Appendix

ADS 503 Team 6 – Predicting Electrical Output in Power Plants

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
```

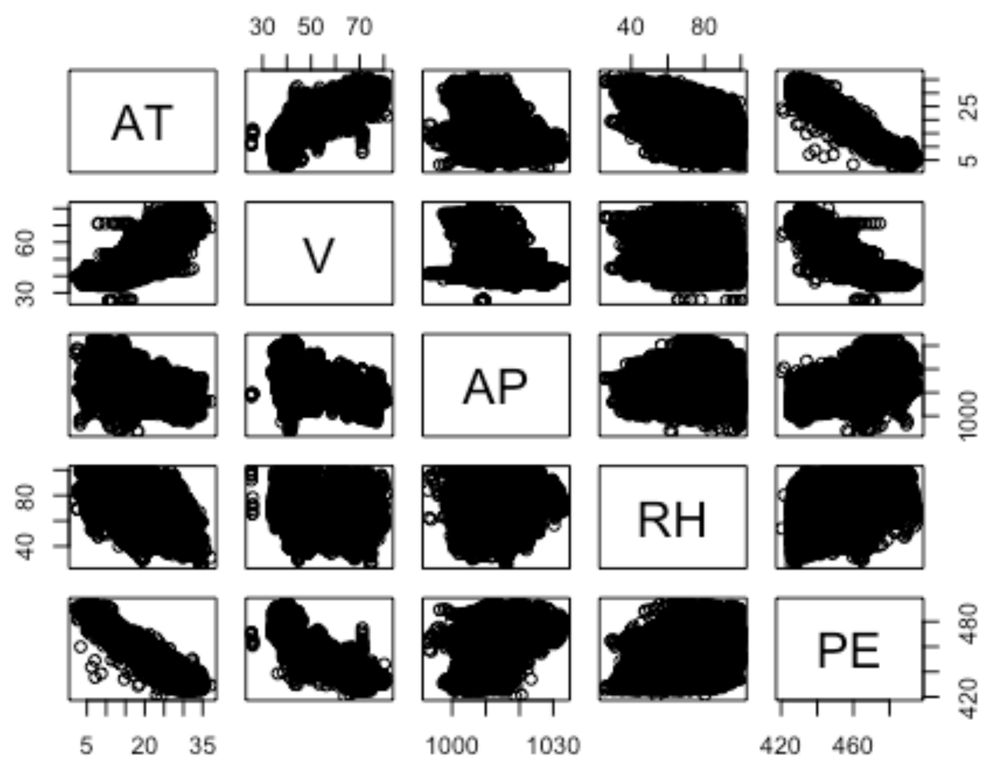
```
str(power)
```

