**House-Hold Appliances Energy Prediction**

**Harika Polaki**

**1.0 Abstract**

The demand for cost-effective energy and resource management continues to grow. There is a need for intelligent automated building solutions to decrease energy consumption which increases alternative energy sources, lessen operational costs, and find interoperable solutions that integrate with legacy equipment without huge investments. The ability to interpret, understand and predict building behavior offers enormous opportunities to demonstrate and confirm increased energy efficiencies, which may ease many extreme pressures taxing the Grid. In today's world, the ability to predict the household energy consumption can not only save money for the customer but can also help in generating money for them by giving back excess energy to Grid. To achieve this, we have implemented Gradient Boosting, Extra Tree Regressor, Random forest, Support Vector Machine, Multivariate Adaptive splines, Ridge, Lasso, Elastic Net Regularization, Partial Least Squares Regression, KNN, and Decision Tree. Data used include measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station, and recorded energy use of lighting fixtures.

**2.0 Data**

The dataset chosen for this project is Appliance’s Energy Prediction dataset from the UCI Machine Learning Repository. The goal of this project is to predict the energy utilization of appliances. Data was collected by surveying the temperature and humidity using wireless sensor networks. The data was obtained every ten minutes in order to record any immediate fluctuations in energy consumption. The house temperature and humidity conditions were monitored by a ZigBee wireless sensor network.

Each wireless node transmitted temperature, humidity conditions around 3.3 min and the wireless data was averaged for 10 minutes periods. The weather data from the nearest airport weather station is joined with this data to assess its influence on the energy consumption of appliances. Two random variables are used in this dataset to test the regression model and to eliminate the non-predictive attributes (UCI Machine Learning Repository, n.d.).

**Source:** Dataset has been chosen from UCI Machine Learning Repository - [Center for Machine Learning and Intelligent Systems](http://cml.ics.uci.edu/).

**Website Link:** <https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction>

**2.1 Predictors**

 1. Date time – represents the date and time when the temperature was calculated

2. Appliances - shows the energy utilization in Watts

3. lights - shows the energy utilization of light fixtures in the house in Watts

4. T1 - Temperature in the kitchen area, in Celsius

5. RH\_1- Humidity in the kitchen area, in %

6. T2 - Temperature in the living room area, in Celsius

7. RH\_2- Humidity in living room area, in %

8. T3 - Temperature in the laundry room area

9. RH\_3 - Humidity in laundry room area, in %

10. T4 - Temperature in office room, in Celsius

11. RH\_4 - Humidity in office room, in %

12. T5 - Temperature in the bathroom, in Celsius

13. RH\_5 - Humidity in the bathroom, in %

14. T6 - Temperature outside the building (north side), in Celsius

15. RH\_6 - Humidity outside the building (north side), in %

16. T7 - Temperature in ironing room, in Celsius

17. RH\_7 - Humidity in ironing room, in %

18. T8 - Temperature in teenager room 2, in Celsius

19. RH\_8 - Humidity in teenager room 2, in %

20. T9 - Temperature in parent’s room, in Celsius

21. RH\_9 - Humidity in parent’s room, in %

22. To - Temperature outside (from Chievres weather station), in Celsius

23. Pressure (from Chievres weather station) - in mmHg

24. RH\_out - Humidity outside (from Chievres weather station), in %

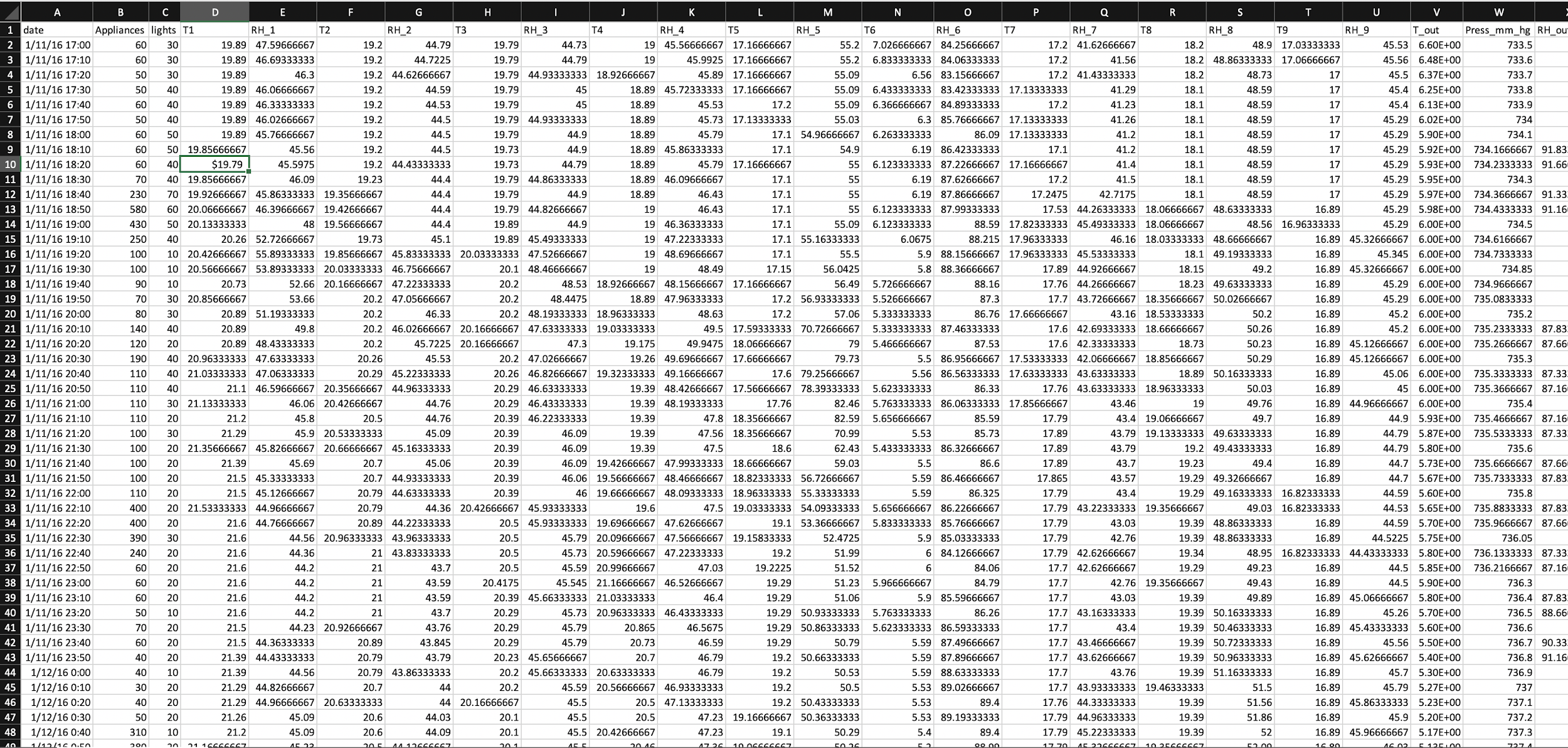
25. Wind speed (from Chievres weather station), in m/s

26. Visibility (from Chievres weather station), in km

27. Tdewpoint (from Chievres weather station), Â°C

RH refers to humidity variables, and T refers to Temperature variables. The predictors mentioned above helps in understanding the appliances used in which room or environment utilizes less energy and the factors behind it to inculcate these methods and methodologies in our day to day life to provide to the energy-saving resources.

**Raw Dataset**



**Figure 2.1: The snapshot includes the entire values of predictors and response variable from the raw dataset.**

**2.2 Floor Dimensions**

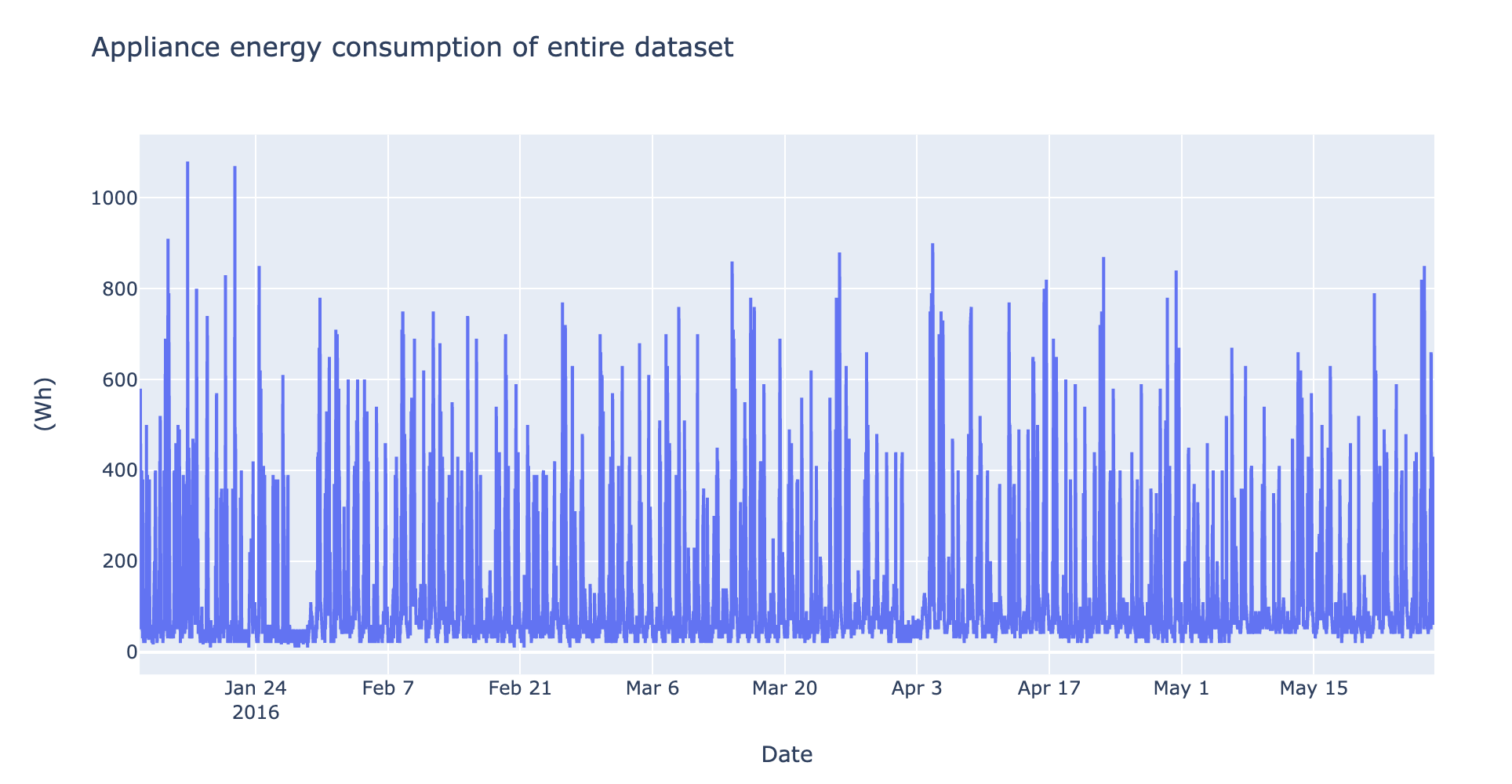
Figure 2.2 shows the dimensions of the building and also the position of the sensors which are used to record the temperature and humidity readings for even 10 minutes interval.The total building which is selected for this experiment is about 1,196 square foot. There are sensors that are also placed outside the building which records the visibility, dew point, Windspeed, temperature, humidity, and pressure outside the building These are also considered vital to conduct this predictive analytics.

