MACHINE LEARNING ENGINEER NANODEGREE
CAPSTONE PROJECT
TITLE : APPLIANCES ENERGY PREDICTION

AKSHIT JINDAL

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. 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:
http://dx.doi.org/10.1016/j.enbuild.2017.01.083
(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.

Metrics

The metric used is "Coefficient of Determination" also denoted as R2-score. It is commonly used to measure accuracy of regression problems.Mathematically,

$$R^2 = 1 - \frac{\overset{\text{Sum Squared Regression Error}}{SS_{Regression}}}{\underset{\text{Sum Squared Total Error}}{SS_{Total}}}$$

I have used the r2_score() function in metrics module of the scikit-learn library.

ANALYSIS

Data Exploration

Number of attributes : 29
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
- 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 %
- T_out, Temperature outside (from Chievres weather station), in Celsius
- Pressure (from Chievres weather station), in mm Hg
- RH_out, Humidity outside (from Chievres weather station), in %
- Wind speed (from Chievres weather station), in m/s
- Visibility (from Chievres weather station), in km
- Tdewpoint (from Chievres weather station), Â°C
- rv1, Random variable 1, non-dimensional
- rv2, Random variable 2, non-dimensional

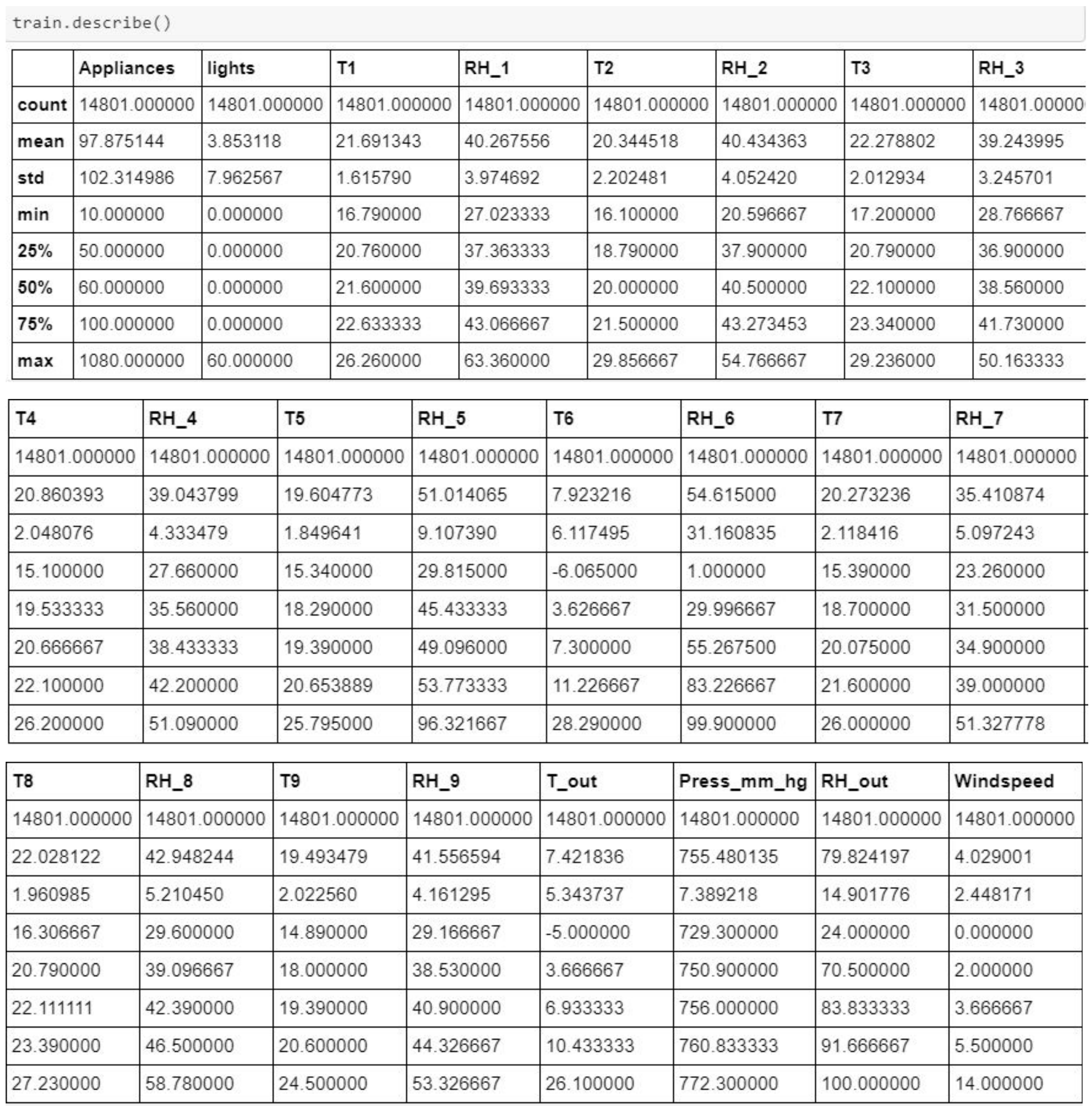
I didn't use the following variables from the start, for the rest I figured out the importance later.