```
## '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 ...
```

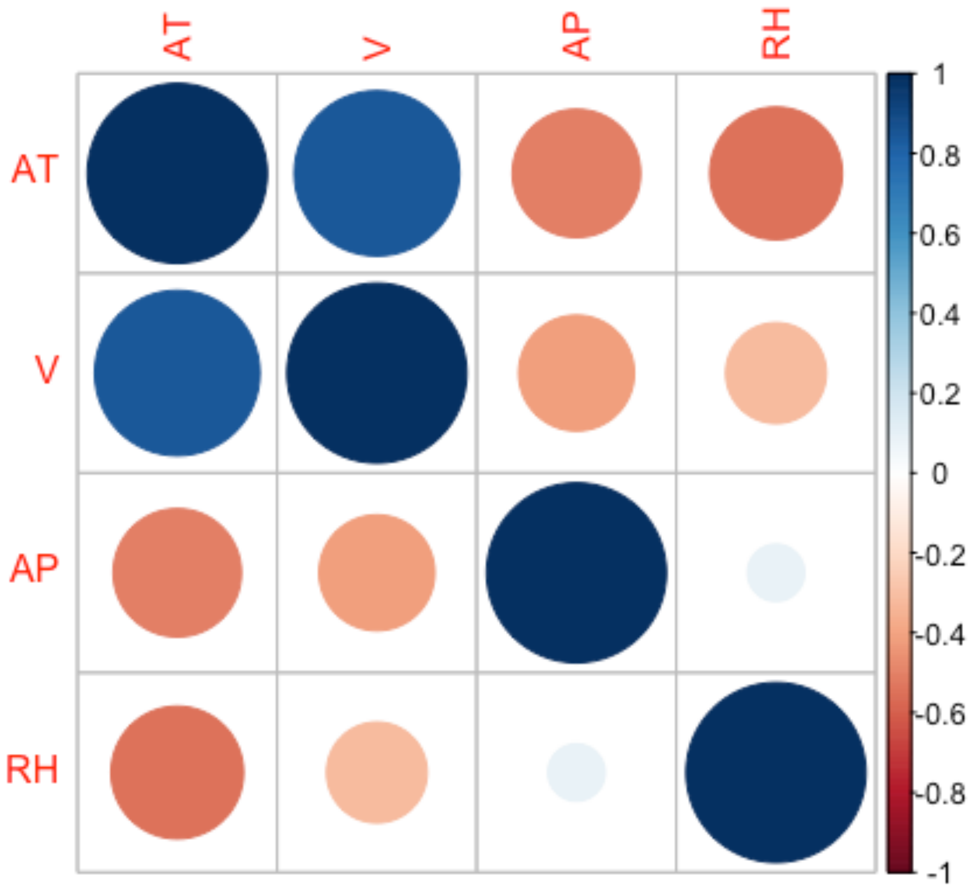
Exploratory Data Analysis:

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