****

**Figure 2.2: Showing the floor dimensions of the building on which the sensor recordings are taken.**

**3.0 Pre-Processing:**

Our Pre-Processing steps involved identifying the missing values and imputing them with centralized measures like mean and median. We have also removed the unnecessary data as our dataset has the time series data where the first predictor is the date with a recording for every 10 minutes. Some of the recordings happened missing due to technical issues, and the values aren’t recorded properly, and those recordings were skipped. We haven’t weighed the date column for further analysis as we had missing data, and also our main aim and goal is to predict the energy consumed by the appliances. The below picture helps in knowing the time series representation of the complete dataset.

**  
 Figure 3.1: Time series representation of the raw dataset.**

**3.1 Unnormalized data**

For detecting the unnormalized data we are using the histogram density plot and used pre-processing methods like center and scaling which helped to normalize the data and dealing with the outliers. As we have skewness to the response variable, we divided the entire dataset into 2 parts and dealt with them separately to make the models work efficiently. In the below figure we can clearly identify a few predictor variables whose distributions are not normal but unnormalized with irregular distributions like RH\_6, Visibility, RH\_out, Windspeed. These variables are to be further normalized and to find the relation between them, we are using the correlation plot.

A close up of a logo

Description automatically generated

**Figure 3.1.1: Histogram distribution showing the normalized and unnormalized data.**

**3.2 Feature Selection**

We are using feature selection algorithms to identify and select the significant features. The feature selection and scaling algorithms like Boruta and relative importance are implemented to check the significance of the predictor which helps to identify the importance of each predictor variable in the analysis process.

In machine learning and statistics, feature selection, also known as variable selection, is the process of selecting a subset of relevant features(variables, predictors) used in model construction. Feature selection techniques are used for simplification of models which are easy to interpret and to avoid the curse of dimensionality, enhanced generalization by reducing overfitting. We have worked on Boruta Algorithm for feature selection in our dataset.

* + 1. **The Boruta Algorithm:**

The Boruta algorithm is a wrapper built around the random forest classification algorithm. It is used for feature selection. It tries to capture all the important, interesting features you might have in your dataset with respect to an outcome variable. It works well for both classification and regression problems. The Boruta Algorithm includes plot for all the predictor variables against the response variable.From the below figure we can conclude that lights, Pressure, Humidity in Bathroom and Parent’s Room are the most important predictor variables.

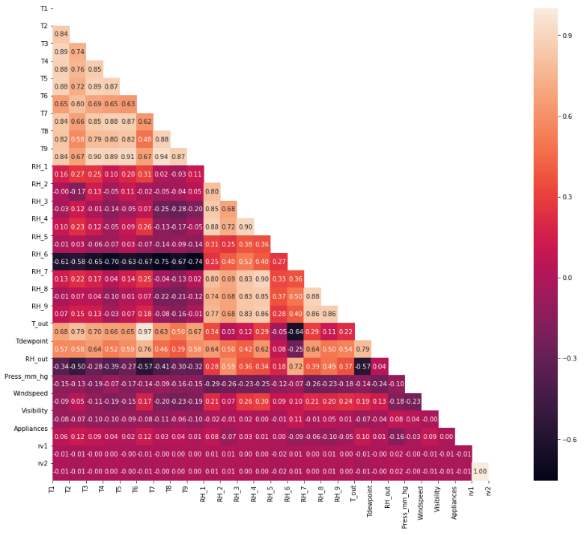
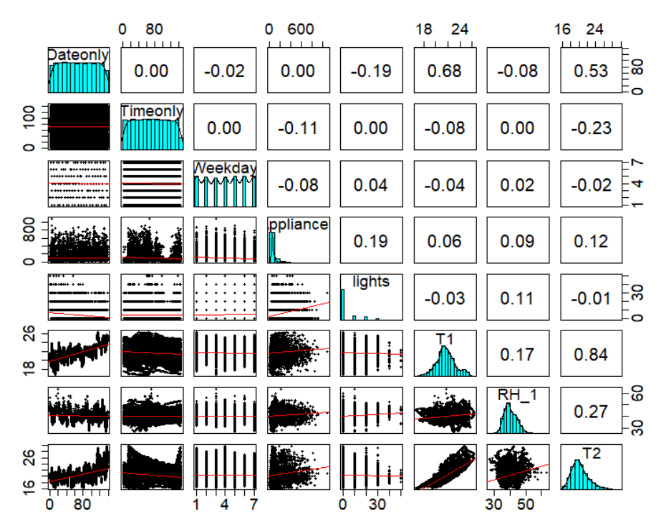
A screenshot of a cell phone

Description automatically generated

**Figure 3.2.1: Identifying the important variables using the Boruta variable importance methods.**

**3.3 Correlation Plots**

We have taken a few more additional steps to normalize the data where we have removed the highly correlated variables with a cut off at 0.9 and 0.8 to get the optimized results. We found the predictor T9 to be highly correlated compared to the rest of the predictors as it is showing correlation with T6 and T8, as a result the predictor T9 is removed from the dataset. The other predictors like visibility and wind speed are also removed as they are also highly influencing the other predictor variables and they are considered to remove from the dataset on which the analysis must be performed.

 ****

**Figure 3.3.1: Correlation plots helping to identify the highly correlated variables**

In the above figure 3.3.1, the first plot shows the relation between the predictor variables and the variables from the upper triangles whose values are greater than 0.8 and 0.9 and they are removed. Out of all the predictor variables the variables like T9 which is the temperature in the parent room and the two temperature outside the building seems to be highly correlated with every other variable in the dataset and thus they are removed.

**4.0 Modeling Techniques**

**4.1 Gradient Boosting Method**

Boosting is one of the best ensemble methods where each new tree is fitted on a modified version of the original data set. This method helped us to improve the model efficiency compared to other models and ensemble models are advantageous in this aspect compared to other models as they combine different models and retrieve the best values out of it.

Once the values are retrieved, to tune the model K-fold cross validation is used which is helpful across different tuning parameters with shrinkage values 0.01 and 0.10 and then with max tree depth of 1 and 5 the optimum values are retrieved. The optimal solution is found for number of trees = 500, learning rate = 0.10, interaction depth = 5 with least RMSE (**0.0041**) and high Rsquared (**0.998**).

A close up of a map

Description automatically generated

**Figure 4.1: Showing the Plot which is plotted with the tuning parameters and RMSE on Y axis.**

GBM is plotted boosting iterations on X-axis and RMSE on Y-axis. As the number of trees increases, RMSE drastically drops from 0.06 to 0.01 for learning rate 0.01 when tree depth is 1. As the number of trees increases, RMSE gradually drops from 0.02 to 0.01(approx.) for learning rate 0.10 when tree depth is 5. Using the variable importance, GBM predicts that **Pressure as highly important** and **Temperature Outside as a least important predictor.**

**4.2 Extra Tree Regressor**

Extra Tree Regressor can be very well explained when compared with the random forest. In random forest, we chose to sample with a replacement for bootstrap datasets and consider best split for each iteration whereas in extra tree regressor it takes the samples without replacement where duplications aren’t allowed, and the random split is considered for each iteration of the subset of the dataset.

Fine tuning is applied to the models after the initial values are obtained. Randomized CV and grid search CV are used to tune this model with different tuning parameters with the estimators ranging from 200 – 2000 and the depth from 80 – 250 in the parameter grid. Grid search evaluates all possible combinations of the parameter values where it combines the hyperparameter values of a model that gives us the best values. Grid search CV obtained the best values when tuned the model which is the best values of prediction compared to remaining models with the best RMSE and R squared values.

A close up of a piece of paper

Description automatically generated

**Figure 4.2: Plot for Extra tree regressor showing a branch with depth of 4 and 10 estimators**

The obtained RMSE and R-Squared values we received on our dataset with this Extra Tree Regressor are **0.004** and **0.999**for tuning parameters, number of estimators at 250 and maximum depth of tree at 80. Using the variable importance, Extra Tree Regressor predicts that **Humidity Outside as highly important** and **Temperature in the Kitchen Area as a least important predictor.**

**4.3 Random Forest**

Random forest helps in reducing the overfitting by bootstrap aggregation and significantly lowering overfitting. Also, it has its traces in reducing the variance whereby using the multiple trees, the chance of stumbling across the classifier is reduced.

A screenshot of a cell phone

Description automatically generated**A close up of a map

Description automatically generated  
Figure 4.3: Showing the plot for random forest with the trees on X axis and the error rate on y axis and also the random forest tree.**

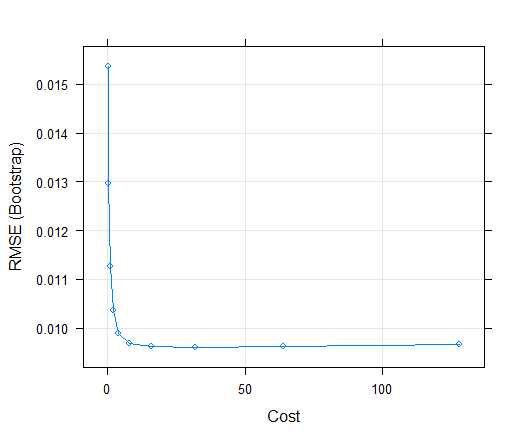
The tuning parameter for Random Forest is the number of trees. The Random Forest plot is plotted against the number of trees on the X-axis and error rate on the Y-axis. The trees vary from 0 to 500. The error rate decreases as the no. of trees increase until it reaches its optimal value and remains constant as trees are increased. The tuning to the model is done by giving the estimators and or trees and mtry values.

