Appliance Energy Consumption Prediction

HarvardX PH125.9x Data Science: Capstone Project

Farah Fauzia

October, 2020

Contents

Abstract

Several types of machine learning algorithms were evaluated to predict appliance energy consumption with only considering light energy consumption and environment data from inside and outside a low-energy house. Tree-based models exhibited better performance than other type of algorithm models in this project. eXtreme Gradient Boosting models after hyperparameters optimization was the best model, which was able to explain 54.41% variances on training set, with only 2.51% difference of RMSE value between training and test set, and 8.55% better MAE value than benchmark model. Humidity were indicated as most important variable to explain energy consumption.

1 Introduction

This report was compiled as part of the Capstone course assignment of Data Science Professional Certificate program from HarvardX (1). The main objective of the assignment was to use publicly available dataset to apply machine learning techniques that go beyond standard linear regression as well as to showcase the skills obtained from the previous courses. For this assignment, predicting appliances energy consumption from a low energy building was chosen as project.

1.1 Project Overview

Appliances energy consumption in buildings represent significant portion of electrical energy demand (2), and therefore energy prediction becomes important for providing higher energy and cost saving for smart homes and smart cities (3). This project focused on regression problem–predicting energy consumption of household appliances based on light energy consumption and numerous environment parameters: temperature, humidity, and weather conditions from certain building. The dataset used was Appliances Energy Prediction Dataset (available at UCI Machine Learning Repository) (4), which contain 4.5 months data for electrical energy consumption and environmental data from a low-energy house in Belgium. While the main relevant paper of the dataset (2) already discussed extensive relationship between the features using several algorithms, this project would focus on regression problem without using time-related information of the dataset.

1.2 Specific Objectives

This specific aims of this project were: a) to apply several types of machine learning algorithms for predicting appliances energy consumption based on only light energy consumption and environment data from inside and outside the building; b) to discuss the general performance of the best model based on benchmarking of evaluation metrics with the main related work (2); and c) to understand what features are important for energy consumption prediction from low-energy house.

1.3 Key Steps

Several steps that would be carried out to achieve the specific goals include:

1. Data Preparation: Dataset would be loaded into `R` version 4.0.2 under Windows 10 x64, carried out on R Studio version 1.3.1093 (full environment can be found on Appendix) and be subjected into pre-processing to ensure its tidiness.

2. Explanatory Data Analysis: initial exploration and basic visualization of dataset would be carried out and feature reduction of dataset would also be carried out for possible effective computation.

3. Modeling Approaches: dataset would be split into two subsets: train and test subset. Control parameters, evaluation metrics, and benchmark target of the models would be defined. Several types of machine learning algorithms would be applied: linear regression model, distance-based model (K-Nearest Neighbor), neural network model (Artificial Neural Network), and tree-based models (Random Forest, Gradient Boost Machine, and eXtreme Gradient Boosting).

4. Model Testing & Performance Summary: the best models with satisfying evaluation metrics on train dataset would be fitted to test dataset and the performance would be evaluated and discussed to draw conclusion based on previously stated specific objectives.

2 Explanatory Data Analysis

2.1 Dataset Overview

The Appliances Energy Prediction Dataset (4) consists of numerous measurement of energy consumption and environment parameters from a low energy house in Stamburges, Belgium (2). Several measurements were collected from wireless sensors inside and outside house, and joined with information from nearby weather station, which were recorded every 10 minutes for 137 days (3). Table 1 summarize the variables description of the dataset after pre-processing:

Table 1: Dataset Variables and Description [a]

Variable Names	**Description**	**Unit** [b]
Date_Time	Date & time information	Y-M-D h:min:s
Appliances	Appliances energy consumption	Wh
Lights	Lights energy consumption	Wh
T_kitchen	Temperature in kitchen	Celcius
H_kitchen	Humidity in kitchen	%
T_living	Temperature in living room	Celcius
H_living	Humidity in living room	%
T_laundry	Temperature in laundry room	Celcius
H_laundry	Humidity in laundry doom	%
T_office	Temperature in office room	Celcius
H_office	Humidity in office room	%
T_bathroom	Temperature in bathroom	Celcius
H_bathroom	Humidity in bathroom	%
T_outNorth	Temperature of north side outside house	Celcius
H_outNorth	Humidity of north side outside house	%
T_ironRoom	Temperature in ironing room	Celcius
H_ironRoom	Humidity in ironing room	%
T_teenRoom2	Temperature in teenager room 2	Celcius
H_teenRoom2	Humidity in teenager room 2	%
T_parentRoom	Temperature in parents room	Celcius
H_parentRoom	Humidity in parents room	%
T_outside	Temperature outside (from nearby weather station)	Celcius
Pressure	Pressure (from nearby weather station)	mmHg
H_outside	Humidity outside (from nearby weather station)	%
WindSpeed	WindSpeed (from nearby weather station)	m/s
Visibility	Visibility (from nearby weather station)	km
T_dewPoint	Temperature dew point (from nearby weather station)	Celcius
RandVar1	Random Variable 1	Non dimensional
RandVar2	Random Variable 2	Non dimensional

[a] modified from (2)

[b] Temperatures is in degree

2.2 Initial Exploration & Visualization

These tables summarize the first 6 lines of the dataset:

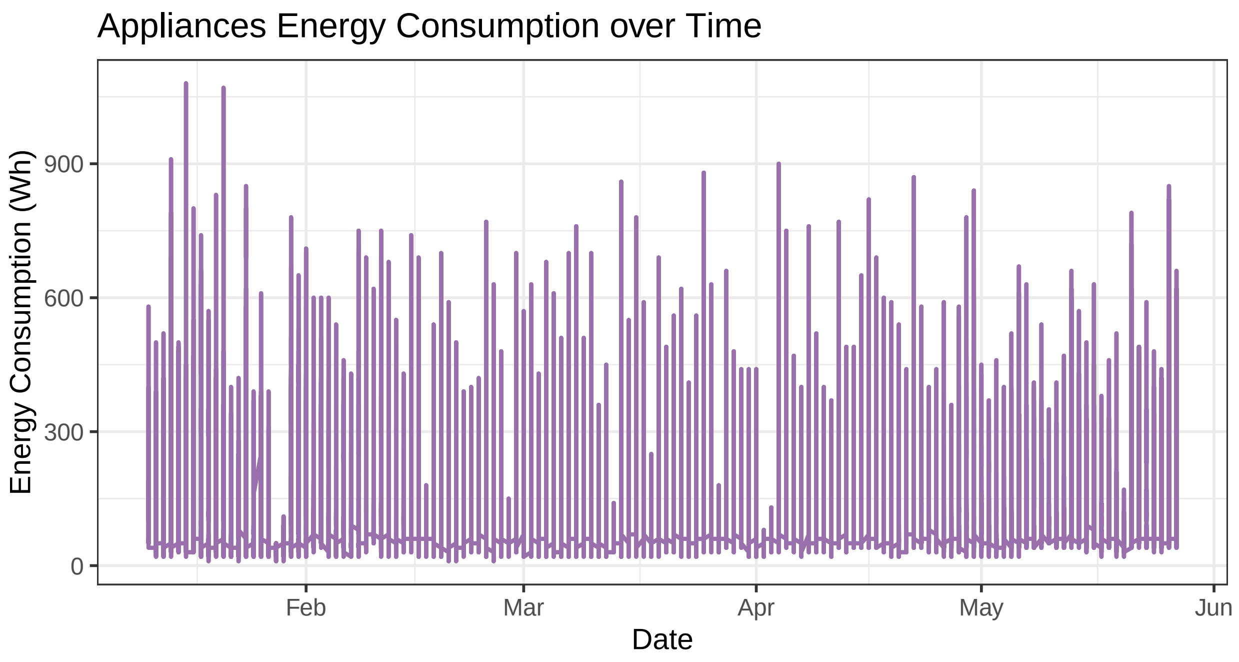
Date_Time	Appliances	Lights	T_kitchen	H_kitchen	T_living	H_living	T_laundry	H_laundry	T_office	H_office	T_bathroom	H_bathroom	T_outNorth	H_outNorth
2016-01-11 17:00:00	60	30	19.9	47.6	19.2	44.8	19.8	44.7	19.0	45.6	17.2	55.2	7.03	84.3
2016-01-11 17:10:00	60	30	19.9	46.7	19.2	44.7	19.8	44.8	19.0	46.0	17.2	55.2	6.83	84.1
2016-01-11 17:20:00	50	30	19.9	46.3	19.2	44.6	19.8	44.9	18.9	45.9	17.2	55.1	6.56	83.2
2016-01-11 17:30:00	50	40	19.9	46.1	19.2	44.6	19.8	45.0	18.9	45.7	17.2	55.1	6.43	83.4
2016-01-11 17:40:00	60	40	19.9	46.3	19.2	44.5	19.8	45.0	18.9	45.5	17.2	55.1	6.37	84.9
2016-01-11 17:50:00	50	40	19.9	46.0	19.2	44.5	19.8	44.9	18.9	45.7	17.1	55.0	6.30	85.8

T_ironRoom	H_ironRoom	T_teenRoom2	H_teenRoom2	T_parentRoom	H_parentRoom	T_outside	Pressure	H_outside	WindSpeed	Visibility	T_dewPoint	RandVar1	RandVar2
17.2	41.6	18.2	48.9	17.0	45.5	6.60	734	92	7.00	63.0	5.3	13.3	13.3
17.2	41.6	18.2	48.9	17.1	45.6	6.48	734	92	6.67	59.2	5.2	18.6	18.6
17.2	41.4	18.2	48.7	17.0	45.5	6.37	734	92	6.33	55.3	5.1	28.6	28.6
17.1	41.3	18.1	48.6	17.0	45.4	6.25	734	92	6.00	51.5	5.0	45.4	45.4
17.2	41.2	18.1	48.6	17.0	45.4	6.13	734	92	5.67	47.7	4.9	10.1	10.1
17.1	41.3	18.1	48.6	17.0	45.3	6.02	734	92	5.33	43.8	4.8	44.9	44.9

This dataset contains no missing values, already in tidy format with each rows representing one observation which was taken in 10 minutes interval time. There are total 19,735 observation of 29 variables, and with following structures:

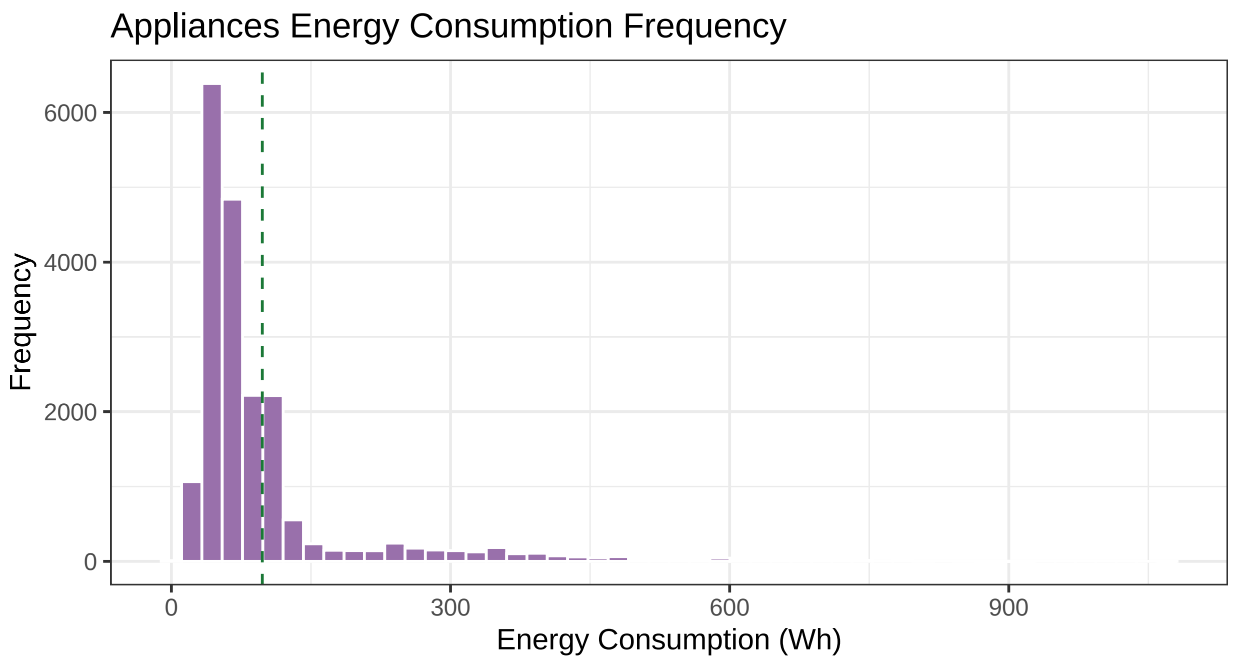
```
## 'data.frame':    19735 obs. of  29 variables:
##  $ Date_Time   : chr  "2016-01-11 17:00:00" "2016-01-11 17:10:00" "2016-01-11 17:20:00" "2016-01-11
##  $ Appliances  : int  60 60 50 50 60 50 60 60 60 70 ...
##  $ Lights      : int  30 30 30 40 40 40 50 50 40 40 ...
##  $ T_kitchen   : num  19.9 19.9 19.9 19.9 19.9 ...
##  $ H_kitchen   : num  47.6 46.7 46.3 46.1 46.3 ...
##  $ T_living    : num  19.2 19.2 19.2 19.2 19.2 ...
##  $ H_living    : num  44.8 44.7 44.6 44.6 44.5 ...
##  $ T_laundry   : num  19.8 19.8 19.8 19.8 19.8 ...
##  $ H_laundry   : num  44.7 44.8 44.9 45 45 ...
##  $ T_office    : num  19 19 18.9 18.9 18.9 ...
##  $ H_office    : num  45.6 46 45.9 45.7 45.5 ...
##  $ T_bathroom  : num  17.2 17.2 17.2 17.2 17.2 ...
##  $ H_bathroom  : num  55.2 55.2 55.1 55.1 55.1 ...
##  $ T_outNorth  : num  7.03 6.83 6.56 6.43 6.37 ...
##  $ H_outNorth  : num  84.3 84.1 83.2 83.4 84.9 ...
##  $ T_ironRoom  : num  17.2 17.2 17.2 17.1 17.2 ...
##  $ H_ironRoom  : num  41.6 41.6 41.4 41.3 41.2 ...
##  $ T_teenRoom2 : num  18.2 18.2 18.2 18.1 18.1 18.1 18.1 18.1 18.1 18.1 ...
##  $ H_teenRoom2 : num  48.9 48.9 48.7 48.6 48.6 ...
##  $ T_parentRoom: num  17 17.1 17 17 17 ...
##  $ H_parentRoom: num  45.5 45.6 45.5 45.4 45.4 ...
##  $ T_outside   : num  6.6 6.48 6.37 6.25 6.13 ...
##  $ Pressure    : num  734 734 734 734 734 ...
##  $ H_outside   : num  92 92 92 92 92 ...
##  $ WindSpeed   : num  7 6.67 6.33 6 5.67 ...
##  $ Visibility  : num  63 59.2 55.3 51.5 47.7 ...
##  $ T_dewPoint  : num  5.3 5.2 5.1 5 4.9 ...
##  $ RandVar1    : num  13.3 18.6 28.6 45.4 10.1 ...
##  $ RandVar2    : num  13.3 18.6 28.6 45.4 10.1 ...
```

Appliances energy consumption over the measurement period shows the consumption data varies greatly with no particular trend in time series:



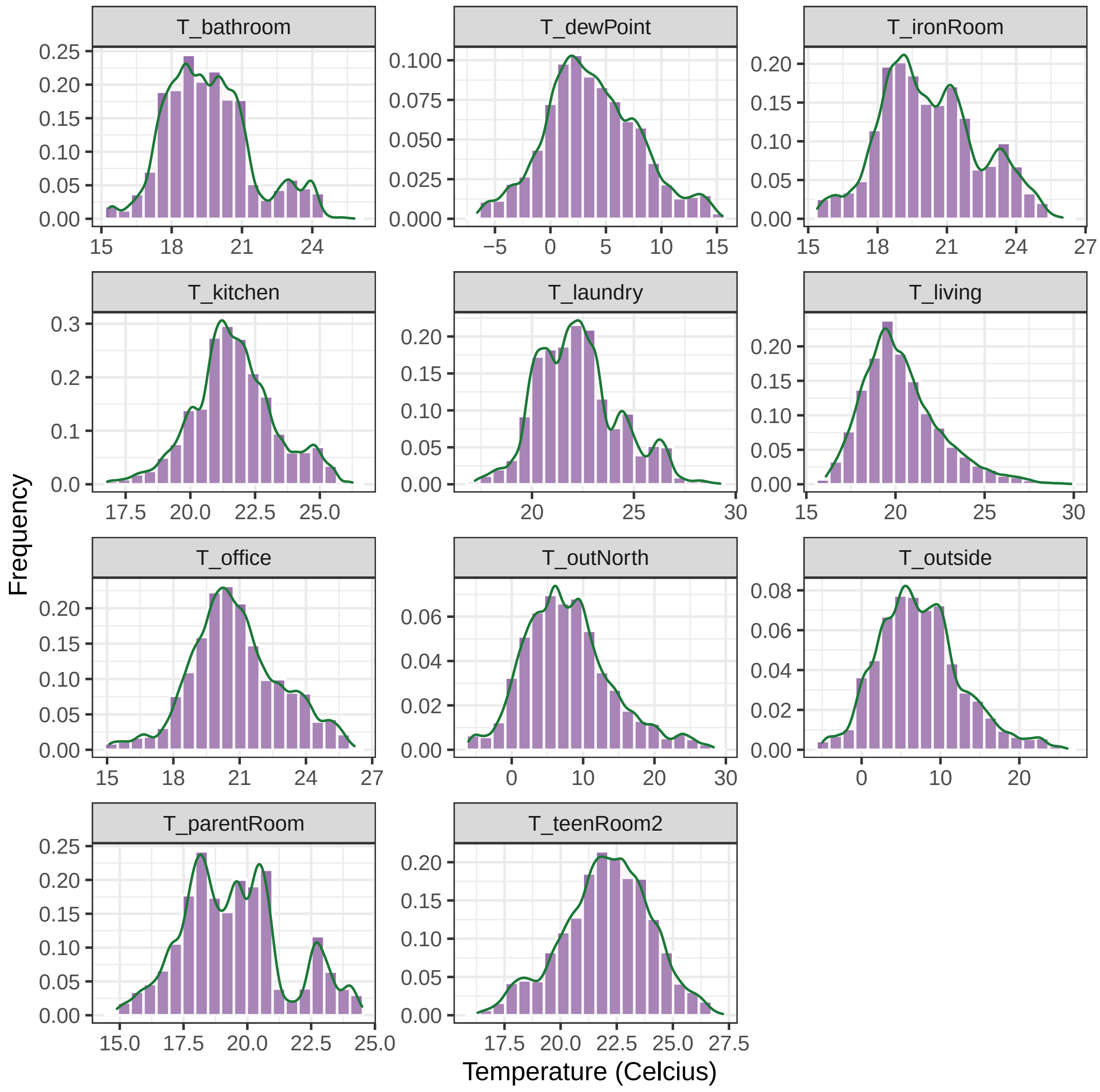
For this project, the time-related feature would not be explored and the main focus would be regression problem based on other available features.

Average energy consumption of appliances is 97.69 Wh, with the frequency of the appliances energy consumption of the house can be seen in the following figure:



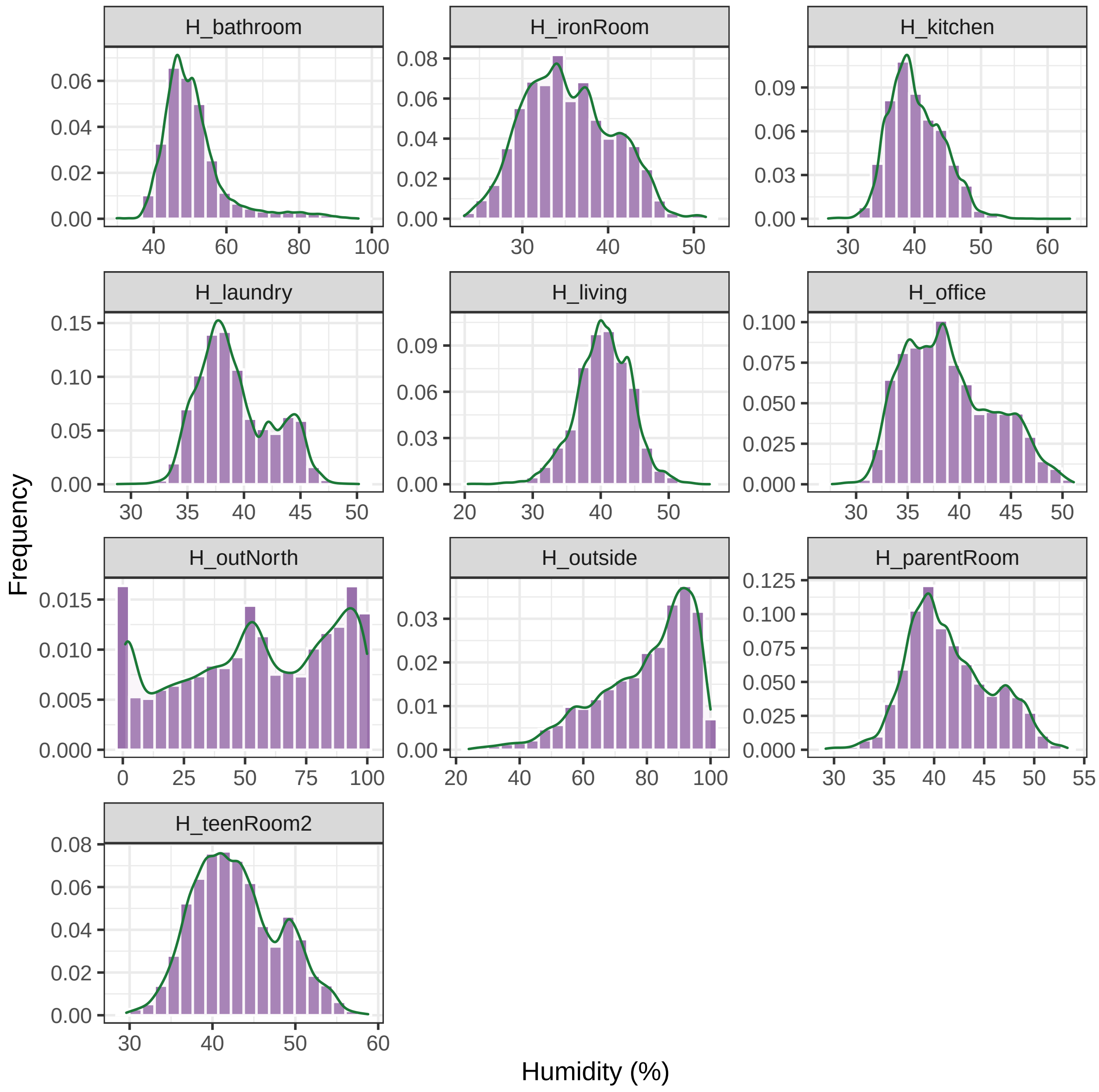
The plot above indicates that the house is indeed representative as low energy building as most consumption value was below 100 Wh.

The distribution of temperature conditions from inside and outside the house can be seen in the following figures:



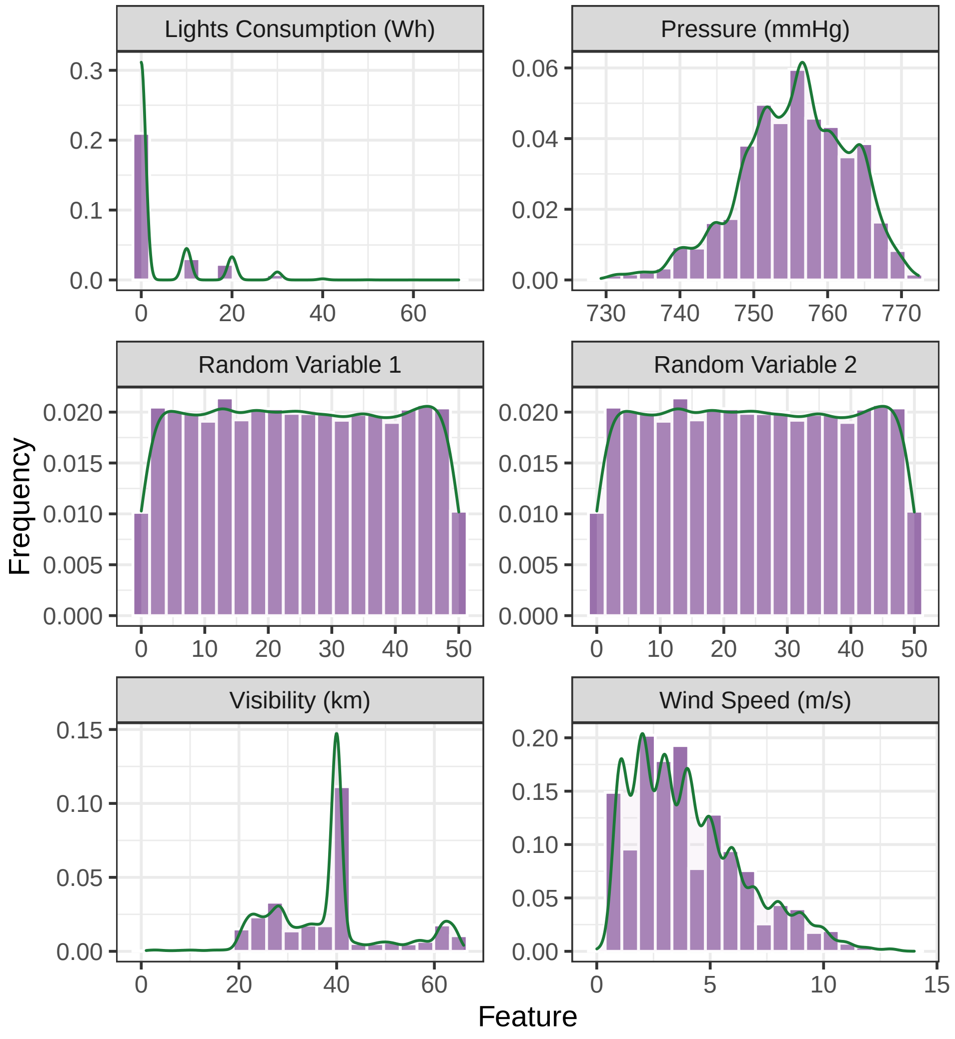
Most of the measurements showed almost normal distribution, excluding the `T_parentRoom`. It was also observed that `T_outNorth` (measurement from sensor outside of the house) and `T_outside` (measurement from nearby weather station) exhibit very similar distribution, which indicates the possibility of linear relationship between these features.

Similarly, distribution of humidity conditions inside and outside the house can be seen in the figures on the following page.



Humidity measurements from inside of the house showed normal distribution tendency. However, unlike the previous temperature distribution, measurement from outside sensor of the house (`H_outNorth`) and from nearby weather station (`H_outside`) did not show similar trend. This indicates that the two variables may not highly correlate to each other.

Distribution from other features can be seen in the following figure:



From plot above it can be seen that the except `Pressure`, remaining features exhibited irregular distribution and almost no distribution for Random Variable 1 & Random Variable 2.

From the visualizations, some features had possibility of having close relationship between each other. Therefore, it might be possible to reduce some similar features in order to optimize modeling computation.

2.3 Features Reduction

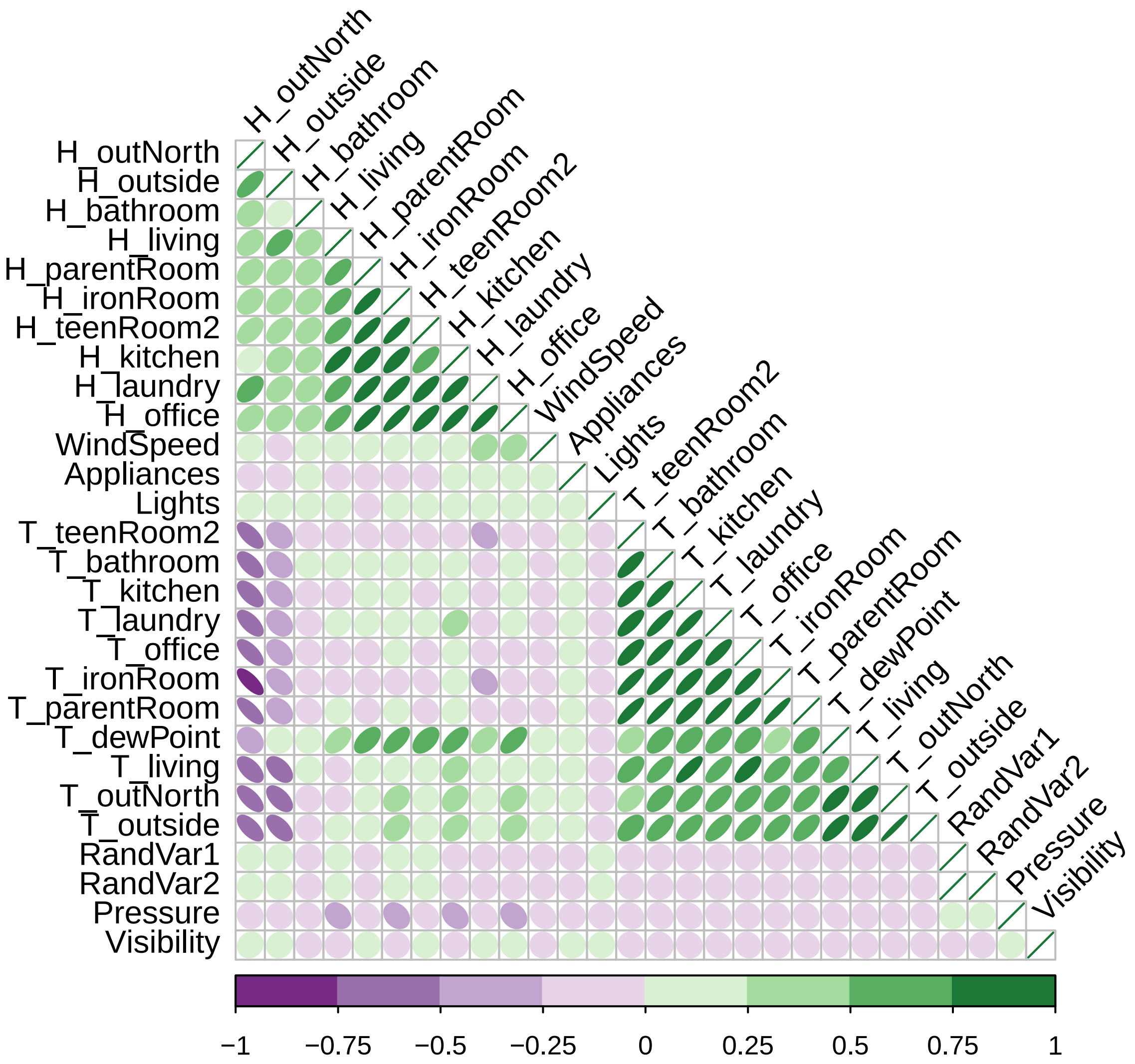
Aside from time-related feature, this dataset consists of 27 features and 1 target variable. Before applying the machine learning algorithm, it is necessary to do pre-processing to remove the features that are clearly not useful. Pre-processing can be done by removing features with very few non-unique values or close to zero variation (5). The `nearZero` function from `Caret` package was used and the following result was obtained:

	freqRatio	**percentUnique**	**zeroVar**	**nzv**
Lights	6.89	0.041	FALSE	FALSE
T_kitchen	1.06	3.658	FALSE	FALSE
H_kitchen	1.06	12.906	FALSE	FALSE
T_living	1.02	8.361	FALSE	FALSE
H_living	1.09	17.107	FALSE	FALSE
T_laundry	1.16	7.226	FALSE	FALSE
H_laundry	1.18	13.266	FALSE	FALSE
T_office	1.01	7.043	FALSE	FALSE
H_office	1.02	15.136	FALSE	FALSE
T_bathroom	1.03	11.467	FALSE	FALSE
H_bathroom	1.10	38.363	FALSE	FALSE
T_outNorth	1.02	22.529	FALSE	FALSE
H_outNorth	1.93	49.197	FALSE	FALSE
T_ironRoom	1.01	9.906	FALSE	FALSE
H_ironRoom	1.11	29.851	FALSE	FALSE
T_teenRoom2	1.28	11.290	FALSE	FALSE
H_teenRoom2	1.19	33.691	FALSE	FALSE
T_parentRoom	1.02	4.682	FALSE	FALSE
H_parentRoom	1.04	17.167	FALSE	FALSE
T_outside	1.11	8.766	FALSE	FALSE
Pressure	1.29	11.092	FALSE	FALSE
H_outside	1.37	2.868	FALSE	FALSE
WindSpeed	1.06	0.958	FALSE	FALSE
Visibility	16.09	2.093	FALSE	FALSE
T_dewPoint	1.12	7.140	FALSE	FALSE
RandVar1	1.00	100.000	FALSE	FALSE
RandVar2	1.00	100.000	FALSE	FALSE