```
pairs(power)
```

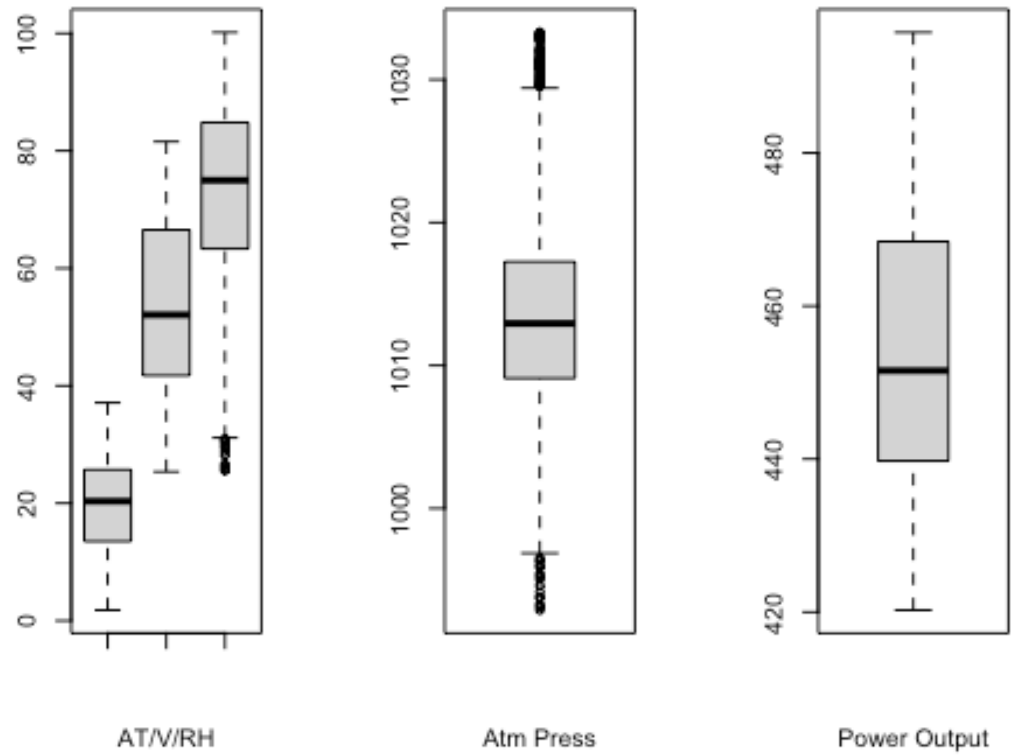



```
corrplot::corrplot(cor(power[,-5]))
```



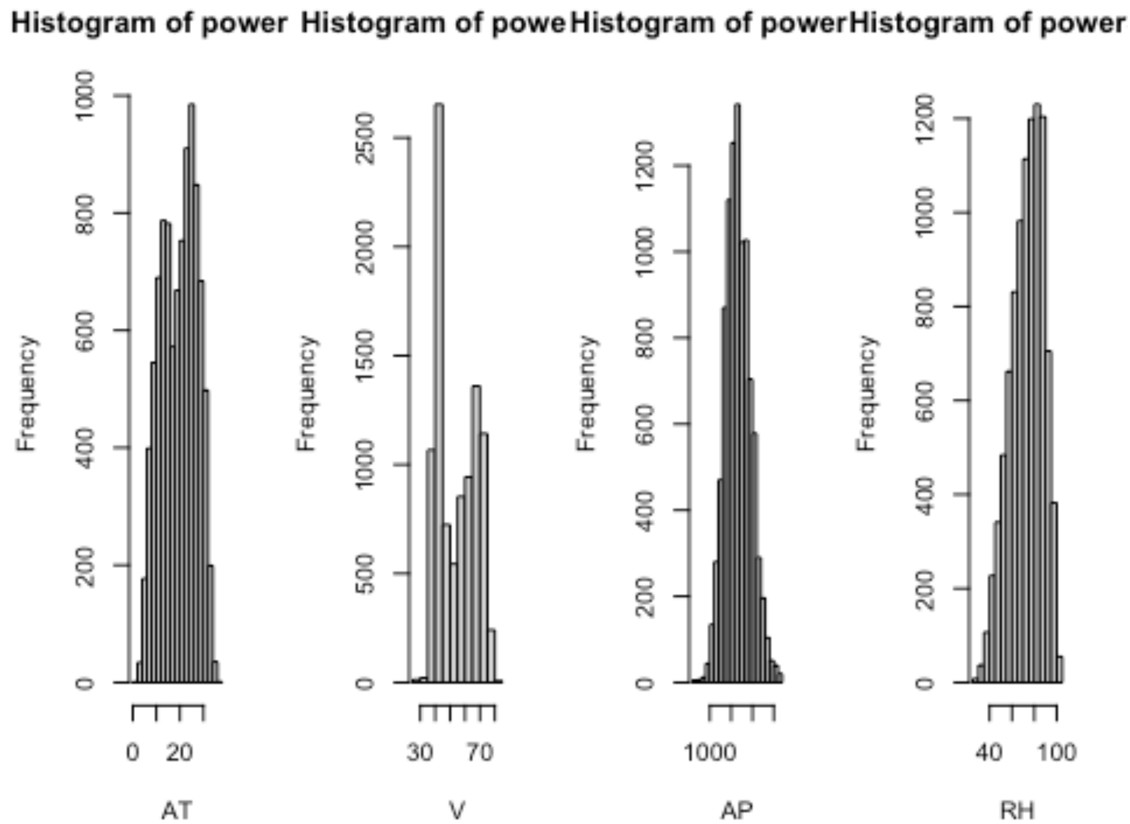
##Look for outliers via box plots:

```
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")
```



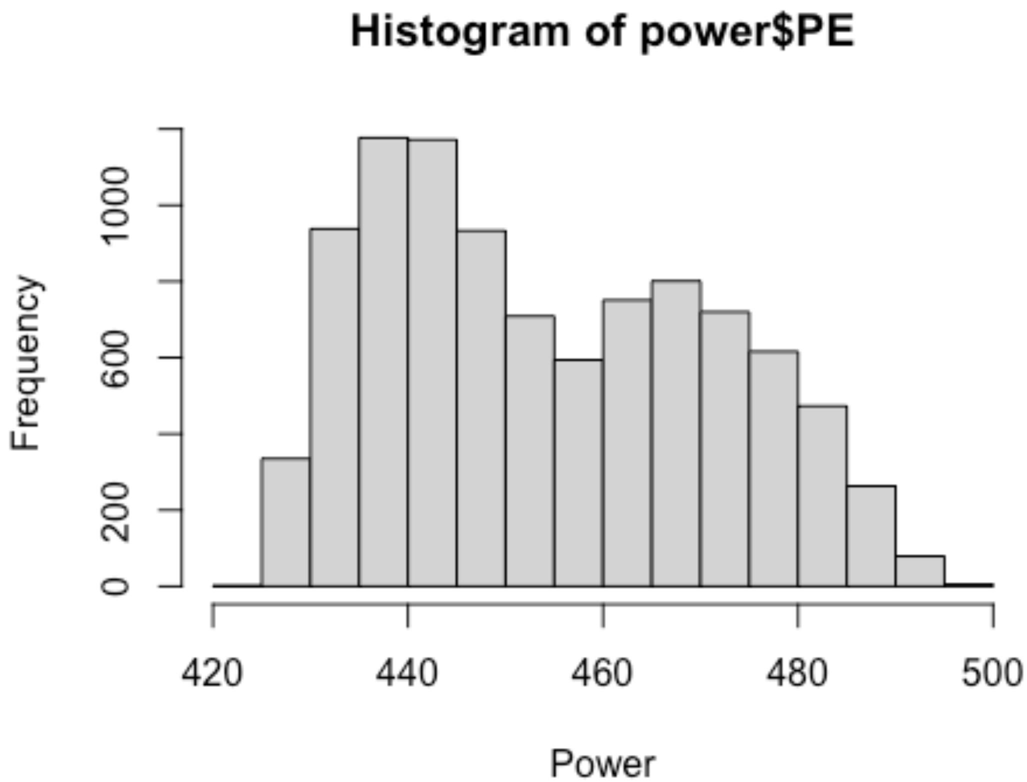
Build histograms of predictor variables to check the distributions of the predictor variables for skewness:

```
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", "sc
ale"))
transXTest
```

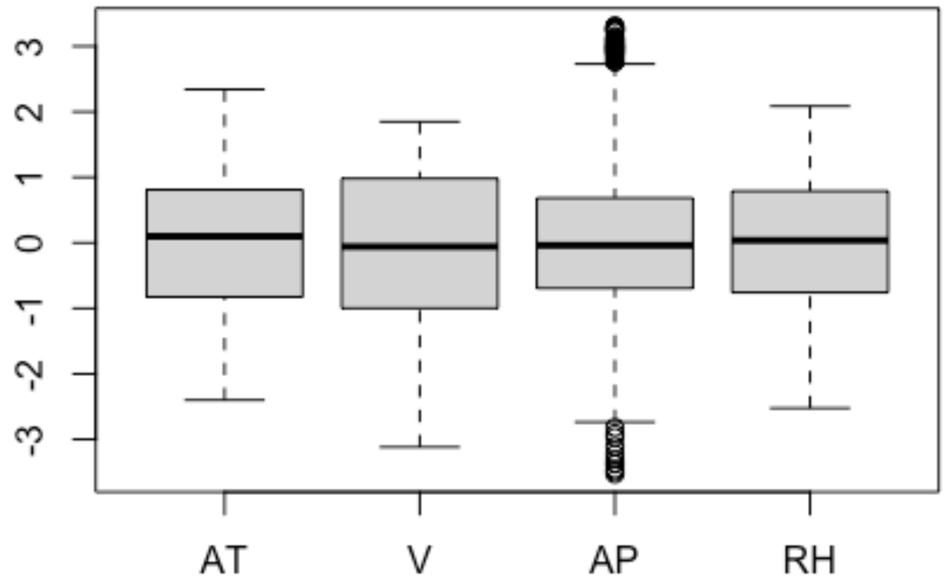
```
## Created from 1912 samples and 4 variables
##
## Pre-processing:
##   - Box-Cox transformation (4)
##   - centered (4)
##   - ignored (0)
##   - scaled (4)
##
## Lambda estimates for Box-Cox transformation:
## 0.9, -0.1, -2, 1.9
```

