

# **MACHINE LEARNING ENGINEER NANODEGREE**

## **CAPSTONE PROJECT**

### **TITLE: APPLIANCES ENERGY PREDICTION**

**Alhasan Gamal Mahmoud**

#### **DEFINITION**

##### **Project Overview**

The goal is to predict the electricity usage of heating and cooling appliances in a household based on internal and external temperatures and other weather conditions. With increase in the number of smart homes these days, accurate energy prediction can really help cutting down energy costs by optimizing the state of appliances. This is a case of regression analysis. Appliance energy usage is target data and inputs from various sensors are the features.

##### **Dataset:**

<http://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction>

Each observation measures electricity in a 10-minute interval. The temperatures and humidity

have been averaged for 10-minute intervals.

##### **Related research and previous work:**

<https://www.sciencedirect.com/science/article/pii/S0378778816308970?via%3Dihub>

**(Research Paper)**

<https://github.com/LuisM78/Appliances-energy-prediction-data>

**(Previous work by the author)**

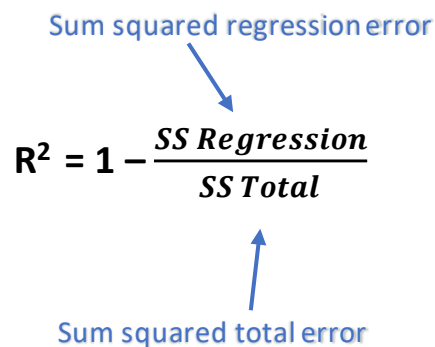
## Problem Statement

Predict the electricity usage of heating and cooling appliances in a household based on internal and external temperatures and other weather conditions. Develop a supervised learning model using regression algorithms to fit the data. Regression is preferred over classification as the target variable is continuous and we have to predict the output value as opposed to classifying the output value. Also, there are no definitive classes of energy consumption in the dataset.

## Metrics

Since this is a regression problem, the metric used will be “Coefficient of Determination”, in other words denoted as (R squared) which gives a measure of the variance of target R<sup>2</sup> variable that can be explained using the given features.

It can be mathematically defined as:



Sum squared regression error

$$R^2 = 1 - \frac{SS \text{ Regression}}{SS \text{ Total}}$$

Sum squared total error

For this project, I will use 'r2\_score()' function of the metrics module of scikit learn library.

While “Coefficient of Determination” provides relative a measure of the how well the model fits the data, the RMSE (Root Mean Squared Error) gives absolute measure of how well model fits the data i.e. how close are the predicted values to the actual values.

Mathematically, RMSE can be defined as:

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

where,

$n$  = number of observations

$y_j$  = Actual value of target variable

$y_j'$  = Predicted value of target variable

In this project, I will calculate RMSE by calculating square root of `mean_squared_error()` function provided in the metrics module of scikit-learn library. Therefore, the metrics to be used are:

- i. R2 score
- ii. RMSE

These two metrics are helpful for this problem because of the following reasons:

- i. It is a Regression based problem.
- ii. R2 score will show the statistical robustness of the model.
- iii. RMSE will give an idea about how accurate the predictions are to actual values

## **ANALYSIS**

### **Data Exploration**

The dataset was collected by sensors placed inside the house and outside readings came from the nearby weather station. The main types of attributes are temperature readings, humidity and pressure. Each observation measures electricity in a 10-minute interval. The temperatures and humidity have been averaged for 10-minute intervals.

**Number of input attributes: 28 (11 temperature, 10 humidity, 1 pressure, 2 randoms, etc.)**

**Target variable: 1 (Appliances)**

#### **Attribute Information:**

- date: time year-month-day hour: minute: second
- lights: energy use of light fixtures in the house in Wh
- T1: Temperature in kitchen area, in Celsius
- T2: Temperature in living room area, in Celsius
- T3: Temperature in laundry room area
- T4: Temperature in office room, in Celsius

- T5: Temperature in bathroom, in Celsius
- T6: Temperature outside the building (north side), in Celsius
- T7: Temperature in ironing room, in Celsius
- T8: Temperature in teenager room 2, in Celsius
- T9: Temperature in parents' room, in Celsius
- T\_out: Temperature outside (from Chievres weather station), in Celsius
- Tdewpoint: (from Chievres weather station),  $\hat{A}^{\circ}\text{C}$
- RH\_1: Humidity in kitchen area, in %