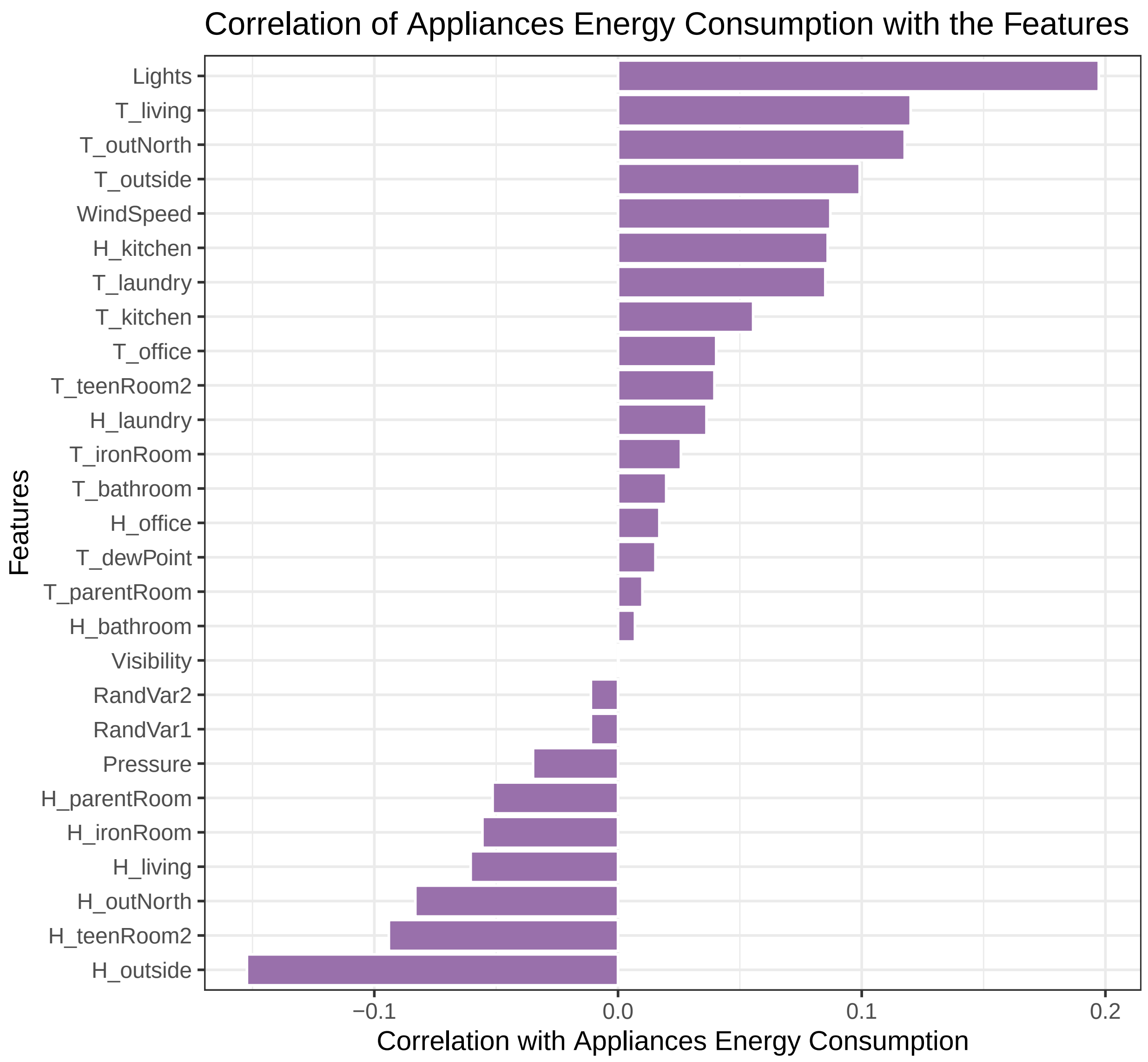
The results shown that there is no particular features recommended to be removed. Therefore, another approach of removing features that are highly correlated with others (5) would be used.

Correlation can be generally defined as a measure of how strongly one variable depends on another (6). For this approach, the correlation matrix of the variables, excluding `Date_Time` feature, was generated using `corrplot` package with Pearson correlation coefficient as indicators. Pearson correlation coefficient summarizes the strength of linear relationship between two data variables (ranged between -1, positive correlation, and 1, negative correlation) (6).

In order to generate correlation matrix, all values would need to be converted into numerical class. The visualization of all pairs correlation of the dataset can be seen as follow:



For easier interpretation, `corrr` package (7) was also used. First, relationship between target correlation with all features was examined. The result can be summarized as following visualization:



From the plot above, it is clear that there is no particular feature with high correlation (> absolute 0.9) with target variable. Highest positive correlation was found with light energy consumption `Lights` (around `0.2`) and highest negative correlation with outside humidity `H_outside` (`-0.15`). It is also important to note discuss about `RandVar1` & `RandVar2` features, which were originally included in the dataset only for the purpose of verifying method used to filter variable features (2). While it was expected that these two random variables exhibit almost no correlation (absolute value of `0.01`) with the target variable, another feature, `Visibility`, shows no systematic relationship with target variable at all (value `0`). For this reason, the `Visibility` feature would be removed.

If two features are highly correlated among themselves, it is possible to make accurate prediction on the target with just one of them (6). Therefore, for next step, the two highly correlated (> absolute 0.9) features were identified, which can be summarized by the following table:

rowname	colname	cor
T_parentRoom	T_laundry	0.901
T_parentRoom	T_bathroom	0.911
T_outside	T_outNorth	0.975
T_parentRoom	T_ironRoom	0.945
T_laundry	T_parentRoom	0.901
T_bathroom	T_parentRoom	0.911
T_ironRoom	T_parentRoom	0.945
T_outNorth	T_outside	0.975
RandVar2	RandVar1	1.000
RandVar1	RandVar2	1.000

As previously mentioned, `RandVar1` & `RandVar2` features were only random variables. Since they also perfectly correlated to each other, these two features would be removed. Another highly correlated features was between `T_outside` and `T_outNorth`, which has been previously indicated in the features distribution visualization. Based on this, `T_outNorth` would be removed since it can be represented by measurement data from weather station outside the house `T_outside`. For the remaining correlation, it is indicated that `T_parentRoom` generally exhibit linear relationship with temperature measurements in other rooms (i.e. iron room, bathroom, laundry room). Since it can be represented by those similar features, `T_parentRoom` would be removed.

To summarize, through correlation matrix analysis, 5 features `Visibility`, `RandVar1`, `RandVar2`, `T_outNorth` and `T_parentRoom` would be removed from dataset, resulting final 23 variables.

3 Modeling Approaches

3.1 Preparation

The dataset would be split into train set and test set. The splitting ratio would mimic the setup used in the main relevant paper (2), where 75% of dataset would serve as train set and the remaining 25% would be test set. These sub-dataset include 23 variables (1 target and 22 features), with 14799 observations in train dataset and 4936 observations in test dataset.

Several machine learning algorithm types (linear regression model, distance-based model, neural network model, and tree-based models) using `Caret` package would be applied into train set and the best model would be evaluated on the test set. All models would be fitted using same train control parameter, through 5 folds cross-validation. `doParallel` package would also be used to enable parallel processing when running model for speeding up the computations.

3.2 Evaluation Metrics & Benchmark

Basic evaluation of machine learning algorithms is comparing the obtained predicted value with the actual outcome, which can be measured using loss function or the difference between these values. For this project, root mean squared error (RMSE) and mean absolute error (MAE) would be used as loss function parameters to compare performance of the models. These metrics were chosen to meet one of the specific objectives of the project: to discuss general performance of best model by benchmarking to main related work (2).

RMSE can be defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{n}(Y_i - \hat{Y}_i)^2}$$

and MAE can be defined as:

$$MAE = \frac{1}{n}\sum_{n}|Y_i - \hat{Y}_i|$$

Where Y_i is the actual energy consumption measurement, $\hat{Y}_i$ is the predicted value, and n is number of measurements. The lower RMSE and MAE values indicate better models.

In this project, coefficient of determination or R-squared (R^2) would also be discussed to evaluate how many variances a model could explain. This metric would be pulled from train result using `Caret` package. Higher R^2 value indicates better performance.

In addition, running time of the models would also be compared by utilizing `proc.time` function. Running time would be measured for each model under active parallel processing condition.

