Predicting Full Power Output from a Combined Cycle Power Plant

Team 6: Amin Fesharaki, Benjamin Earnest, and Jeffrey Joyner

Shiley-Marcos School of Engineering, University of San Diego

Jun 27, 2022

Predicting Full Power Output from a CCPP

Abstract

Predicting full load power available from a power plant, based on assumed or forecasted environmental or ambient conditions, has strong implications for improved operational efficiencies and profit from available megawatt hours. This paper conducts a secondary data analysis, utilizing operating data from a combined cycle power plant over a six year period. There are four main parameters (predictors) that influence power plant performance. These parameters include ambient temperature, atmospheric pressure, relative humidity, and steam turbine exhaust pressure (vacuum). The data is used to build both linear and non-linear regression models that predict hourly full load electrical power from the combined cycle power plant. Model performance is measured by mean absolute error (MAE), Root Mean Squared Error (RMSE), and R squared. After analyzing the performance metrics on the test data set for each model used in this study, it was determined that Random Forest tuned by out-of-bag estimates yielded the best results. The final model used had a mtry value of 2 with an RMSE of 3.1709 and an R^2 of .965 on the training data set. However, the performance metrics on the testing data sets had a slightly higher RMSE of 3.7197, a lower R^2 of .952, and an MAE of 2.6460. The Random Forest model tuned by out-of-bag estimates slightly outperformed the most successful model in Tufekci's 2014 study while using the same feature subset. Furthermore, the prediction accuracy for the optimal machine learning model is suitable enough to replace thermodynamic approaches to model a real world system. By doing so, the time and effort to model thermodynamic systems analytics can be greatly reduced. This data was compared to a 2014 study, which found the optimal model to be a bagging algorithm with REPTree (Tufekci, 2014). The optimal model for this study slightly outperformed the 2014 study.

Problem Statement

Combined cycle power plants (CCPP) are thermodynamically complex systems. By definition, they utilize more than one means to generate power, and one of the generators will use waste heat or energy from the other generator as its energy source for electric power generation. This is advantageous due to the efficiencies gained. For our analysis, we revisit a problem first explored by Tufecki (2014), who did a predictive analysis on a combined cycle power plant made up of two gas turbines (GT) and one steam turbine (ST). Figure 1 shows a functional diagram of the combined cycle power plant under consideration. Two GT generators output approximately 160 MW of electrical power each. The exhaust from the GT generators is then run through steam generators as a heat source, creating high pressure steam that is used to power an ST generator.

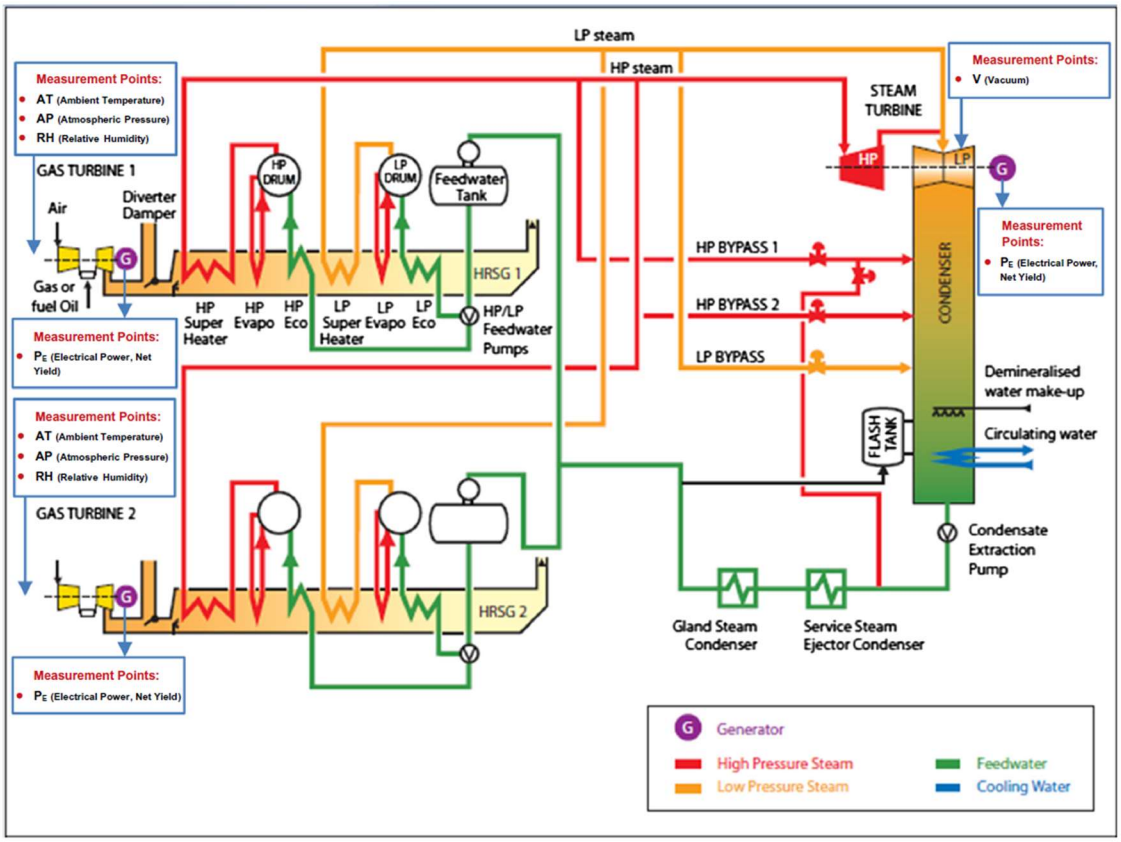


Figure 1 Combined Cycle Power Plant (Tufekci, 2014)

Predicting Full Power Output from a CCPP

Predicting power plant electrical output provides benefits from both an economic and a plant efficiency perspective. The opportunity to plan for and maximize output at least one day prior, as well as income from the available mega-watt hours is a desirable tool. The problem, as Tufekci (2014) found, requires many assumptions to deal with the thermodynamic complexities that come with combined cycle power plants. Without these assumptions, the machine learning problem would be much more challenging, and the thermodynamic analysis would require thousands of non-linear equations (Tufekci, 2014). To avoid these complexities, Tafukci (2014) considered primarily, for GTs, the ambient parameters that impact performance, including ambient temperature, atmospheric pressure, and relative humidity. ST generator performance has a relationship with exhaust vacuum, which will be utilized in this analysis. Our goal for this analysis is to identify the optimal predictive model for full power output from a CCPP, then compare optimal model performance to the study done by Tufekci, and determine if the modeling tools available in 2022 provide opportunities to create more effective predictive models to solve the same problem.

Exploratory Data Analysis

This effort is a secondary data analysis. The data used was downloaded from the UCI Machine Learning Repository. It consists of 5 variables (4 predictors and 1 target) and 9568 observations. The predictor variables include ambient temperature (AT), atmospheric pressure (AP), relative humidity (RH), and steam turbine exhaust vacuum (V). The target variable is full load electrical power output (PE). The observations represent data collected from the CCPP over a six year period, each observation representing hourly measurements over the six year period. Note, each observation represents maximum output of the CCPP, based on the conditions of that day. Table 1 shows the descriptive statistics for the predictor and target variables.

Table 1 *Descriptive Statistics for Predictors and Target Variables*

Characteristic	Descriptive Statistic			
	Mean	Standard Deviation	Range	Median
Ambient Temperature (AT)	19.65	7.45	35.3	20.34
Atmospheric Pressure (AP)	1013.3	5.93	40.4	1012.9
Relative Humidity (RH)	73.31	14.60	74.6	74.97
Turbine Exhaust Pressure (V)	54.31	12.71	56.2	52.08
Electrical Power Output (PE)	454.4	17.07	75.5	451.6

Note. Units are AT(degrees C), AP (mbar), RH (%), V (cm Hg), PE (MW)

Our first step in exploring the data set was to check for any missing values that may require removal, imputation, or other actions to ensure they don't inappropriately influence or interfere with model development. We found there were no missing values. Further, we also found that the data in the predictor and target variable columns were numerical and continuous. This lends itself well to regression models.

Now that we understand the structure and quality of the data, we looked for relationships between variables, investigated the distributions, and looked for outliers that may influence model training. Figure 2 shows the pairwise scatter plots of each predictor and the target variable.

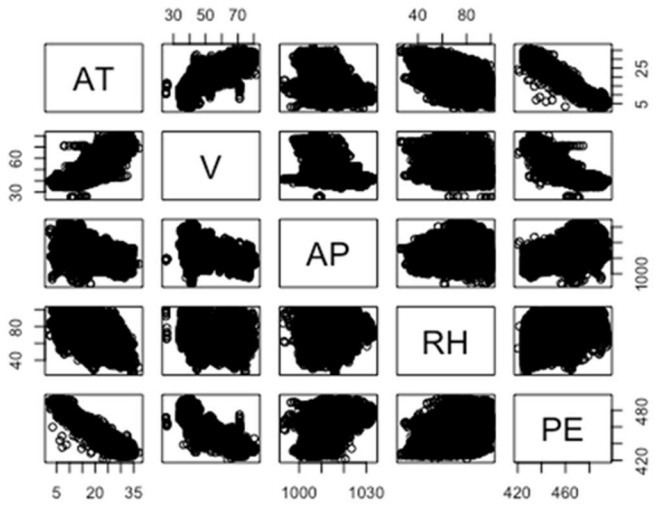


Figure 2 Pairwise scatter plots of variables

This shows linear relationships may exist between the PE and AT, as well as PE and V. We verified this by checking the correlation between all variables. The correlations were calculated to be -0.94 between PE/AT, and -0.87 between PE/V. We also checked the distribution of the variables, using histograms, and looked for outliers using box plots. Figure 3 shows the histograms of the predictor variables. They generally follow a normal distribution, and do not appear skewed. Vacuum (V) shows the least normal distribution, and will need to be addressed in preprocessing.