- RH\_2: Humidity in living room area, in %
- RH\_3: Humidity in laundry room area, in %
- RH\_4: Humidity in office room, in %
- RH\_5: Humidity in bathroom, in %
- RH\_6: Humidity outside the building (north side), in %
- RH\_7: Humidity in ironing room, in %
- RH\_8: Humidity in teenager room 2, in %
- RH\_9: Humidity in parents' room, in %
- RH\_out: Humidity outside (from Chievres weather station), in %
- Pressure: (from Chievres weather station), in mm Hg
- Wind speed: (from Chievres weather station), in m/s
- Visibility : (from Chievres weather station), in km
- Rv1: Random variable 1, non-dimensional
- Rv2: Random variable 2, non-dimensional

### Target Variable:

- Appliances: Total energy used by appliances, in Wh

### Sample of data:

```
data.head()
```

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_mm_hg	RH_out
0	2016-01-11 17:00:00	60	30	19.89	47.598667	19.2	44.790000	19.79	44.730000	19.000000	...	17.033333	45.53	6.600000	733.5	92.0
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	17.068667	45.56	6.483333	733.6	92.0
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	17.000000	45.50	6.366667	733.7	92.0
3	2016-01-11 17:30:00	50	40	19.89	46.068667	19.2	44.590000	19.79	45.000000	18.890000	...	17.000000	45.40	6.250000	733.8	92.0
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	...	17.000000	45.40	6.133333	733.9	92.0

5 rows × 29 columns

**I didn't use the following features as they were not relevant to the problem.**

- **Date:** As the problem is regression not time-series, date of the record does not matter.
- **Lights:** The goal is to predict overall energy use and not category-wise.

**Other variables like random variable 1&2, Visibility, Windspeed seem to be irrelevant but can't be discarded without some analysis.**

**Therefore, the number of features became 26.**

**I chose a 75-25 split for the train-test data out of the complete data.**

**Rows in train data - 14801**

**Rows in test data - 4934**

**Total - 19735**

**The size of the dataset seems to be enough to model a good enough solution.**

**All the features have numerical values. There are no categorical or ordinal features.**

**The numerical values are of floating type, which seems pretty obvious as they are temperature and pressure readings.**

**Count of NULL values = 0**

**Count of MISSING values = 0**

**Moving on to the attribute analysis.**

## Train data description

### I. Ranges of all variables

```
train.describe()
```

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3
count	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
mean	97.875144	3.853118	21.691343	40.267556	20.344518	40.434363	22.278802	39.243995
std	102.314986	7.962567	1.615790	3.974692	2.202481	4.052420	2.012934	3.245701
min	10.000000	0.000000	16.790000	27.023333	16.100000	20.596667	17.200000	28.766667
25%	50.000000	0.000000	20.760000	37.363333	18.790000	37.900000	20.790000	36.900000
50%	60.000000	0.000000	21.600000	39.693333	20.000000	40.500000	22.100000	38.560000
75%	100.000000	0.000000	22.633333	43.066667	21.500000	43.273453	23.340000	41.730000
max	1080.000000	60.000000	26.260000	63.360000	29.856667	54.766667	29.236000	50.163333

T4	RH_4	T5	RH_5	T6	RH_6	T7	RH_7
14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
20.860393	39.043799	19.604773	51.014065	7.923216	54.615000	20.273236	35.410874
2.048076	4.333479	1.849641	9.107390	6.117495	31.160835	2.118416	5.097243
15.100000	27.660000	15.340000	29.815000	-6.065000	1.000000	15.390000	23.260000
19.533333	35.560000	18.290000	45.433333	3.626667	29.996667	18.700000	31.500000
20.666667	38.433333	19.390000	49.096000	7.300000	55.267500	20.075000	34.900000
22.100000	42.200000	20.653889	53.773333	11.226667	83.226667	21.600000	39.000000
26.200000	51.090000	25.795000	96.321667	28.290000	99.900000	26.000000	51.327778

T8	RH_8	T9	RH_9	T_out	Press_mm_hg	RH_out	Windspeed
14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
22.028122	42.948244	19.493479	41.556594	7.421836	755.480135	79.824197	4.029001
1.960985	5.210450	2.022560	4.161295	5.343737	7.389218	14.901776	2.448171
16.306667	29.600000	14.890000	29.166667	-5.000000	729.300000	24.000000	0.000000
20.790000	39.096667	18.000000	38.530000	3.666667	750.900000	70.500000	2.000000
22.111111	42.390000	19.390000	40.900000	6.933333	756.000000	83.833333	3.666667
23.390000	46.500000	20.600000	44.326667	10.433333	760.833333	91.666667	5.500000
27.230000	58.780000	24.500000	53.326667	26.100000	772.300000	100.000000	14.000000