Main related work (2) used four models considering all features (including time-related information): multiple linear regression, support vector machine with radial kernel, random forest (RF), and gradient boosting machines (GBM), with the best model have following performance:

	Train		**Test**	
Model	**RMSE**	**MAE**	**RMSE**	**MAE**
Benchmark (GBM)	17.6	12	66.7	35.2

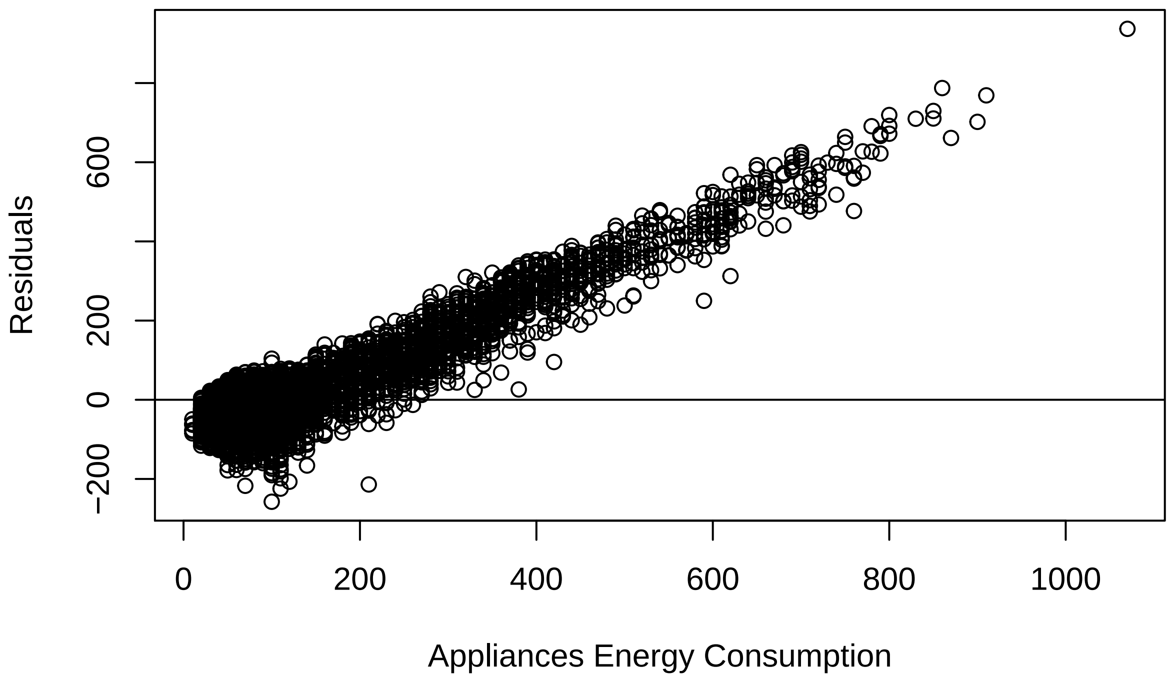
Since due to computational limitation this project could not setup similar modeling condition (10 fold cross-validation with 3 repeats), the benchmarking would be discussed in general manner.

3.3 Linear Regression Model

The first type of model is simple Multiple Linear Regression (MLR) model, which would use all available 22 features to find appropriate slope quantifying effect of each of them and respective response. The obtained evaluation metrics was:

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
Multiple Linear Regression (MLR)	93.2	52.9	0.158	2.12

MLR could only explain about 15.81% of the variance. In addition, the residual plot of the MLR model can be generated:



Residuals were difference between the true observed data and the fitted values of dependent variable. From plot above, it can be seen that the residuals are not normally distributed around the horizontal axis, which indicates that this model could not properly represent the relationship between the predictors and appliances energy consumption (2).

3.4 Distance-based Model

Distance-based model can also be used in regression problem, and in this project K-Nearest Neighbor (KNN) model was chosen as representative of this type. KNN algorithm predicts outcome of new observation through comparison to k similar cases in the training dataset (8), with k is tuning parameter that can be determined.

As one example of distance-based algorithm, KNN is highly affected by the range of features, since it use distance between data points to determine the similarity (9). This dataset consists of different range of values from various metrics, therefore, before applying KNN it is necessary to preprocess the data through scaling. The dataset would be normalized to scale the values so they end up ranging between 0 and 1, using `preProcess` on `Caret` package. For determination of parameter k, automatic tuning using `tuneLength` would also be carried out. `tuneLength` argument tells caret how many possible values to test.

The best k value which return the lowest RMSE was 5, with obtained evaluation metrics:

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
k-Nearest Neighbor (KNN)	77	36.1	0.432	8.98

KNN model already improved the RMSE and MAE of train set considerably, and was able to explain 43.18% of the variance.

3.5 Neural Network Model

As representative of neural network type, Artificial Neural Network (ANN) model was chosen. It was mentioned that ANNs were the most used algorithms for energy consumption in buildings (3) and therefore, worth to be explored for this project. For ANN, the dataset would be pre-processed by scaling using standardization, and automatic tuning would be carried out.

The best `size` and `decay` for ANN parameters which return the lowest RMSE were 5 and 0 respectively. The obtained evaluation metrics were:

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
Artificial Neural Network (ANN)	92	50.4	0.182	8.98

From result above, ANN model only provided slight improvement from MLR model and were outperformed by previous KNN model. Therefore, neural network type models would not be explored any further.

3.6 Tree-Based Models

3.6.1 Random Forest (RF) Model

Next, tree-based models would be evaluated. RFs are very popular machine learning algorithm that improve prediction performance and reduce instability through averaging multiple decision trees, on a forest of trees constructed with randomness (5). RF is a special type of bagging models (8) in tree-based model, which have concept of random sampling with replacement (10).

Tree-based models generally do not need scaling. However, several hyperparameters in RF would need to be tuned in order to minimize RMSE. For initial exploration, `mtry` (number of features to be considered at any given split) parameter would be tuned manually through expand grid.

The best `mtry` value which return the lowest RMSE was 3, with obtained evaluation metrics:

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
Random Forest (RF)	70.6	33.5	0.527	435

RF was able to improve RMSE 24.26% from MLR model, with was a better performance than KNN model. It was able to explain 52.73% of the variance (22.13% improvement compared to KNN model).

This result indicates the worth of further exploring tree-based models. While it is also possible to further optimize the hyperparameters of RF, the computation time was a challenge. Therefore, different approach of tree-based models would be evaluated.

3.6.2 Gradient Boosting Machine (GBM) Model

GBM models also known as boosting, which try to improve the prediction by utilizing the information from the first trees (2). In boosting method, the trees are grown sequentially, where each successive tree is grown using information from previously grown trees (8). Different with bagging, random sampling in boosting will include replacement over weighted data (10).

GBM also called stochastic gradient boosting, and can improve generalizability of model by utilizing gradient descent to minimize loss function (11). In the benchmark work (2), GBM was chosen as the best model which provided the best RMSE. For this project, GBM model would be applied using `gbm` method in `Caret` package. Determination of its hyperparameters would be carried out through automatic tuning using `tuneLength.`

Best tune for GBM parameters which return the lowest RMSE were:

	n.trees	interaction.depth	shrinkage	n.minobsinnode
100	500	10	0.1	10

With obtained evaluation metrics:

	Train Performance			
Model	RMSE	MAE	R-squared	Train Time (s)
Gradient Boosting Machine (GBM)	76	39.5	0.441	91.4

With auto-tuning, GBM did not perform better than RF model on the test set. Therefore, another tree-based model with boosting approach would be evaluated.

3.6.3 eXtreme Gradient Boosting (XGBoost) Model

XGBoost model is one of boosting approaches which use more accurate approximations to find the best tree model through utilization of second order derivative of loss function and advanced regularization to improve model generalization (11). For this project, XGBoost model would be applied using `xgbtree` method in `Caret` package. XGBoost have several hyperparameters which can be determined by using auto or manual tuning.

3.6.3.1 Auto-tuned XGBoost First, hyperparameters would be determined through automatic tuning using `tuneLength.` Best tune for XGBoost parameters which return the lowest RMSE based using auto-tuning were:

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
250	250	5	0.3	0	0.8	1	1

with the following obtained evaluation metrics:

	Train Performance			
Model	RMSE	MAE	R-squared	Train Time (s)
XGBoost (Auto-tuned)	73.9	37.6	0.472	113

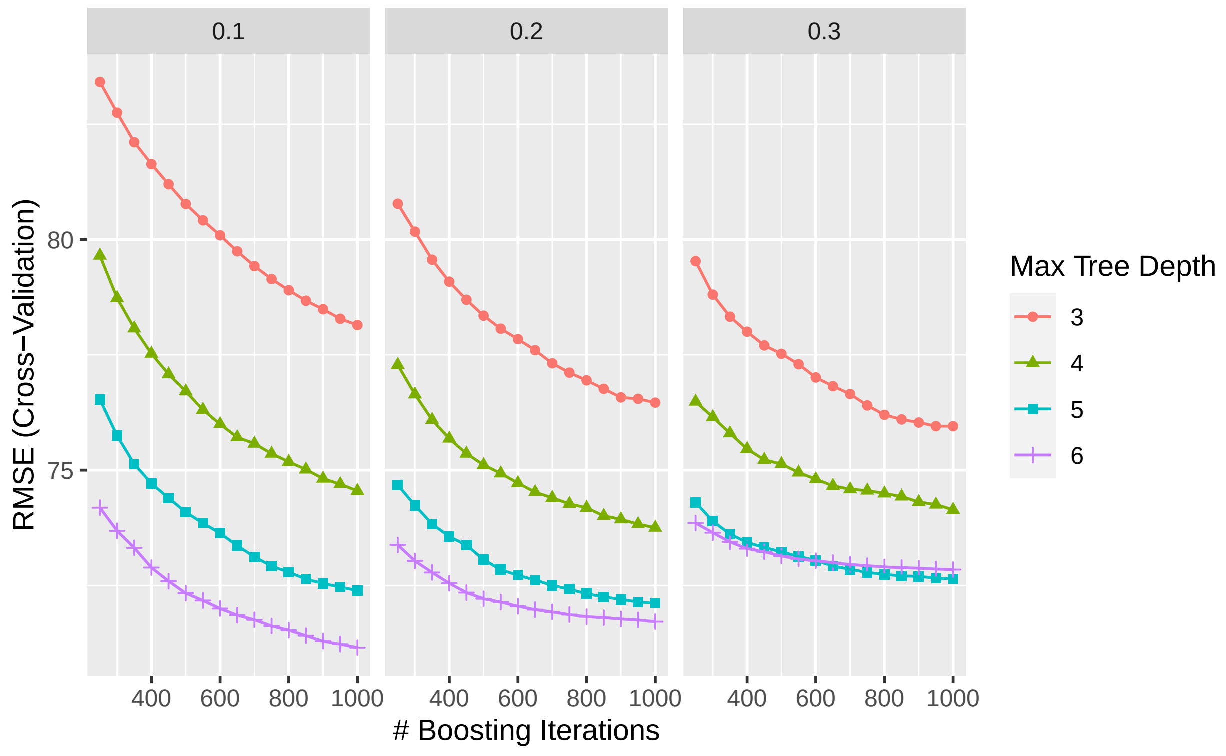
With auto-tuning, XGBoost model did not perform better than RF model but it performed 2.79% better in RMSE than previous GBM model. Since the training time of XGBoost were about 3.8 times faster than RF, it would be worth to further optimize the hyperparameters of this model and evaluate its performance.