- 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.

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
All the features have **numerical values**, and there are **no null or missing values.**

Test 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	14801.00000
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

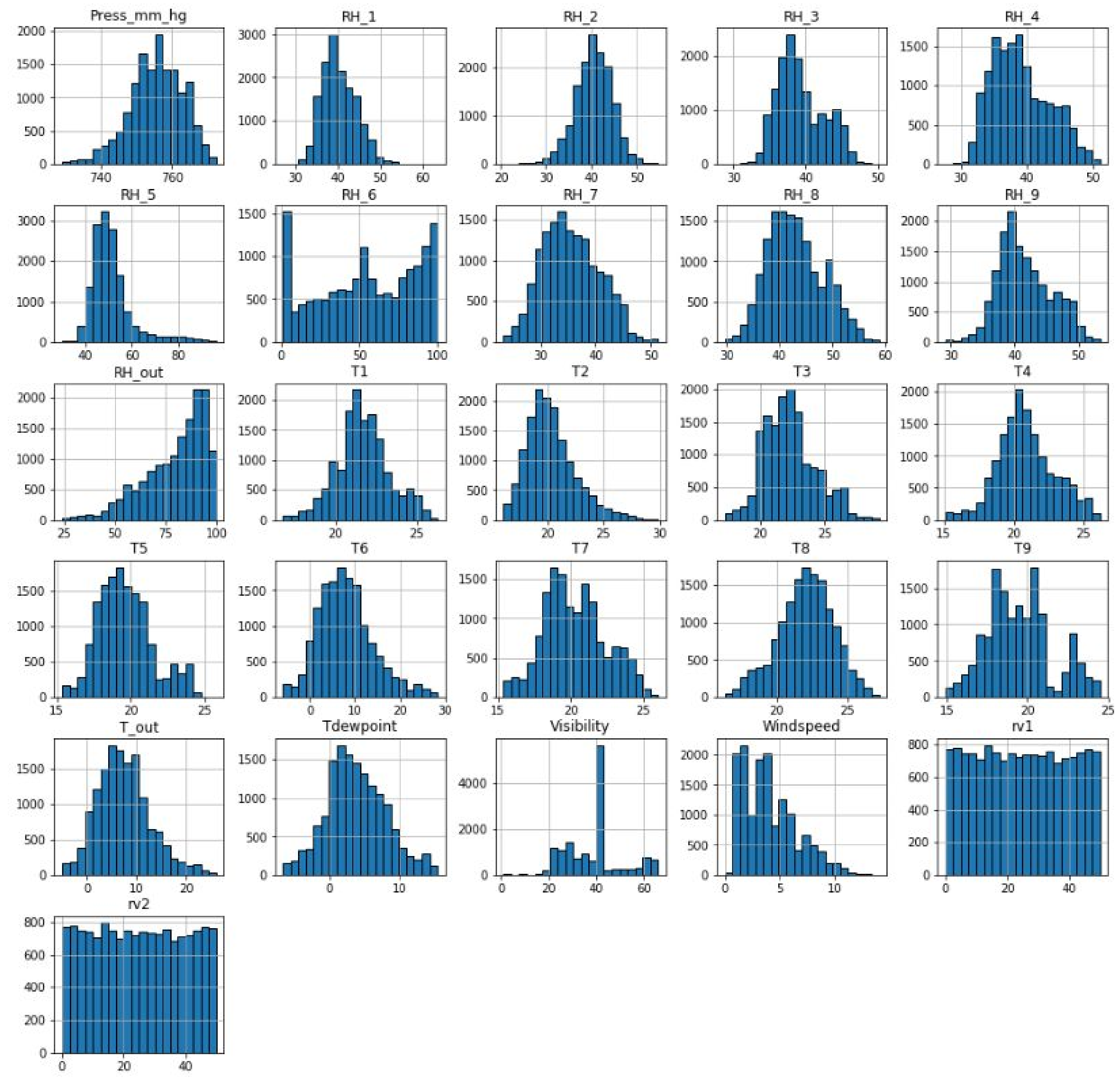
T4	RH_4	T5	RH_5	T6	RH_6	T7	RH_7
14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
20.860393	39.043799	19.604773	51.014065	7.923216	54.615000	20.273236	35.410874
2.048076	4.333479	1.849641	9.107390	6.117495	31.160835	2.118416	5.097243
15.100000	27.660000	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