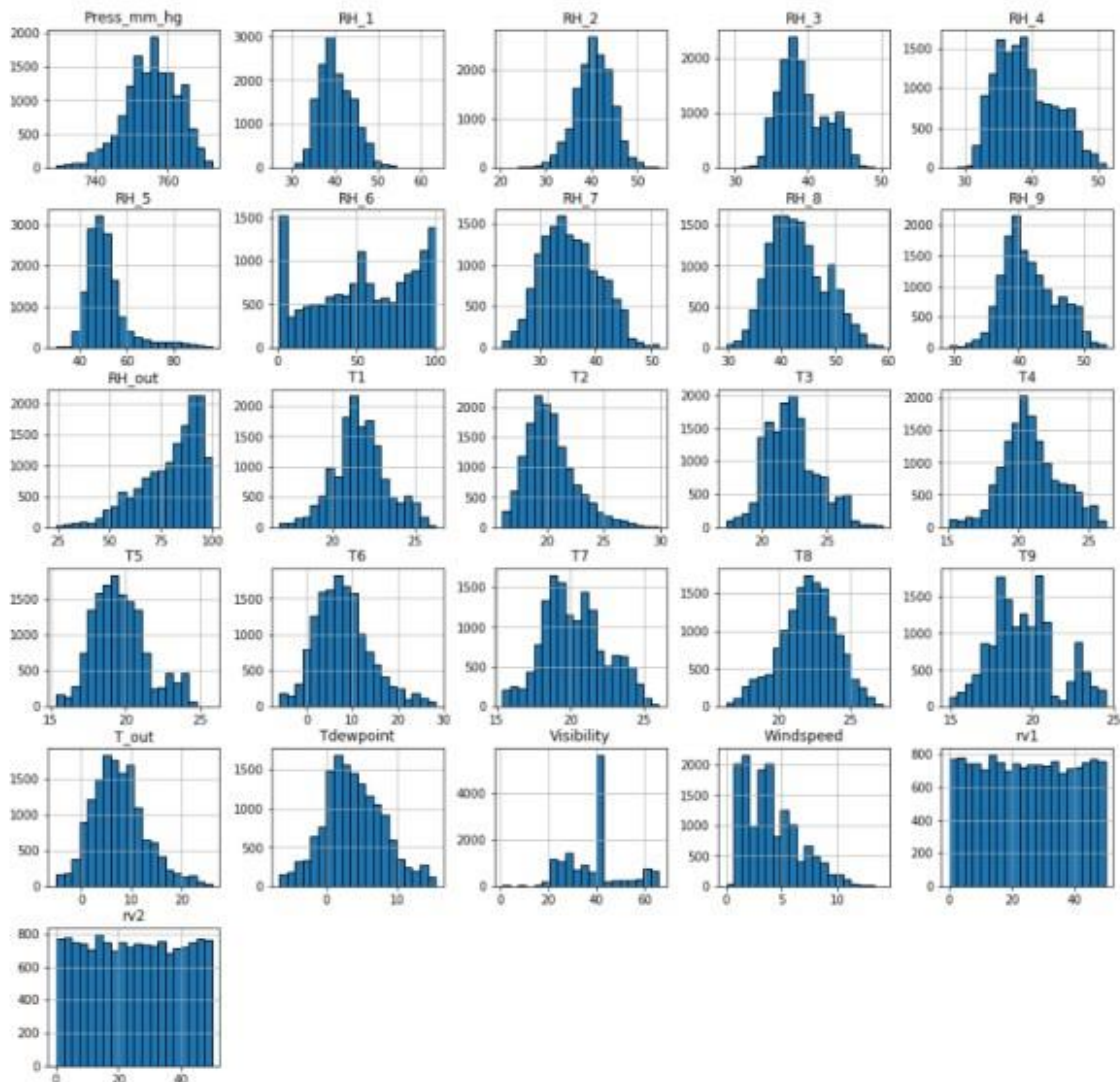
Visibility	Tdewpoint	rv1	rv2
14801.000000	14801.000000	14801.000000	14801.000000
38.290284	3.782509	24.893132	24.893132
11.789650	4.194994	14.539772	14.539772
1.000000	-6.600000	0.006033	0.006033
29.000000	0.933333	12.352580	12.352580
40.000000	3.483333	24.878409	24.878409
40.000000	6.600000	37.534894	37.534894
66.000000	15.316667	49.996530	49.996530

## Ranges of features irrespective of units

1. Temperature: - 6 to 30
2. Humidity: 1 to 100
3. Windspeed: 0 to 14
4. Visibility: 1 to 66
5. Pressure: 729 to 772
6. Appliance Energy Usage: 10 to 1080