3.6.3.2 Optimized XGBoost In this step, hyperparameters of XGBoost would optimized through manual tuning using grid search. This process would be carried out step-by-step (12) based on initial best values of auto-tuned model. Detail values of grid search on each steps would be attached on Appendix.

1. Number of Boosting Iterations, Learning Rate, and Maximum Tree Depth Tuning

This step would serve as starting point:, the number of boosting iterations would be limited to 1000 in order not to overstretch the running time while narrow down possible maximum tree depth and learning rate parameters. Learning rate (`eta`) or shrinkage ranged from 0 to 1 and controls how much information from a new tree will be used in boosting (10) while `max_depth` control the depth of tree, where larger depth indicates more complex model (13).

The following plot visualizes tuning result of this step:



The obtained evaluation metrics were:

	Train Performance			
Model	RMSE	MAE	R-squared	Train Time (s)
XGBoost (Optimization Step 1)	71.1	34.6	0.51	85.6

For `nrounds = 1000`, `max_depth = 6`, and `eta = 0.1`. This step slightly lower the RMSE and improved the R^2 from auto-tuned model.

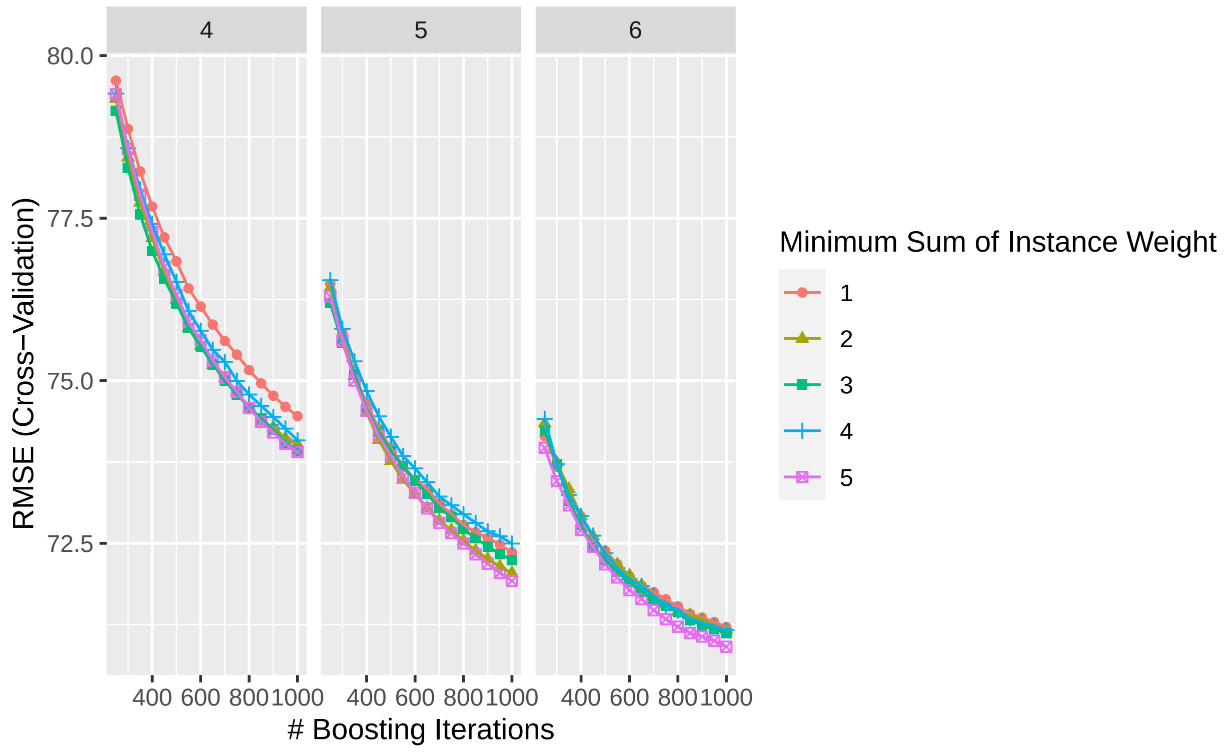
Lower learning rate means smaller piece of information will be used from each new tree (10). From the visualization, it can be concluded that low learning rate works better for this dataset. Therefore, learning rate would be tuned again for wider range of lower value and higher number of iterations after other hyperparameters were optimized.

It was also indicated in `max_depth` that deeper trees may results in lower RMSE. However, trees that are too deep give higher chances of overfitting (13), as the model will use more information from the first trees and final trees will be less important (10). Therefore, instead of further deepening the trees, other parameters would be optimized.

2. Maximum Tree Depth and Minimum Child Weight Tuning

Since it is not desirable to have trees that are too deep, previously obtained `max_depth` would be set as maximum and possible minimum child weight would be narrowed down. Minimum child weight in regression refers to minimum number of instances required in a child node (13).

The following plot visualizes tuning result of this step:



The obtained evaluation metrics were:

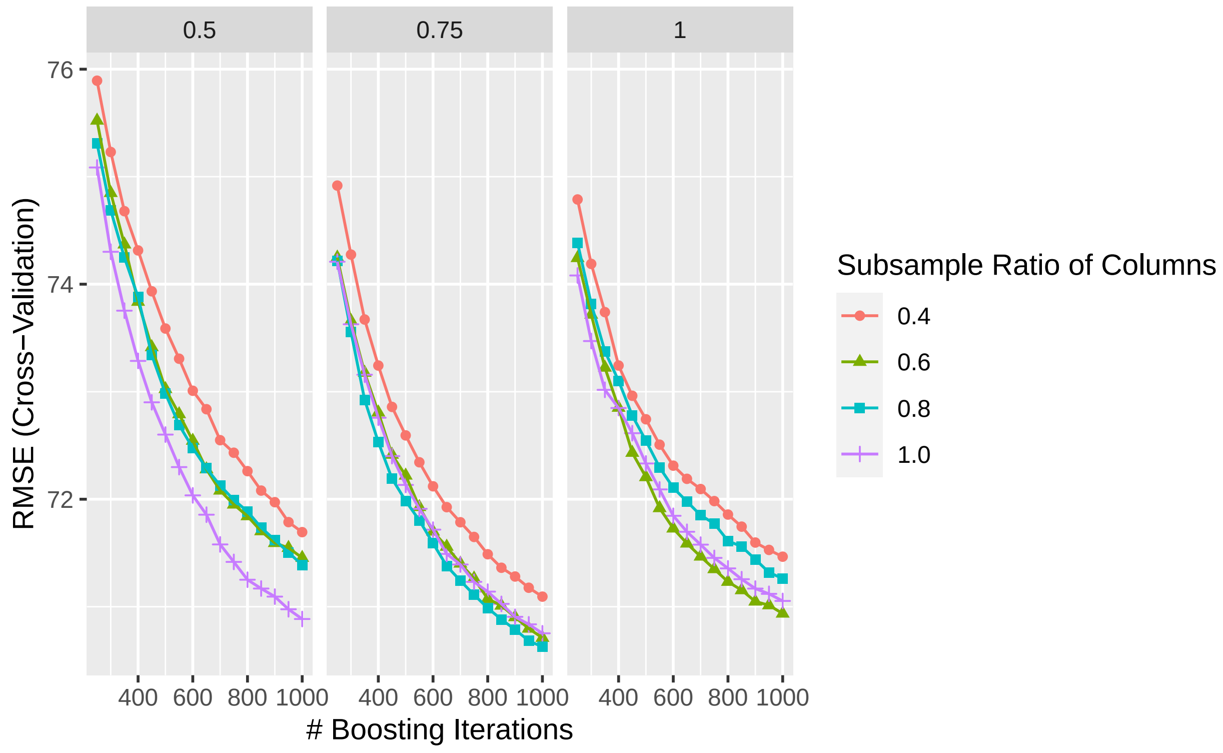
	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
XGBoost (Optimization Step 2)	70.9	35.2	0.513	116

This step slightly lower the RMSE from previous step, for `max_depth = 6`, and `min_child_weight = 5`. It was expected that `min_child_weight` would be larger for deeper trees.

3. Number of Samples and Number of Features Tuning

`subsample` control number of samples or observation, while `colsample_bytree` control number of features or variables supplied to a tree (13). Several possible values ranged from 0 to 1 were explored.

The following plot visualizes tuning result of this step:



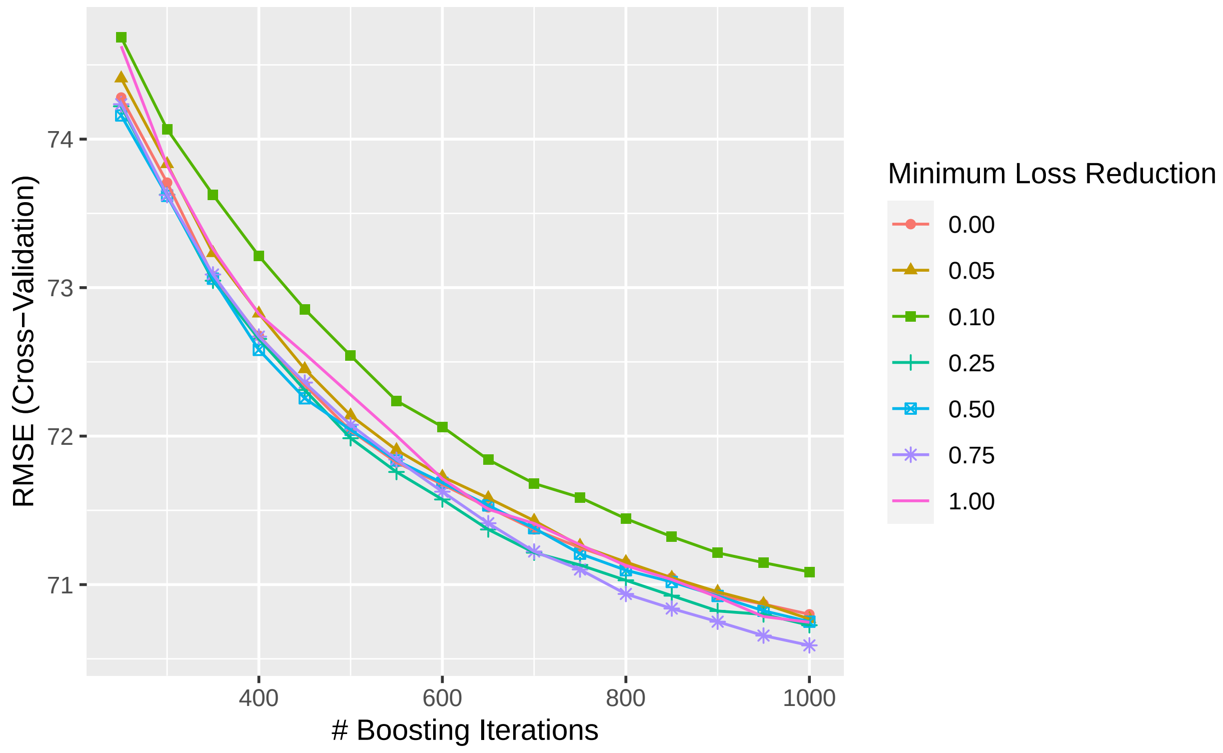
The obtained evaluation metrics were:

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
XGBoost (Optimization Step 3)	70.6	35.3	0.517	108

for `subsample = 0.75` and `colsample_bytree = 0.8`. This step also slightly lower the RMSE from previous step.