II. Distribution of Input Attributes

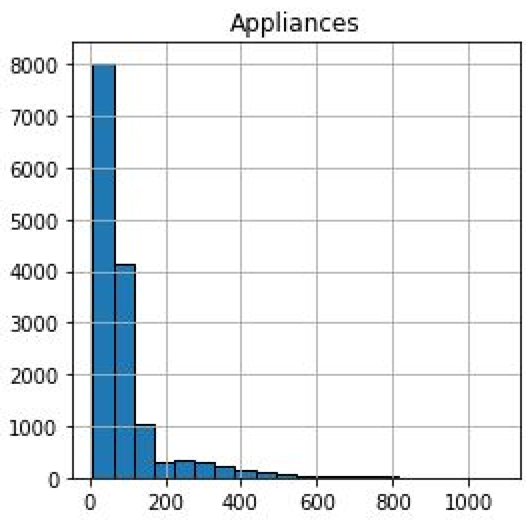
```
hists = input_vars.hist(figsize=(16, 16), bins=20,edgecolor='black')
```



III. Output Variable distribution

```
output_var.hist(figsize=(4,4), bins=20,edgecolor='black')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0
      dtype=object)
```



It can be observed from Histograms that:-

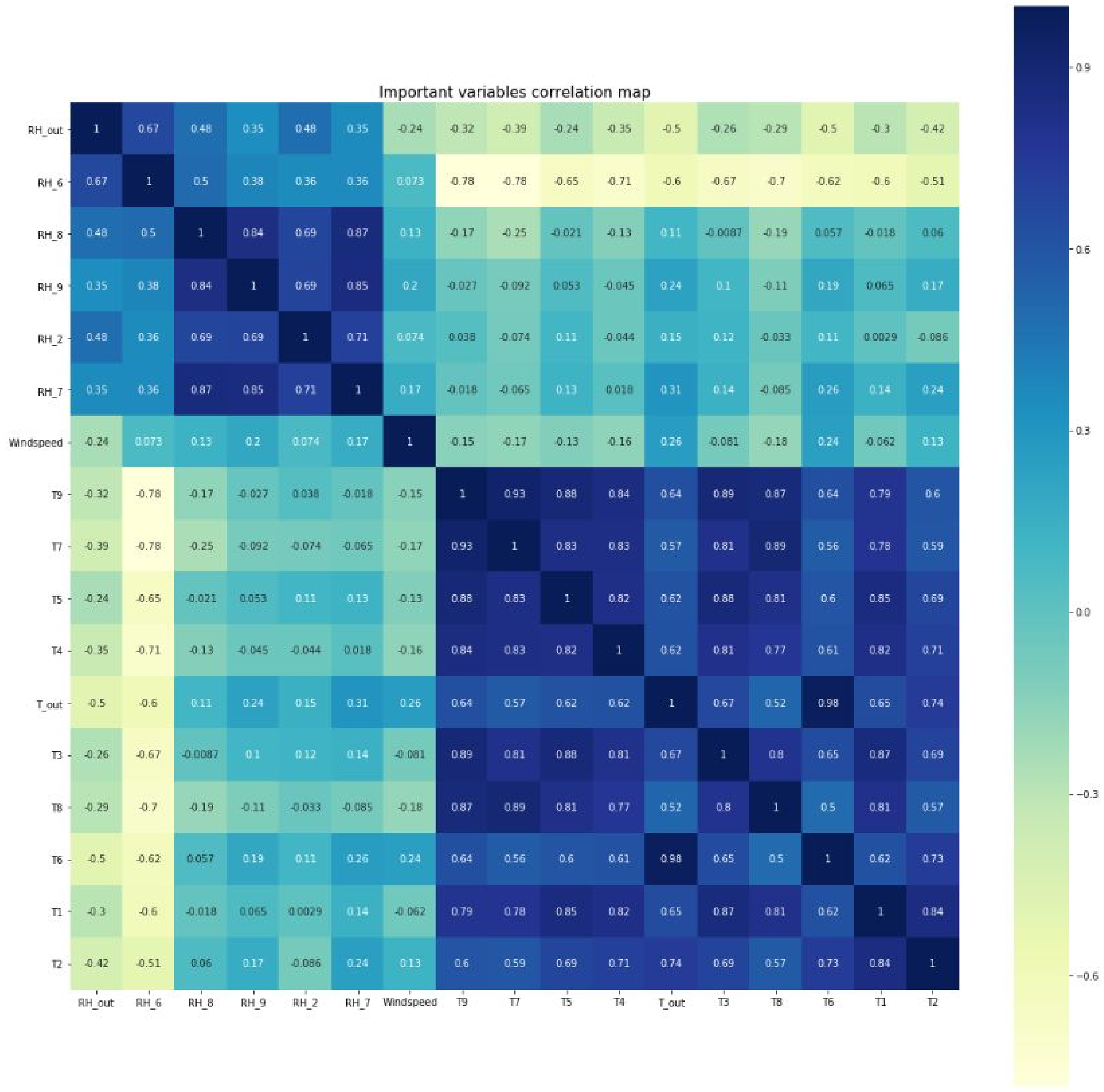
- All humidity values except RH_6 and RH_out follow a Normal distribution, i.e., all the readings from sensors inside the home are from a Normal distribution.
- Similarly, all temperature readings follow a Normal distribution except for T9.
- Out of the remaining columns, we can see that Visibility, 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.

Exploratory Visualisation

Co-relation plot

- The random variables rv1, rv2 and Visibility, Tdewpoint, Press_mm_hg have 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 >= 0.1.