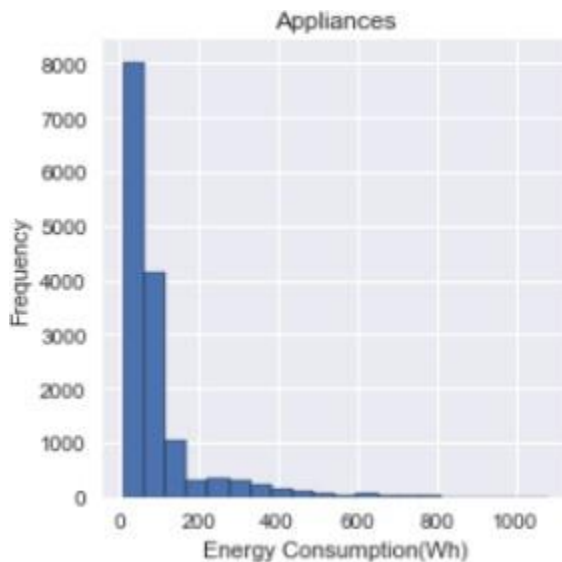
## II. Distribution of Input Attributes

```
hists = input_vars.hist(figsize=(16, 16), bins=20,edgecolor='black')
```





### III. Output Variable distribution



It can be observed from Histograms that: -

- All humidity values except RH\_6 and RH\_out follow a Normal distribution, i.e., all the readings from sensors inside the home are from a Normal distribution.
- Similarly, all temperature readings follow a Normal distribution except for T9.
- Out of the remaining columns, we can see that Visibility, Windspeed and Appliances are skewed.
- The random variables rv1 and rv2 have more or less the same values for all the recordings.
- The output variable Appliances has most values less than 200Wh, showing that high energy consumption cases are very low.

No column has a distribution like the target variable Appliances.

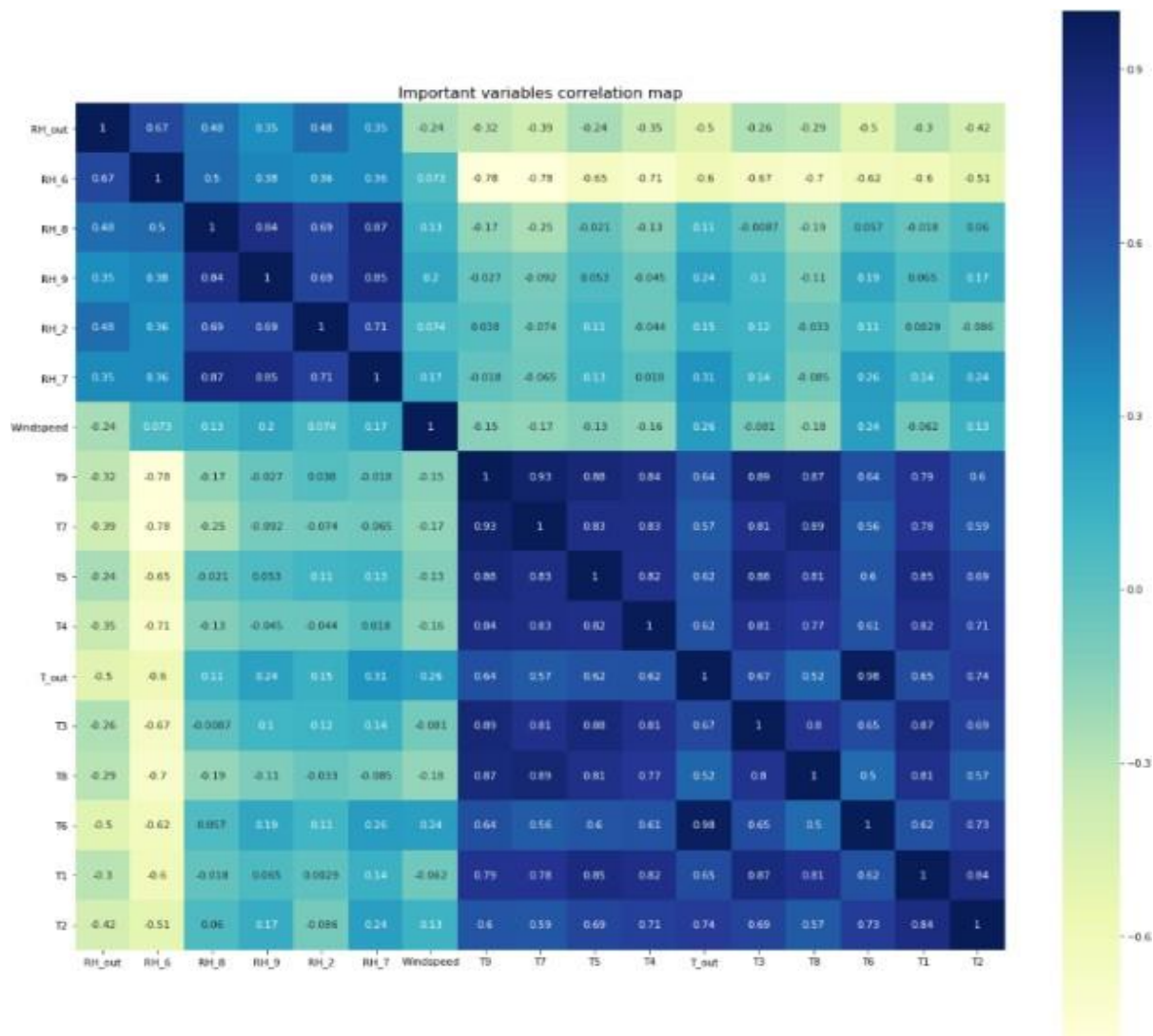
Therefore, we can safely assume that no feature independently has a linear relationship with the target.