4. Regularization Tuning `gamma` controls regularization or prevents overfitting by penalizing larger coefficients that do not improve model's performance (13), with higher value indicating higher regularization. Values of `gamma` ranged from 0 to 1 were explored to see if they could further improve model's performance.

The following plot visualizes tuning result of this step:



The obtained evaluation metrics were:

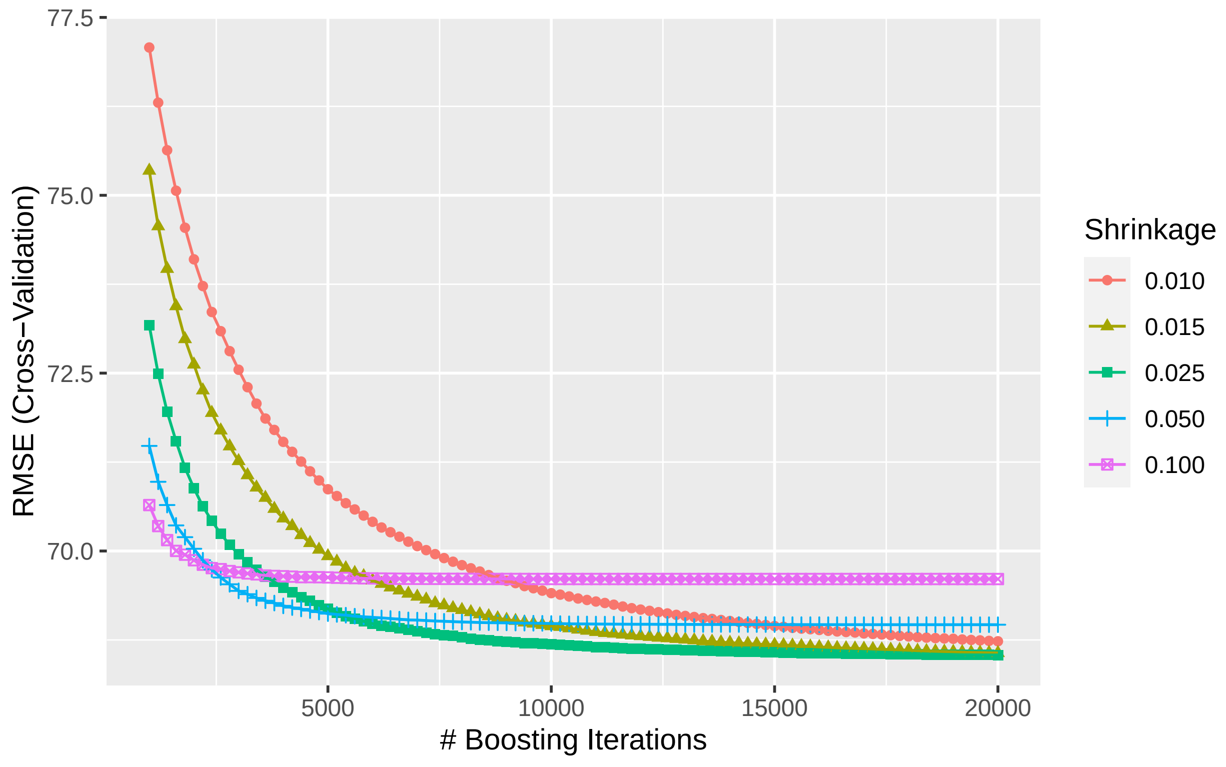
	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
XGBoost (Optimization Step 4)	70.6	35.1	0.518	72.8

for `gamma` =0.75. Changing `gamma` values did not influence the RMSE, and only very slightly improved MAE and R^2.

5. Number of Boosting Iteration and Learning Rate Re-tuning

After finalizing other possible hyperparameters, different values of previously obtained number of boosting iteration and learning rate would be re-tuned. Since it has been indicated that lower learning rate may work better with this dataset, `eta` value ranging from 0.01 to 0.1 would be explored. Lower `eta` leads to slower computation and must be supported by increase in number of boosting iterations (13). For this step, `nrounds` would be maximized up to 20,000 iterations.

The following plot visualizes tuning result of this step:



The obtained evaluation metrics were:

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
XGBoost (Optimization Step 5)	68.5	33.3	0.545	1091

It was indicated that the model has not yet stabilized even after 20,000 iterations. Further exploration up to 30,000 iterations (not printed in this report) did not significantly improve the performance, so the tuning would be finalized for `eta` =0.025 and `nrounds` =0.025.

XGBoost Optimization Summary

Table 2 summarized improvement of XGBoost model on test dataset after manual optimization of the hyperparameters:

Table 2: XGBoost Optimization Summary

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
XGBoost (Auto-tuned)	73.9	37.6	0.472	113.2
XGBoost (Optimization Step 1)	71.1	34.6	0.510	85.6
XGBoost (Optimization Step 2)	70.9	35.2	0.513	116.2
XGBoost (Optimization Step 3)	70.6	35.3	0.517	108.2
XGBoost (Optimization Step 4)	70.6	35.1	0.518	72.8
XGBoost (Optimization Step 5)	68.5	33.3	0.545	1090.7
XGBoost (Optimized)	68.7	33.3	0.544	308.1

Optimized XGBoost was resulting in 7.08% RMSE improvement over auto-tuned model, with the following final hyperparameters:

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
288	20000	6	0.025	0.75	0.8	5	0.75

4 Results & Discussion

Table 3 summarized performance of various types of machine learning algorithms on training set:

Table 3: Performance of Models on Training Set

	Train Performance			
Model	**RMSE**	**MAE**	**R-squared**	**Train Time (s)**
Multiple Linear Regression (MLR)	93.2	52.9	0.158	2.12
k-Nearest Neighbor (KNN)	77.0	36.1	0.432	8.98
Artificial Neural Network (ANN)	92.0	50.4	0.182	8.98
Random Forest (RF)	70.6	33.5	0.527	435.12
Gradient Boosting Machine (GBM)	76.0	39.5	0.441	91.42
XGBoost (Auto-tuned)	73.9	37.6	0.472	113.17
XGBoost (Optimized)	68.7	33.3	0.544	308.07

The best models provide lower RMSE and MAE and highest R^2. Results above indicate tree-based models (RF, GBM, and XGBoosts) gave better performance compared to other type of algorithm models. Among them, bagging approach (RF) gave initial best performance despite the high train time. However, after hyperparameters optimization, boosting approach (Optimized XGBoost) was resulting in 2.78% improvement of RMSE compared to RF with lower training time. Optimized XGBoost model was able to explain the variances 3.17% better, although the MAE value was slightly lower than RF model. Therefore, both optimized XGBoost and RF would be subjected to testing.

The two models would be evaluated on the test set to decide the best model of this project, which would be compared to the benchmark model. The benchmark work reported that the most important variable to predict appliance energy consumption using GBM model was time information (number of seconds from midnight) (2). In contrast, this project was trying to give prediction with only using environment data. The comparison between best models on this project and benchmark model can be summarized on Table 4:

Table 4: Benchmarking Summary

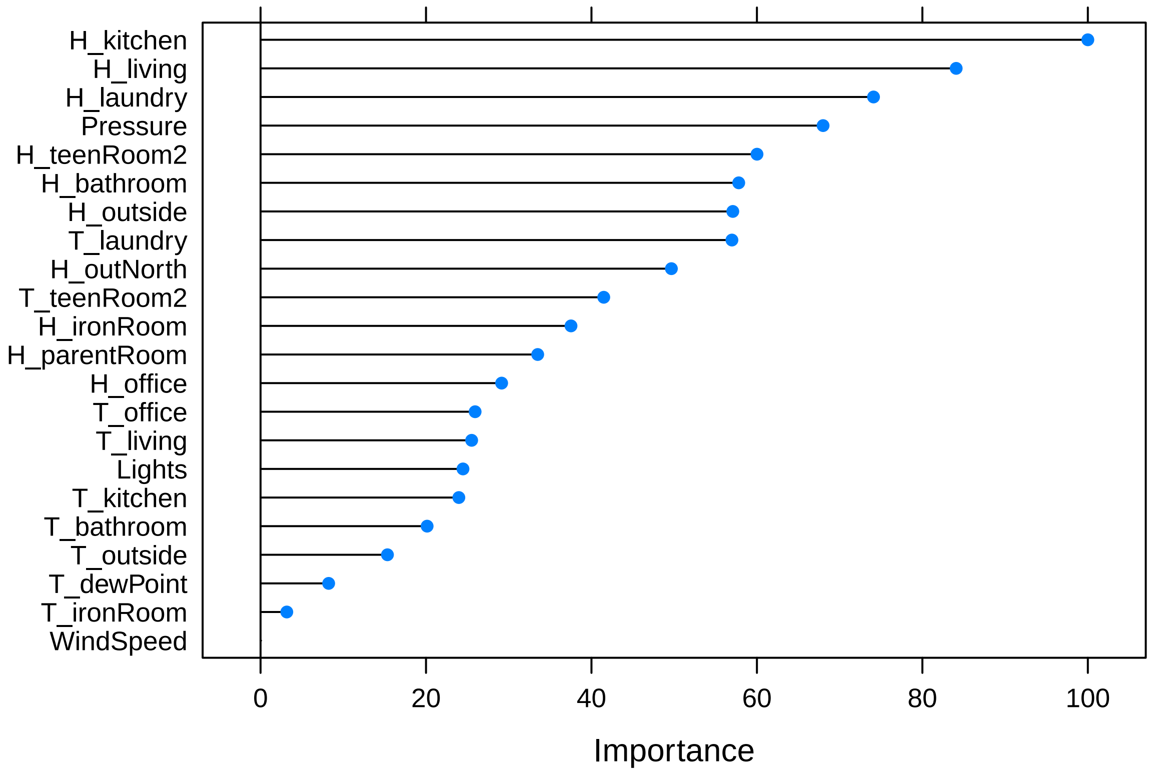
	Train		Test	
Model	**RMSE**	**MAE**	**RMSE**	**MAE**
Benchmark (GBM)	17.6	12.0	66.7	35.2
Random Forest	70.6	33.5	71.3	32.9
Optimized XGBoost	68.7	33.3	66.9	32.2

The result on test set indicates that boosting approach from tree-based models (optimized XGBoost) were the best model for this project. Without considering time information features, optimized XGBoost model was able to perform better than benchmark model by 8.55% improvement of MAE metric and only 0.42% lower on RMSE metric.