The obtained RMSE and R-Squared values we received on our dataset with Random Forest are **0.006** and **0.996**for tuning parameters number of trees = 500 and mtry value = 2.

**4.4 Support Vector Machine - SVM**

Support vector machine is a supervised learning model which is used for two group classification problems by using classification algorithm. SVM is also used for regression and outlier detection. SVM first separates the features and then a hyperplane is formed between these data points. The hyperplane is also called as decision boundary. The best hyperplane for SVM is the line whose distance to nearest element of each feature is the largest (Stecanella, n.d.) (Support Vector Machines, n.d.).

For our SVM model we have used a tuning parameter called Cost Function. Cost function ranges from 0 to 128. You can also see the plot which shows us the RMSE and Cost function for the SVM. As you can observe RMSE value decreases from C=0 to 4 and then it is constant till C=128. From the obtained summary for SVM model we have got an RMSE of 0.0096 and R squared of 0.993. The optimal value for which these results are obtained is at C= 32.

**  
Figure 4.4: Showing the SVM plot with the tuning parameter cost on X axis and the error rate on Y axis.**

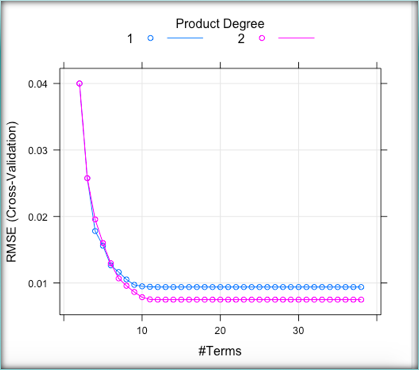
We have also performed variable importance which predicts that the most important variable is Pressure and the least important variable is Temperature(outside). The advantages of SVM are it is effective in high dimensional spaces and it is versatile, which means different kernel functions can be used for decision. The main disadvantage of SVM is that it is not suitable for datasets which are huge as it doesn’t perform well when data has more noise.

**4.5 Multivariate Adaptive Regression Splines (MARS)**

Multivariate adaptive regression spines is a form of regression analysis. MARS is a non-parametric regression technique and can be observed as extension of linear models which automatically models non linearities and interactions between variables. The interaction of variables can be said as knots of every feature. After the knots are known, now we can remove the knots of features which are not significant for predicting. This process is known as pruning. We can implement the MARS model using the package earth (Multivariate adaptive regression spline, n.d.).

For MARS we have two tuning parameters which is maximum degree of interactions and number of terms retained in model. From the summary of our model we got RMSE of 0.007 and R squared of 0.995. The optimal value for which these results are obtained are nprune = 12 and degree = 2. These values were obtained after tuning the model with iterated CV and grid parameters.

We have also performed variable importance which predicts Pressure as most important variable and Humidity in Bathroom as the least important variable. The advantage of MARS is that it is more flexible than linear regression models and it is simple to understand and interpret. MARS requires scaling of features and automated selection of features. We have also plotted a graph of RMSE, number of terms and two-degree interactions.

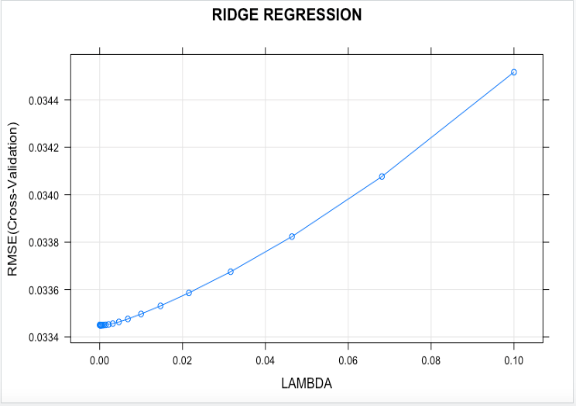
****

**Figure 4.5: Showing the plot of tuning parameters for MARS model.**

**4.6 Ridge Regression**

Ridge Regression is a parsimonious model which uses L2 Regularization. It is also a multi regression technique which is used when the number of predictor variables exceeds number of observations or when the data has multicollinearity. Ridge regression adds enough bias to make estimates reliable approximations to true population values (Stephanie, n.d.).

L2 regularization adds a penalty which equals the square of coefficients and all coefficients will be shrunk by the same factor. This penalty is a tuning parameter known as penalty parameter (). From the summary of our ridge model we got an RMSE score of 0.0342 and R squared of 0.900. The optimal value for which these values are obtained are =0.0004. We have performed K-fold CV to tune the model.



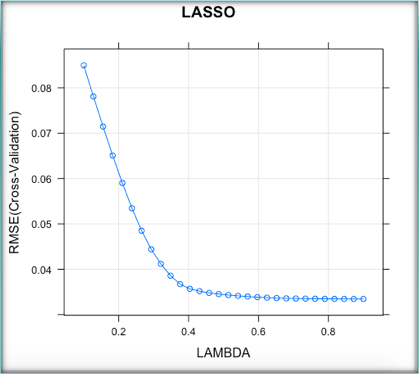
**Figure 4.6: Showing the plot of tuning parameters for Ridge model.**

We plotted a graph which shows RMSE and lambda values from ridge where we can see that RMSE increases as the lambda value increases. We also performed variable importance through which we got Pressure as the most important variable and Temperature(out) as the least important variable. One of the major advantage of ridge regression is that it works well with large multivariate data which has number of predictors greater than number of samples. L2 regularizations will not have sparse models

**4.7 Lasso Regression**

Lasso regression is a parsimonious model which uses L1 Regularization. L1 regularization is adds a penalty which is equal to the absolute value of the coefficients. This will result in removal of coefficients and to get sparse models. From the summary of lasso regression, we got RMSE value of 0.0342 and R squared value of 0.9007. This was for an optimal value of =0.9. These values are obtained when we use k-fold CV method to tune the model which uses train and test datasets (Stephanie, Lasso Regression: Simple Definition, n.d.).

We have plotted a graph which gives us RMSE and lambda where lambda ranges from 0 to 1.0 and have 30 values in between. From the plot we can see that RMSE decreases from 0 to 0.4 and then is constant from 0.4 to 1.0. We have also performed variable importance where we got Pressure as the most important variable and Temperature(out) as the least important variable. Lasso gives us sparse models which is possible through shrinkage and eliminating of coefficients. Lasso is also easier to interpret than the ridge regression.

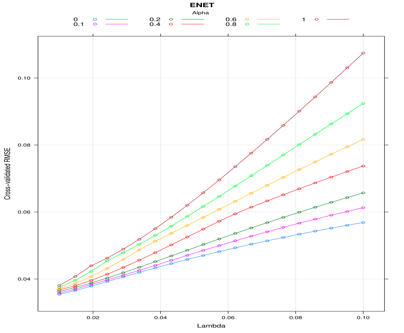


**Figure 4.7: Showing the plot of tuning parameters for Lasso model.**

**4.8 Elastic Net Regularization**

Elastic net regularization is a regression method that linearly combines Lasso and Ridge penalties. Elastic net is also more expensive computationally than Lasso and Ridge. This is because we use cross validation as the relative weight. For example, if grid of alpha values is [0,1] with step size 0.1 then it will be 11 times more computationally expensive (Arunava, n.d.).

From our analysis we got an RMSE of 0.0361 and R squared of 0. 893. The optimal value at which we obtained these results are at =0.01 and α= 0. We also plotted a graph where RMSE, lambda and alpha were shown. There were 125 tuning values and 5 interactions. We also see that RMSE increases as lambda increases. We used k-fold cv for tuning the model with the help of lambda and alpha parameters. We also performed variable importance and we saw Pressure is the most important variable and Humidity in bathroom is the least important variable.

****

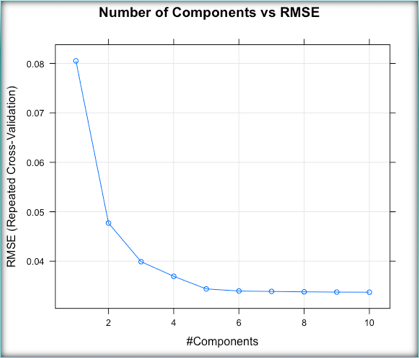
**Figure 4.8: Showing the plot of tuning parameters for ENET model.**

**4.9 Partial Least Squares Regression**

Partial least squares regression is developed from principal component regression. It is used in building models where there is more than one dependent variable. In principal component analysis (PCA) only X(data) is considered whereas in partial least squares regression both X and Y are considered (Partial Least Squares Regression, n.d.)`.

From the summary of our model we have got an RMSE value of 0.0337 and R squared value of 0.907. The optimal value for which we got these values is at ncomp=10. The tuning for the model was done with CV method and by specifying the tuning length.

We have also plotted a graph using RMSE and number of components. We got a total of 10 tuning components. We can see that RMSE decreases till 5 and then it is constant till 10. We also performed variable importance where we got to see that Pressure is the most important variable and Wind Speed is lease important variable.

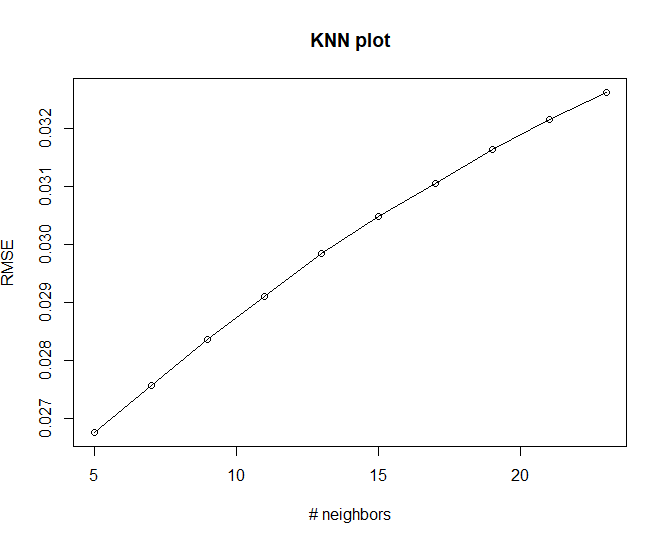


**Figure 4.9: Showing the plot of tuning parameters for PLS model.**

**4.10 K- Nearest Neighbor**

The k-Nearest Neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The tuning parameter is the K value along with the 10-fold cross validation method. This parameters are used to tune the model. The result contains the change of tuning parameter values ranging from 5 to 23 along with their RMSE, Rsquared, RMSESD, Rsquared SD values. The least RMSE and high R-Squared values we received on our dataset with K – Nearest Neighbor are 0.02671164 and 0.9427125 for tuning parameters K = 5.

