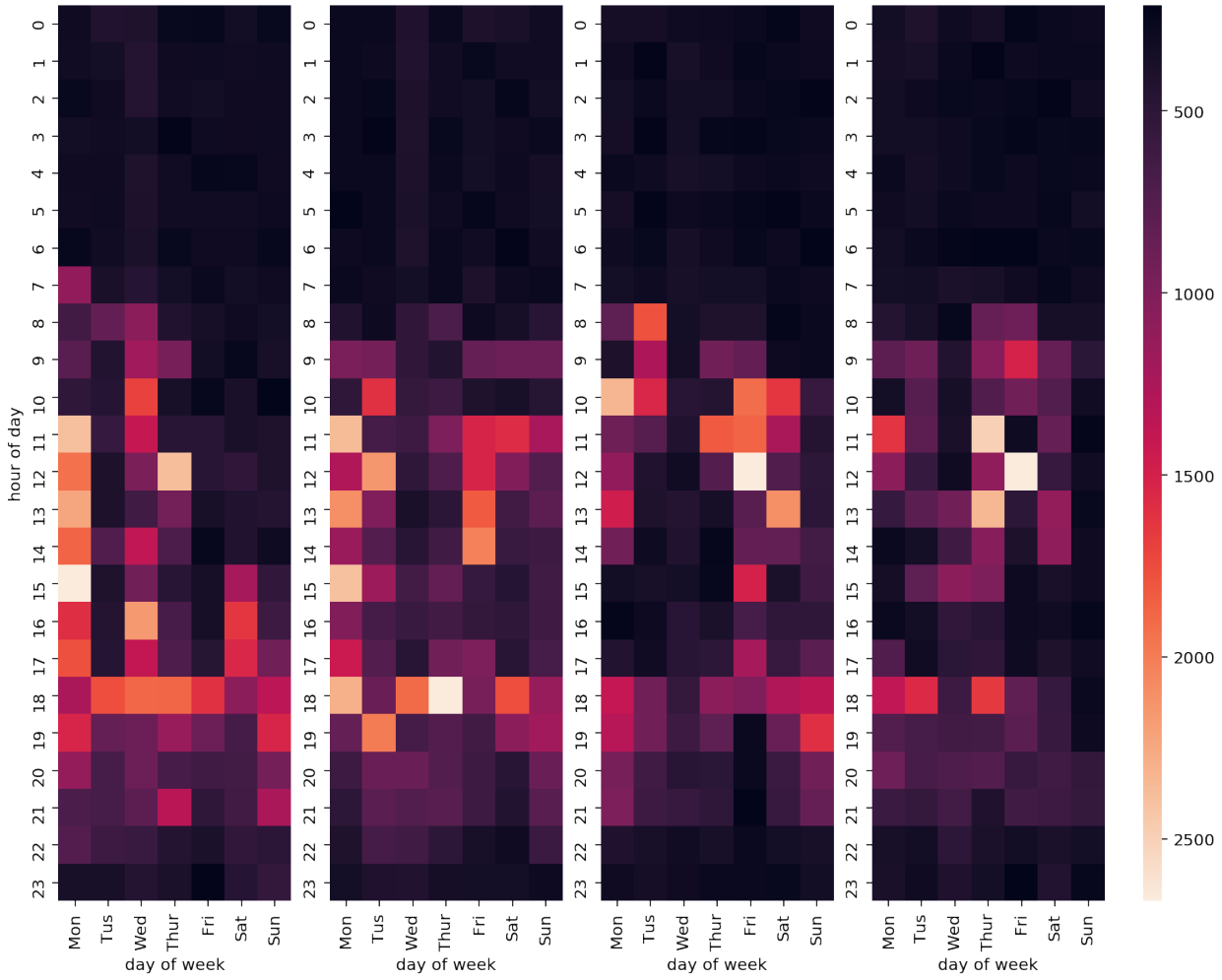



*Figure 6. Hourly energy consumption of appliances heat map for four consecutive weeks.*

# Part 3: Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is an informal topic, but it is considered essential in applied machine learning.

In this part, feature engineering is applied to the dataset we described in the last part. Since the dataset has already been cleaned (no missing value) and there are no text features, no image features in the dataset, the main task of this part is to generate the dummy variables of categorical features.

```
df['NSM'] = pd.to_datetime(df.date)-pd.to_datetime(df.date.str.split('\s+').str[0]+" 00:00:00")


def time_to_second(time):
    return time.total_seconds()

df['NSM'] = df['NSM'].apply(time_to_second)
df.head()
```

### 3.1 Add NSM

Firstly, Number of seconds from midnight (NSM) is added to the dataset by convert the date time to seconds from 12am. Actually, it has already been done in the previous part and the ranking of Pearson Correlation shows that NSM has a significant correlation with appliances energy consumption.

```
df['Dayoftheweek'] = pd.to_datetime(df.date).dt.weekday_name.astype('category')
df.head()
```

### 3.2 Add Day of the Week

Day of the week is added to the data framework using pandas ‘dt.weekday_name’. This function will return the name of the weekday according to the date time it gets from ‘to_datetime(df.date)’. The result is shown as follow:

| T4 | ... | T_out | Press_mm_hg | RH_out | Windspeed | Visibility | Tdewpoint | rv1 | rv2 | NSM | Dayoftheweek |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 19.000000 | ... | 6.600000 | 733.5 | 92.0 | 7.000000 | 63.000000 | 5.3 | 13.275433 | 13.275433 | 61200.0 | Monday |
| 19.000000 | ... | 6.483333 | 733.6 | 92.0 | 6.666667 | 59.166667 | 5.2 | 18.606195 | 18.606195 | 61800.0 | Monday |
| 18.926667 | ... | 6.366667 | 733.7 | 92.0 | 6.333333 | 55.333333 | 5.1 | 28.642668 | 28.642668 | 62400.0 | Monday |
| 18.890000 | ... | 6.250000 | 733.8 | 92.0 | 6.000000 | 51.500000 | 5.0 | 45.410389 | 45.410389 | 63000.0 | Monday |
| 18.890000 | ... | 6.133333 | 733.9 | 92.0 | 5.666667 | 47.666667 | 4.9 | 10.084097 | 10.084097 | 63600.0 | Monday |

### 3.3 Add Weekday Status

Then a function is defined to divide the data into two categories: Weekend and Weekday.

```
def weekday_status(day):
    if day == 'Saturday' or day == 'Sunday':
        return 'Weekend'
    else:
        return 'Weekday'

df['Weekdaystatus'] = df['Dayoftheweek'].apply(weekday_status).astype('category')
df.head()
```

| T4 | ... | Press_mm_hg | RH_out | Windspeed | Visibility | Tdewpoint | rv1 | rv2 | NSM | Dayoftheweek | Weekdaystatus |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | ... | 733.5 | 92.0 | 7.000000 | 63.000000 | 5.3 | 13.275433 | 13.275433 | 61200.0 | Monday | Weekday |
| 000 | ... | 733.6 | 92.0 | 6.666667 | 59.166667 | 5.2 | 18.606195 | 18.606195 | 61800.0 | Monday | Weekday |
| 667 | ... | 733.7 | 92.0 | 6.333333 | 55.333333 | 5.1 | 28.642668 | 28.642668 | 62400.0 | Monday | Weekday |
| 000 | ... | 733.8 | 92.0 | 6.000000 | 51.500000 | 5.0 | 45.410389 | 45.410389 | 63000.0 | Monday | Weekday |
| 000 | ... | 733.9 | 92.0 | 5.666667 | 47.666667 | 4.9 | 10.084097 | 10.084097 | 63600.0 | Monday | Weekday |

### 3.4 Transform Categorical Features into Dummy Features

The very first idea is simply assign the value of day of the week to integer from 1 to 7; and assign the value of weekday status to binary value like 0 and 1. However, the regression models inside the python packages make the fundamental assumption that numerical features reflect algebraic quantities. For example, that Monday < Tuesday <

Wednesday<…, or even that Sunday - Saturday = Monday, which does not make much sense.

Thus, one-hot encoding should be applied here, to generate dummy columns indicating the presence or absence of a category with a value of 1 or 0, respectively.
Then the 'Dayoftheweek' and 'Weekdaystatus' columns are transformed to dummy vavirables using 'pandas.get_dummies':

```
df = pd.get_dummies(df, columns=['Dayoftheweek', 'Weekdaystatus'])
df.head()
```

| NSM | Dayoftheweek_Friday | Dayoftheweek_Monday | Dayoftheweek_Saturday | Dayoftheweek_Sunday | Dayoftheweek_Thursday | D |
|---|---|---|---|---|---|---|
| 61200.0 | 0 | 1 | 0 | 0 | 0 | |
| 61800.0 | 0 | 1 | 0 | 0 | 0 | |
| 62400.0 | 0 | 1 | 0 | 0 | 0 | |
| 63000.0 | 0 | 1 | 0 | 0 | 0 | |
| 63600.0 | 0 | 1 | 0 | 0 | 0 | |

After doing this, we get a total of 39 columns which including 7 dummy variables from day of the week and 2 dummy variables from weekday status.