## Exploratory Visualization

### Co-relation plot

- The random variables rv1, rv2 and Visibility, Tdewpoint, Press\_mm\_hg has low correlation with the target variable.
- All the temperature variables from T1-T9 and T\_out have high correlation with the target.

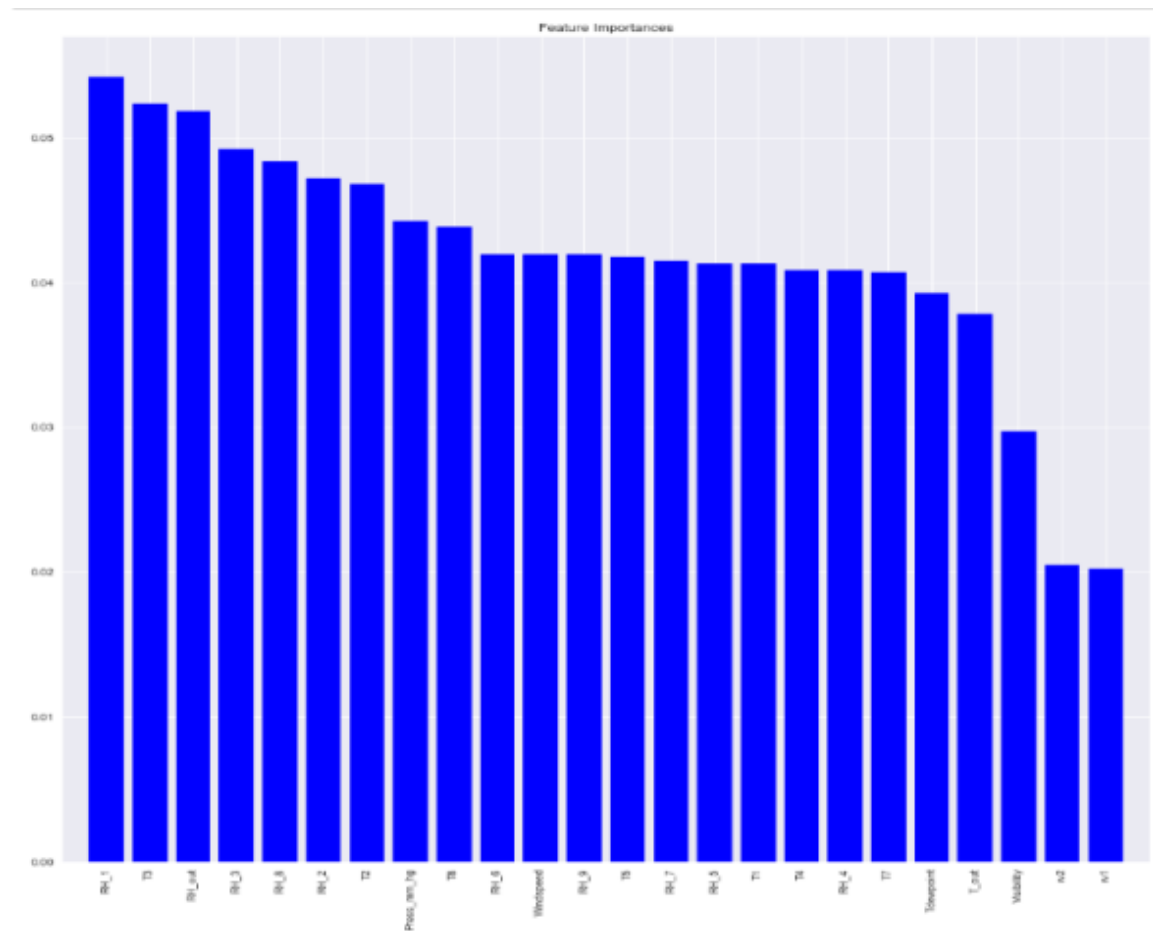
The variables which had a spearman correlation coefficient  $< 0.1$  with the target variable Appliances, will have to be removed as they have little effect on the target. Below, I have plotted inter-variable correlation map for the variables with spearman correlation coefficient  $\geq 0.1$ .