So we can conclude that K=5 is the optimal solution.

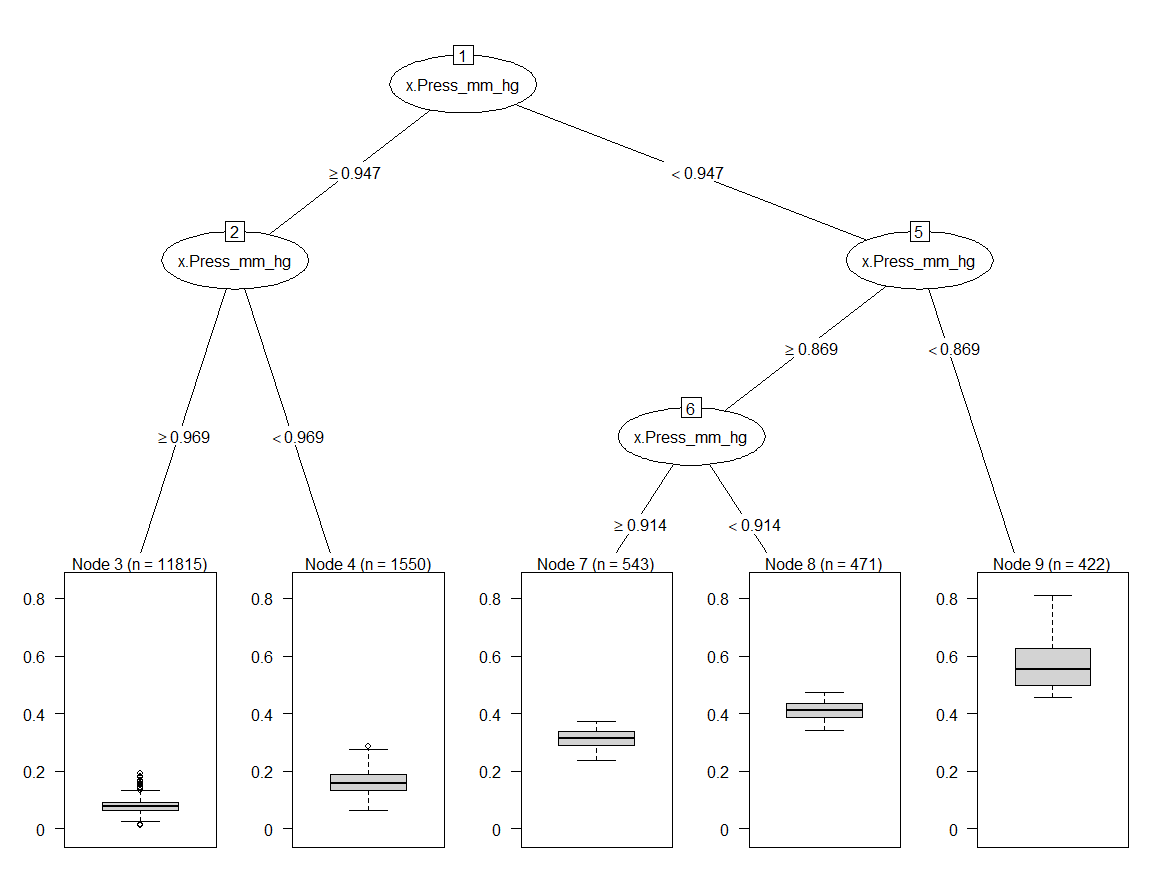


**Figure 4.10: Showing the plot of tuning parameters for KNN model.**

The above screenshot shows the plot of tuning parameters for KNN models. Here the K value (number of neighbours) is plotted on the X axis and RMSE is plotted on the Y axis respectively. It is observed that as the number of neighbours increases, RMSE increases.

**4.11 Decision Tree**

Decision trees have a natural “if … then … else …” construction for easier understanding. It uses different algorithms to identify the variable and the split. The tuning parameters are the maximum depth of the tree and the complexity parameter which is used to select the size of the tree. The highest R-Squared and least RMSE values derived from decision tree are 0.908 and 0.032 for the tuning parameters cp = 0.01 and maximum depth = 30.



**Figure 4.11: Showing the plot of tuning parameters for Decision Tree model.**

In the decision tree plot we can see that Pressure outside the house is considered to be the most important variable since the value is high.

**5.0 Future Work**

Future work includes adding more predictive parameters such as occupancy information, the area of the house, day to day activities performed by the occupant, more indoor and outdoor environmental conditions, by using different datasets can also improve the prediction. Training other machine models such as ANN (Artificial Neural Network) can further boost the predictive capacity, energy consumption is a vast domain and has scope in future to implement the hierarchical and hybrid models to predict the appliances energy on a larger scale.

**6.0 Conclusion**

**A screenshot of a cell phone

Description automatically generated**

**Fig 6.1 Bar graph showing the RMSE, Rsquared values of the predictive models.**RESULT

The bar graph above showcases the RMSE, Rsquared estimations of all the predictive models executed in this venture to anticipate the energy usage of the Appliances, we see that on the test set, Support Vector Machine, Mars, Random Forest, Gradient Boosting Method, Extra Tree Regressor have most elevated R-squared values. Least RMSE values have appeared for all the ensemble techniques like Extra Tree Regressor, Gradient Boosting Method and Random Forest. Out of all the techniques, Extra tree regressor has the least RMSE esteem and the most noteworthy or high R2 value which is viewed as the best model for this dataset and the application of predicting the energy usage of the appliances.

|  |  |  |
| --- | --- | --- |
| MODELS | RMSE | R-SQUARED |
| GBM | 0.0041 | 0.998 |
| EXTRA TREE REGRESSOR | 0.004 | 0.999 |
| SVM | 0.0096 | 0.993 |
| LASSO | 0.0342 | 0.9007 |
| RIDGE | 0.0342 | 0.9008 |
| ENET | 0.0361 | 0.8938 |
| PLS | 0.0345 | 0.899 |
| MARS | 0.007 | 0.995 |
| RANDOM FOREST | 0.0638 | 0.99680 |
| KNN | 0.2470 | 0.9506 |
| DECISION TREE | 0.3289 | 0.9086 |

The above table shows the comparison of various algorithms we have performed. From this we can conclude that Extra TRee Regressor has the least RMSE, and hence the highest R Squared value of 0.999 which means the model explains 99.9% of the variance in the data.

The optimal predictive model for Predicting Appliances Energy Consumption is Extra Tree Regressor with RMSE (**0.004**) and Rsquared (**0.999**).

**REFERENCES**

[1] Realdataweb. (2017, May 23). *Ridge Regression and the Lasso*. Retrieved from RBlogggers: <https://www.r-bloggers.com/ridge-regression-and-the-lasso/>.

[2] Arunava. (n.d.). *ENet — A Deep Neural Architecture for Real-Time Semantic Segmentation*. Retrieved from Towards Data Science: <https://towardsdatascience.com/enet-a-deep-neural-architecture-for-real-time-semantic-segmentation-2baa59cf97e9>.

[3] Boehmke, B. (n.d.). *Multivariate Adaptive Regression Splines*. Retrieved from <https://bradleyboehmke.github.io/HOML/mars.html#fitting-a-basic-mars-model>.

[4] Kassambara. (2018, November 3). *Statistical tools for high-throughput data analysis*. Retrieved from STDHA: <http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/152-principal-component-and-partial-least-squares-regression-essentials/>.

[5] Oleszak, M. (2019, November 12). *Regularization: Ridge, Lasso and Elastic Net*. Retrieved from DataCamp: <https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net>.

[6] Xu, P. J. (2019, November 25). Classification Trees and Rule-based Models. Retrieved from <https://mymasonportal.gmu.edu/webapps/blackboard/content/listContent.jsp?course_id=_375504_1&content_id=_9047730_1&mode=reset>.

**7.0 Appendix:**

**7.1 Gradient Boosting Method:**

A screenshot of a cell phone

Description automatically generated

**Figure 7.1: Showing the results of Gradient Boosting Model**

The image shows the results of Gradient Boosting Model on train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, learning rate(0.01, 0.10), interaction depth(1,5) and number of trees(100,500).

A screenshot of a cell phone

Description automatically generated

**Figure 7.1.1: Showing Variable Importance and RMSE, Rsquared values on test set.**

The image shows the variable importance by Gradient Boosting Model and predicted RMSE, Rsquared values on test set.