### 3.5 Create Feature Engineering Class

Finally, a class is generated for transferring the raw date set to the new data set with 39 columns. This class makes the feature engineering of this data set become reusable: Simply create an instance of this class, pass the raw data framework then call the 'feature_engineering' function.

```
class DfAfterFE:
    def __init__(self, df):
        self.df = df
        self.date = self.df.date

    @staticmethod
    def time_to_second(time):
        return time.total_seconds()

    @staticmethod
    def weekday_status(day):
        if day == 'Saturday' or day == 'Sunday':
            return 'Weekend'
        else:
            return 'Weekday'

    def feature_engineering(self):
        self.df['NSM'] = pd.to_datetime(self.date) - pd.to_datetime(
            self.date.str.split('\s+').str[0] + " 00:00:00")
        self.df['NSM'] = self.df['NSM'].apply(self.time_to_second)
        self.df['Dayoftheweek'] = pd.to_datetime(self.date).dt.weekday_name.astype('category')
        self.df['Weekdaystatus'] = self.df['Dayoftheweek'].apply(self.weekday_status).astype('category')
        self.df = pd.get_dummies(self.df, columns=['Dayoftheweek', 'Weekdaystatus'])
```

# Part 4: Prediction algorithms

### 4.1 Prepare Data

In this part, three different prediction models are applied to the train data. Before fitting models, the feature engineering class created in the last part is used here to transfer the dataset to a usable one with 39 variables.

```
df = pd.read_csv('energydata_complete.csv')
df1 = DF_AFTER_FE(df)
df1.feature_engineering()
new_df = df1.df
```

Then date, appliance energy consumption, random variable 1 and random variable 2 are removed from the data set.

```
X = new_df.drop(columns=['date', 'Appliances', 'rv1', 'rv2'])
y = new_df['Appliances']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

### 4.2 Create Error Metrics Class

Error Metrics class is created in advance to compute the different error metrics of each model. RMS, MAPE, $R^2$ and MAE are included in the function to measure the prediction performance of each model.

Linear regression, random forest and neural network are used here to fit the data set. For random forest, number of estimators are randomly picked as 100.

```
class ErrorMetrics:

    def __init__(self, X_train, y_train, X_test, y_test):
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.error_metric = pd.DataFrame({'rmse_train': [],
                                          'rmse_test': [],
                                          'mae_train': [],
                                          'mae_test': [],
                                          'mape_train': [],
                                          'mape_test': [],
                                          'r_train': [],
                                          'r_test': []})
```

```
def cal_metric(self, modelname, model):
    y_train_pre = model.predict(self.X_train)
    y_test_pre = model.predict(self.X_test)

    rmse_train = math.sqrt(mean_squared_error(self.y_train, y_train_pre))
    rmse_test = math.sqrt(mean_squared_error(self.y_test, y_test_pre))

    mae_train = mean_absolute_error(self.y_train, y_train_pre)
    mae_test = mean_absolute_error(self.y_test, y_test_pre)

    mape_train = np.mean(np.abs((self.y_train - y_train_pre) / self.y_train)) * 100
    mape_test = np.mean(np.abs((self.y_test - y_test_pre) / self.y_test)) * 100

    r_train = r2_score(self.y_train, y_train_pre)
    r_test = r2_score(self.y_test, y_test_pre)

    error_metric_local = pd.DataFrame({'Model': [modelname],
                                       'rmse_train': [rmse_train],
                                       'rmse_test': [rmse_test],
                                       'mae_train': [mae_train],
                                       'mae_test': [mae_test],
                                       'mape_train': [mape_train],
                                       'mape_test': [mape_test],
                                       'r_train': [r_train],
                                       'r_test': [r_test]})

    if self.error_metric.columns[0] == 'Model' and not self.error_metric.loc[self.error_metric['Model'] == modelname].empty:
        self.error_metric = self.error_metric[self.error_metric.Model != modelname]
    else:
        pass

    self.error_metric = pd.concat([self.error_metric, error_metric_local])
```

## 4.3 Fit Models

```
#linear regression
lm = LinearRegression()
lm.fit(X_train, y_train)
cal_metric('Linear Regression', lm, X_train, y_train, X_test, y_test)
print('Regression completed')
```

```
#Random Forest
rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)
cal_metric('Random Forest', rf, X_train, y_train, X_test, y_test)
print('RandomForest completed')
```

```
# Neural network
nn = MLPClassifier()
nn.fit(X_train, y_train)
error_metrics.cal_metric('Neural Network', nn)
print('Neural Network completed')
```

## 4.3 Computation Results

The computation results are shown as below. It can be easily seen that random forest regression has the best performance of prediction for this data set. All the three errors of random forest are lower than the other two models, and the R2 of random forest is

significantly higher than the other two's. Linear regression and neural network have similar performance according to MAE, MAPE and RMSE. Thus, random forest is recommended at this moment. However, further model validation and discussion are needed to determine which model of the three is the best fit for this data set.

| Model | mae_test | mae_train | mape_test | mape_train | r_test | r_train | rmse_test | rmse_train |
|---|---|---|---|---|---|---|---|---|
| Linear Regression | 54.145228 | 51.905148 | 60.588860 | 59.907541 | 0.154850 | 0.180181 | 97.610095 | 91.692803 |
| Random Forest | 34.028861 | 11.824019 | 31.786667 | 11.706005 | 0.519976 | 0.937836 | 73.562965 | 25.249015 |
| Neural Network | 52.626672 | 50.659415 | 53.687344 | 53.263843 | 0.015207 | 0.022670 | 105.365963 | 100.114488 |

# Part 5: Feature Selection

Since there are a lot of variables in the dataset, it is essential to find out which variables are more important for the prediction and how many variables should be counted into the prediction models.

### 5.1 RFE Algorithm

Packages used: sklearn.feature_selection, RFECV

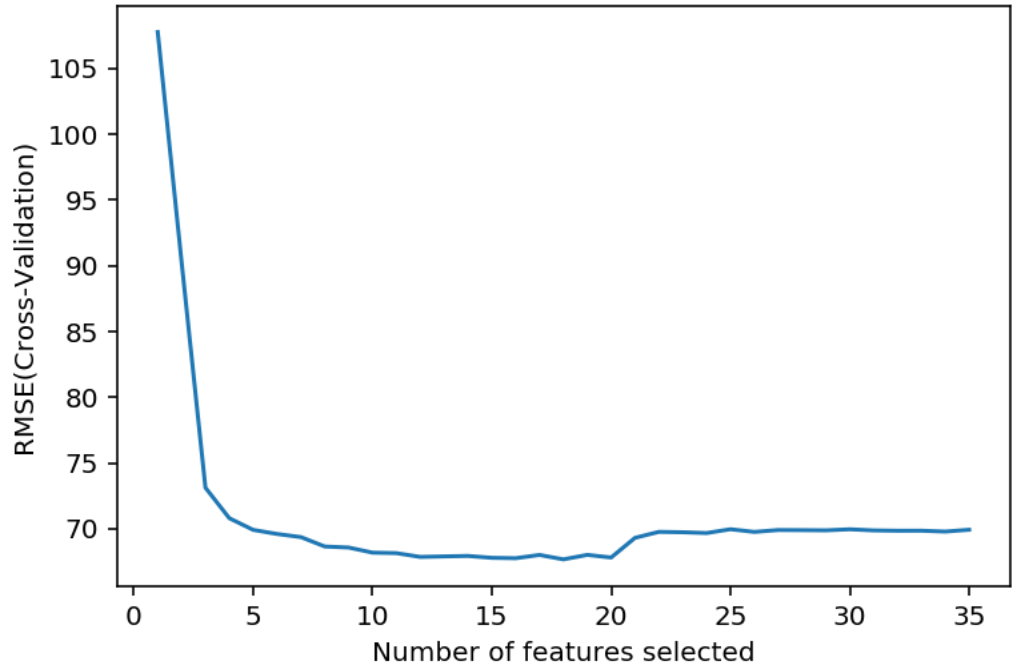



*Figure 7. RMSE using the RFE algorithm. The optimal number of predictors (18) isshown with the filled dot.*

As Figure 7 shows, the blue line is RMSE of test set. according to the RFE method, when the number of features increase, RMSE of the test set begins to decrease and approach to the minimum value when the number of feature is 18, then tends to be a straight line parallel to the x axis. In order to keep the integrity of the data sets, we keep all feature in later random forest regression.

### 5.2 Random Forest Variable Importance

Packages used: sklenarn.ensemble, RandomForestReressor

Random forest regression is a method which use the ensembles of decision tree, which can calculate the relative importance of the features in dataset, thus we use this property to evaluate the importance in this part.

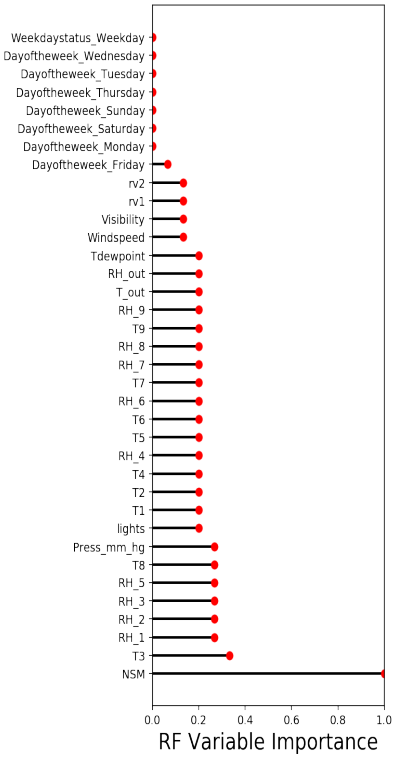



*Figure 8. Variable importance for the different GBM data subsets models*

Figure x depict the importance of the 37 features including two random variables. According to the picture, NSM (number of seconds from midnight is the most important feature in the random forest model), and the features manually added was useless, even lower than the importance of random values.

## 5.3 Exhaustive search, Forward search and Backward search

Packages used: mlxtend.feature_selection SequentialFeatureSelector

### 5.3.1 Exhaustive search

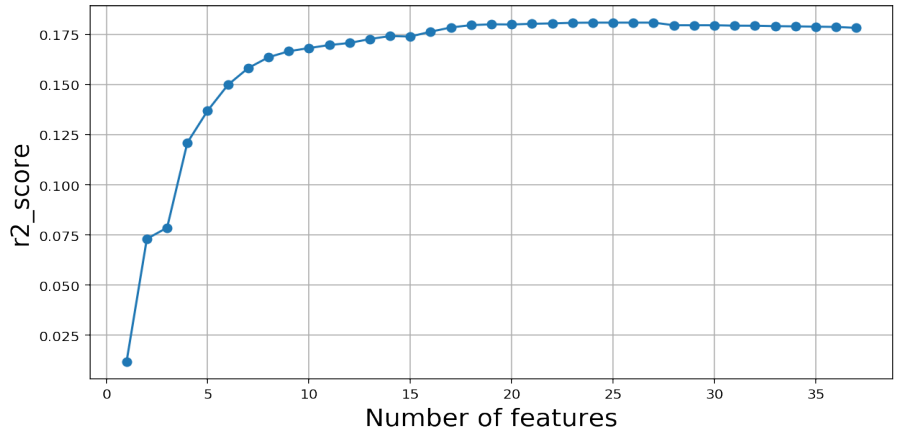



*Figure 9. R2 score using the Exhaustive search algorithm. The optimal number of predictors (25)*

As figure 9 shows, R2 score of the model increase when the number of features increase, and approaches the maximum value when the number of feature is 25, then tends to be a line parallel to x axis.

5.3.2 Forward search

For linear regression, use the forward search, the sequence of the forward search is following, 'NSM' is the first attributes forward search use, which is identical with other approaches.