Some inferences drawn from the above plot are :

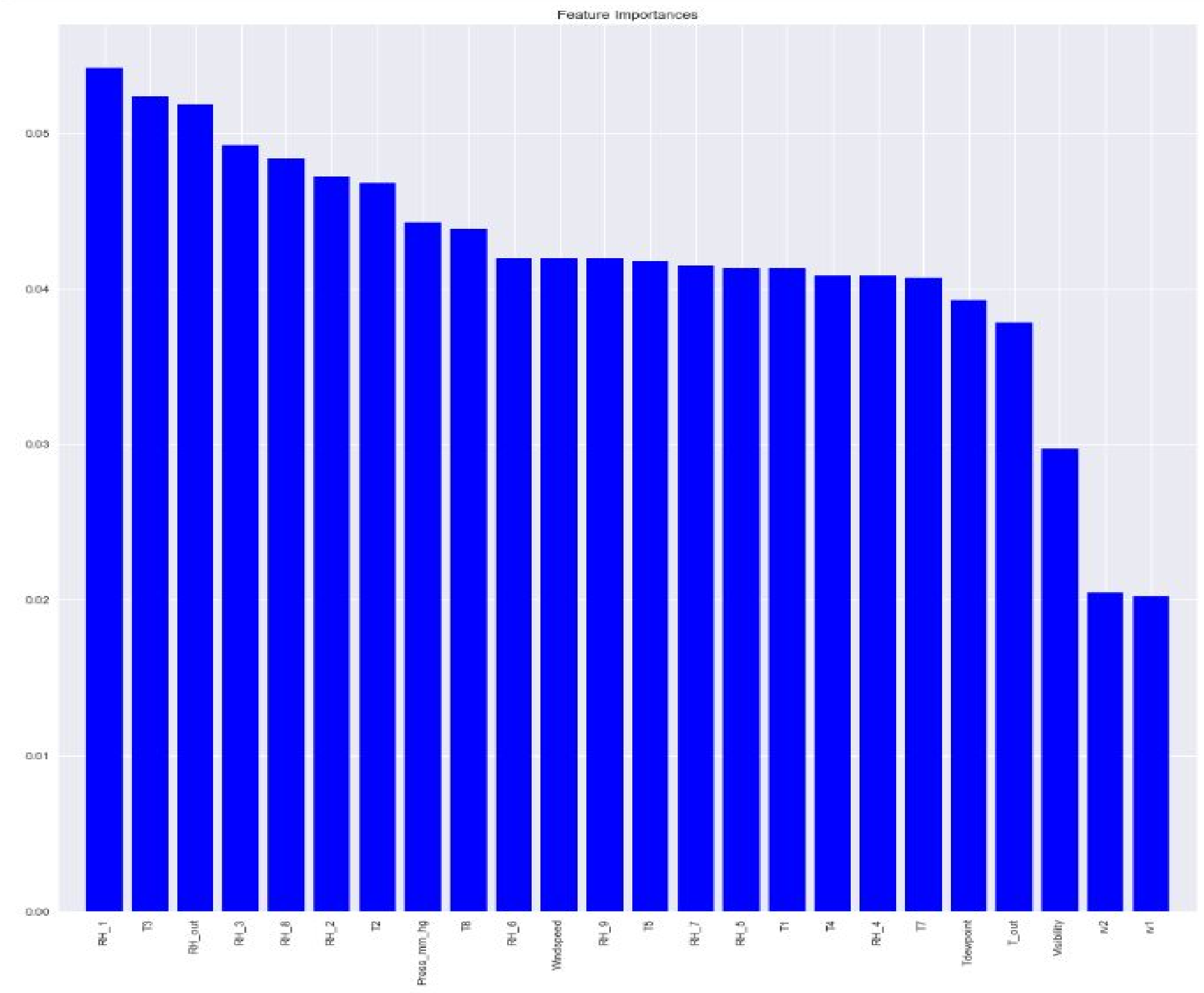
- The inter-attribute correlation is very high(>0.9) b/w T6 and T_out
- A number of variables have high correlation with T9 (T3,T5,T7,T8), so it is clear that T9 is redundant
- T6 and T9 need to be removed

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

```
train_X = train[input_vars.columns]
train_Y = train[output_var.columns]
```

```
train_X = train_X.drop(["T6", "T9"], axis=1)
```

```
from sklearn import ensemble
model = ensemble.ExtraTreesRegressor(n_estimators=200, max_depth=20, max_features=0.5, n_jobs=-1, random_s
tate=0)
model.fit(train_X, train_Y)
```



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.