It was interesting to note that the benchmark model show rather overfitting trend, indicated by low train RMSE and high test RMSE (14), while best model for this project (optimized XGBoost) show slightly lower RMSE and MAE value on the test set. The objective of learning algorithm is optimal fit, with very minimal generalization gap (gap between train and validation loss learning curves) (15). While it is often expected that train RMSE to be lower than test RMSE, this will not always be true due to the randomness of the split (14). The best model of this project gave only about 2.51% difference on RMSE value between train and test set and 3.32% on MAE (in contrast to respectively 279.56% and 194.24% differences on benchmark model), which may still indicate a good fit.

Alternative explanation of lower loss on validation rather than on the training is that the validation dataset may be relatively easier to predict rather than the training dataset (15). This issue can be solved by rechecking the splitting or performing more cross-validation. However, due to limited computation capacity, this would not be further evaluated on this project.

Next, the following figure shows relative variable importance for the best model (optimized XGBoost):



From plot above, it was clear that the 3 most important features were humidity measurements (from kitchen, living room, and laundry room). Outside environment features (pressure and outside humidity) were also considerably important. This results was consistent with findings reported on benchmark work, where pressure and several humidity measurements inside house were most important after time variable (2). The least important features was wind speed measurement.

5 Conclusion

This project managed to explore the possibility to predict appliance energy consumption with only considering light energy consumption and environment data from inside and outside a low-energy house. Several types machine learning algorithms were evaluated: linear model (MLR), distance-based model (KNN), neural network model (ANN), and tree-based models (RF, GBM, and XGBoost). Tree-based models exhibited better performance on test set. XGBoost after hyperparameters optimization was the best model, which was able to explain 54.41% variances on training set, with 2.51% difference of RMSE value between training and test set, and 8.55% better MAE value than benchmark model.

Result of this project indicated importance of humidity measurement in explaining energy consumption. This result gave potent basis for further exploration to evaluate whether better humidity control inside house can lead to better energy consuming behavior on smart homes and cities.

Limitation of this project include limited computational performance which lead to minimum cross-validation and model tuning, and only cover one scenario of involved features. Repeated cross-validation may be further improve the scenario on this project. It may also worth to explore different scenarios of the features (i.e. comparing the effect of certain features presence). In addition, since tree-based models was indicated to suit this dataset, exploration of other similar models (i.e. Extremely Randomized Trees, which are similar to RF with rather challenging computation time) may also worth to be carried out.

6 References

(1) https://www.edx.org/professional-certificate/harvardx-data-science

(2) Candanedo, L.M., et al. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings* 140 (2017) 81–97. http://dx.doi.org/10.1016/j.enbuild.2017.01.083

(3) Chammas, M. et al. An efficient data model for energy prediction using wireless sensors. *Computers & Electrical Engineering* 76 (2019) 249-257. https://doi.org/10.1016/j.compeleceng.2019.04.002

(4) https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

(5) Irizarry RA. *Introduction to Data Science - Data Analysis and Prediction Algorithms with R.* Available at: https://rafalab.github.io/dsbook/machine-learning-in-practice.html

(6) *Hands-on with Feature Selection Techniques: Filter Methods.* Available at: https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-filter-methods-f248e0436ce5

(7) *Exploring correlations in R with corrr.* Available at: https://drsimonj.svbtle.com/exploring-correlations-in-r-with-corrr

(8) *Statistical Machine Learning Essentials.* Available at: http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/

(9) *Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs Standardization.* Available at: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

(10) *Hyperparameter Tuning with XGBoost.* Available at: http://ml-course.kazsakamoto.com/Labs/hyperparameterTuning.html

(11) *Machine Learning Basics - Gradient Boosting & XGBoost.* Available at: https://www.shirin-glander.de/2018/11/ml_basics_gbm/

(12) *Visual XGBoost Tuning with Caret.* Available at: https://www.kaggle.com/pelkoja/visual-xgboost-tuning-with-caret

(13) *Beginner Tutorials on XGBoost and Parameter Tuning in R.* Available at: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/

(14) Dalpiaz, D. *R for Statistical Learning.* Available at: https://daviddalpiaz.github.io/r4sl/regression-for-statistical-learning.html#choosing-a-model

(15) *Diagnosing Model Performance with Learning Curves.* Available at: https://rstudio-conf-2020.github.io/dl-keras-tf/notebooks/learning-curve-diagnostics.nb.html

7 Appendix

7.1 R Environment

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
## system code page: 932
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] xgboost_1.2.0.1     gbm_2.1.8           randomForest_4.6-14
##  [4] doParallel_1.0.15   iterators_1.0.12    foreach_1.5.0
##  [7] corrr_0.4.2         RColorBrewer_1.1-2  corrplot_0.84
## [10] caret_6.0-86        lattice_0.20-41     lubridate_1.7.9
## [13] kableExtra_1.2.1    scales_1.1.1        forcats_0.5.0
## [16] stringr_1.4.0       dplyr_1.0.2         purrr_0.3.4
## [19] readr_1.3.1         tidyr_1.1.2         tibble_3.0.3
## [22] ggplot2_3.3.2       tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-148         fs_1.5.0             webshot_0.5.2
##  [4] httr_1.4.2           tools_4.0.2          backports_1.1.10
##  [7] R6_2.4.1             rpart_4.1-15         DBI_1.1.0
## [10] colorspace_1.4-1     nnet_7.3-14          withr_2.3.0
## [13] tidyselect_1.1.0     compiler_4.0.2       cli_2.0.2
## [16] rvest_0.3.6          xml2_1.3.2           labeling_0.3
## [19] digest_0.6.25        rmarkdown_2.4        pkgconfig_2.0.3
## [22] htmltools_0.5.0      dbplyr_1.4.4         rlang_0.4.7
## [25] readxl_1.3.1         rstudioapi_0.11      farver_2.0.3
## [28] generics_0.0.2       jsonlite_1.7.1       ModelMetrics_1.2.2.2
## [31] magrittr_1.5         Matrix_1.2-18        Rcpp_1.0.5
## [34] munsell_0.5.0        fansi_0.4.1          lifecycle_0.2.0
## [37] stringi_1.5.3        pROC_1.16.2          yaml_2.2.1
## [40] MASS_7.3-51.6        plyr_1.8.6           recipes_0.1.13
## [43] grid_4.0.2           blob_1.2.1           crayon_1.3.4
## [46] haven_2.3.1          splines_4.0.2        hms_0.5.3
## [49] knitr_1.30          pillar_1.4.6        reshape2_1.4.4
## [52] codetools_0.2-16    stats4_4.0.2        reprex_0.3.0
## [55] glue_1.4.2          evaluate_0.14       data.table_1.13.0
## [58] modelr_0.1.8        vctrs_0.3.4         cellranger_1.1.0
## [61] gtable_0.3.0        assertthat_0.2.1    xfun_0.18
## [64] gower_0.2.2         prodlim_2019.11.13  broom_0.7.1
## [67] class_7.3-17        survival_3.1-12     viridisLite_0.3.0
## [70] timeDate_3043.102   lava_1.6.8          ellipsis_0.3.1
## [73] ipred_0.9-9
```

7.2 Grid Search Values for XGBoost Optimization

```
# ------------------------------------------------------------------------------
# Tune Grid for XGBoost Optimization Step 1
# ------------------------------------------------------------------------------
tune_xgb1 <- expand.grid(
  nrounds = seq(from = 250, to = 1000, by = 50),
  max_depth = c(3, 4, 5, 6),
  eta = c(0.1, 0.2, 0.3),
  gamma = 0,
  colsample_bytree = 0.8,
  min_child_weight = 1,
  subsample = 1)

# ------------------------------------------------------------------------------
#Tune Grid for XGBoost Optimization Step 2
# ------------------------------------------------------------------------------
tune_xgb2 <- expand.grid(
  nrounds = seq(from = 250, to = 1000, by = 50),
  max_depth = c(train_xgb1$bestTune$max_depth - 2,
                train_xgb1$bestTune$max_depth - 1,
                train_xgb1$bestTune$max_depth),
  eta = train_xgb1$bestTune$eta,
  gamma = 0,
  colsample_bytree = 0.8,
  min_child_weight = c(1, 2, 3, 4, 5),
  subsample = 1)

# ------------------------------------------------------------------------------
# Tune Grid for XGBoost Optimization Step 3
# ------------------------------------------------------------------------------
tune_xgb3 <- expand.grid(
  nrounds = seq(from = 250, to = 1000, by = 50),
  max_depth = train_xgb2$bestTune$max_depth,
  eta = train_xgb1$bestTune$eta,
  gamma = 0,
  colsample_bytree = c(0.4, 0.6, 0.8, 1.0),
  min_child_weight = train_xgb2$bestTune$min_child_weight,
  subsample = c(0.5, 0.75, 1.0))

# ------------------------------------------------------------------------------
# Tune Grid for XGBoost Optimization Step 4
# ------------------------------------------------------------------------------
tune_xgb4 <- expand.grid(
```

```r
  nrounds = seq(from = 250, to = 1000, by = 50),
  max_depth = train_xgb2$bestTune$max_depth,
  eta = train_xgb1$bestTune$eta,
  gamma = c(0, 0.05, 0.1, 0.25, 0.5, 0.75, 1.0),
  colsample_bytree = train_xgb3$bestTune$colsample_bytree,
  min_child_weight = train_xgb2$bestTune$min_child_weight,
  subsample = train_xgb3$bestTune$subsample)

# ---------------------------------------------------------------------------------
# Tune Grid for XGBoost Optimization Step 5
# ---------------------------------------------------------------------------------
tune_xgb5 <- expand.grid(
  nrounds = seq(from = 1000, to = 20000, by = 200),
  max_depth = train_xgb2$bestTune$max_depth,
  eta = c(0.01, 0.015, 0.025, 0.05, 0.1),
  gamma = train_xgb4$bestTune$gamma,
  colsample_bytree = train_xgb3$bestTune$colsample_bytree,
  min_child_weight = train_xgb2$bestTune$min_child_weight,
  subsample = train_xgb3$bestTune$subsample)

# ---------------------------------------------------------------------------------
# Final Tune Grid Value for Optimized XGBoost
# (Note: this will be directly used in accompanying R file of this report)
# ---------------------------------------------------------------------------------
tune_xgbfinal <- expand.grid(
  nrounds = 20000,
  max_depth = 6,
  eta = 0.025,
  gamma = 0.75,
  colsample_bytree = 0.8,
  min_child_weight = 5,
  subsample = 0.75)
```