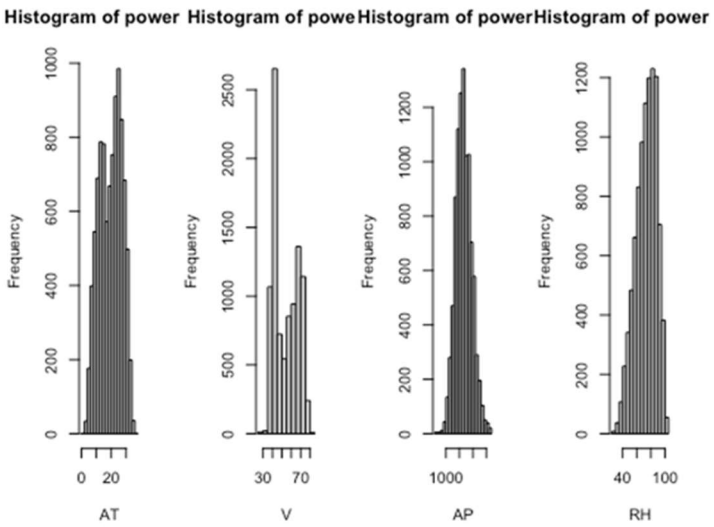


Figure 3 Distribution of predictor variables

The boxplots in figure 4 show very few outliers, but make it clear the variables are on very different scales, which will also need to be dealt with during preprocessing, before data splitting and model training.

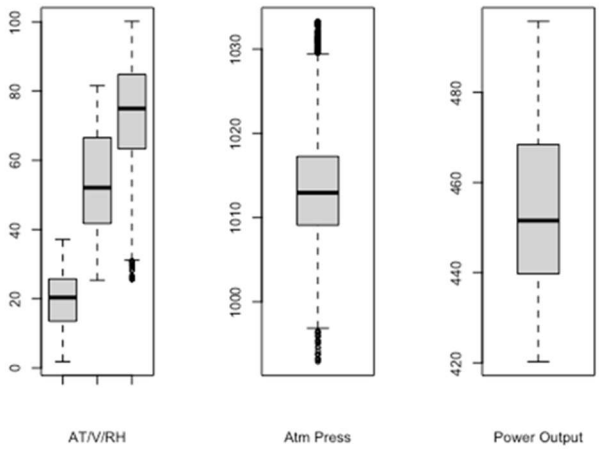


Figure 4 Box plots of all variables

Data Pre-Processing

To preprocess, we first checked for any near zero variance (degenerate) predictors that may not add value to the model training. We used the nearZeroVar() function in R, passed in the data set, and

found there are not any degenerate predictors. Next we checked for any highly correlated predictors, and found that AT and RH had a fairly strong correlation. So strong we could potentially remove AT as a predictor. This was confirmed by principal component (PC) analysis, which identified three PC's that account for 95% of variance in the target. However, the 2014 study included all of the predictors in the optimal model, so we also will for comparison. We will investigate variable importance later on for the optimal model to validate whether AT should stay, or recommend removing in future analysis.

The predictors were centered, scaled, and transformed using the Box Cox method, to prepare the data for model training. This is done to improve numerical stability for many of the modeling calculations. Many of the linear regression models are sensitive to the scale of predictors, while others, like tree based methods, are not. The only real downside to centering, scaling, and transforming is the loss of interpretation of the data, as it will no longer be in its original units (Kuhn & Johnson, 2013). The output can be seen in the new boxplots, which show the data each having normal distributions and on the same scale in figure 5.

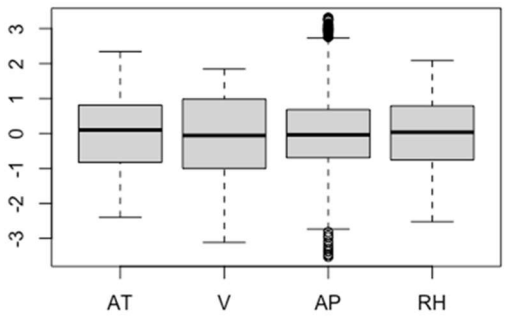


Figure 5 Box plots of centered, scaled, and transformed predictor variables

Data Splitting

The data was split into training and test data sets, first by separating the predictors into "x" or independent variables, then the target variable (PE) as the "y" or independent variable. The predictors

were centered, scaled, and transformed after partitioning into training and test data sets. We used the createDataPartition() function in R to create an 80/20 split of the data, where 80% of the 9568 observations were partitioned into the training set, and the remaining 20% were set aside as the test data set.

Modeling Strategies

A mix of regression modeling strategies was implemented for comparison of best model performance according to the lowest Root Mean Square Error (RMSE), Mean Absolute Error (MAE), highest R-squared values. Our goal was to produce a model that would best predict power output given our predictor variables and to see which predictor variables have the most influence on the full power output of the CCPP.

With the data now pre-processed and partitioned, the training and testing variables created within the R-language code were incorporated into the various models for computation and analysis. We estimated the level of performance of the models by analyzing how it performed on data it has not yet seen, which is our test set of data. We used 10-fold cross validation as the resampling method across all models, in which data were partitioned into a training set of data and a test set of data. A random number seed of 100 was also used across all models for reproducibility of results. The algorithms used the training set of data to train the various machine learning models and to find their parameters. Then each model's performance against the test set of data was evaluated.

We also intended to evaluate if linear or non-linear models perform better. If the data were linear then linear regression would have outperformed the non-linear models. On the contrary, if the data were non-linear, the non-linear models would outperform the linear regression model.

Validation and Testing

Several linear regression models were fitted to the data via R programming language in an attempt to match the pattern of the relationship between the predictor variables and the response variable. This category of models works best when the nature of the relationship between the predictor variables and the response variable is linear in nature, or rather, can be represented by a straight line on a graph whose algebraic equation uses the variable to the power of 1. They work by finding the straight line that produces the smallest amount of RSME, or x-axis distance between the data points and the regression line. Linear regression models tried were Ordinary Least Squares (OLS), Ridge, Lasso, ElasticNet, Partial Least Squares (PLS), Principal Component Regression (PCR), and OLS Principal Component Analysis (PCA). The linear model function was used as the training method for OLS and OLS PCA models. With the linear regression function we trained the model on the training data for the OLS model. We then used the resulting tuning variable to rank the variables in order of importance. For the Ridge model, the penalty variable of lambda was the tuning parameter. We made a grid data frame of every combination of variables to use for tuning in the Ridge, ElasticNet, PLS and PCR models.

Non-linear regression models used to model the data were Support Vector Machines (SVM) Radial, K-Nearest Neighbor (KNN), Neural Network, Multivariate Adaptive Regression Splines (MARS), and SVM Polynomial. The “earth” package was used to train the MARS model. The train function was used to tune the MARS model via external resampling. We were also able to rank the variables in terms of importance according to the MARS model. For the KNN model, the k value was used for tuning grind in the train function. For SVM Radial and SVM Polynomial, the tuneLength argument uses 14 cost values as a default.

Tree Regression models, which use multiple, long series of if-then statements for predictor variable values and corresponding response variable outcome values, were also applied to the data to determine if this style of modeling would produce results superior to those of the non-linear models.

The Tree Regression models used on the data were Random Forest, Boosting, Cubist, Classification and Regression Tree (CART), and Bagged. With the Random Forest model we defined the number of trees. Similarly in the Bagged Trees model we defined the number of bags.

Results and Final Model Selection

Prediction accuracy using root mean-squared error (RMSE) will indicate the best performance models. RMSE is used to measure the difference between values predicted by the model and the values that are observed. In addition, 10-fold cross validation was used for all models. Moreover, linear, nonlinear, and tree regression models were used to analyze the data. With respect to the lower RMSE score, tree and nonlinear regression models outperformed the linear regression models. The only exception was the bagged tree model as it was ranked lower than five linear models. Linear models include ordinary least squares, partial least squares, principal component regression, and three penalized linear models: Lasso, Ridge, and Elastic Net. Support vector machines (radial and poly), multivariate adaptive regression splines, K-Nearest neighbors, and neural network made up the nonlinear regression models. Lastly, the tree regression models used were CART, random forest, boosting, bagged, and cubist.

Table 2

Performance Metrics On Test Set For All Models

Table 2

Model	Type	RMSE	R^2	MAE	
Random Forest	Tree	3.8025	0.9501	2.7439	
SVM Radial	Nonlinear	4.0610	0.9430	2.9680	
KNN	Nonlinear	4.0612	0.9429	2.8716	
Boosting	Tree	4.0679	0.9426	2.9963	
Neural Network	Nonlinear	4.1510	0.9402	3.1339	
MARS	Nonlinear	4.3579	0.9342	3.3181	
Cubist	Tree	4.3640	0.9342	3.0827	
SVM Poly	Nonlinear	4.4659	0.9309	3.4597	
CART	Tree	4.5106	0.9295	3.4101	
OLS	Linear	4.6642	0.9247	3.6306	
Ridge	Linear	4.6642	0.9247	3.6306	
ElasticNet	Linear	4.6642	0.9247	3.6306	
Lasso	Linear	4.7216	0.9230	3.6852	
PLS	Linear	4.9263	0.9160	3.8140	
Bagged	Tree	5.1254	0.9091	3.9202	
PCR	Linear	5.5115	0.8947	4.2382	
OLS PCA	Linear	32.640	0.8962	29.075	

Table 2 indicates RMSE, R^2, and MAE for each model. The range of the target variable (electrical power output) is 75.5 with a mean of 454.4. This indicates that all of the models have suitable RMSE values with the exception of the Ordinary Least Squares model using Principal Component Analysis. As shown in the table, nonlinear models outperform linear models, which indicates that the data should be considered nonlinear. If the data were linear, the linear models would outperform the other models. Moreover, Ordinary Least Squares yielded the best linear regression with an RMSE score of 4.664. The best nonlinear regression was the Support Vector Machine (SVM) with radial basis function kernel and a RMSE score of 4.061. For the SVM model, the tuning parameter 'sigma' was held constant at a value of 0.4113507 with a C of 4. However, the best performing model was Random Forest which had the lowest RMSE score of 3.803. The data concludes that Random Forest models achieved the lowest RMSE and highest R^2. Therefore, two additional models were built to see what variation of Random Forest was

most optimal. The original model had no tuning and was set at 500 trees with 1 variable tried at each split.