```
#linear_model with forward search
linear_forward = LinearRegression()
sfs = SFS(linear_forward, k_features=37, forward=True,scoring='r2' )
sfs.fit(X_train.values, y_train.values)
```

```
Index(['NSM', 'lights', 'RH_out', 'RH_1', 'RH_7', 'RH_2',
       'Dayoftheweek_Tuesday', 'RH_8', 'Dayoftheweek_Thursday',
       'Dayoftheweek_Wednesday', 'T3', 'T9', 'T2', 'RH_3', 'T6', 'T_out', 'T8',
       'Dayoftheweek_Sunday', 'Dayoftheweek_Monday', 'Windspeed', 'Visibility',
       'RH_6', 'Press_mm_hg', 'T1', 'Tdewpoint', 'T7', 'RH_9', 'T4', 'RH_4',
       'Dayoftheweek_Friday', 'Weekdaystatus_Weekend', 'Weekdaystatus_Weekday',
       'Dayoftheweek_Saturday', 'T5', 'rv1', 'rv2', 'RH_5'],
      dtype='object')
```

### 5.3.3 Backward search

Back search was also used for linear regression, but it got a different sequence from forward search.

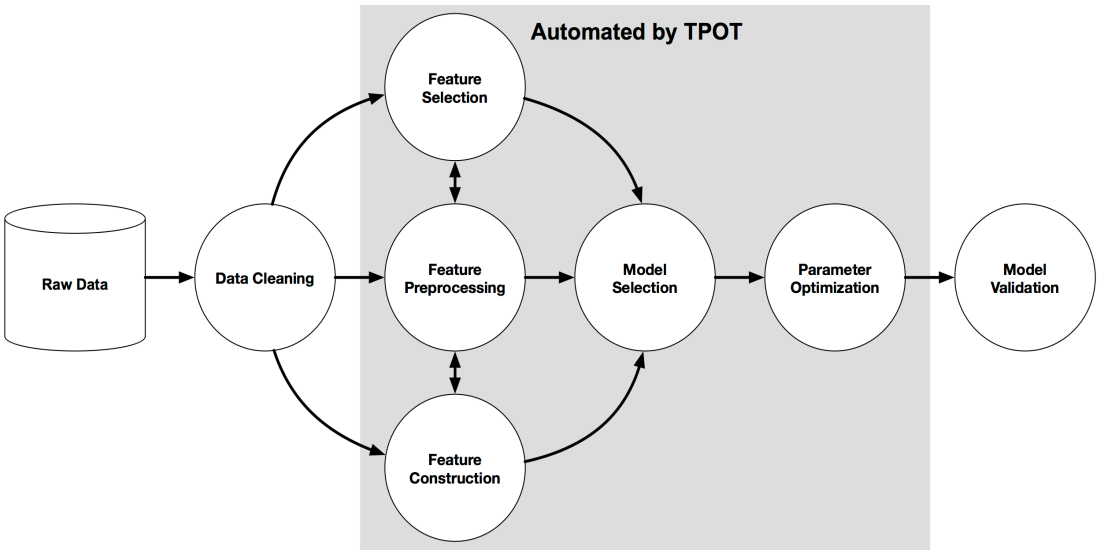
```
#linear_model with backward search
linear_backward = LinearRegression()
sbs = SFS(linear_backward, k_features=1, forward=False,scoring='r2' )
sbs.fit(X_train.values, y_train.values)
```

```
SequentialFeatureSelector(clone_estimator=True, cv=5,
             estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False),
             floating=False, forward=False, k_features=1, n_jobs=1,
             pre_dispatch='2*n_jobs', scoring='r2', verbose=0)
```

## 5.4 Other Feature Selection Packages

In this part, some advanced approach was explored and compared for feature engineering.

### 5.4.1 Tpot




*Process of tpot*

Tpot can automatically generate an appropriate pipeline by exploring thousands of possible pipelines.

```
In [82]: from tpot import TPOTRegressor
         tpot = TPOTRegressor(generations=5, population_size=20, verbosity=2)
         tpot.fit(X_train, y_train)
         print(tpot.score(X_test, y_test))
         tpot.export('tpot_pipeline.py')
         Generation 1 - Current best internal CV score: -5155.070537565258
         Optimization Progress:  50%|     | 60/120 [13:12<20:53, 20.89s/pipeline]
         Generation 2 - Current best internal CV score: -5155.070537565258
         Optimization Progress:  68%|     | 81/120 [25:33<43:15, 66.55s/pipeline]
         Generation 3 - Current best internal CV score: -5155.070537565258
         Optimization Progress:  85%|     | 102/120 [47:12<25:07, 83.74s/pipeline]
         Generation 4 - Current best internal CV score: -4955.58726380178

         Generation 5 - Current best internal CV score: -4955.58726380178

         Best pipeline: RandomForestRegressor(input_matrix, bootstrap=False, max_features=0.8, min_samples_leaf=1, min_samples
         _split=7, n_estimators=100)
         -5613.374074543417
Out[82]: True
```

After five generations, tpot generate a pipeline document. as we can see the best estimator tpot generate is Random Forest Regressor with the parameters illustrate in the above figure.

### 5.4.2 Featuretools

Featuretools is a package achieve the goal of automatically process feature engineering. In this part Featuretools are used to generate more complexed data. First, an entity set named ‘appliance’ which contains single entity named ‘date’ was created to loads the

information of energy datasets. The interesting features which might be useful for regression can be generated by DFS when the target_entity was specified as ‘date’ entity.