```r
# 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
```

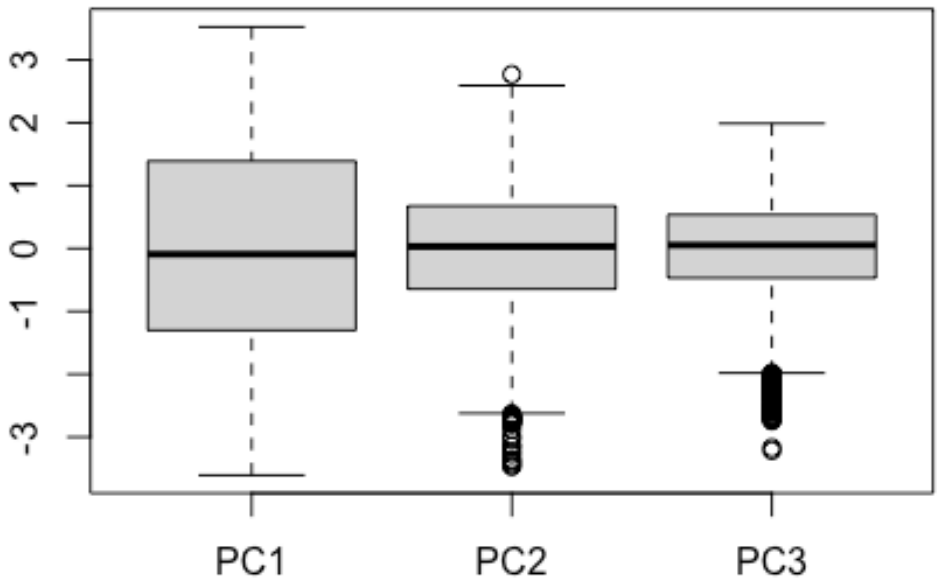
Apply the transformation:

```r
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)
```

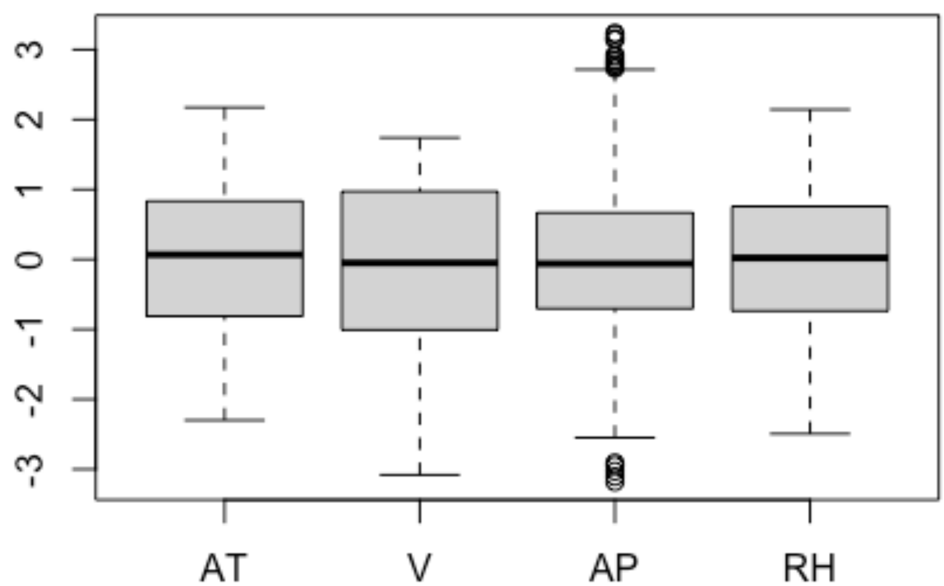