Table 3

Performance Metrics On Test Set For Random Forest Models

Table 3

Random Forest Model	RMSE	R Squared	MAE	
Tuned Model Using Out-of-Bag Estimates	3.7197	0.9521	2.6460	
Tuned Model Using Cross Validation	3.7257	0.9519	2.6488	
Original Model	3.8027	0.9501	2.7439	

In addition, the Random Forest model was tuned using out-of-bag (OOB) estimates and using cross validation. Among all three Random Forest models, the one tuned using out-of-bag had the lowest RMSE with the highest R^2. Performance metric values on the test set are shown in Table **3.** The OOB model was tuned to find the optimal mtry value; the final model used 500 trees with 2 variables tried at each split. In addition, the model explained 96.56% variance using these parameters.

Table 4

Variable Importance for Optimal Model - Random Forest (OOB)

Table 4

Predictor	Overall Importance
Relative Humidity	105.554
Ambient Temperature	70.8531
Ambient Pressure	52.3890
Turbine Exhaust Pressure	40.1479

Variable importance gives insights on how much a specific variable influences the model to make accurate predictions. Table **4** indicates the predictor variable importance used in the out-of-bag tuned Random Forest mode: in descending order, relative humidity with an importance of 105; ambient temperature with 70; ambient temperature with 52; and ambient pressure with 40. Therefore, we can assume that relative humidity was the most influential predictor used in the model.

Discussion and Conclusions

Machine learning models can be implemented to replace the need to incorporate thermodynamic approaches. Thermodynamic approaches often use assumptions in real life application in order to use appropriate nonlinear equations relating to the system. Therefore, machine learning models can create easier and more effective predictions by removing the obstacles surrounding assumptions and nonlinear equations when modeling the system.

The goal of this study is to improve upon Tufekci's study on the prediction of electrical power using machine learning methods. Tufekci's study concluded that the most successful model was bagging REPtree (using all predictors) with a RMSE of 3.787 and MAE of 2.818. To achieve this goal, the study focuses on finding the best model with a secondary objective of determining the predictor impact on the model. The most successful model will be regarded as the model with the lowest RMSE value. After analyzing the performance metrics on the test data set for each model used in this study, it was determined that Random Forest tuned by out-of-bag estimates yielded the best results. The final model used had a mtry value of 2 with an RMSE of 3.1709 and an R^2 of .965 on the training data set. However, the performance metrics on the testing data sets had a slightly higher RMSE of 3.7197, a lower R^2 of .952, and an MAE of 2.6460. The range of electrical power output (75.5) and mean (454.4) indicates that the RMSE value is suitable for accurate model prediction.

Furthermore, each model included all four predictors to obtain the best model. In addition, the predictor variable importance, which gives insight on how influential each predictor is, was calculated for each of the predictors in the optimal model. Among all the predictors, relative humidity had the highest importance by a significant amount, followed by ambient temperature, ambient pressure, and turbine exhaust pressure. With this insight, plant operators can potentially focus on manipulating the relative humidity and ambient temperature to reach a desired electrical output level.

In conclusion, the Random Forest model tuned by out-of-bag estimates slightly outperformed the most successful model in Tufekci's 2014 study while using the same feature subset. Furthermore, the prediction accuracy for the optimal machine learning model is suitable enough to replace thermodynamic approaches to model a real world system. By doing so, the time and effort to model thermodynamic systems analytics can be greatly reduced. Additionally, eliminating assumptions can improve prediction accuracy as well. Future works on this study should include different tuned models in efforts to perfect the ability to predict electrical output. As time goes on, better models are introduced to the data science industry, which can lead to better results than what the model provided in this study. In addition, data should be gathered for different types of power plants with similar unit designs since different unit designs can drastically affect the predictor values.

-

References

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. New York: Springer.

Tufekci, P. (2014). Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *Electrical Power and Energy Systems,* 60, 126-140.

Appendix

ADS 503 Team 6

Team 6

6/25/2022

Install Librarys:

```
library(caret)

## Loading required package: ggplot2

## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'tibble'

## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'

## Loading required package: lattice

library(Hmisc)

## Warning: package 'Hmisc' was built under R version 4.1.2

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: Formula

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, units

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:Hmisc':
##
##     src, summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(gbm)

## Loaded gbm 2.1.8

library(lars)

## Warning: package 'lars' was built under R version 4.1.2

## Loaded lars 1.3

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin

library(AppliedPredictiveModeling)
library(rpart)
library(rpart.plot)
library(partykit)

## Loading required package: grid

## Loading required package: libcoin

## Loading required package: mvtnorm

library(Cubist)
library(partykit)

power <- read.csv('/Users/datascience/Desktop/Project/Power_Plant_DS.csv')
```

Load the data set

Inspect for NA's, structure of the data, and data frame dimentions. Also load as data frame since the data set was downloaded as an excel spreadsheet.

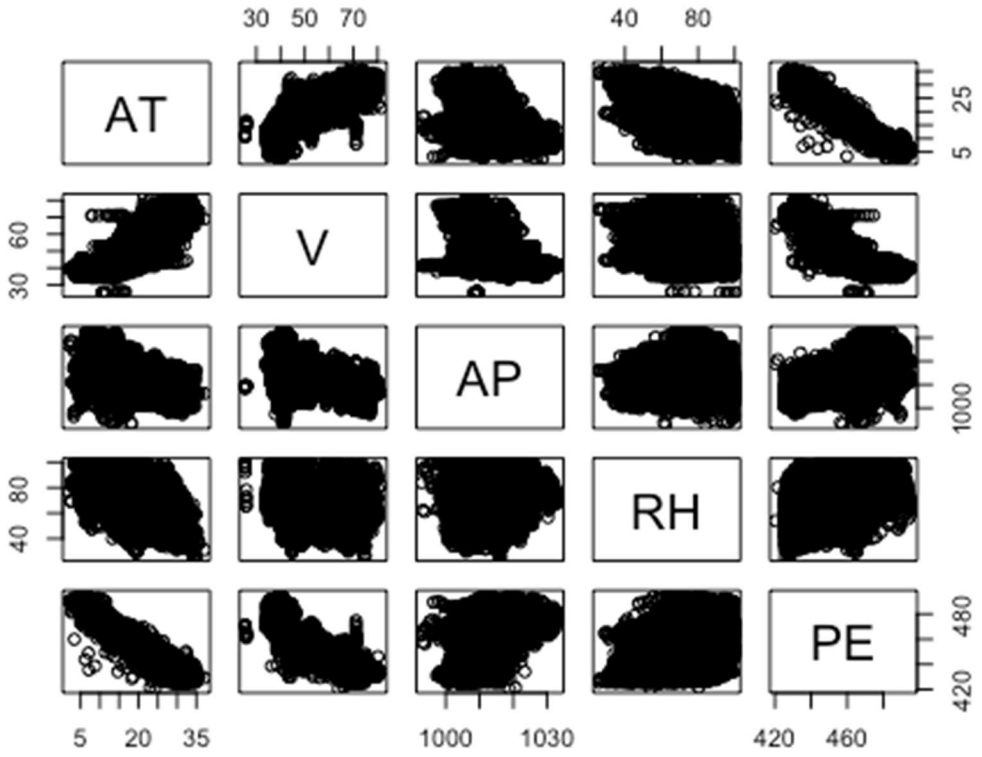
```
#power <- data.frame(Power_Plant_DS)
head(power)

##      AT     V      AP    RH     PE
## 1 14.96 41.76 1024.07 73.17 463.26
## 2 25.18 62.96 1020.04 59.08 444.37
## 3  5.11 39.40 1012.16 92.14 488.56
## 4 20.86 57.32 1010.24 76.64 446.48
```

```
## 5 10.82 37.50 1009.23 96.62 473.90
## 6 26.27 59.44 1012.23 58.77 443.67

sum(is.na(power))

## [1] 0

summary(power)

##        AT               V               AP               RH
##  Min.   : 1.81   Min.   :25.36   Min.   : 992.9   Min.   : 25.56
##  1st Qu.:13.51   1st Qu.:41.74   1st Qu.:1009.1   1st Qu.: 63.33
##  Median :20.34   Median :52.08   Median :1012.9   Median : 74.97
##  Mean   :19.65   Mean   :54.31   Mean   :1013.3   Mean   : 73.31
##  3rd Qu.:25.72   3rd Qu.:66.54   3rd Qu.:1017.3   3rd Qu.: 84.83
##  Max.   :37.11   Max.   :81.56   Max.   :1033.3   Max.   :100.16
##        PE
##  Min.   :420.3
##  1st Qu.:439.8
##  Median :451.6
##  Mean   :454.4
##  3rd Qu.:468.4
##  Max.   :495.8

dim(power)

## [1] 9568    5

str(power)

## 'data.frame':    9568 obs. of  5 variables:
##  $ AT: num  14.96 25.18 5.11 20.86 10.82 ...
##  $ V : num  41.8 63 39.4 57.3 37.5 ...
##  $ AP: num  1024 1020 1012 1010 1009 ...
##  $ RH: num  73.2 59.1 92.1 76.6 96.6 ...
##  $ PE: num  463 444 489 446 474 ...
```