```
from featuretools.primitives import (Day, Hour, Minute, Month, Weekday, Week, Weekend)

es.add_interesting_values()

trans_primitives = [Minute, Hour, Day, Week, Month, Weekday, Weekend]

feature_matrix, features = ft.dfs(entityset=es,
                                  target_entity="date",
                                  trans_primitives=trans_primitives,
                                  verbose=True,
                                  approximate='36d')
```

```
Building features: 37it [00:00, 8679.00it/s]
Progress: 100%|██████████| 1/1 [00:00<00:00,  2.11cutoff time/s]
```

```
feature_matrix.head()
```

| T4 | RH_4 | ... | rv1 | rv2 | NSM | MINUTE(date) | HOUR(date) | DAY(date) | WEEK(date) | MONTH(date) | WEEKDAY(date) | IS_WEEKEND(date) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19.000000 | 45.566667 | ... | 13.275433 | 13.275433 | 61200.0 | 0 | 17 | 11 | 2 | 1 | 0 | False |
| 19.000000 | 45.992500 | ... | 18.606195 | 18.606195 | 61800.0 | 10 | 17 | 11 | 2 | 1 | 0 | False |
| 18.926667 | 45.890000 | ... | 28.642668 | 28.642668 | 62400.0 | 20 | 17 | 11 | 2 | 1 | 0 | False |

In this case, 7 features are automatically added to the Dtaframe. With the help of Featuretools, data analysis can save a lot of times.

### 5.4.3 BorutaPy

Different from the Boruta in R, BorutaPy is based on the scikit-learn package, so it has a faster behavior. Boruta is a relevant feature selection method, it tries to find all usable information for prediction.

```
X_train_bo_ =X_train_bo.values
y_train_bo_ = y_train_bo.values.ravel()
rf_boruta =  RandomForestRegressor(n_estimators = 100)
feat_selector = BorutaPy(rf_boruta, n_estimators='auto', verbose=2, random_state=1)
feat_selector.fit(X_train_bo_, y_train_bo_)
```

```
Confirmed:      20
Tentative:      0
Rejected:       17
```

```
/Users/chenlianxu/anaconda3/lib/python3.6/site-packages/boruta/boruta_py.py:418: RuntimeWarning: invalid value encoun
tered in greater
  hits = np.where(cur_imp[0] > imp_sha_max)[0]
```

```
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=63, n_jobs=1,
           oob_score=False,
           random_state=<mtrand.RandomState object at 0x10a630cf0>,
           verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x10a630cf0>,
     two_step=True, verbose=2)
```

```
X_bo.columns[feat_selector.support_]
```

```
Index(['lights', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T5',
       'RH_5', 'RH_6', 'T7', 'RH_7', 'T8', 'T9', 'RH_9', 'Press_mm_hg',
       'RH_out', 'Tdewpoint', 'NSM'],
      dtype='object')
```

```
feat_selector.ranking_
```

```
array([ 1,  6,  1,  1,  1,  1,  1,  1,  1,  1,  1,  3,  1,  1,  1,  1,  2,
        1,  1,  4,  1,  1,  5,  7,  1,  8,  9,  1, 10, 11, 12, 18, 15, 13,
       14, 16, 17])
```

Here a list of attributes which is selected by Boruta is listed, as above figure in instruct, 20 features are used by Boruta, also ranked by Bruta. the feature with rank 1 means its usable for the regression. From the figure, the rank for two random variables was 8 and 9, features manually added was proved to be useless.

### 5.4.4 tsfresh

Tsfresh is the abbreviation of ‘Time Series Feature extraction based on scalable hypothesis tests’. Building features by extracting them from time series automatically tsfresh save a lot time.

```
df_shift, y_t = make_forecasting_frame(df['Appliances'], kind="Wh", max_timeshift=20, rolling_direction=1)
```

```
X_feature = extract_features(df_shift, column_id="id", column_sort="time", column_value="value", impute_function=impute
                     show_warnings=False)
```

```
 'value__fft_coefficient__coeff_95__attr_"angle"'
 'value__fft_coefficient__coeff_95__attr_"imag"'
 'value__fft_coefficient__coeff_95__attr_"real"'
 'value__fft_coefficient__coeff_96__attr_"abs"'
 'value__fft_coefficient__coeff_96__attr_"angle"'
 'value__fft_coefficient__coeff_96__attr_"imag"'
 'value__fft_coefficient__coeff_96__attr_"real"'
 'value__fft_coefficient__coeff_97__attr_"abs"'
 'value__fft_coefficient__coeff_97__attr_"angle"'
 '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"'] did not have any finite values. Filling with zeros.
```

According to the figure, tsfresh extract 99 features in appliance time series. using the features generated by tsfresh, feature regression model can be established.

# Part 6: Model Validation and Selection

5.1 Degree of Polynomial for Linear Model

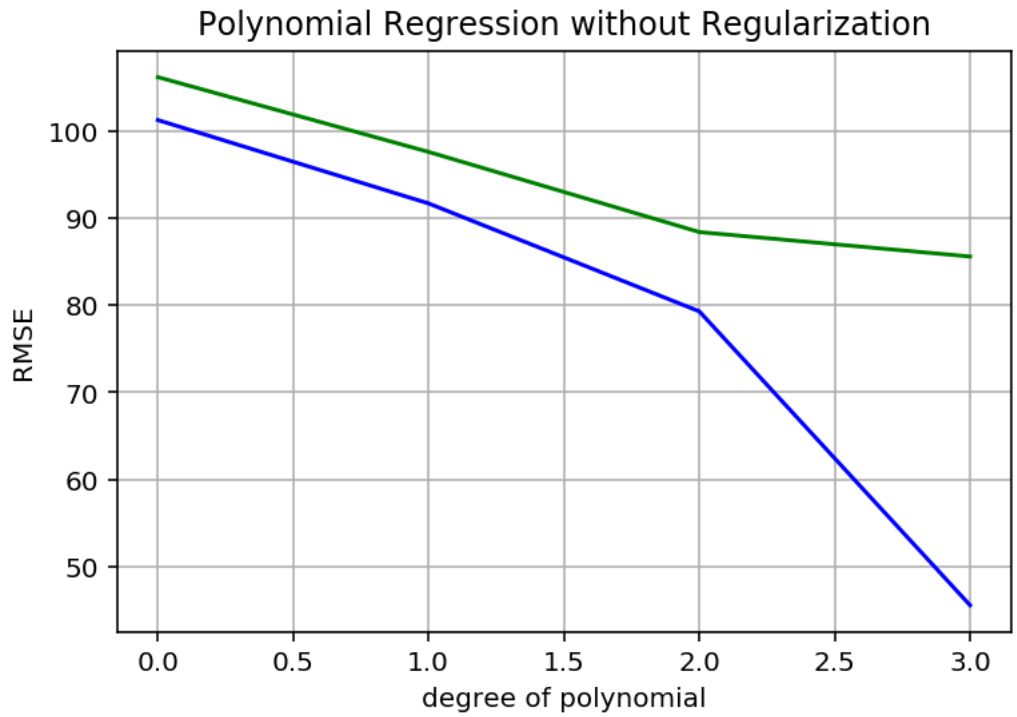



*Figure 10. Polynomial Regression without Regularization*

As Figure 10 shows, the green line is RMSE of test set, the blue line is RMSE of train set. RMSE of linear model decreases with increasing of model complexity. However, a sharp decreasing of train set RMSE appears after quadratic model, which indicates that the model is going to be overfitting after cubic model. Biquadratic model is overfitting for this data set and will have a very big RMSE which cannot be placed in the above graph. Actually, the trend of the green line also shows that the decreasing speed of test set error is slowed after quadratic model. Thus, the conclusion is that cubic model is the best one of linear regression model.

5.2 Regularization with Cross Validation for Linear Model

Since the model is not overfitting at this time, the RMSE of test set would not be decreased significantly after regularization. Thus, for simplifying the computation, quadratic model is selected here to perform regularization and cross validation.

Below is the code of Lasso Regularization. Code of Ridge Regularization is very similar to this one by just changing the fitting model from 'Lasso( )' to 'Ridge( )'. Also the ranges of alpha need to be changed by estimation.