```
powerTrain_Xtrans_pca <- predict(transXTrain_pca, powerXTrain[,1:4])
head(powerTrain_Xtrans_pca)

##           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)
```

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

Linear Regression model using principal components from preprocessing:

```r
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

```r
#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

```r
#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

```r
# 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:

```r
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

```r
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

```r
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

```r
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

```r
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

```r
#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$Linear_Regression, obs = testResults$o
bs)
OLS_PCA <- postResample(pred = testResults$Linear_Regression_pca, obs = testR
esults$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$neural_net, obs = testResult
s$obs)

# Regression Trees
CART <- postResample(pred = testResults$cart, obs = testResults$obs)
Cubist <- postResample(pred = testResults$cubist, obs = testResults$obs)
RandomForest <- postResample(pred = testResults$randomforest, obs = testResul
ts$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,
```

```r
                                      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
```

```r
#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

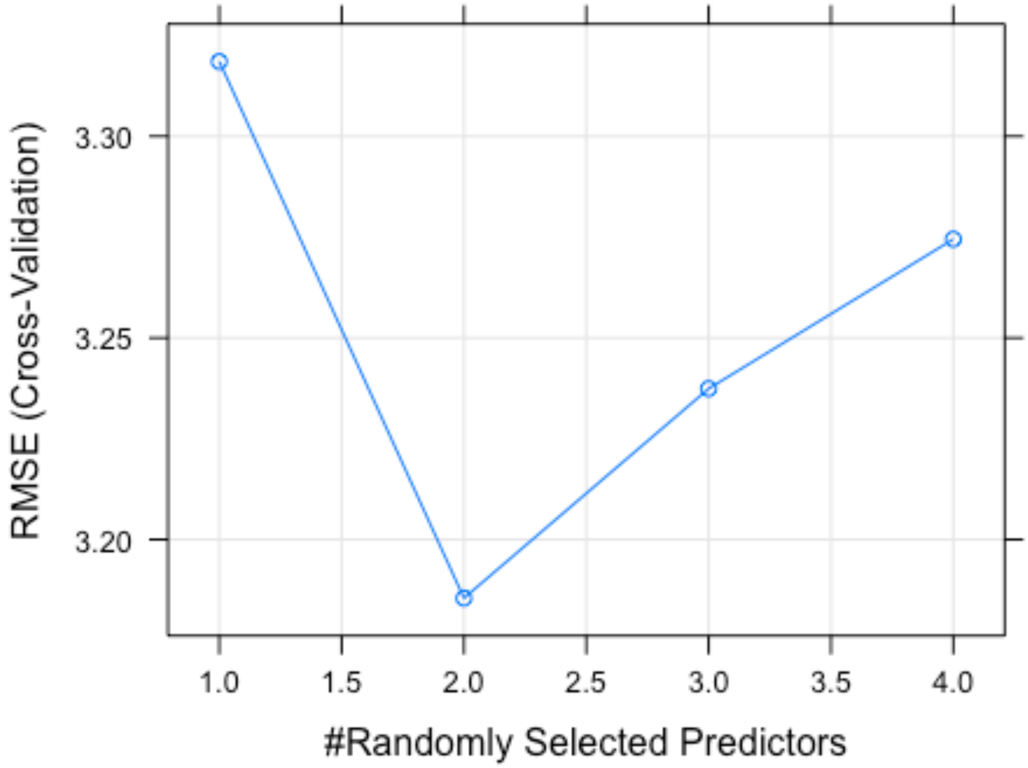
(With respect to the lowest RMSE Score)
Best Linear Model: OLS
Best Penalized Model: Ridge
Best Non Linear Regression Model: KNN
Best Tree Model: Random Forest

*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
##
```