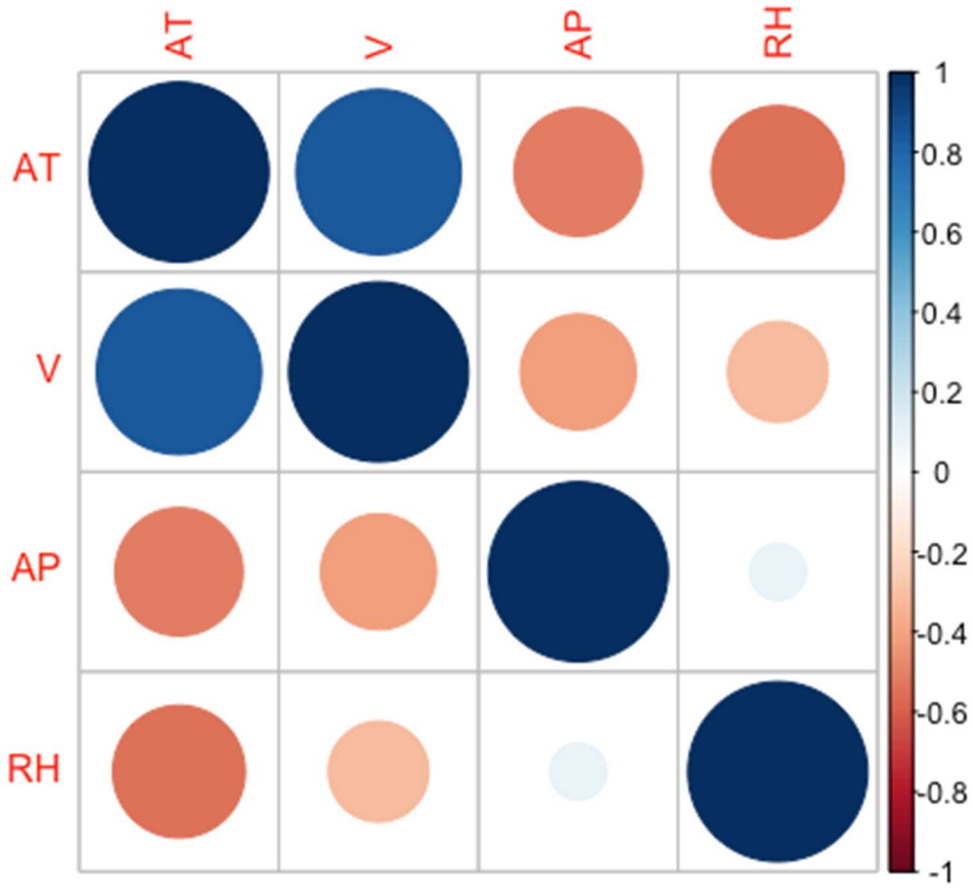
Exploratory Data Analysis:

Check pairwise distributions and check for correlations with the predictor variables.

```
pairs(power)
```

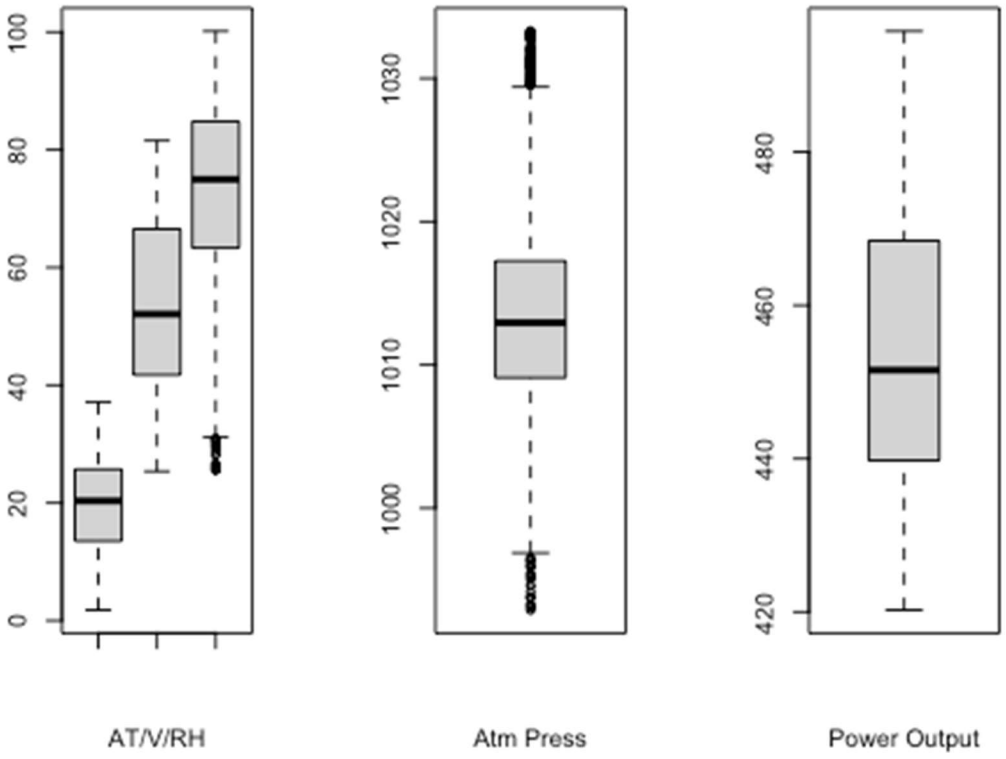



```
corrplot::corrplot(cor(power[,-5]))
```



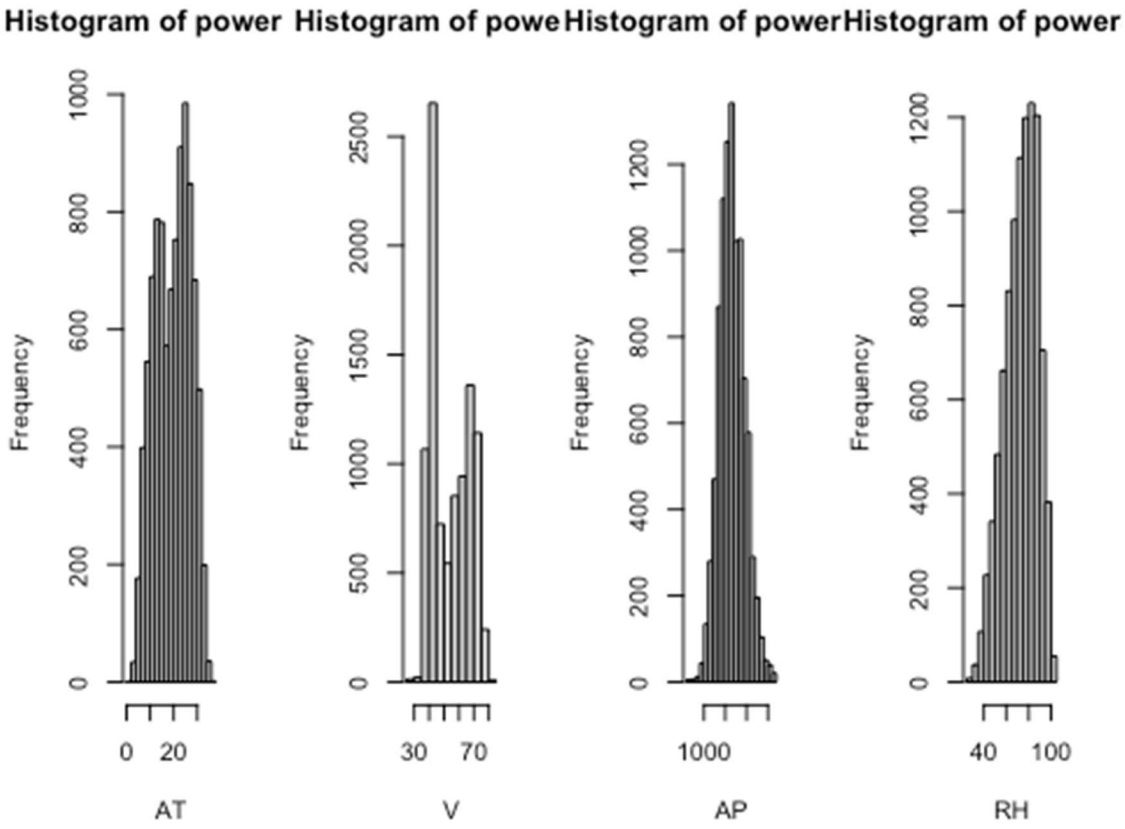
##Look for outliers via box plots:

```
par(mfrow = c(1,3))
boxplot(power$AT, power$V, power$RH, xlab = "AT/V/RH")
boxplot(power$AP, xlab = "Atm Press")
boxplot(power$PE, xlab = "Power Output")
```