```
model = PolynomialFeatures(degree=2)
X_train_ = model.fit_transform(X_train)
X_test_ = model.fit_transform(X_test)

scores_lasso = []
alphas_lasso = np.arange(0.001,0.02, 0.002)
for alpha in alphas_lasso:
    lasso = Lasso()
    lasso.alpha = alpha
    this_score = np.mean(model_selection.cross_val_score(lasso, X_train_, y_train, cv=10))
    scores_lasso.append(this_score)
```

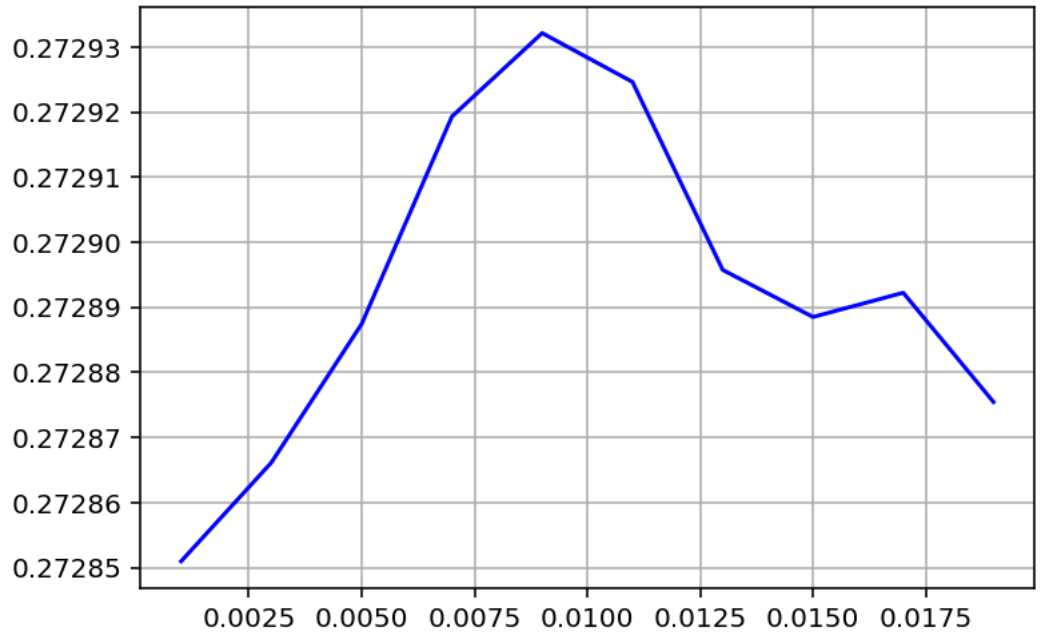



*Figure 11. Optimal Alpha for L1*

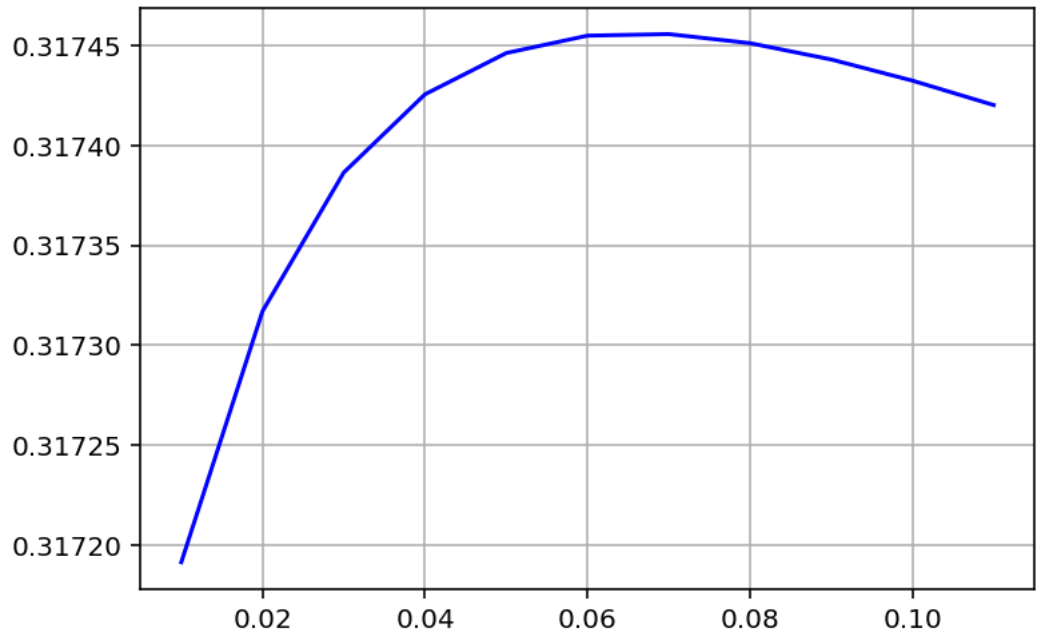



*Figure 12. Optimal Alpha of L2*

The optimal alphas for Lasso and Ridge Regularizations are 0.009, 0.07 correspondently. Then two different regularization regressions are applied to the quadratic model with their optimal alpha values. Error metrics are also calculated after fitting the model with regularization. The computation results are shown as follow:

| | Model | mae_test | mae_train | mape_test | mape_train | r_test | r_train | rmse_test | rmse_train |
|---|---|---|---|---|---|---|---|---|---|
| **0** | lasso | 50.548743 | 47.719088 | 54.854065 | 53.922543 | 0.252223 | 0.302998 | 91.815024 | 84.546041 |
| **0** | ridge | 50.183745 | 46.194405 | 55.766621 | 53.337758 | 0.306217 | 0.386342 | 88.438121 | 79.330397 |
| **0** | no regularization | 50.305300 | 46.264116 | 56.003835 | 53.470374 | 0.306722 | 0.386887 | 88.405929 | 79.295166 |

Before applying regularization, RMSE of test set is 88.41. After Lasso regularization, test RMSE becomes 91.82. After Ridge regularization, test RMSE becomes 88.44. Thus, we could find that regularization cannot reduce prediction error when the model is not overfitting.

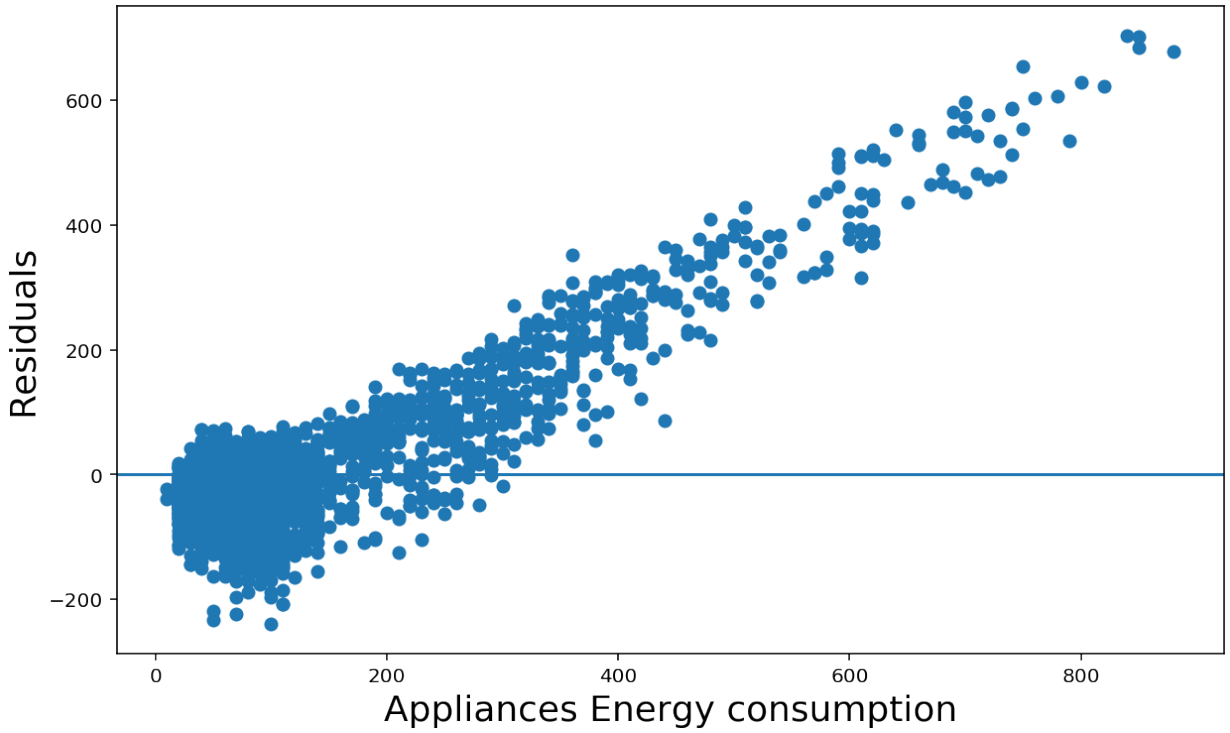



*Figure 13. Residuals and appliances' energy consumption plot of the Lasso regularized quadratic model*

Figure 13 is Residuals and appliances' energy consumption plot of the Lasso regularized quadratic model. It can be easily seen that the variables and the energy consumption of appliances is not well represented by this model since the residuals are not normally distributed around the x=0.

5.3 Cross Validation to Find Appropriate Parameters for Random Forest

There are two parameters discussed in our study, which determine the prediction performance of Random Forest: number of estimators and maximum depth of the trees.