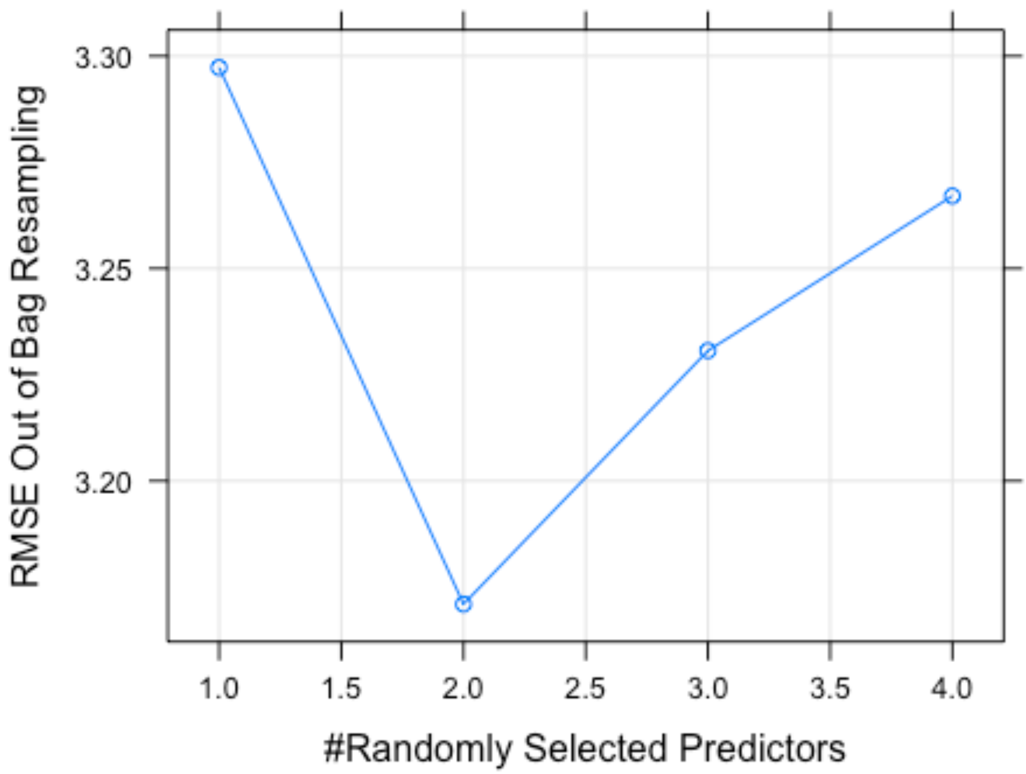
```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

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)
```