- The most **basic** Regression algorithm is Linear Regression. If a Linear model can explain the data well, there is no need for further complexity.
- **Regularization** techniques which penalize the coefficient values of the features, since higher values lead to overfitting. With Regularization, Linear Regression is transformed into Lasso or Ridge Regression depending on how we modify our loss function.
- **Tree-based Regression**: Tree based models are less affected by outliers as compared to Linear models.
- A simple Decision Tree will surely overfit here owing to the number of features. So I used ensemble methods like **Random Forest** and **Extreme Trees Regression**.
- **Gradient Boosting Machines** is a type of Boosting method.
- **Neural networks** work great when there is a complex nonlinear relationship between the inputs and the output. I have used a Multi-Layer Perceptron.

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 0.97 on the training data and 0.58 on the test dataset.

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

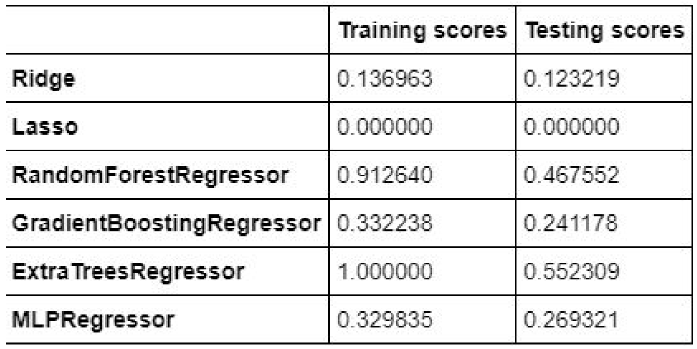
I used the following functions from scikit-learn to test each regression model:

- 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 were stored 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 DataFrame.

Result:



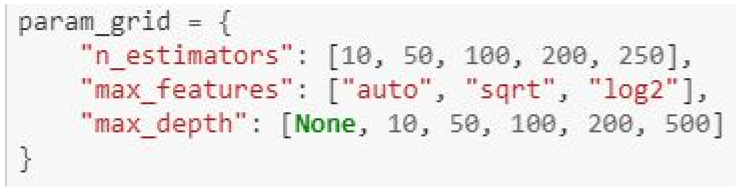
	Training scores	Testing scores
Ridge	0.136963	0.123219
Lasso	0.000000	0.000000
RandomForestRegressor	0.912640	0.467552
GradientBoostingRegressor	0.332238	0.241178
ExtraTreesRegressor	1.000000	0.552309
MLPRegressor	0.329835	0.269321

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 max_features=n_features.
- If “sqrt”, then max_features=sqrt(n_features).
- If “log2”, then max_features=log2(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%.

RESULTS

Model Evaluation and Validation

Un-tuned model parameters:

- n_estimators = 10
- max_features = auto
- max_depth = None

Refined model parameters:

- n_estimators = 250
- max_features = sqrt
- max_depth = 50

Robustness Check

Using only the 5 most important features reported by tuned ExtraTrees, the R2 score came out to be 50.37%. The untuned model had a score of 55.2%.
So even after a drastic reduction in features from 21 to just 5, the loss in performance is less.