```
scores = []
estimators = np.arange(1,500,100)
for estimator in estimators:
    rf = RandomForestRegressor(n_estimators = estimator , random_state = 0)
    this_score = np.mean(model_selection.cross_val_score(rf, X_train, y_train, cv=10))
    scores.append(this_score)
```

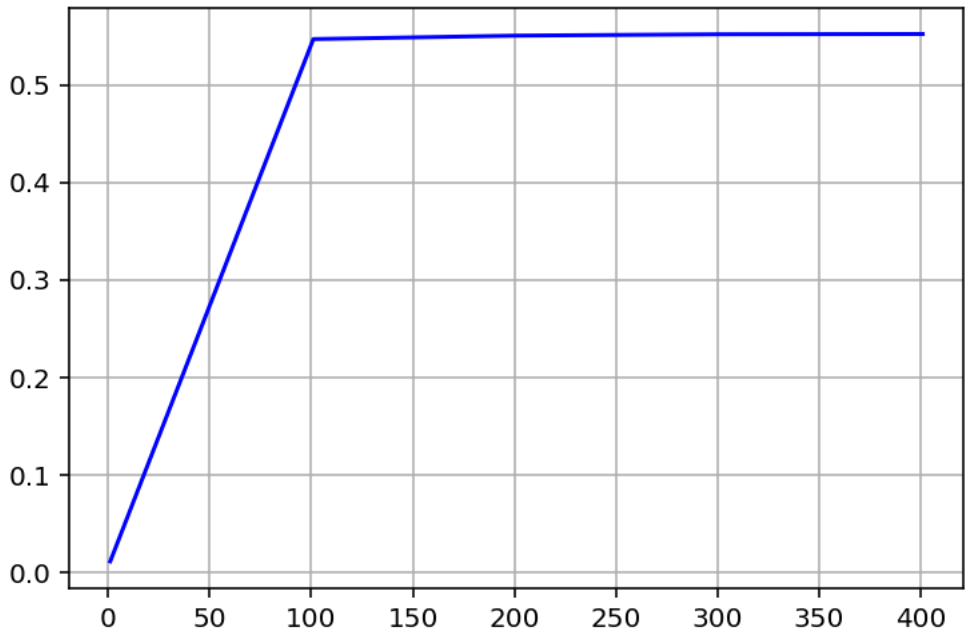



*Figure 14. CV Score - Number of estimators*

```
print scores
```

```
[0.011294046003472221, 0.54681793699715, 0.5503834130395859, 0.5517797097805641, 0.5520494464939416]
```

As Figure 14 shows, there is not big difference in cross validation score after the number of estimators equals to 200. Thus, 200 estimators will be used in the following Random Forest Regression model to save the computation time.

```
depth_list = []
rmse_test_list = []
for i in range(25, 35):
    rf = RandomForestRegressor(n_estimators=200, max_depth=i)
    depth_list.append(i)
    rmse_test = model_selection.cross_val_score(rf, X_train, y_train, n_jobs=5, cv=3, scoring='neg_mean_squared_error')
    rmse_test_list.append(math.sqrt(np.mean(-rmse_test)))
    print rmse_test_list

plt.plot(depth_list, rmse_test_list, '--r')
```

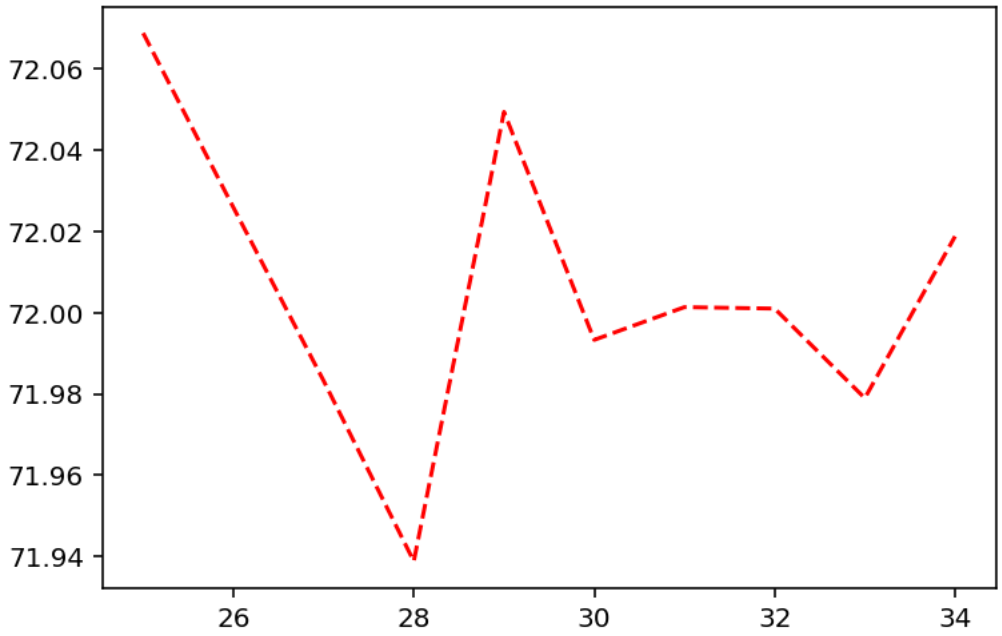



*Figure 15. RMSE - maximum depth*

Figure 15 indicates that 28 is the optimal maximum depth for the Random Forest model in this case. 3-Fold Cross Validation is used here to speed up the computation. Cross Validation with more folds are recommended here to get a better trend of the RMSE curve.

### 5.4 Grid Search to Find Appropriate Parameters for Neural Network

Three parameters are adjusted here using Grid Search to find the optimal Neural Network model: L2 penalty (regularization term) parameter; The initial learning rate used. It controls the step-size in updating the weights; Momentum for gradient descent update. Should be between 0 and 1. The last two parameters are only used when the solver for weight optimization is stochastic gradient descent.

```
param_grid = { "alpha":[.1,.01,.01],
               "momentum":[.7,.9],
               "learning_rate_init":[.1,.01,.01]
}
mlp= MLPClassifier(hidden_layer_sizes=(50,), max_iter=10000,batch_size=600,
                   solver='sgd',  random_state=1,activation="logistic",learning_rate="constant")
random_search = RandomizedSearchCV(mlp, param_distributions= param_grid,n_iter = 8, cv=5)
random_search.fit(X_train, y_train)
report(random_search.grid_scores_,10)
best = random_search.best_estimator_
```

Here are the optimal parameters calculated by the above code.

```
Best alpha: 0.010000
Best momentum: 0.700000
Best initial learning rate: 0.010000
```

## 5.5 Computation of RMSE and R-square for each model

In conclusion, the optimal model of Linear Regression is cubic model without regularization according current research. Actually, biquadratic model with regularization might have a better prediction performance than cubic model. However, the test set error of biquadratic model with regularization would be just close to the test set error of cubic model, which is still much higher than Random Forest and Neural Network. Thus, quadratic model is used here to compute the test set error as a comparator to the other two models.

The optimal model of random forest has 200 estimators and a maximum depth of 28.

The optimal neural network has the following parameters: L2 penalty equals to 0.01; the initial learning rate equals to 0.01; momentum for gradient descent update is 0.7.

```
quadratic_scores = model_selection.cross_val_score(lm, X_train_, y_train, cv=10, scoring='neg_mean_squared_error')
```

```
rf_scores = model_selection.cross_val_score(optimal_rf, X_train, y_train, cv=10, scoring='neg_mean_squared_error')
```

```
nn_scores = model_selection.cross_val_score(optimal_nn, X_train, y_train, cv=10, scoring='neg_mean_squared_error')
```

10-fold cross validation is used here to compute the RMSE of each model for comparison.

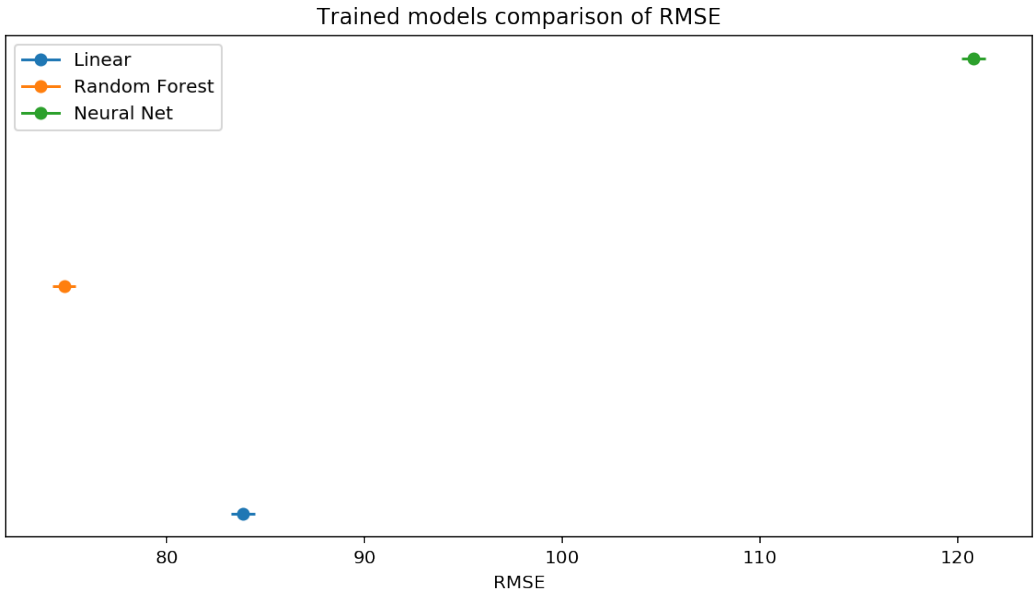



*Figure 16. RMSE - Model*

As Figure 16 shows, Random Forest with optimal number of estimators and maximum depth is the best model of prediction for this data set.

# Part 7: Final pipeline

## 5.1 Optimal Model

The optimal model of random forest has 200 estimators and a maximum depth of 28.