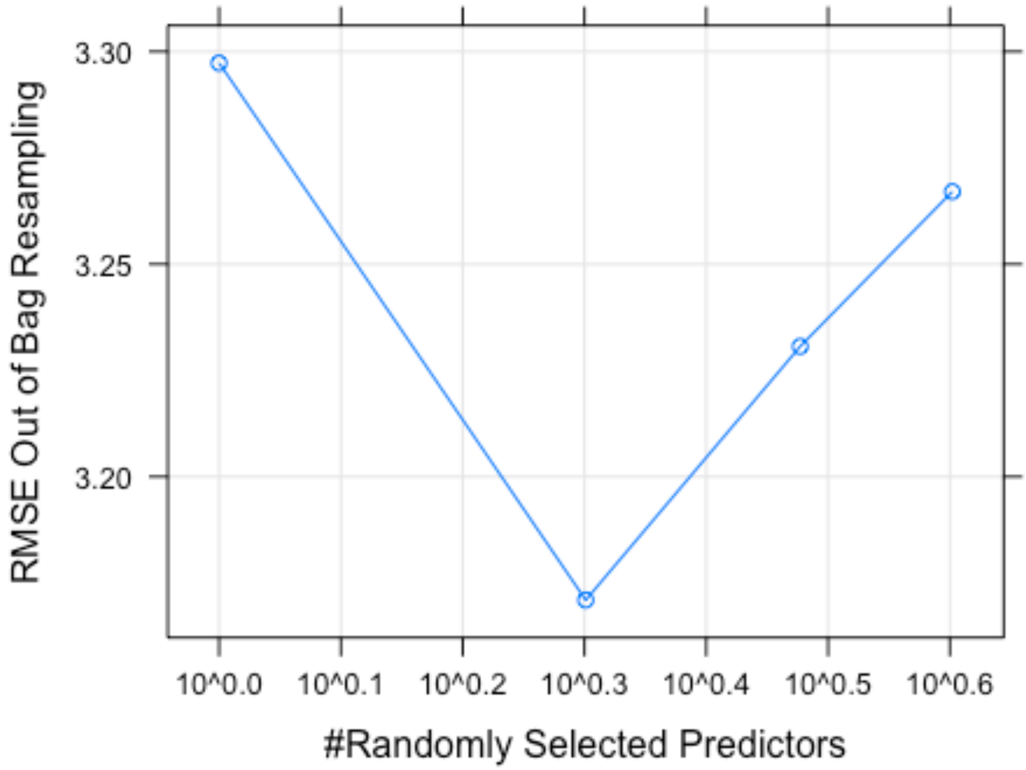



```
testResults$rfOOB <- predict(rfTuneOOB, powerTest_Xtrans)

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

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

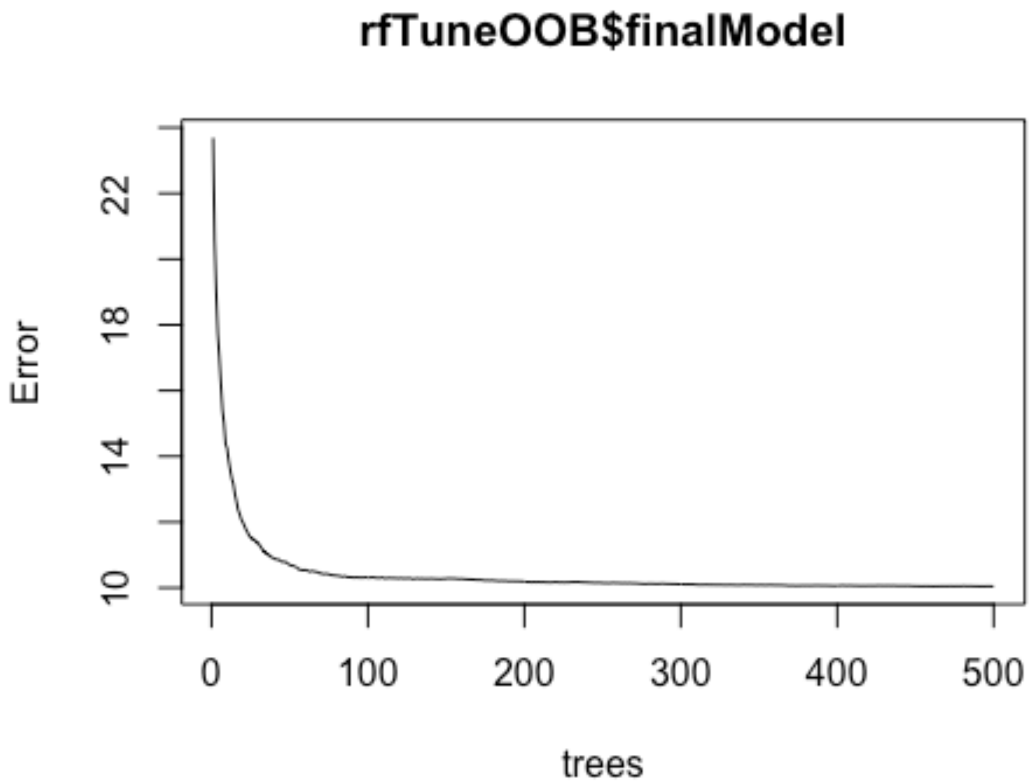
```
##                       RMSE  Rsquared      MAE
## RandomForest_OOB 3.719788 0.9520561 2.646000
## RandomForest_CV  3.725710 0.9519033 2.648830
## RandomForest     3.802477 0.9501324 2.743886
```

#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)
```




```r
#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
```