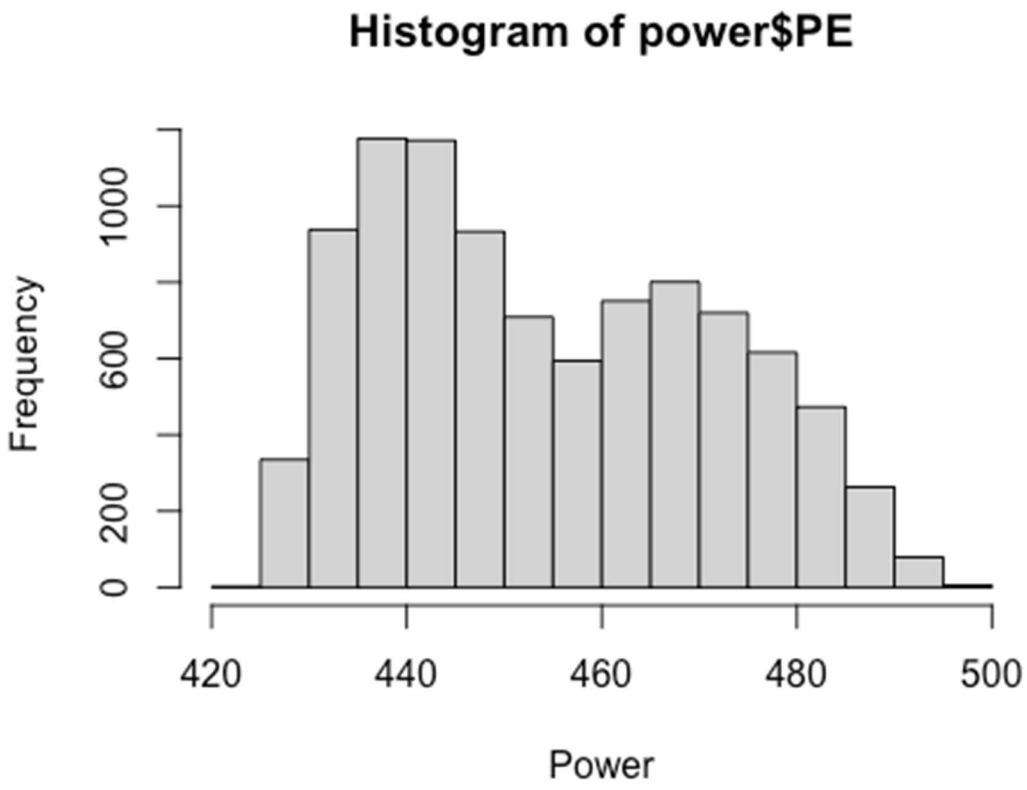
Build histograms of predictor variables to check the distributions of the predictor variables for skewness:

```
par(mfrow = c(1,4))
hist(power$AT, xlab = "AT")
hist(power$V, xlab = "V")
hist(power$AP, xlab = "AP")
hist(power$RH, xlab = "RH")
```



Histogram of the target variable, which is full power output at different operating conditions:

```
hist(power$PE, xlab = "Power")
```



Data Preprocessing

Check for zero variance in the columns, suggesting which could be removed:

```
degeneratecols <- nearZeroVar(power)
degeneratecols

## integer(0)
```

Look for highly correlated predictors and create a filtered data set that removes them. In this case, AT was highly correlated to both the response (PE) and V, so AT was filtered out. The filtered data set will be kept for comparison to models that are unfiltered.

```
correlations <- cor(power[,-5])
highCorr <- findCorrelation(correlations, cutoff = 0.75)
length(highCorr)

## [1] 1

head(highCorr)

## [1] 1

correlations
```

```
##            AT          V          AP          RH
## AT  1.0000000  0.8441067 -0.50754934 -0.54253465
## V   0.8441067  1.0000000 -0.41350216 -0.31218728
## AP -0.5075493 -0.4135022  1.00000000  0.09957432
## RH -0.5425347 -0.3121873  0.09957432  1.00000000
```

```
filtered <- power[,-highCorr]
head(filtered)
```

```
##       V      AP    RH     PE
## 1 41.76 1024.07 73.17 463.26
## 2 62.96 1020.04 59.08 444.37
## 3 39.40 1012.16 92.14 488.56
## 4 57.32 1010.24 76.64 446.48
## 5 37.50 1009.23 96.62 473.90
## 6 59.44 1012.23 58.77 443.67
```

Convert PE target to “Derated”, “Nominal”, and “High” output ordinal values for other modeling options. We can modify the bins based on the distribution later.

```
filtered_ord <- filtered



PE_ord <- case_when(filtered_ord$PE <= 440 ~ 'Derated',
                    between(filtered_ord$PE, 440, 480) ~ 'Nominal',
                    filtered_ord$PE >= 480 ~ 'High'
                    )
PE_ord <- as.factor(PE_ord)
power1 <- cbind(power, PE_ord)

table(power$PE_ord)
```

```
## < table of extent 0 >
```

Create the training/test split prior to pre-processing the data:

```
set.seed(100)
trainingRows <- createDataPartition(power1$PE, p = .8, list = FALSE)

powerXTrain <- power1[trainingRows,]
powerXTest <- power1[-trainingRows,]

powerYTrain <- powerXTrain$PE
powerYTest <- powerXTest$PE

powerYTrain_ord <- powerXTrain$PE_ord
powerYTest_ord <- powerXTest$PE_ord

table(powerXTrain$PE_ord)
```

```
##
## Derated    High Nominal
##    1962     659    5035
```

```
table(powerXTest$PE_ord)
```

```
##
## Derated    High Nominal
##     490     161    1261
```

```
head(powerXTrain)
```

```
##      AT     V      AP    RH     PE  PE_ord
## 2 25.18 62.96 1020.04 59.08 444.37 Nominal
## 3  5.11 39.40 1012.16 92.14 488.56    High
## 4 20.86 57.32 1010.24 76.64 446.48 Nominal
## 5 10.82 37.50 1009.23 96.62 473.90 Nominal
## 6 26.27 59.44 1012.23 58.77 443.67 Nominal
## 9 14.64 45.00 1021.78 41.25 475.98 Nominal
```

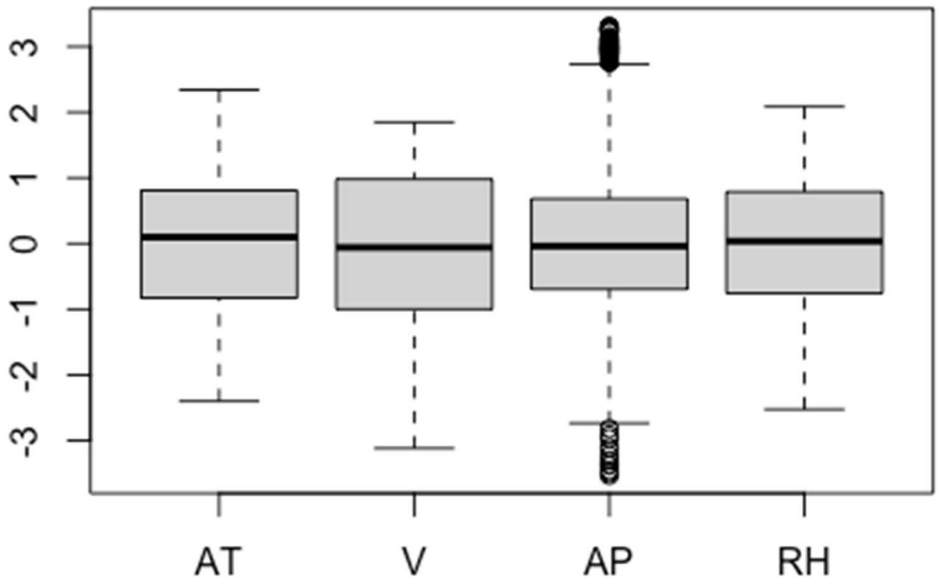
Transform Data, include Principal Component Analysis:

```
#preprocess (normalize, center, scale):
transXTrain <- preProcess(powerXTrain[,1:4], method = c("BoxCox", "center", "scale"))
transXTrain

## Created from 7656 samples and 4 variables
##
## Pre-processing:
##   - Box-Cox transformation (4)
##   - centered (4)
##   - ignored (0)
##   - scaled (4)
##
## Lambda estimates for Box-Cox transformation:
## 1, 0, -2, 1.8

transXTest <- preProcess(powerXTest[,1:4], method = c("BoxCox", "center", "scale"))
transXTest

## Created from 1912 samples and 4 variables
##
## Pre-processing:
##   - Box-Cox transformation (4)
##   - centered (4)
##   - ignored (0)
##   - scaled (4)
##
## Lambda estimates for Box-Cox transformation:
## 0.9, -0.1, -2, 1.9

# preprocess with pca:
transXTrain_pca <- preProcess(powerXTrain[,1:4], method = c("BoxCox", "center", "scale", "pca"
))
transXTrain_pca

## Created from 7656 samples and 4 variables
##
## Pre-processing:
##   - Box-Cox transformation (4)
##   - centered (4)
##   - ignored (0)
##   - principal component signal extraction (4)
##   - scaled (4)
##
## Lambda estimates for Box-Cox transformation:
## 1, 0, -2, 1.8
## PCA needed 3 components to capture 95 percent of the variance
```

```
transXTest_pca <- preProcess(powerXTest[,1:4], method = c("BoxCox", "center", "scale", "pca"))
transXTest_pca

## Created from 1912 samples and 4 variables
##
## Pre-processing:
##   - Box-Cox transformation (4)
##   - centered (4)
##   - ignored (0)
##   - principal component signal extraction (4)
##   - scaled (4)
##
## Lambda estimates for Box-Cox transformation:
## 0.9, -0.1, -2, 1.9
## PCA needed 3 components to capture 95 percent of the variance
```

Apply the transformation:

```
powerTrain_Xtrans <- predict(transXTrain, powerXTrain[,1:4])
head(powerTrain_Xtrans)

##           AT          V         AP        RH
## 2  0.7384096  0.7483214  1.1525420 -1.005857
## 3 -1.9567102 -1.2442877 -0.1674296  1.382188
## 4  0.1582942  0.3493607 -0.4937359  0.158416
## 5 -1.1899372 -1.4543952 -0.6661346  1.769097
## 6  0.8847814  0.5037494 -0.1555680 -1.024224
## 9 -0.6769647 -0.6793378  1.4398993 -1.933836

dim(powerTrain_Xtrans)

## [1] 7656    4

boxplot(powerTrain_Xtrans)
```