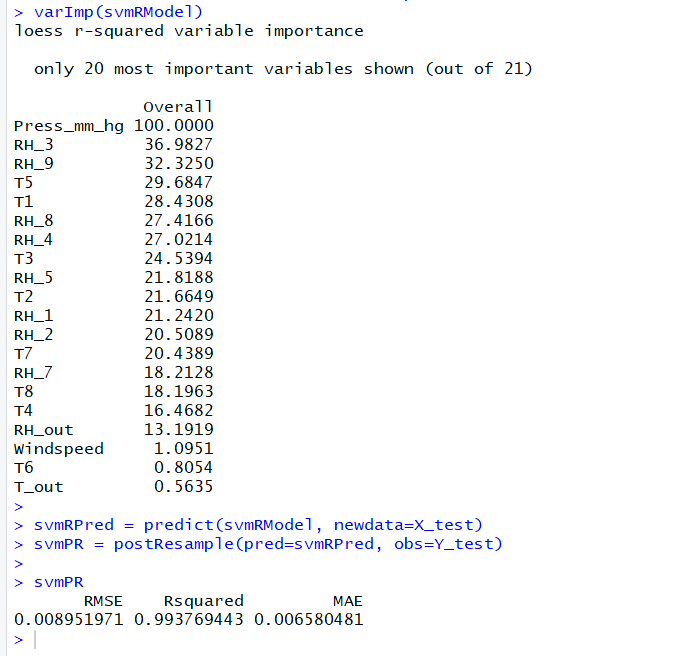
**7.2 Support Vector Machines:**

**A close up of text on a black background

Description automatically generated**

**Figure 7.2: Showing the results of Support Vector Regression**

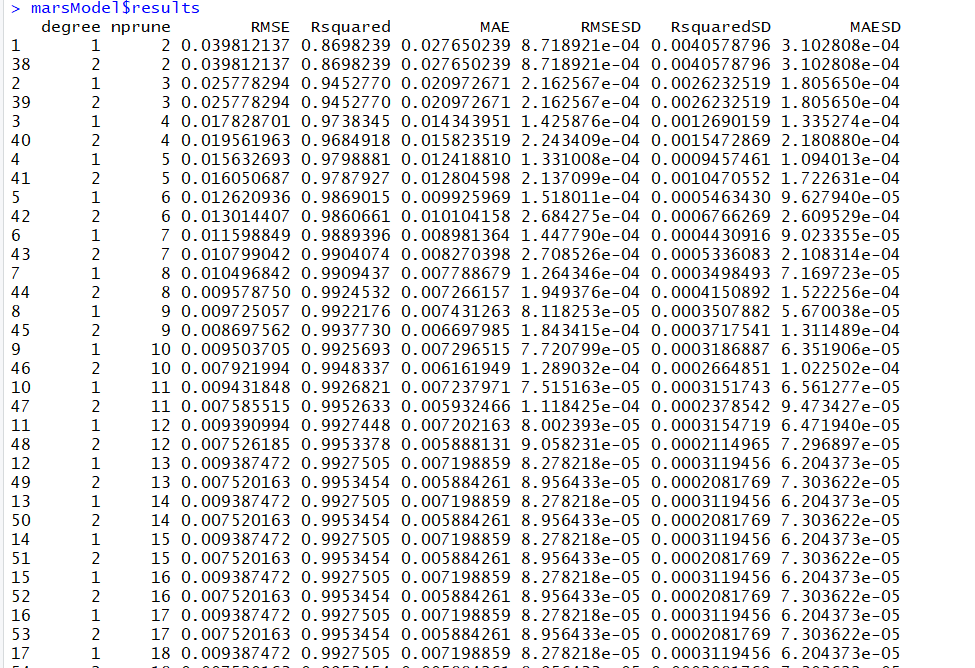
The image shows the results of Support Vector Regression on train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, cost function(0.25, 0.50 1.00….128).



**Figure 7.2.1: Showing Variable Importance and RMSE, Rsquared values on test set.**

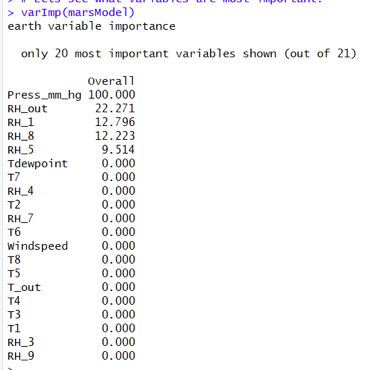
The image shows the variable importance by Support Vector Regression and predicted RMSE, Rsquared values on the test set.

**7.3 MARS:**



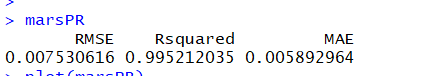
**Figure 7.3: Showing the results of MARS model**

The image shows the results of MARS model on a train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, degree (1,2) and nprune (2,30).

****

**Figure 7.3.1: Showing Variable Importance for MARS model**

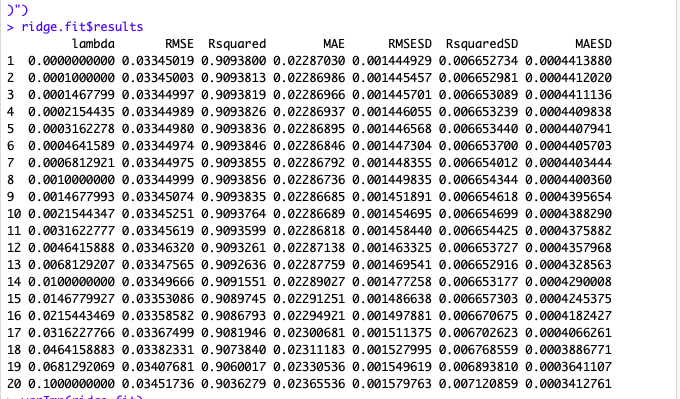
The image shows the variable importance by MARS model and predicted RMSE, Rsquared values on the test set.



**Figure 7.3.2: Showing RMSE, Rsquared on test set.**

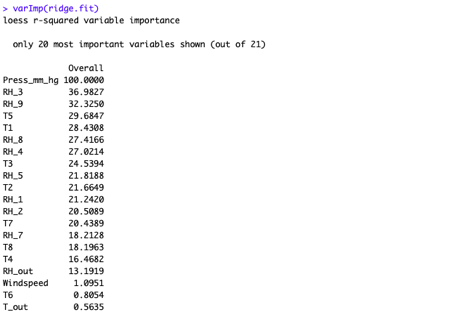
The image shows the RMSE, Rsquared by MARS model on a test set.

**7.4 RIDGE:**

****

**Figure 7.4: Showing the results of Ridge Regression**

The image shows the results of Ridge Regression on train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, lambda (0,0.1)



**Figure 7.4.1: Showing Variable Importance for Ridge model**

The image shows the variable importance by Ridge model.

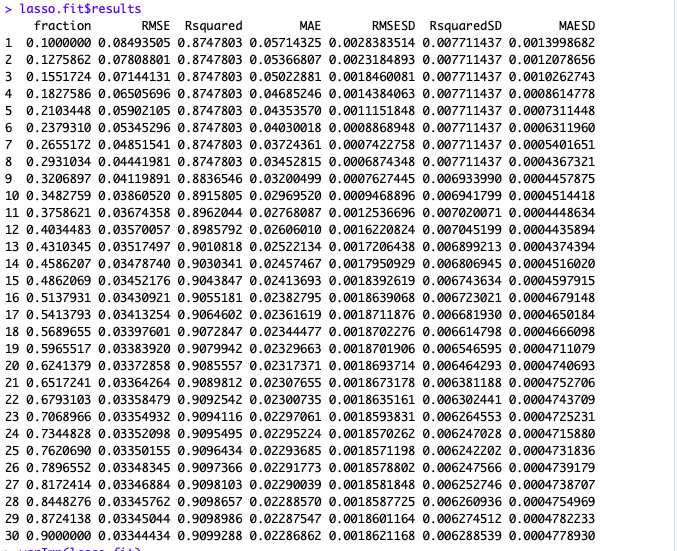
A screenshot of a cell phone

Description automatically generated

**Figure 7.4.2: Showing RMSE, Rsquared on test set.**

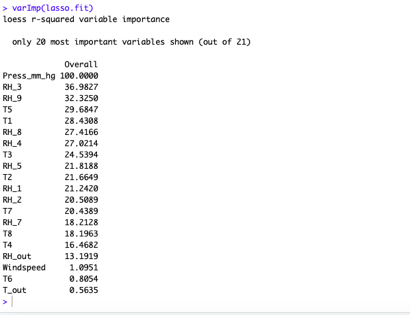
The image shows the RMSE, Rsquared by Ridge model on a test set.

**7.5 LASSO:**

****

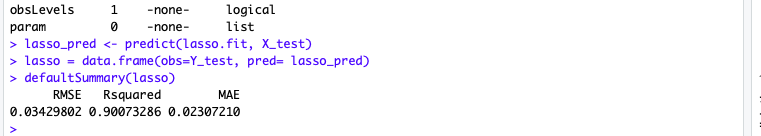
**Figure 7.5: Showing the results of Lasso Regression**

The image shows the results of Lasso Regression on train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, lambda (0,0.9)



**Figure 7.5.1: Showing Variable Importance for Lasso model**

The image shows the variable importance by the Lasso model.



**Figure 7.5.2: Showing RMSE, Rsquared on test set.**

The image shows the RMSE, Rsquared by Lasso model on a test set.

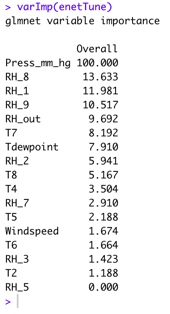
**7.6 ENET:**

A picture containing window, large, white

Description automatically generated

**Figure 7.6: Showing the results of ENET Regression**

The image shows the results of ENET Regression on train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, alpha(0,1) and lambda(0.01, 0.02).

****

**Figure 7.6.1: Showing Variable Importance for ENET model**

The image shows the variable importance by ENET model.

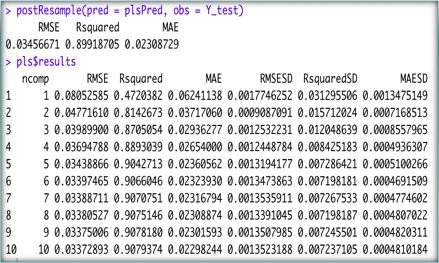
A screenshot of a cell phone

Description automatically generated

**Figure 7.6.2: Showing RMSE, Rsquared on test set.**

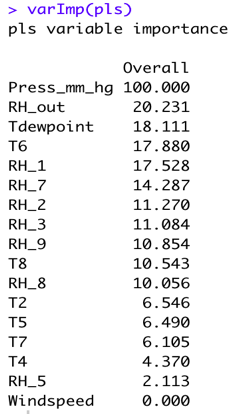
The screenshot shows the RMSE, Rsquared by ENET model on the test set.

**7.7 PLS:**

****

**Figure 7.7: Showing the results of PLS Regression and RMSE, Rsquared on test set.**

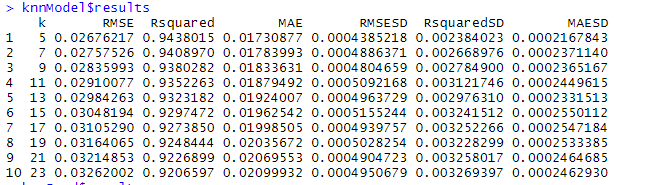
The image shows the results of PLS Regression on a train set that includes RMSE, Rsquared, Rsquared, RMSESD values against tuning parameters, number of components, ncomp(1,10).



**Figure 7.7.1: Showing Variable Importance for PLS model**

The image shows the variable importance by PLS model.