```
optimal_rf = RandomForestRegressor(n_estimators=200, max_depth=28)
```

After training the random forest model with extra optimal parameters generated by t-pot. All the error metrics show that the original one performs better. Thus, the one we selected in the last part is the optimal one.

```
optimal_rf2 = RandomForestRegressor(n_estimators=200, max_depth=28, max_features=0.8,
                                    min_samples_leaf=1, min_samples_split=7)
optimal_rf2.fit(X_train, y_train)
```

| Model | mae_test | mae_train | mape_test | mape_train | r_test | r_train | rmse_test | rmse_train |
|---|---|---|---|---|---|---|---|---|
| Quadratic Linear Regression | 50.305300 | 46.264116 | 56.003835 | 53.470374 | 0.306722 | 0.386887 | 88.405929 | 79.295166 |
| Random Forest | 34.031849 | 11.954068 | 31.901336 | 12.071304 | 0.519952 | 0.938554 | 73.564821 | 25.102843 |
| Neural Network | 53.737333 | 51.801905 | 35.312893 | 35.235218 | -0.215938 | -0.216746 | 117.080241 | 111.705973 |
| Random Forest2 | 35.055201 | 16.466893 | 33.017050 | 16.356763 | 0.505697 | 0.878334 | 74.649081 | 35.323346 |

## 5.2 Create Pipeline

```
import pandas as pd
from sklearn.pipeline import make_pipeline, make_union
from Feature_Engineering import DfAfterFE
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

df = pd.read_csv('energydata_complete.csv')
df1 = DfAfterFE(df)
df1.feature_engineering()
new_df = df1.df
X = new_df.drop(columns=['date', 'Appliances', 'rv1', 'rv2'])
y = new_df['Appliances']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