Some inferences drawn from the above plot are:

- The inter-attribute correlation is very high(>0.9) b/w T6 and T\_out
- A number of variables have high correlation with T9 (T3,T5,T7,T8), so it is clear that T9 is redundant
- T6 and T9 need to be removed

Although I've got the important variables based on correlation, it's better to get a second opinion using a Tree based regressor (Extra Randomized Trees in this case).



From this, it becomes clear that rv1, rv2 and Visibility are the least important of the lot. But Press\_mm\_hg, which had a low correlation, turns out more important than others. So, I dropped rv1, rv2, Visibility, T6, T9.

## **Data after Feature Engineering:**

Number of Input Variables - 21 (reduced from 26)

## **Algorithms and Techniques**

The problem is Regression based, but there are a lot of ways to apply regression. I will try the

following algorithms and find out which works best.

### **● Linear Models:**

The most basic Regression algorithm is Linear Regression. If a Linear model can explain the data well, there is no need for further complexity. This model is usually the first choice when solving a regression problem.

Modifying the original least squares regression, Regularization techniques which penalize the coefficient values of the features, since higher values lead to overfitting, can be applied. If the basic linear model doesn't perform well, these techniques help increase performance.

With Regularization, Linear Regression is transformed into Lasso or Ridge Regression depending on how we modify our loss function. If we add absolute values of coefficients, we get Lasso, and if we add their squares, we get Ridge Regression.

### **● Tree-based Regression:**

Tree based models are less affected by outliers as compared to Linear models. We saw earlier that there isn't a linear relation between any input and the target variable, so it is likely that Trees will work better than Linear models.

A simple Decision Tree will surely overfit here owing to the number of features (21). So, I used ensemble methods, which work in 2 ways:

1. Bagging: In this method multiple regressors are built on copies of the same training data and the output is combined through either

mean, median or mode. Ex. Random Forest and Extreme Trees Regression. Random Forest works well on high dimensional data, so it is a good fit for this dataset as it has a large number of features (21). Extreme trees goes further by making splits random.

2. Boosting : In this method each tree is built sequentially, new tree being built from data of the previous tree. Using weighted average of these weak learners (Weak because accuracy is just above 50%) gives the final answer. Ex. Gradient Boosting Machine, which builds the model in such a way that performance always increases.

- **Neural networks** work great when there is a complex nonlinear relationship between the inputs and the output. They have higher performance in most cases but take a long time to train. I have used a Multi-Layer Perceptron as present in the MLPRegressor of scikit-learn.
- The ones which I didn't use, like SVM Regression, Nearest Neighbor Regression, I didn't use because I learnt about them after the review. I always thought they were used for classification only. The ones which I did use, are the ones which I have come across before while dabbling with kernels on Kaggle.
- Other algorithms like XGBoost, lightgbm etc., I tried initially but none of them gave satisfactory results even after tuning them. Either I didn't tune them right, or I was doing something wrong, or they just aren't the right tools for the job.

## **Benchmark**

The benchmark is the R2 score of the Gradient Boosting technique used by the author in his original research paper. The author obtained a R2 score of 0.58 on the test dataset, which I try to beat.

## **METHODOLOGY**

### **Data Preprocessing**

The variables present in the data have varying ranges, with some having very low ranges like Windspeed (0 to 14), while others like Pressure have high range (729-772). So, the variables need to be scaled else some features may dominate the result. I scaled all the input variables to mean 0 and 1 variance using StandardScaler in scikit-learn's preprocessing module.

Also, I had removed certain variables based on importance and redundancy as explained above.

As a result, the data has 21 input attributes.