**7.8 KNN:**



**Figure 7.8: Showing the results of K-Nearest Neighbors.**

The above screenshot shows the results of K-Nearest Neighbors on the train set. The results include RMSE, Rsquared, MAE, RMSESD, RsquaredSD, MAESD values against tuning parameter K, number of neighbors.



**Figure 7.8.1: Showing RMSE, Rsquared on test set.**

The image shows the RMSE, Rsquared by K-Nearest Neighborson the test set.

A close up of a map

Description automatically generated

**Figure 7.8.2: Showing the plot of tuning parameters for KNN model.**

The above screenshot shows the plot of tuning parameters for KNN models. Here the K value (number of neighbours) is plotted on the X axis and RMSE is plotted on the Y axis respectively. It is observed that as the number of neighbours increases, RMSE increases.

k= 5 has the least RMSE and is considered to be the optimal solution.

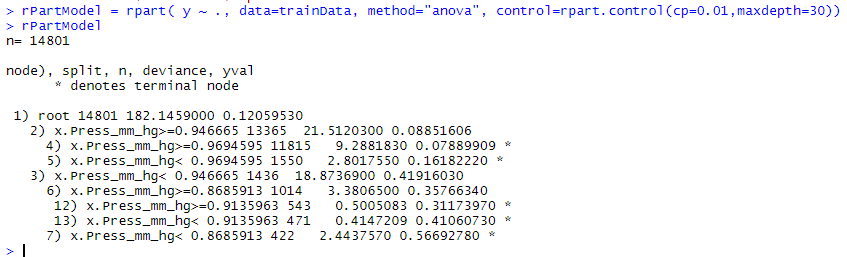
A screenshot of a cell phone

Description automatically generated

**Figure 7.8.3: Showing the results of K-Nearest Neighbors.**

The screenshot shows the results of K-Nearest Neighbors on the train set that includes RMSE, Rsquared, values against tuning parameter, K, number of neighbors.

**7.9 Decision tree:**



**Figure 7.9: Showing the results of Decision Trees.**

The above screenshot shows the results of Decision Trees on the train against tuning parameters, cost complexity parameter (0.01) and maximum node depth(30).



**Figure 7.9.1: Showing RMSE, Rsquared on test set.**

The image shows the RMSE, Rsquared by Decision Treeson the test set.

**7.10 Random Forest:**



**Figure 7.10: Showing RMSE, Rsquared on test set.**

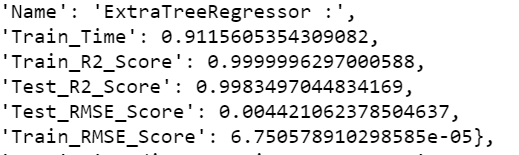
The image shows the RMSE, Rsquared by Decision Treeson a test set.



**Figure 7.7.1: Showing Variable Importance for Random Forest**

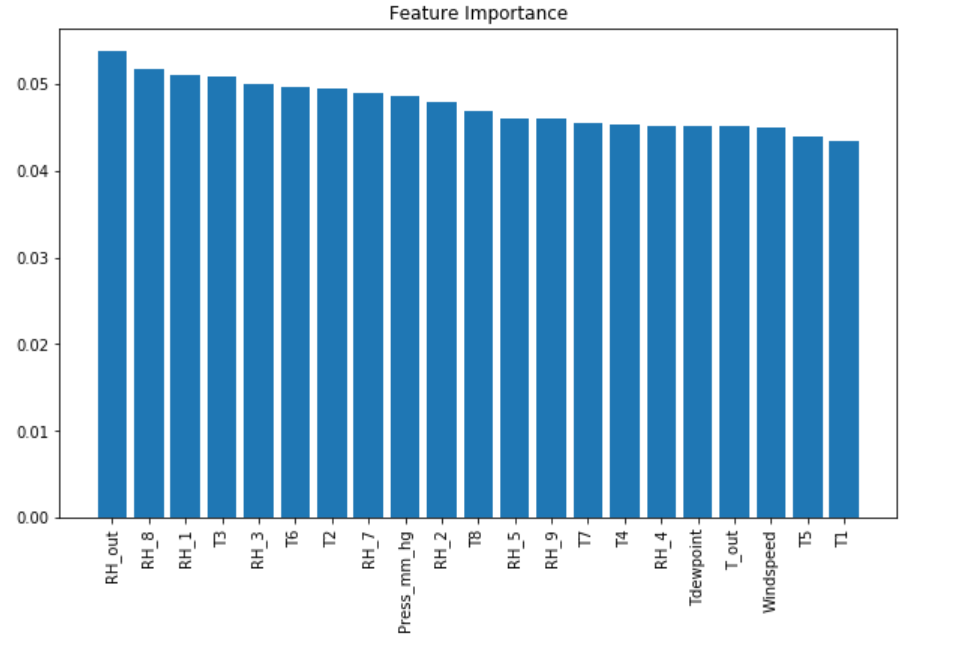
The image shows the variable importance by Random Forest.

**7.11 Extra Tree Regressor:**

****

**Figure 7.11: Showing RMSE, Rsquared on test and train set.**

The image shows the RMSE, Rsquared by Extra Tree Regressoron test and train set.

****

**Figure 7.11.1: Showing Variable Importance for Extra Tree Regressor**

The image shows the variable importance by Extra Tree Regressor.

**8.0 How to run code:**

**Step 1:** The raw data is included in the data folder within the zip folder and a link to the dataset is likewise given in the main body of the document under the dataset segment.

**Step 2:** The python code is appended where it incorporates all the pre-processing steps, normalizing and standardizing and removing the highly correlated variables etc.

**Step 3:** Subsequent to running the raw dataset through the python code first we have saved the data which is pre-processed and splitted as train\_x, train\_y, test\_x and test\_y in the local directory.

**Step 4:** These 4 datasets where train\_x and test\_x contains the data of the predictors whereas train\_y and test\_y contains the data regarding the response variable. This data is shared among the other people in the team to run their models on.

**Step 5:** The datasets which are pre-processed and splitted, which are used to run the R code, are also included in the Zip file with the same names.

**Step 6:** After loading those 4 datasets which are extracted from the raw dataset after preprocessing and normalizing into the R studio then please run the rest of the models.

**Step 7:** Most of the models took so much time to run them. We have also tuned the models using random search CV, grid search CV and K fold CV and repeated CV using number of repeats.

**CODE**:

**Pre-Processing code in jupiter notebook:**

import pandas as pd

app\_energy = pd.read\_csv("/Users/anusha/Downloads/energydata\_complete.csv")

app\_energy.head()

app\_energy.shape

y = app\_energy.head(20)

y.head()

y.plot() # can enhance grid and see the changes

Convert the date into date time and save them in a separate column

app\_energy.head()

type(app\_energy['date'])

import dask.dataframe as dd

app\_energy['date'] = pd.to\_datetime(app\_energy['date'])

app\_energy.set\_index(app\_energy['date'],inplace=True)

app\_energy.index

data\_sample\_from\_pd = dd.from\_pandas(app\_energy, npartitions=1)

data\_sample\_from\_pd.index.head()

weekly = app\_energy.resample('W').sum()

weekly.plot(style = [':', '--', '-'])

daily = app\_energy.resample('D').sum()

daily.rolling(30, center = True).sum().plot(style =[':', '--', '-'])

## Part 2 Code

import numpy as np

import pandas as pd

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import preprocessing, model\_selection, metrics

import warnings

warnings.filterwarnings("ignore")

app\_energy.info()

## Splitting the data into training and splitting

from sklearn.model\_selection import train\_test\_split

data = app\_energy

# 75% of the data is usedfor the training of the models and the rest is used for testing

train, test = train\_test\_split(data,test\_size=0.25,random\_state=40)

train.shape

## Dividing the columns based on their features

# Dividing the columns based on their features like temperature, humidity anf weather (outside the house) and lights, random variables and appliances.

temp = ["T1","T2","T3","T4","T5","T6","T7","T8","T9"]

hum = ["RH\_1","RH\_2","RH\_3","RH\_4","RH\_5","RH\_6","RH\_7","RH\_8","RH\_9"]

weather = ["T\_out", "Tdewpoint","RH\_out","Press\_mm\_hg",

"Windspeed","Visibility"]

light = ["lights"]

randoms = ["rv1", "rv2"]

target = ["Appliances"]

Separating the target variables

feature\_vars = train[temp + hum + weather + light + randoms ]

target\_vars = train[target]

feature\_vars.drop(columns = ['lights'], axis = 1, inplace = True)

feature\_vars.head(2)

Time series representation of the entire dataset

import chart\_studio.plotly as py

from plotly.offline import init\_notebook\_mode, iplot

init\_notebook\_mode(connected=True)

import plotly.graph\_objs as go

# To understand the timeseries variation of the applaince energy consumption

Visulaization = go.Scatter( x= data.date , mode = "lines", y = data.Appliances )

viz\_layout = go.Layout(title = 'Appliance energy consumption of entire dataset' , xaxis=dict(title='Date'), yaxis=dict(title='(Wh)'))

fig = go.Figure(data=[Visulaization],layout=viz\_layout)

iplot(fig)

Creating Weekday Variable for timeseries representation

data['WEEKDAY'] = ((pd.to\_datetime(data['date']).dt.dayofweek)//5).astype(float)

# There are 5472 weekend recordings

data['WEEKDAY'].value\_counts()

data.head()

Time series representation of all weekdays

weekday\_timeseries = data[data['WEEKDAY'] == 0]

# To understand the timeseries variation of the applaince energy consumption

visData = go.Scatter( x= weekday\_timeseries.date , mode = "lines", y = weekday\_timeseries.Appliances )

layout = go.Layout(title = 'Appliance energy consumption - weekdays' , xaxis=dict(title='Date'), yaxis=dict(title='(Wh)'))

fig = go.Figure(data=[visData],layout=layout)

iplot(fig)

Time series representation of all Weekends

weekend\_timeseries = data[data['WEEKDAY'] == 1]

# To understand the timeseries variation of the applaince energy consumption

visData = go.Scatter( x= weekend\_timeseries.date , mode = "lines", y = weekend\_timeseries.Appliances )

layout = go.Layout(title = 'Appliance energy consumption - weekends ' , xaxis=dict(title='Date'), yaxis=dict(title='(Wh)'))

fig = go.Figure(data=[visData],layout=layout)

iplot(fig)

Histogram density function

# Histogram of all the features to understand the distribution

feature\_vars.hist(bins = 20 , figsize= (12,16)) ;

RH\_6, RH\_out, visibility, windspeed and T9 have irregular distributions

Pairwise plots distribution

sns.pairplot(feature\_vars);

# focussed displots for RH\_6 , RH\_out , Visibility , Windspeed, T9 due to irregular distribution

f, ax = plt.subplots(2,2,figsize=(12,10))

den1 = sns.distplot(feature\_vars["RH\_6"],bins=10, ax= ax[0][0])

den2 = sns.distplot(feature\_vars["RH\_out"],bins=10, ax=ax[0][1])

den3 = sns.distplot(feature\_vars["Visibility"],bins=10, ax=ax[1][0])

den4 = sns.distplot(feature\_vars["Windspeed"],bins=10, ax=ax[1][1])

# Distribution of values in Applainces column

Appliances\_dis = plt.figure(figsize=(12,5))

plt.xlabel('Appliance consumption in Whatts')

plt.ylabel('Frequency')

sns.distplot(target\_vars , bins=10 ) ;

# postive skewed

# count of appliances grater than 200 wh

data['Appliances'][data['Appliances'] > 200].count()

# feature variables have all columns and target\_vars have the appliance info.

print('Percentage of the appliance consumption is less than 200 Wh')

print(((target\_vars[target\_vars <= 200].count()) / (len(target\_vars)))\*100 )

More than 90% of the appliances consume less than 200 wh.