model = make_pipeline(RandomForestRegressor(n_estimators=200, max_depth=28))
```

```
model.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('randomforestregressor', RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=28,
           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))])
```

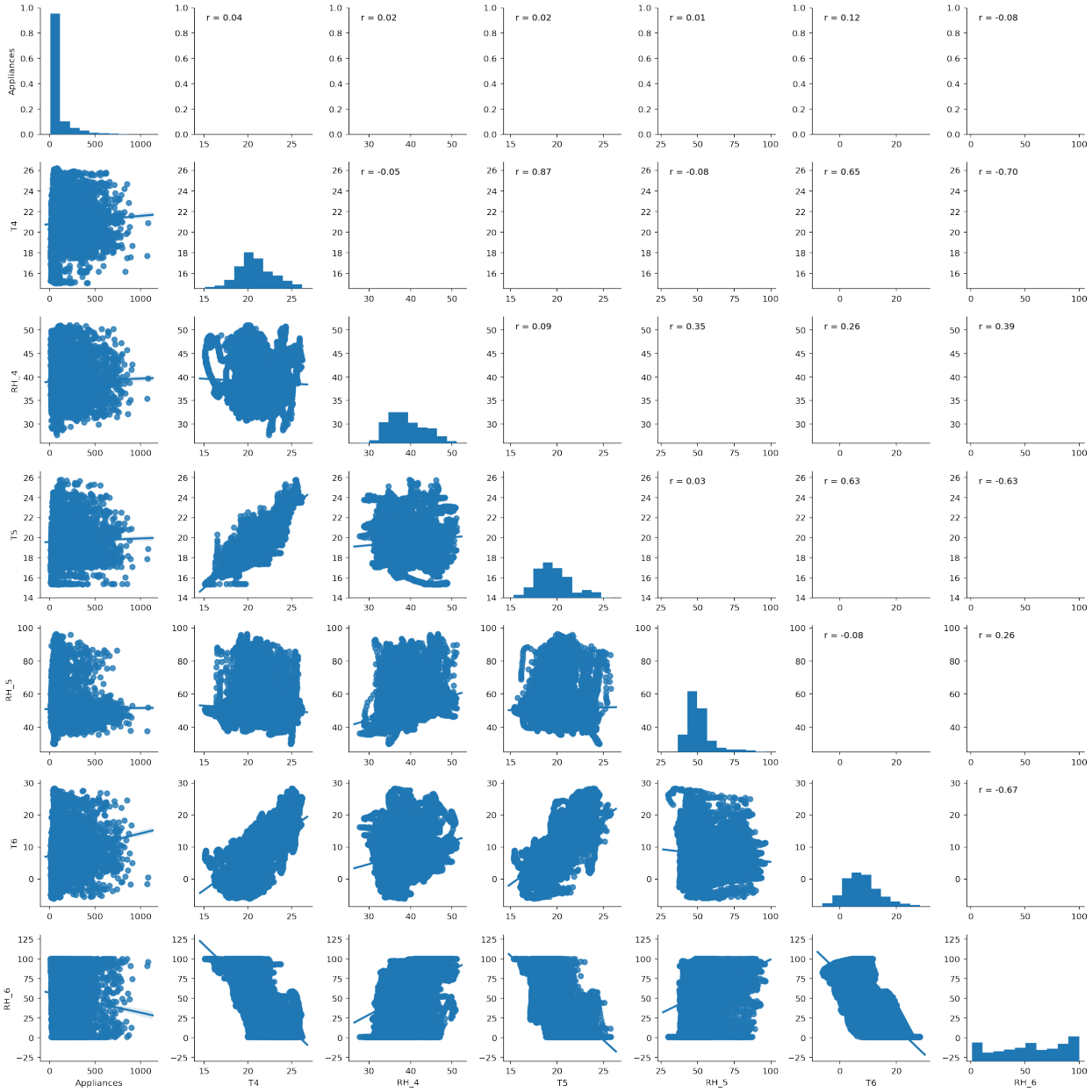
Appendix A.

Data variables and description.

| Data Variables | Units | Number of features |
|---|---|---|
| Appliances energy consumption | Wh | 1 |
| Light energy consumption | Wh | 2 |
| T1, Temperature in kitchen area | ℃ | 3 |
| RH1, Humidity in kitchen area | % | 4 |
| T2, Temperature in living room area | ℃ | 5 |
| RH2, Humidity in living room area | % | 6 |
| T3, Temperature in laundry room area | ℃ | 7 |
| RH3, Humidity in laundry room area | % | 8 |
| T4, Temperature in office room | ℃ | 9 |
| RH4, Humidity in office room | % | 10 |
| T5, Temperature in bathroom | ℃ | 11 |
| RH5, Humidity in bathroom | % | 12 |
| T6, Temperature outside the building (north side) | ℃ | 13 |
| RH6, Humidity outside the building (north side) | % | 14 |
| T7, Temperature in ironing room | ℃ | 15 |
| RH7, Humidity in ironing room | % | 16 |
| T8, Temperature in teenager room 2 | ℃ | 17 |
| RH8, Humidity in teenager room 2 | % | 18 |
| T9, Temperature in parents room | ℃ | 19 |
| RH9, Humidity in parents room | % | 20 |
| To, Temperature outside (from Chièvres weather station) | ℃ | 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) | ℃ | 26 |
| Random Variable 1 (RV 1) | Non dimensional | 27 |
| Random Variable 2 (RV 2) | Non dimensional | 28 |
| Number of seconds from midnight (NSM) | s | 29 |
| Week status (weekend (0) or a weekday (1)) | Factor/categorical | 30 |
| Day of week (Monday, Tuesday. . . Sunday) | Factor/categorical | 31 |
| Date time stamp | year-month-day hour:min:s | - |

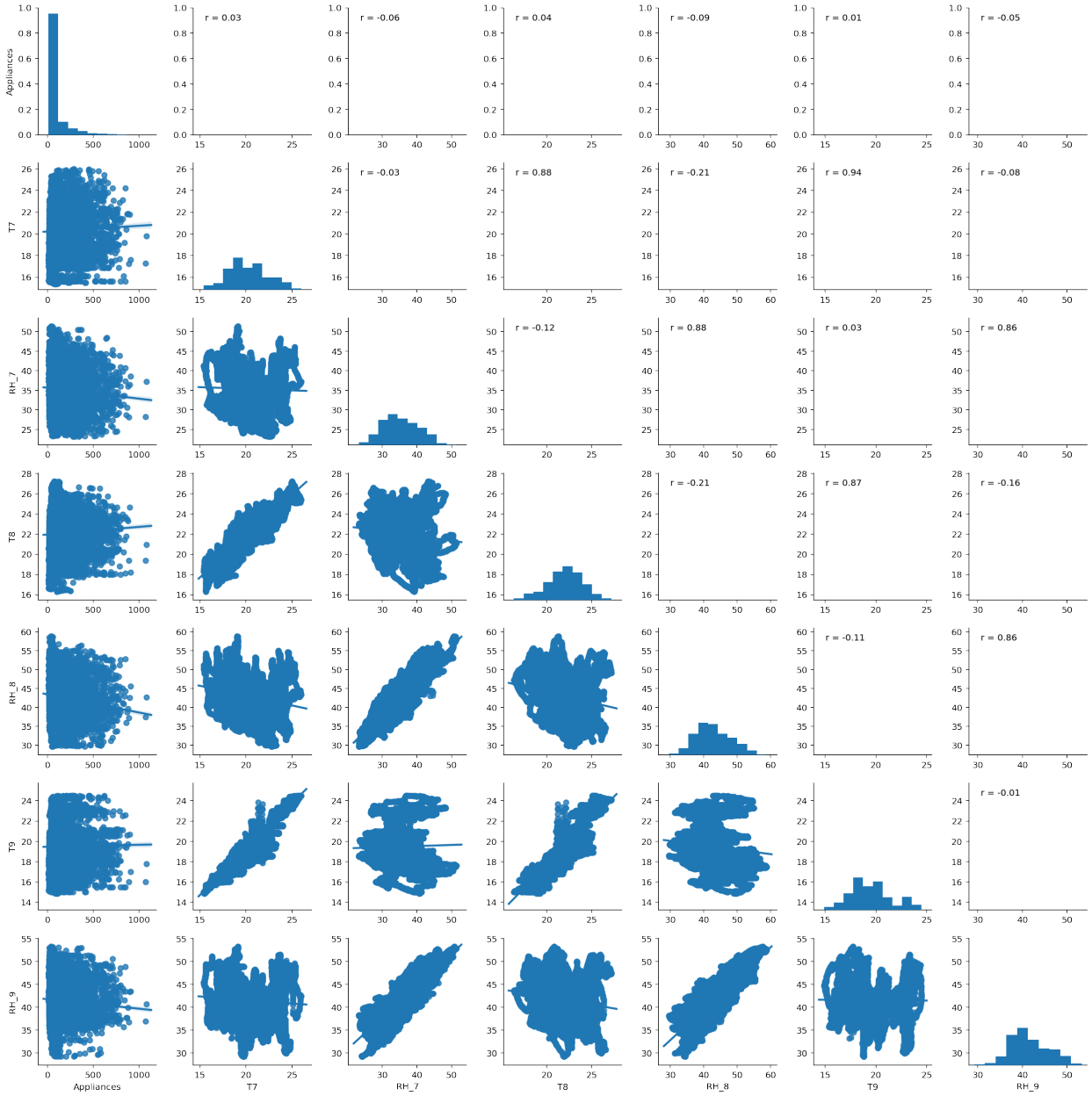
Appendix B.

Pairs plot showing relationships between the energy consumption of appliances with: T4, RH4, T5, RH5, T6, RH6.



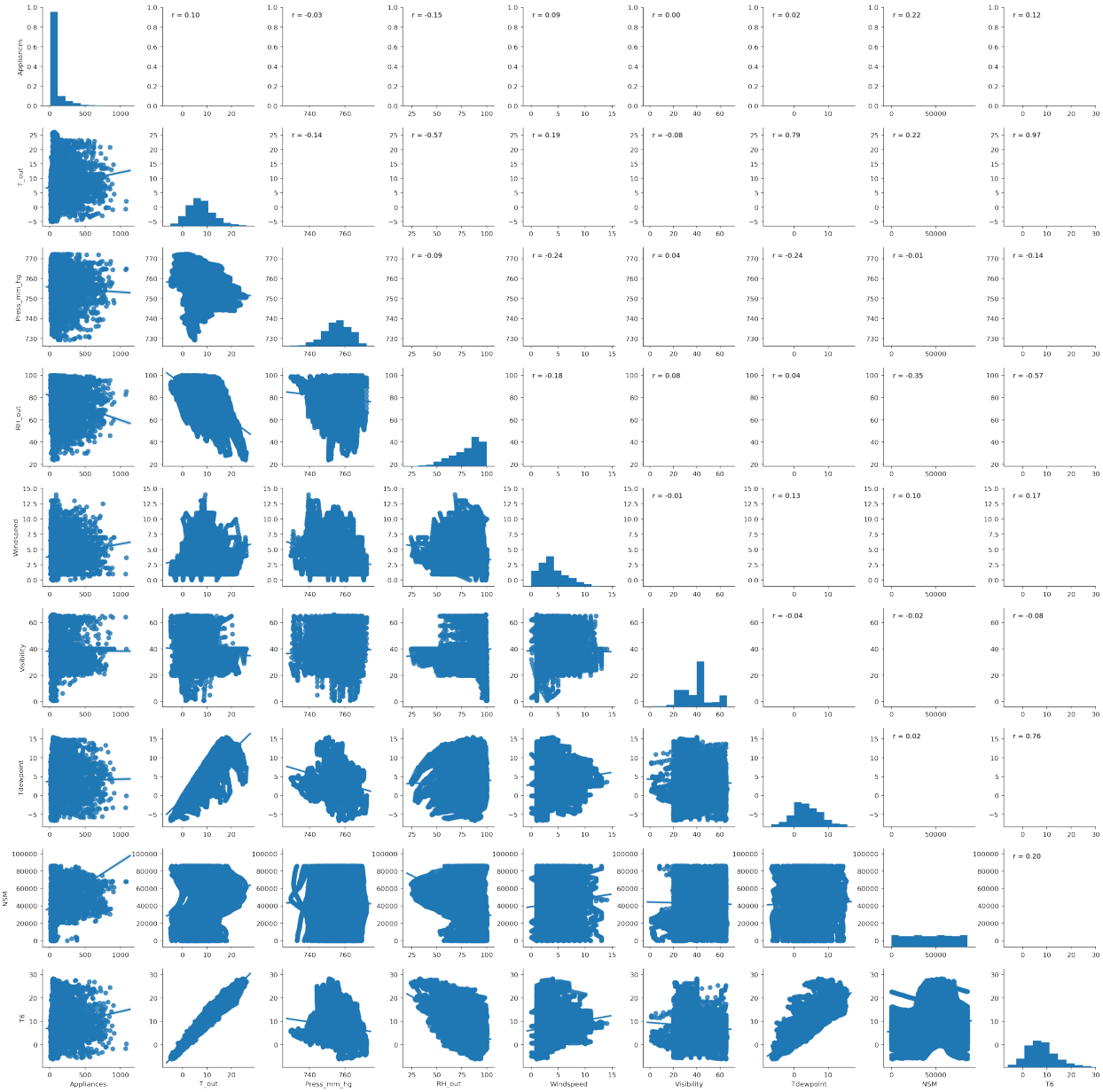

Appendix C.

Pairs plot showing relationships between the energy consumption of appliances with: T7, RH7, T8, RH8, T9, RH9.




Appendix D.

Pairs plot showing relationships between the energy consumption of appliances with: T out, Pressure, RH out, Windspeed, Visibility, TDewpoint, NSM and T6.


Appliances
T_out
Press_mm_hg
RH_out
Windspeed
Visibility
Tdewpoint
NSM
T6
r = 0.10
r = -0.03
r = -0.15
r = 0.09
r = 0.00
r = 0.02
r = 0.22
r = 0.12
r = -0.14
r = -0.57
r = 0.19
r = -0.08
r = 0.79
r = 0.22
r = 0.97
r = -0.09
r = -0.24
r = 0.04
r = -0.24
r = -0.01
r = -0.14
r = -0.18
r = 0.08
r = 0.04
r = -0.35
r = -0.57
r = -0.01
r = 0.13
r = 0.10
r = 0.17
r = -0.04
r = -0.02
r = -0.08
r = 0.02
r = 0.76
r = 0.20