### **Implementation**

I used the following functions from scikit-learn to test each regression model:

- `sklearn.linear_model.Ridge`
- `sklearn.linear_model.Lasso`
- `sklearn.ensemble.RandomForestRegressor`
- `sklearn.ensemble.GradientBoostingRegressor`
- `sklearn.ensemble.ExtraTreesRegressor`
- `sklearn.neural_network.MLPRegressor`

Pipeline:

- I stored all the regressor names in a list, then iterated over the list, picking up one regressor at a time.
- The regressor's `random_state` was initialized with a seed so that the results are the same every time. Rest all parameters were default.
- Then the regressor was made to fit on the data which had been already divided into two parts, i.e., input variables and output.

- The properties of the regressor , Name, score on training set and score on testing set werestored in a dictionary variable as key-value pairs.
- The dictionary was appended to a global list of all dictionaries.
- Then the dictionary was converted to a Data Frame.

### Result:

	Training scores	Testing scores
Ridge	0.136963	0.123219
Lasso	0.000000	0.000000
RandomForestRegressor	0.912640	0.467552
GradientBoostingRegressor	0.332238	0.241178
ExtraTreesRegressor	1.000000	0.552309
MLPRegressor	0.329835	0.269321

Extra trees Regressor outperforms every other regressor. The training score of 1 may lead to initial doubt that it is overfitting, but it scores the best on the testing set.

### Refinement

Extra Trees Regressor performed the best. But it was using default parameters. So did a grid search cross validation using the GridSearchCV function of the sklearn.model\_selection library.

The parameters which I tuned were :

- n\_estimators: The number of trees to be used.
- max\_features: The number of features to be considered at each split.
- max\_depth: The maximum depth of the tree.

The parameter grid looked like this:

```
param_grid = {
    "n_estimators": [10, 50, 100, 200, 250],
    "max_features": ["auto", "sqrt", "log2"],
    "max_depth": [None, 10, 50, 100, 200, 500]
}
```

n\_estimators: The number of trees in the forest.

max\_features: The number of features to consider when looking for the best split:

- If “auto”, then  $\text{max\_features} = \text{n\_features}$ .
- If “sqrt”, then  $\text{max\_features} = \text{sqrt}(\text{n\_features})$ .
- If “log2”, then  $\text{max\_features} = \text{log2}(\text{n\_features})$ .

max\_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

Before tuning, R2 score was 55.2%. After tuning, it rose to 60.8%, a gain in performance of 5.6%.

#### **Regarding the challenges faced during the journey:**

- Feature Scaling is a must when using regression. Initially, I trained models without scaling, and the results I got were nowhere near where they are now. Then, after reading some blogs on regression and learning how to scale data, I was able to get satisfactory results.
- Seed helps in reproducing results. When I began, I was training each model separately, and every time I ran the model, it gave different results. So, I was thinking of running each model several times, and outputting the average score as the final answer. Then a friend told me the concept of seed. I had seen it used earlier, but never figured out what it was used for.
- Dummy variables help avoid silly overwriting and save a lot of time. Many a times I've had to restart my Kernel and run cells again to get the original values back. I wasn't using dummy variables because I have a habit of using as little memory as possible. But after I started using dummy variables, and thus preferring time over space, it has made coding a little easier.
- Inter-correlation between input attributes is a must to remove as some redundant features with high correlation get removed.



- Ipython Notebooks make life easier as compared to writing code in python and running as whole.
- I was using RandomizedSearch CV earlier. But a little bit of reading up led to the conclusion that GridSearchCV has much better performance, although it takes more time.
- Grid Search tuning can be made much faster by utilising all CPU's. Just set `n_jobs = -1`. This point isn't mentioned in the documentation. I was setting `n_jobs` to a positive value everytime, but it started giving errors. Then while resolving those errors, I came across this "hack". Then training was much faster and no errors.

## RESULTS

### a. Model Evaluation and Validation

Features of the untuned model:

- `n_estimators = 10`
- `max_features = n_features = 22`
- `max_depth = None`

Features of best model after hyper parameter tuning:

- `n_estimators = 250`
- `max_features = log2(n_features) = log2(22) ~ 4`
- `max_depth = None`

Robustness check:

The best model is trained on reduced feature space having only 5 highest ranked features in terms of importance instead of 22 features.

R2 score on test data = 0.499

R2 score of untuned model = 0.558.

Difference = 0.059 or 5.9%.