Correlation Matrix

corr\_plot = train[temp + hum + weather +target+randoms]

corr = corr\_plot.corr()

# Mask the repeated values

mask = np.zeros\_like(corr, dtype=np.bool)

mask[np.triu\_indices\_from(mask)] = True

f, ax = plt.subplots(figsize=(16, 14))

#Generate Heat Map, allow annotations and place floats in map

sns.heatmap(corr, annot=True, fmt=".2f" , mask=mask,)

#Apply xticks

plt.xticks(range(len(corr.columns)), corr.columns);

#Apply yticks

plt.yticks(range(len(corr.columns)), corr.columns)

#show plot

plt.show()

RH6 and T9 are highly correlated so remove the highly correlated variables

corrmatplot = data.corr()

f, ax = plt.subplots(figsize =(9, 8))

sns.heatmap(corrmatplot, ax = ax, cmap ="YlGnBu", linewidths = 0.1)

Identifying the variables which are highly correlated

data.corr().unstack().sort\_values(ascending = False).drop\_duplicates()

app\_energy.head()

Average of temperatures and humidity

app\_energy.head()

'''df = energy\_dat

df['avg\_temp'] = df[['T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9', 'T\_out']].mean(axis = 1)

df['avg\_RH'] = df[['RH\_1', 'RH\_2', 'RH\_3', 'RH\_4', 'RH\_5', 'RH\_6', 'RH\_7', 'RH\_8', 'RH\_9', 'RH\_out']].mean(axis = 1)

df.head()'''

feature\_vars.columns

train.columns

Splitting the data into training and test dataset

x\_train = train[feature\_vars.columns]

y\_train = train[target\_vars.columns]

x\_test = test[feature\_vars.columns]

y\_test = test[target\_vars.columns]

Four columns have a high degree of correlation with T9 - T3,T5,T7,T8 also T6 & T\_Out.

rh6 most negatively correlated and T9 most positively correlated.

x\_train.drop(["rv1","rv2","Visibility","RH\_6","T9"],axis=1 , inplace=True)

x\_test.drop(["rv1","rv2","Visibility","RH\_6", "T9"], axis=1, inplace=True)

x\_train.columns

x\_test.columns

Normalizing the dataset and preprocessing using Scaling method

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import Normalizer

sc=StandardScaler()

n = Normalizer()

# Create test and training set by including Appliances column

train = train[list(x\_train.columns.values) + target ]

test = test[list(x\_test.columns.values) + target ]

# Create dummy test and training set to hold scaled values

sc\_train = pd.DataFrame(columns=train.columns , index=train.index)

sc\_train[sc\_train.columns] = n.fit\_transform(train)

sc\_test= pd.DataFrame(columns=test.columns , index=test.index)

sc\_test[sc\_test.columns] = n.fit\_transform(test)

x\_train = sc\_train.drop(['Appliances'] , axis=1)

y\_train = sc\_train['Appliances']

x\_test = sc\_test.drop(['Appliances'] , axis=1)

y\_test = sc\_test['Appliances']

x\_test.to\_csv(r'/Users/anusha/Desktop/Assignments-Spring2020/SYST-568/Assignments/Project/test\_x.csv')

y\_test.to\_csv(r'/Users/anusha/Desktop/Assignments-Spring2020/SYST-568/Assignments/Project/test\_y.csv')

x\_train.to\_csv(r'/Users/anusha/Desktop/Assignments-Spring2020/SYST-568/Assignments/Project/train\_x.csv')

y\_test.to\_csv(r'/Users/anusha/Desktop/Assignments-Spring2020/SYST-568/Assignments/Project/test\_y.csv')

y\_train

Using SKlearn to implement all the models

from sklearn.linear\_model import Ridge, Lasso

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, ExtraTreesRegressor

from sklearn.neural\_network import MLPRegressor

import xgboost as xgb

from sklearn import neighbors

from sklearn.svm import SVR

from sklearn.ensemble import RandomForestRegressor

randomforest = RandomForestRegressor(random\_state = 42)

from pprint import pprint

print('Parameters currently in use:\n')

pprint(randomforest.get\_params())

Using Random Search CV method for tuning the methods

from sklearn.model\_selection import RandomizedSearchCV

# Number of trees in random forest

n\_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split

max\_features = ['auto', 'sqrt']

# Maximum number of levels in tree

max\_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max\_depth.append(None)

# Minimum number of samples required to split a node

min\_samples\_split = [2, 5, 10]

# Minimum number of samples required at each leaf node

min\_samples\_leaf = [1, 2, 4]

# Method of selecting samples for training each tree

bootstrap = [True, False]

# Create the random grid

grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap}

pprint(grid)

# Use the random grid to search for best hyperparameters

# First create the base model to tune

randomforest = RandomForestRegressor()

# Random search of parameters, using 3 fold cross validation,

# search across 100 different combinations, and use all available cores

rf\_model = RandomizedSearchCV(estimator = randomforest, param\_distributions = grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

# Fit the random search model

rf\_model.fit(train\_X, train\_y)

rf\_model.best\_params\_

best\_grid = rf\_model.best\_params\_

#grid\_accuracy = evaluate(best\_grid, test\_X, test\_y)

# Import the model we are using

from sklearn.ensemble import RandomForestRegressor

# Instantiate model with 1000 decision trees

randomforest = RandomForestRegressor(n\_estimators = 1000, random\_state = 42)

# Train the model on training data

randomforest.fit(x\_train, y\_train);

# Use the forest's predict method on the test data

predictions = rf.predict(x\_test)

# Calculate the absolute errors

errors = abs(predictions - y\_test)

# Print out the mean absolute error (mae)

print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

import os

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'

For retrieving the random forest tree graph

# Limit depth of tree to 3 levels

rf\_small = RandomForestRegressor(n\_estimators=10, max\_depth = 7)

rf\_small.fit(x\_train, y\_train)

# Extract the small tree

tree\_small = rf\_small.estimators\_[9]

# Save the tree as a png image

export\_graphviz(tree\_small, out\_file = 'D:\\aSemester - 3\\OR 568\\Data\_Project\\small\_tree.dot', feature\_names = list(x\_train.columns), rounded = True, precision = 1)

(graph, ) = pydot.graph\_from\_dot\_file('D:\\aSemester - 3\\OR 568\\Data\_Project\\small\_tree.dot')

graph.write\_png('D:\\aSemester - 3\\OR 568\\Data\_Project\\small\_tree2.png');

# Limit depth of tree to 3 levels

rf\_small = ExtraTreesRegressor(n\_estimators=10, max\_depth = 4)

rf\_small.fit(train\_X, train\_y)

# Extract the small tree

tree\_small = rf\_small.estimators\_[9]

# Save the tree as a png image

export\_graphviz(tree\_small, out\_file = 'D:\\aSemester - 3\\OR 568\\Data\_Project\\small3tree.dot', feature\_names = list(train\_X.columns), rounded = True, precision = 1)

(graph, ) = pydot.graph\_from\_dot\_file('D:\\aSemester - 3\\OR 568\\Data\_Project\\small3tree.dot')

graph.write\_png('D:\\aSemester - 3\\OR 568\\Data\_Project\\smalltree3.png');

model = rf.fit(train\_X, train\_y)

results = rf.predict(test\_)

MSE = mean\_squared\_error(test\_X, results) #Calculate Mean Squared Error

RMSE = sqrt(MSE)

print ("RMSE : ",RMSE)

models = [

['Lasso: ', Lasso()],

['Ridge: ', Ridge()],

['KNeighborsRegressor: ', neighbors.KNeighborsRegressor()],

['SVR:' , SVR(kernel='rbf')],

['RandomForest ',RandomForestRegressor()],

['ExtraTreeRegressor :',ExtraTreesRegressor()],

['GradientBoostingRegressor: ', GradientBoostingRegressor()] ,

['XGBRegressor: ', xgb.XGBRegressor()] ,

['MLPRegressor: ', MLPRegressor( activation='relu', solver='adam',learning\_rate='adaptive',max\_iter=1000,learning\_rate\_init=0.01,alpha=0.01)]

]

import time

from math import sqrt

from sklearn.metrics import mean\_squared\_error

model\_data = []

for name,curr\_model in models :

curr\_model\_data = {}

curr\_model.random\_state = 78

curr\_model\_data["Name"] = name

start = time.time()

curr\_model.fit(train\_X,train\_y)

end = time.time()

curr\_model\_data["Train\_Time"] = end - start

curr\_model\_data["Train\_R2\_Score"] = metrics.r2\_score(train\_y,curr\_model.predict(train\_X))

curr\_model\_data["Test\_R2\_Score"] = metrics.r2\_score(test\_y,curr\_model.predict(test\_X))

curr\_model\_data["Test\_RMSE\_Score"] = sqrt(mean\_squared\_error(test\_y,curr\_model.predict(test\_X)))

curr\_model\_data["Train\_RMSE\_Score"] = sqrt(mean\_squared\_error(train\_y,curr\_model.predict(train\_X)))

model\_data.append(curr\_model\_data)

from sklearn.metrics import mean\_squared\_error

model\_data

df = pd.DataFrame(model\_data)

df.iloc[1:,]