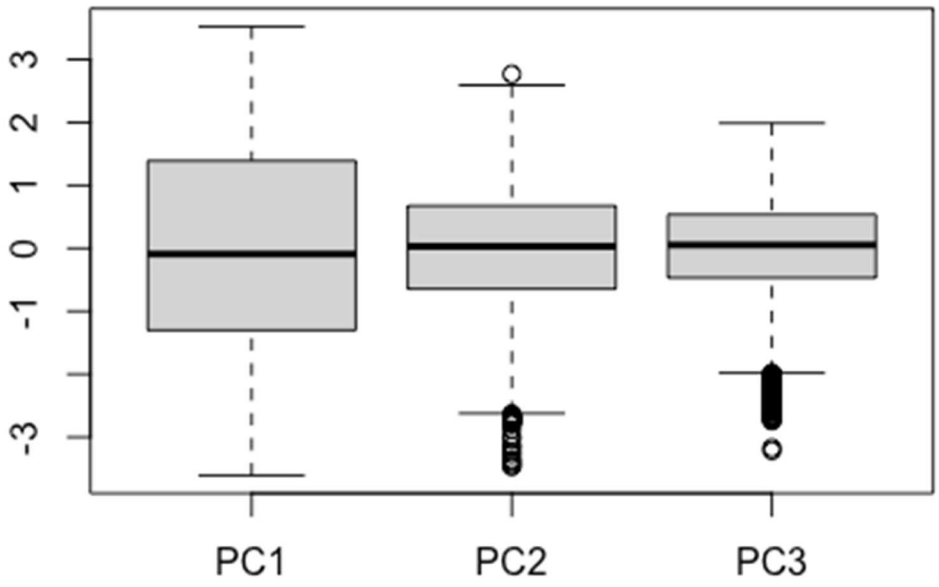

```
powerTrain_Xtrans_pca <- predict(transXTrain_pca, powerXTrain[,1:4])
head(powerTrain_Xtrans_pca)
```

```
##          PC1       PC2          PC3
## 2 -0.7937461  1.473832  0.798359347
## 3  2.3617878 -1.118876 -0.499418143
## 4 -0.4308583 -0.466138  0.004503088
## 5  1.9542278 -1.684169 -0.625252142
## 6 -1.2809218  0.658879 -0.144735707
## 9  0.6362357  2.423267 -0.551797152
```

```
dim(powerTrain_Xtrans_pca)
```

```
## [1] 7656    3
```

```
boxplot(powerTrain_Xtrans_pca)
```




```
powerTest_Xtrans <- predict(transXTest, powerXTest[,1:4])
head(powerTest_Xtrans)
```

```
##            AT          V         AP          RH
## 1  -0.6114519 -0.9923359  1.7586415 -0.11176192
## 7  -0.4870526 -0.7773508  0.0977769  0.03881562
## 8  -1.3444720 -0.7065126  0.9467441 -0.57575966
## 17 -0.1767229 -0.6794399  1.5612657 -1.59466223
## 18 -1.1358021 -0.9943418  1.5187629  0.20828132
## 23 -1.5745440 -0.9405161 -0.8322486  0.66180488
```

```
dim(powerTest_Xtrans)
```

```
## [1] 1912    4
```

```
boxplot(powerTest_Xtrans)
```

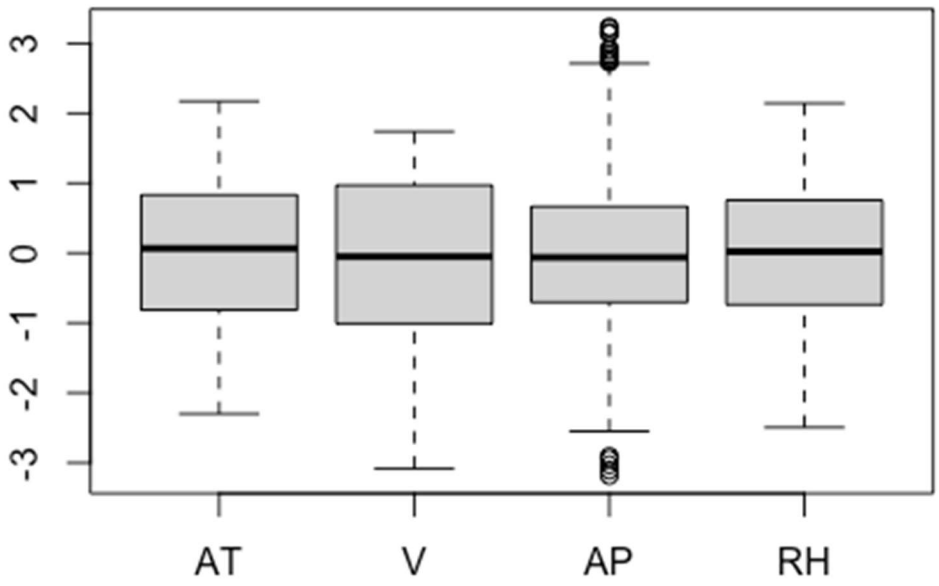
```
powerTest_Xtrans_pca <- predict(transXTest_pca, powerXTest[,1:4])
head(powerTest_Xtrans)
```

```
##            AT          V         AP          RH
## 1  -0.6114519 -0.9923359  1.7586415 -0.11176192
## 7  -0.4870526 -0.7773508  0.0977769  0.03881562
## 8  -1.3444720 -0.7065126  0.9467441 -0.57575966
## 17 -0.1767229 -0.6794399  1.5612657 -1.59466223
## 18 -1.1358021 -0.9943418  1.5187629  0.20828132
## 23 -1.5745440 -0.9405161 -0.8322486  0.66180488
```

```
dim(powerTest_Xtrans)
```

```
## [1] 1912    4
```

```
boxplot(powerTest_Xtrans)
```



Linear Regression Models

Linear Regression model

```
indx <- createFolds(powerYTrain, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)

set.seed(100)
lmTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                method = "lm",
                trControl = ctrl)

#LM Predictions
testResults <- data.frame(obs = powerYTest,
                          Linear_Regression = predict(lmTune, powerTest_Xtrans))
```

Linear Regression model using principal components from preprocessing:

```
indx <- createFolds(powerYTrain, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)
```

```
set.seed(100)
lmTune_pca <- train(x = powerTrain_Xtrans_pca, y = powerYTrain,
                    method = "lm",
                    trControl = ctrl)

testResults$Linear_Regression_pca <- predict(lmTune_pca, powerTest_Xtrans_pca)
```

PCR

```
#PCR Tune
set.seed(100)
pcrTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                    method = "pcr",
                    tuneLength =  30,
                    trControl = ctrl)

#PCR Prediction
testResults$pcr <- predict(pcrTune, powerTest_Xtrans)
```

PLS

```
#PLS Tune
set.seed(100)
plsTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                    method = "pls",
                    tuneLength =  30,
                    trControl = ctrl)

#PLS Prediction
testResults$pls <- predict(plsTune, powerTest_Xtrans)
```

Penalized Linear Models

Lasso

```
# Lasso
set.seed(100)

LassoTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                   method = "lasso",
                   trControl = ctrl,
                   preProc = c("center", "scale"))
#Lasso Prediction
testResults$lasso <- predict(LassoTune, powerTest_Xtrans)
```

Ridge Model:

```
set.seed(100)
ridgeGrid <- expand.grid(lambda = seq(0, .1, length = 15))

ridgeTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                   method = "ridge",
                   tuneGrid = ridgeGrid,
                   trControl = ctrl,
                   preProc = c("center", "scale"))
```

```
testResults$Ridge <- predict(ridgeTune, powerTest_Xtrans)
```

Elastic Net

```
indx <- createFolds(powerYTrain, returnTrain = TRUE)

ctrl <- trainControl(method = "cv", index = indx)

enetGrid <- expand.grid(lambda = c(0, 0.01, .1),
                        fraction = seq(.05, 1, length = 20))
set.seed(100)
enetTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                  method = "enet",
                  tuneGrid = enetGrid,
                  trControl = ctrl,
                  preProc = c("center", "scale"))

testResults$enet <- predict(enetTune, powerTest_Xtrans)
```

Non Linear Regression Models

MARS

```
ctrl <- trainControl(method = "cv", index = indx)

set.seed(100)
marsTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                  method = "earth",
                  tuneGrid = expand.grid(degree = 1, nprune = 2:38),
                  trControl = ctrl)

## Loading required package: earth

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos

##
## Attaching package: 'TeachingDemos'

## The following objects are masked from 'package:Hmisc':
##
##     cnvrt.coords, subplot

testResults$MARS <- predict(marsTune, powerTest_Xtrans)
```

Support Vector Machine:

```
set.seed(100)
svmRTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                  method = "svmRadial",
                  preProc = c("center", "scale"),
                  tuneLength = 5,
                  trControl = ctrl)
testResults$SVM <- predict(svmRTune, powerTest_Xtrans)
```

```
svmGrid <- expand.grid(degree = 1:2,
                       scale = c(0.01, 0.005, 0.001),
                        C = 2^(-2.5))

set.seed(100)
svmPTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                  method = "svmPoly",
                  preProc = c("center", "scale"),
                  tuneGrid = svmGrid,
                  trControl = ctrl)

testResults$svmPTune <- predict(svmPTune, powerTest_Xtrans)
```

KNN

```
set.seed(100)

knnTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                 method = "knn",
                 preProc = c("center", "scale"),
                 tuneGrid = data.frame(k = 1:20),
                 trControl = ctrl)

testResults$Knn <- predict(knnTune, powerTest_Xtrans)
```

Neural Network

```
#Neural Network Tune
set.seed(100)
nnetGrid <- expand.grid(decay = c(0, .1, 1),
                        size = c(3, 6, 12, 15))
MaxSize <- max(nnetGrid$size)
nwts <- 1*(MaxSize*(length(powerTrain_Xtrans)+1) + MaxSize + 1) #For MaxNWTS

nnetTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                  method = "nnet",
                  tuneGrid = nnetGrid,
                  trControl = ctrl,
                  preProc = c("center", "scale"),
                  linout = TRUE,
                  trace = FALSE,
                  MaxNWts = nwts,
                  maxit = 1000)
#Neural Prediction
testResults$neural_net <- predict(nnetTune, powerTest_Xtrans)
```