Justification

Difference in the benchmark and my solution:

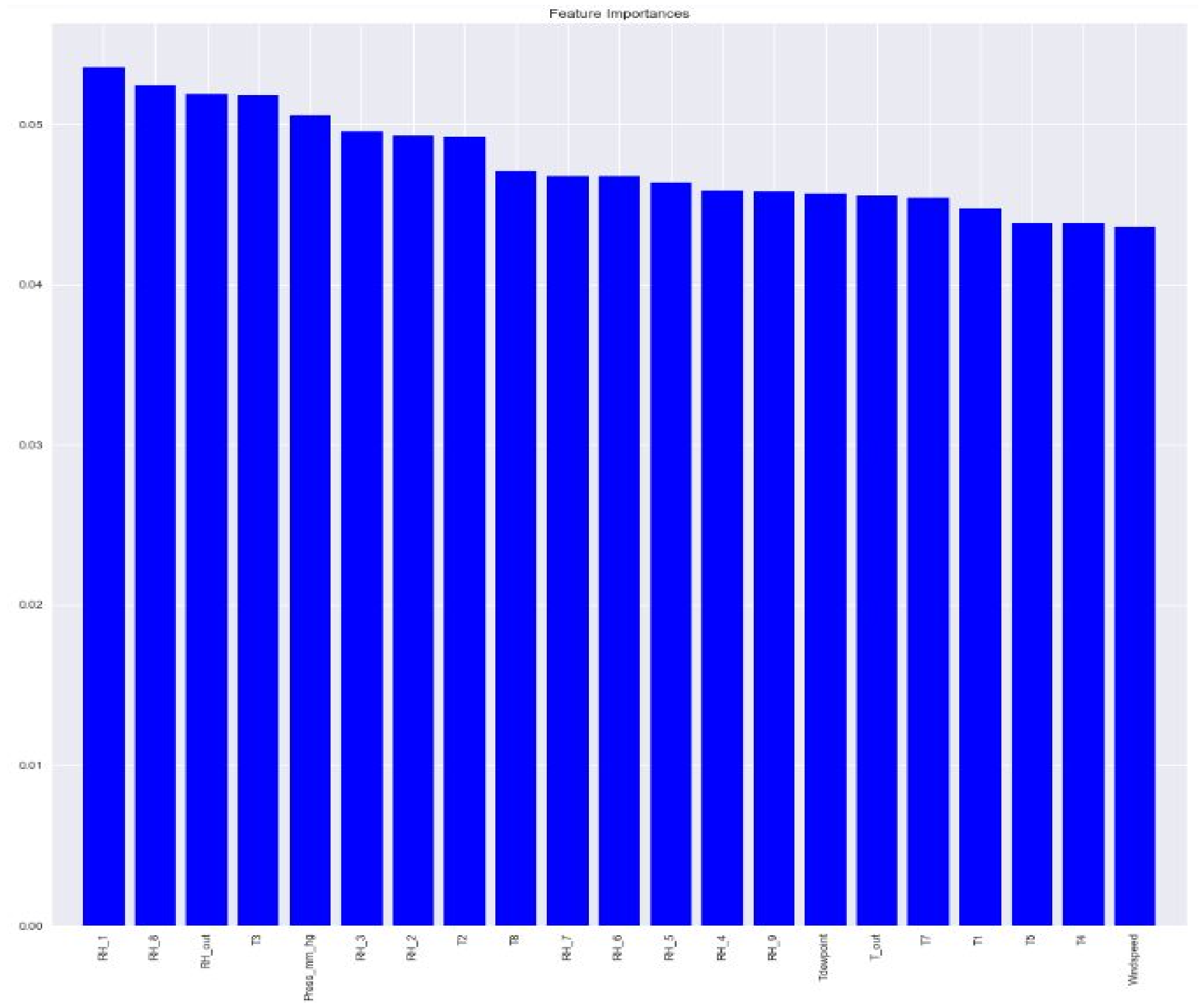
Models/Parameters	Training R2	Testing R2
Final Model	1.0	0.60
Benchmark	0.97	0.58
Difference	0.03	0.02

Given that the benchmark I used was the best possible result as discovered by the author of the original research paper, the improvement of about 2% is significant enough to consider my solution a satisfactory one.

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.

Improvement

- Performing better feature engineering.
- Adding more parameters to the Grid Search parameter space.