RMSE on test data = 0.708

RMSE of untuned model = 0.665

Difference = 0.343

Therefore, we can see that even though the feature space is reduced drastically (by more than 75%), the relative loss in performance on test data is less.

b. Justification

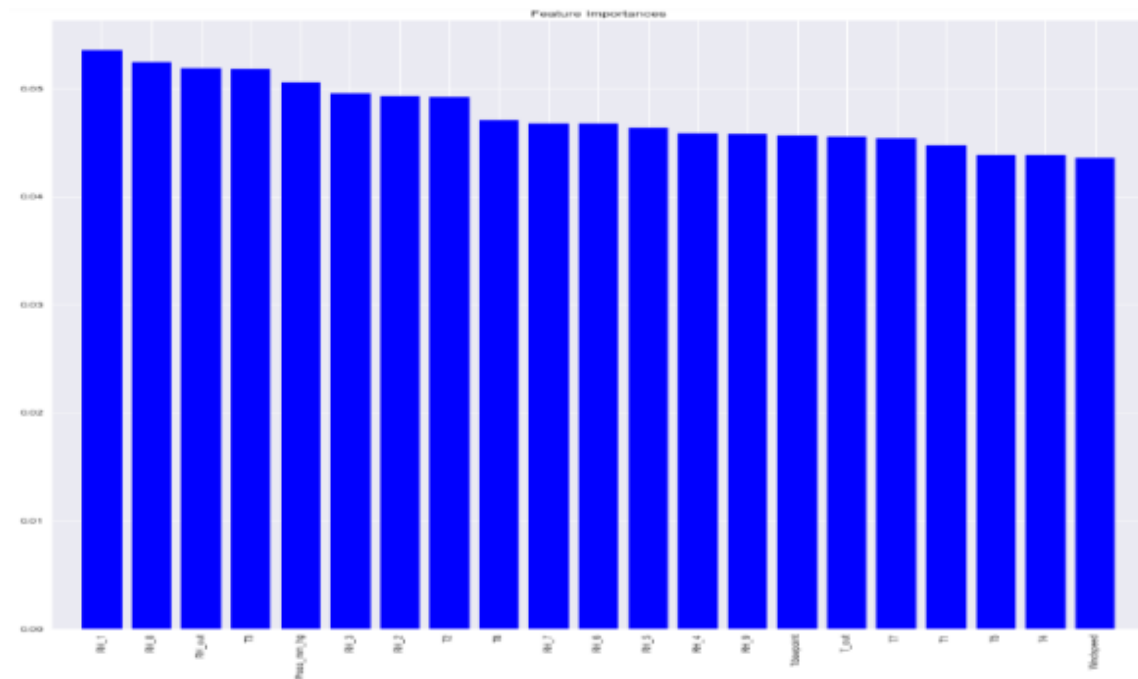
Parameters/Models	Final Model	Benchmark Model	Difference
Training R2 Score	1.0	0.174	0.853
Testing R2 Score	0.61	0.142	0.468
RMSE on test Data	0.624	0.926	0.302

Based on the improvements recorded above, the final tuned model can be deemed as a satisfactory solution.

## **CONCLUSION**

### **Free form visualization**

The best model yields the following feature importance graph:



Thus, the most important feature is RH\_1 and the least important is Windspeed.

The top 3 important features are humidity attributes, which leads to the conclusion that humidity affects power consumption more than temperature.

Windspeed is least important as the speed of wind doesn't affect power consumption inside the house.

So, controlling humidity inside the house may lead to energy savings.

### **Reflection**

The whole project experience can be summarized as follows:

- The problem on which I didn't perform well during the coding challenge led me to explore it further.
- Found the dataset on the UCI ML repository and the author's work.

- Decided to work on it and improve upon the author's work.
- Visualized the data, did preprocessing by learning from other regression contests from Kaggle.
- Preprocessing was an integral part of this project, finding out inter-variable correlation and removing it led to much better results.
- Using a seed to reproduce the results was an important discovery.
- Applying selected algorithms and tuning the best performing model.
- Using GridSearchCV instead of RandomizedSearchCV as it yields better results.
- Comparing my tuned model against the author's result which I used as the benchmark for this project.

Interesting part was learning new preprocessing techniques and visualization techniques.

Difficult was figuring out to tune the model as I had never done it before.

Improvement

- Performing better feature engineering.
- Modifying parameters of the Grid Search parameter space.
  - max\_features parameter can have more set of values like int, float and None
  - max\_depth can have more values
  - adding more parameters like min\_samples\_split, min\_impurity\_decrease etc.

## **REFERENCES:**

- [https://scikit-learn.org/stable/supervised\\_learning.html#supervised-learning](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)
- <http://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction>
- <https://linkinghub.elsevier.com/retrieve/pii/S0378778816308970>
- <https://github.com/LuisM78/Appliances-energy-prediction-data>