Created a dataframe and the data points are the values which are obtained from the models in the R code except extrta regressor

import pandas as pd

data = [['MARS', 0.075, 0.995], ['Ridge', 0.0334, 0.909], ['Lasso', 0.0334, 0.909], ['GBM', 0.004, 0.998], ['SVM', 0.0096,0.993],

['KNN',0.02470946, 0.95065926], ['Decision Trees', 0.03289199, 0.90866495], ['PLS', 0.03372,0.9079], ['Enet', 0.0354,0.9038],

['Extratree Regressor', 0.004421, 0.998350], ['Random Forest', 0.0063, 0.9968]]

df = pd.DataFrame(data, columns = ['Name', 'RMSE', 'R2'])

df = df.sort\_values(by = ['R2'])

df.plot(x="Name", y=['R2' , 'RMSE'], kind="bar" ,

title = 'RMSE, R2 Score Results' , figsize= (6,6)) ;

from sklearn.model\_selection import GridSearchCV

param\_grid = [{

'max\_depth': [80, 150, 200,250],

'n\_estimators' : [100,150,200,250],

'max\_features': ["auto", "sqrt", "log2"]

}]

reg = ExtraTreesRegressor(random\_state=40)

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator = reg, param\_grid = param\_grid, cv = 5, n\_jobs = -1 , scoring='r2' , verbose=2)

grid\_search.fit(train\_X, train\_y)

from sklearn.model\_selection import GridSearchCV

grid\_search.best\_params\_

grid\_search.best\_estimator\_

grid\_search.best\_estimator\_.score(train\_X,train\_y)

grid\_search.best\_estimator\_.score(test\_X,test\_y)

np.sqrt(mean\_squared\_error(test\_y, grid\_search.best\_estimator\_.predict(test\_X)))

tuning xtree regressor :

R2: 0.998901

RMSE: 0.003607

np.sqrt(mean\_squared\_error(test\_y,grid\_search.best\_estimator\_.predict(test\_X)))

Feature Importance

feature\_indices = np.argsort(grid\_search.best\_estimator\_.feature\_importances\_)

importances = grid\_search.best\_estimator\_.feature\_importances\_

indices = np.argsort(importances)[::-1]

names = [train\_X.columns[i] for i in indices]

# Create plot

plt.figure(figsize=(10,6))

# Create plot title

plt.title("Feature Importance")

# Add bars

plt.bar(range(train\_X.shape[1]), importances[indices])

# Add feature names as x-axis labels

plt.xticks(range(train\_X.shape[1]), names, rotation=90)

# Show plot

plt.show()

train\_important\_feature = train\_X[names[0:5]]

test\_important\_feature = test\_X[names[0:5]]

from sklearn.base import clone

cloned\_model = clone(grid\_search.best\_estimator\_)

cloned\_model.fit(train\_important\_feature , train\_y)

**R code for Models:**

install.packages("caret")

install.packages("caret")

install.packages("ISLR")

install.packages("caTools")

install.packages("dplyr")

install.packages("mlbench")

install.packages("earth")

install.packages("kernlab")

install.packages("pROC")

install.packages("corrplot")

install.packages("tseries")

install.packages("Ecdat")

library(tseries)

library(Ecdat)

library(lattice)

library(ggplot2)

library(caret)

library(ISLR)

library(caTools)

library(dplyr)

library(Formula)

library(plotrix)

library(TeachingDemos)

library(plotmo)

library(mlbench)

library(nnet)

library(earth)

library(kernlab)

library(pROC)

## The datatsets included here are the datasets after preprocessing and

## the model EXTRA TREE REGRESSOR are done using python code due to the time complexity in R

X\_train <- read.csv("~/Downloads/train\_x.csv")

View(X\_train)

X\_train <- X\_train[, 2:22]

View(X\_train)

X\_test <- read.csv("~/Downloads/test\_x.csv")

View(X\_test)

X\_test <- X\_test[, 2:22]

View(X\_test)

#splitting our dependent variable into Y\_train and Y\_test

Y\_train <- read.csv("~/Downloads/train\_y.csv", header= FALSE)

Y\_train <- Y\_train[, 2]

View(Y\_train)

Y\_test <- read.csv("~/Downloads/test\_y.csv", header= FALSE)

Y\_test<- Y\_test[, 2]

View(Y\_test)

ctrl <- trainControl(method = "repeatedcv", repeats = 10)

# MARS model:

library(earth)

marsGrid = expand.grid(.degree=1:2, .nprune=2:30)

set.seed(100)

marsModel = train(x=X\_train, y=Y\_train, method="earth", preProc=c("center", "scale"),tuneGrid=marsGrid,trControl = ctrl )

marsModel

marsModel$results

plot(marsModel)

# Lets see what variables are most important:

varImp(marsModel)

marsPred = predict(marsModel, newdata=testData$x)

marsPR = postResample(pred=marsPred, obs=testData$y)

marsPR

#Ridge Regression

set.seed(100)

ctrl <- trainControl(method = "cv", number = 10)

ridge.fit <- train(X\_train, Y\_train, method = "ridge", trControl = ctrl, preProc = c("center", "scale"), tuneLength = 20)

ridge.fit

summary(ridge.fit)

ridge\_pred <- predict(ridge.fit, X\_test)

ridge = data.frame(obs=Y\_test, pred= ridge\_pred)

defaultSummary(ridge)

plot(ridge.fit, main= "RIDGE REGRESSION" , xlab = "LAMBDA", ylab = "RMSE(Cross-Validation)")

ridge.fit$results

varImp(ridge.fit)

#lasso\_regression

lasso.fit <- train(X\_train, Y\_train, method= "lasso", trControl = ctrl, preProc = c("center", "scale"), tuneLength = 30)

lasso.fit

summary(lasso.fit)

lasso\_pred <- predict(lasso.fit, X\_test)

lasso = data.frame(obs=Y\_test, pred= lasso\_pred)

defaultSummary(lasso)

plot(lasso.fit, main= "LASSO" , xlab = "LAMBDA", ylab = "RMSE(Cross-Validation)")

lasso.fit$results

varImp(lasso.fit)

# A Support Vector Machine (SVM):

set.seed(300)

# tune against the cost C

svmRModel = train(x=X\_train, y=Y\_train, method="svmRadial", preProc=c("center", "scale"), tuneLength=10)

svmRModel

svmRModel$results

plot(svmRModel)

# Lets see what variables are most important:

varImp(svmRModel)

svmRPred = predict(svmRModel, newdata=X\_test)

svmPR = postResample(pred=svmRPred, obs=Y\_test)

svmPR

#Random Forest

install.packages("randomForest")

library(randomForest)

library(rpart)

random.fit <- randomForest(X\_train, Y\_train, control=cforest\_unbiased(mtry=2,ntree=500))

random.fit

importance(random.fit)

varImp(random.fit)

rf\_yHat = predict(random.fit, newdata= X\_test)

## performance evaluation

rfPR = postResample(pred=rf\_yHat, obs=Y\_test)

rfPR

# KNN model

install.packages("caret")

library(caret)

set.seed(100)

knnModel = train(x=X\_train, y=Y\_train, method="knn",preProc=c("center","scale"), tuneLength=10)

knnModel

knnModel$results

# Lets see what variables are most important:

varImp(knnModel)

# plot the RMSE performance against the k

plot(knnModel$results$k, knnModel$results$RMSE, type="o",xlab="# neighbors",ylab="RMSE", main="KNN plot")

# we try the model on the test data

knnPred = predict(knnModel, newdata = X\_test)

knnPred

knnPR = postResample(pred=knnPred, obs = Y\_test)

knnPR

# Decision trees

install.packages("rpart")

library(rpart)

install.packages("party")

library(party)

install.packages("partykit")

library(partykit)

# set up training data

trainData = data.frame( x=X\_train, y=Y\_train )

rPartModel = rpart( y ~ ., data=trainData, method="anova", control=rpart.control(cp=0.01,maxdepth=30))

# tree plotting

rpartTree = as.party(rPartModel)

dev.new()

plot(rpartTree)

# predict test with this regression tree:

rPart\_yHat = predict(rPartModel,newdata=data.frame(x=X\_test))

## performance evaluation

rtPR = postResample(pred=rPart\_yHat, obs=Y\_test)

rtPR

#PLS

library(pls)

set.seed(567)

pls <- train(X\_train, Y\_train, method="pls",tuneLength=10, preProcess= c("zv","center","scale"), trControl= ctrl)

pls$results

# Lets see what variables are most important:

varImp(pls)

plot(pls,main = "Number of Components vs RMSE")

pls$bestTune

summary(pls$resample)

plsPred = predict(pls, X\_test)

postResample(pred = plsPred, obs = Y\_test)

#enet

library(glmnet)

enetGrid <- expand.grid(alpha = c(0, .1, .2, .4, .6, .8, 1), lambda = seq(.01, .1, length = 20) )

set.seed(123)

enetTune <- train(x = X\_train, y = Y\_train,

method = "glmnet",

tuneGrid = enetGrid,

trControl = ctrl)

enetTune

# Lets see what variables are most important:

varImp(enetTune)

enetTune$bestTune

enetTune$results

plot(enetTune, xlab = "Lambda", ylab = "Cross-validated RMSE", main = "ENET")

enetPred = predict(enetTune, X\_test)

postResample(pred = enetPred, obs = Y\_test)

#GBM

gbmGrid <- expand.grid(n.trees = c(100,500), interaction.depth = c(1, 5),

n.minobsinnode = c(10), shrinkage = c(.01, .1))

ctrl <- trainControl(method = "repeatedcv", repeats = 10)

set.seed(476)

gbmFit <- train(x = X\_train,

y = Y\_train,

method = "gbm",

tuneGrid = gbmGrid,

verbose = FALSE,

trControl = ctrl)

gbmFit

gbmFit$results

plot(gbmFit)

plot(gbmFit, type = "s", print.thres = c(.5), print.thres.pch = 3,

print.thres.pattern = "", print.thres.cex = 1.2,add = TRUE, col = "red",

print.thres.col = "red", legacy.axes = TRUE)

gbm\_pred <- predict(gbmFit, X\_test)

gbmValues1 = data.frame(obs = Y\_test, pred = gbm\_pred)

defaultSummary(gbmValues1)