Regression Trees

Random Forest

```
#Random Forest
set.seed(100)
rfModel <- randomForest(powerTrain_Xtrans, powerYTrain,
                        importance = TRUE,
                        ntrees = 1000)
#Random Forest Prediction
testResults$randomforest <- predict(rfModel, powerTest_Xtrans)
```

Bagged Tres

```
set.seed(100)

### Bagged Trees

treebagTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                     method = "treebag",
                     nbagg = 25,
                     trControl = ctrl)

#Tree Bag Prediction
testResults$treebag <- predict(treebagTune, powerTest_Xtrans)
```

CART

```
set.seed(100)
ctrl <- trainControl(method = "cv", index = indx)
cartTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                  method = "rpart",
                  tuneLength = 25,
                  trControl = ctrl)

### Save the test set results in a data frame
testResults$cart <-  predict(cartTune, powerTest_Xtrans)
```

#Boosted Tree

```
gbmGrid <- expand.grid(interaction.depth = seq(1, 7, by = 2),
                       n.trees = seq(100, 500, by = 50),
                       shrinkage = c(0.01, 0.1),
                       n.minobsinnode = 10)

set.seed(100)
gbmTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                 method = "gbm",
                 tuneGrid = gbmGrid,
                 trControl = ctrl,
                 verbose = FALSE)
testResults$gbm <- predict(gbmTune, powerTest_Xtrans)
```

Cubist

```
#Cubist
set.seed(100)
cubistModel <- cubist(powerTrain_Xtrans, powerYTrain)
#Cubist Prediction
testResults$cubist <- predict(cubistModel, powerTest_Xtrans)
```

Results

```
#Calculate RMSE, Rsquared, and MAE
set.seed(100)

# Linear Models
OLS <- postResample(pred = testResults$Linear_Regression, obs = testResults$obs)
OLS_PCA <- postResample(pred = testResults$Linear_Regression_pca, obs = testResults$obs)
```

```
PCR <- postResample(pred = testResults$pcr, obs = testResults$obs)
PLS <- postResample(pred = testResults$pls, obs = testResults$obs)

# Penalized Linear Models
Lasso <- postResample(pred = testResults$lasso, obs = testResults$obs)
Ridge <- postResample(pred = testResults$Ridge, obs = testResults$obs)
ElasticNet <- postResample(pred = testResults$enet, obs = testResults$obs)

# Non Linear Regression Models
MARS <- postResample(pred = testResults$MARS, obs = testResults$obs)
SVM <- postResample(pred = testResults$SVM, obs = testResults$obs)
SVM_Tune <- postResample(pred = testResults$svmPTune, obs = testResults$obs)
KNN <- postResample(pred = testResults$Knn, obs = testResults$obs)
NeuralNetwork <- postResample(pred = testResults$neural_net, obs = testResults$obs)

# Regression Trees
CART <- postResample(pred = testResults$cart, obs = testResults$obs)
Cubist <- postResample(pred = testResults$cubist, obs = testResults$obs)
RandomForest <- postResample(pred = testResults$randomforest, obs = testResults$obs)
BoostedTrees <- postResample(pred = testResults$gbm, obs = testResults$obs)
BaggedTrees <- postResample(pred = testResults$treebag, obs = testResults$obs)

#Combine Model Results Table
Model_Results <- as.data.frame(rbind(OLS,
                                     OLS_PCA,
                                     PCR,
                                     PLS,
                                     Lasso,
                                     Ridge,
                                     ElasticNet,
                                     MARS,
                                     SVM,
                                     SVM_Tune,
                                     KNN,
                                     NeuralNetwork,
                                     CART,
                                     Cubist,
                                     RandomForest,
                                     BoostedTrees,
                                     BaggedTrees))
Model_Results <- round(Model_Results,4) #Round table

Model_Results
```

```
##                   RMSE Rsquared     MAE
## OLS             4.6642   0.9247  3.6306
## OLS_PCA        32.6409   0.8962 29.0758
## PCR             5.5115   0.8947  4.2382
## PLS             4.9263   0.9160  3.8140
## Lasso           4.7216   0.9230  3.6852
## Ridge           4.6642   0.9247  3.6306
## ElasticNet      4.6642   0.9247  3.6306
## MARS            4.3579   0.9342  3.3181
## SVM             4.0610   0.9430  2.9680
## SVM_Tune        4.4659   0.9309  3.4597
## KNN             4.0612   0.9429  2.8716
## NeuralNetwork   4.1510   0.9402  3.1339
## CART            4.5106   0.9295  3.4101
## Cubist          4.3640   0.9342  3.0827
## RandomForest    3.8025   0.9501  2.7439
```

```
## BoostedTrees   4.0679   0.9426  2.9963
## BaggedTrees    5.1254   0.9091  3.9202

#Order by RMSE in Descending Order
Model_Results[order(Model_Results$RMSE),]

##                   RMSE Rsquared     MAE
## RandomForest   3.8025   0.9501  2.7439
## SVM            4.0610   0.9430  2.9680
## KNN            4.0612   0.9429  2.8716
## BoostedTrees   4.0679   0.9426  2.9963
## NeuralNetwork  4.1510   0.9402  3.1339
## MARS           4.3579   0.9342  3.3181
## Cubist         4.3640   0.9342  3.0827
## SVM_Tune       4.4659   0.9309  3.4597
## CART           4.5106   0.9295  3.4101
## OLS            4.6642   0.9247  3.6306
## Ridge          4.6642   0.9247  3.6306
## ElasticNet     4.6642   0.9247  3.6306
## Lasso          4.7216   0.9230  3.6852
## PLS            4.9263   0.9160  3.8140
## BaggedTrees    5.1254   0.9091  3.9202
## PCR            5.5115   0.8947  4.2382
## OLS_PCA       32.6409   0.8962 29.0758
```

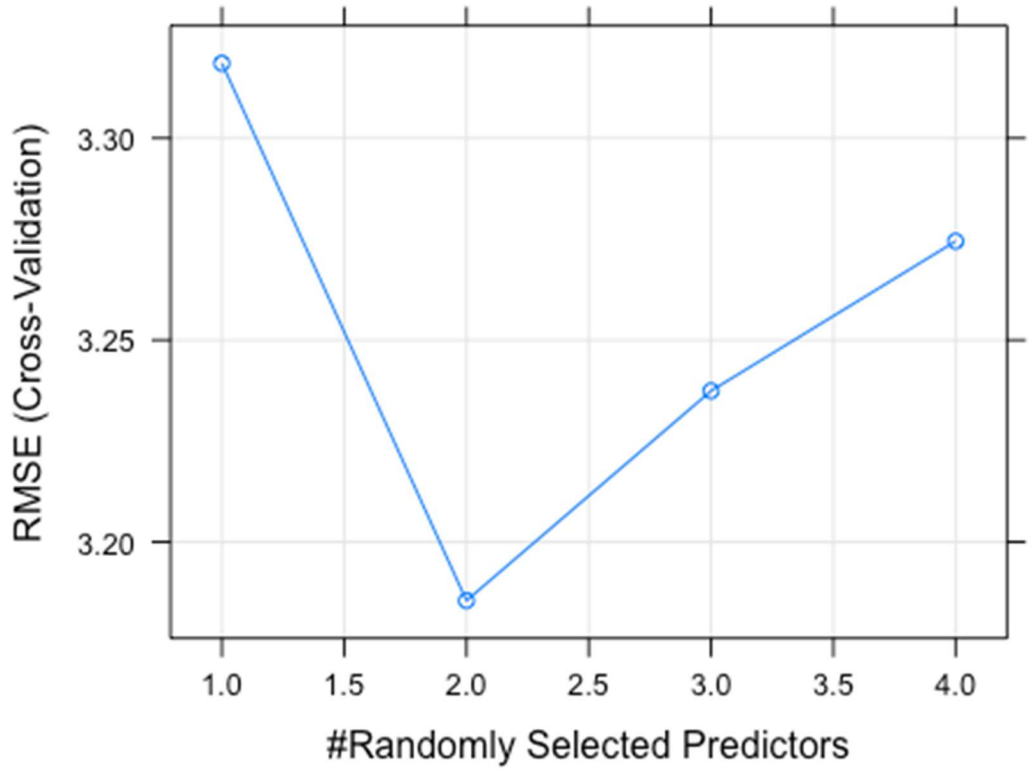
Best Models

(With respect to the lowest RMSE Score)
Best Linear Model: OLS
Best Penalized Model: Ridge
Best Non Linear Regression Model: KNN
Best Tree Model: Random Forest

*Best Overall Model is the Random Forest Model (Regression Trees)

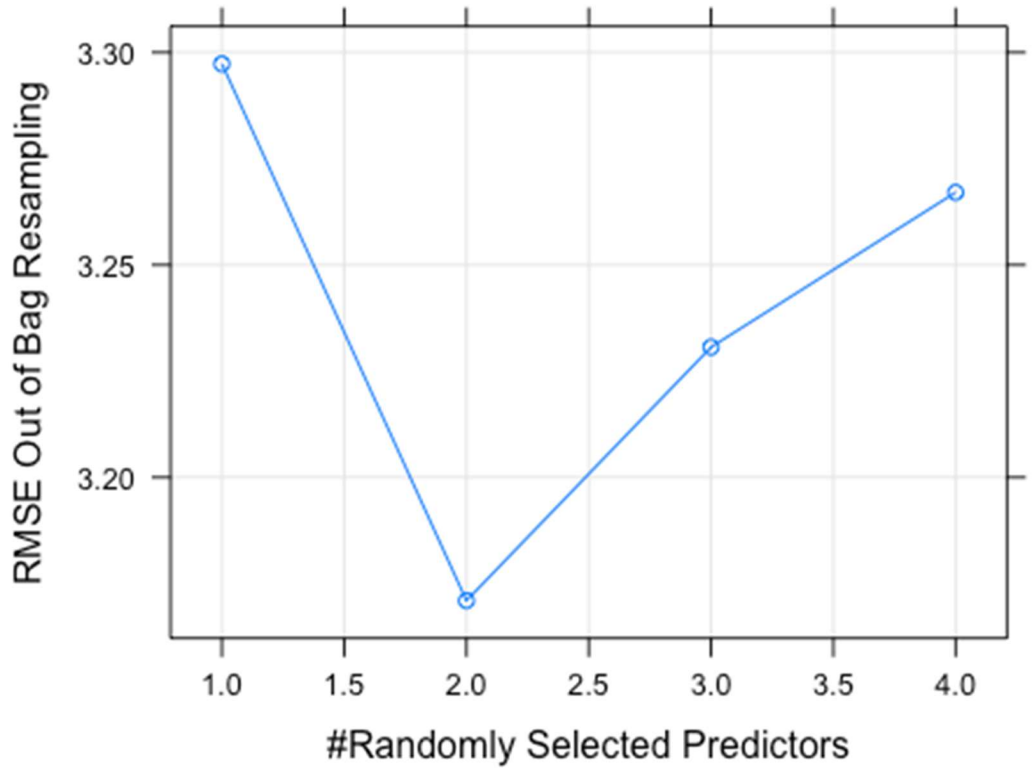
```
rfModel

##
## Call:
##  randomForest(x = powerTrain_Xtrans, y = powerYTrain, importance = TRUE,      ntrees = 1000
)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 10.88991
##                     % Var explained: 96.27

mtryGrid <- data.frame(mtry = floor(seq(1, 4, length = 4)))

### Tune the model using cross-validation
set.seed(100)
rfTune <- train(x = powerTrain_Xtrans, y = powerYTrain,
                method = "rf",
                tuneGrid = mtryGrid,
                ntree = 500,
                importance = TRUE,
```

```
                trControl = ctrl)
rfTune

## Random Forest
##
## 7656 samples
##    4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 6890, 6891, 6890, 6891, 6891, 6891, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   1     3.318530  0.9625350  2.435997
##   2     3.185480  0.9652369  2.300329
##   3     3.237423  0.9640714  2.335818
##   4     3.274481  0.9632345  2.363303
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

plot(rfTune)
```




```
testResults$rfCV <- predict(rfTune, powerTest_Xtrans)

### Tune the model using the OOB estimates
ctrlOOB <- trainControl(method = "oob")
```

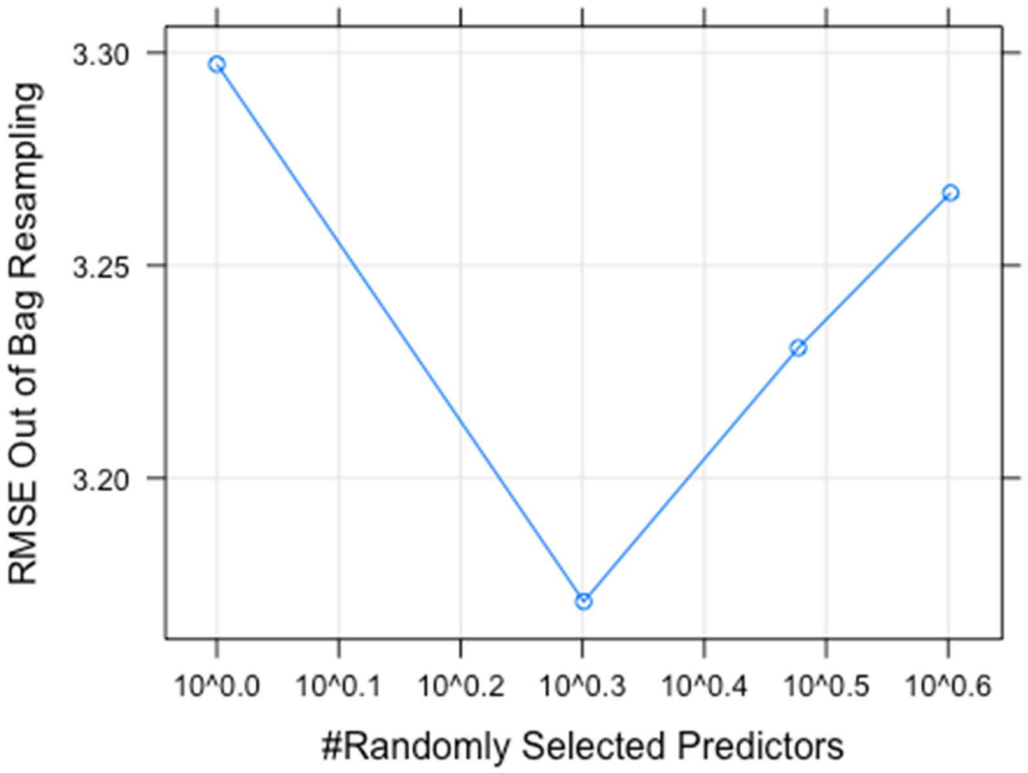
```
set.seed(100)
rfTuneOOB <- train(x = powerTrain_Xtrans, y = powerYTrain,
                   method = "rf",
                   tuneGrid = mtryGrid,
                   ntree = 500,
                   importance = TRUE,
                   trControl = ctrlOOB)
rfTuneOOB

## Random Forest
##
## 7656 samples
##    4 predictor
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared
##   1     3.297295  0.9627667
##   2     3.170989  0.9655646
##   3     3.230645  0.9642567
##   4     3.267067  0.9634462
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

plot(rfTuneOOB)
```



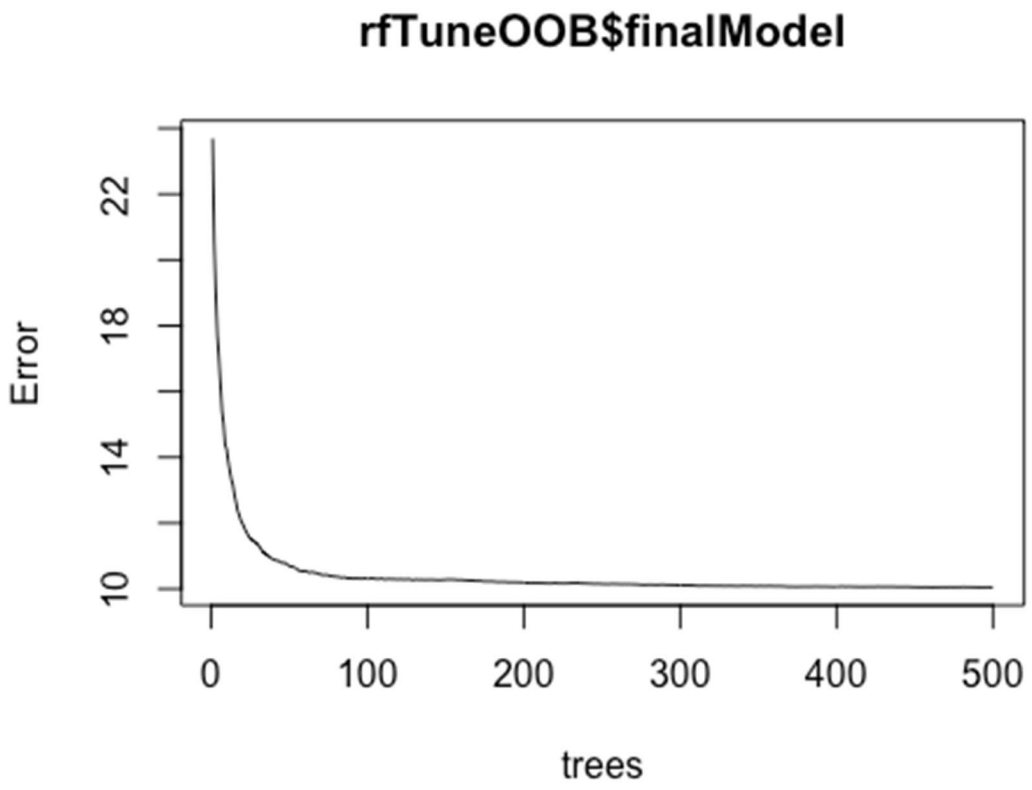
```
testResults$rfOOB <- predict(rfTuneOOB, powerTest_Xtrans)

RandomForest_CV <- postResample(pred = testResults$rfCV, obs = testResults$obs)
RandomForest_OOB <- postResample(pred = testResults$rfOOB, obs = testResults$obs)

#Combine Model Results Table
Model_Results_rf <- as.data.frame(rbind(RandomForest, RandomForest_CV, RandomForest_OOB))
Model_Results_rf[order(Model_Results_rf$RMSE),]

##                       RMSE  Rsquared      MAE
## RandomForest_OOB 3.719788 0.9520561 2.646000
## RandomForest_CV  3.725710 0.9519033 2.648830
## RandomForest     3.802477 0.9501324 2.743886
```

#Random Forest OOB anaylysis

```
rfTuneOOB

## Random Forest
##
## 7656 samples
##    4 predictor
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared
##   1     3.297295  0.9627667
##   2     3.170989  0.9655646
##   3     3.230645  0.9642567
##   4     3.267067  0.9634462
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

### Plot the tuning results
plot(rfTuneOOB, scales = list(x = list(log = 10)))
```




```
rfTuneOOB$finalModel

##
## Call:
##  randomForest(x = x, y = y, ntree = 500, mtry = min(param$mtry,      ncol(x)), importance =
TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 10.04671
##                     % Var explained: 96.56

plot(rfTuneOOB$finalModel)
```




```
#Variable Importance for Random Forest OOB
rfOOBImp <- varImp(rfTuneOOB, scale = FALSE, competes = FALSE)
rfOOBImp

## rf variable importance
##
##    Overall
## RH  105.55
## AT   70.85
## AP   52.39
## V    40.15
```