# A close up of a sign Description automatically generated

# Data 624 Group3 Project 2

#### Alain Kuiete Tchoupou, Jeff Littlejohn, Samriti Malhotra, Rajwant Mishra, Jimmy Ng

#### Due: 7/18/2020

Contents

[Introduction 2](#_Toc45892736)

[Keywords and Phrases 3](#_Toc45892737)

[Approach 4](#_Toc45892738)

[Exploratory Data Analysis 4](#_Toc45892739)

[Data Loading 4](#_Toc45892740)

[Data Exploration 6](#_Toc45892741)

[Data Processing 11](#_Toc45892742)

[Linear Regression 12](#_Toc45892743)

[Linear Model 13](#_Toc45892744)

[Ridge Regression 17](#_Toc45892745)

[Lasso Regression 26](#_Toc45892746)

[Elastic Net Model 36](#_Toc45892747)

[Best model 46](#_Toc45892748)

[Top 5 Variables 49](#_Toc45892749)

[Test from the training set 53](#_Toc45892750)

[Test with Given Test set 56](#_Toc45892751)

[Nonlinear Regression 58](#_Toc45892752)

[MARS 58](#_Toc45892753)

[SVM 60](#_Toc45892754)

[KNN 62](#_Toc45892755)

[MARS model summary 65](#_Toc45892756)

[SVM model summary 65](#_Toc45892757)

[KNN model summary 65](#_Toc45892758)

[Regression Trees and Rule-Based Models 66](#_Toc45892759)

[Random Forest Tree-Based Model 67](#_Toc45892760)

[Extreme Gradient Boosting Tree 68](#_Toc45892761)

[Model Comparison 69](#_Toc45892762)

[Conclusion 70](#_Toc45892763)

[References 70](#_Toc45892764)

## Introduction

As most of us know, ensuring our beverages are produced at the correct potential for hydrogen (pH) level is an essential driver to our business. This pH score, the measure of acidity and alkalinity in our liquids, must be within a narrow, critical range to ensure long-term sales.

The objective of this project is to consider a number of measures and data points involved in the production of our beverages and build a model to use those factors to be able to predict the pH level of the beverage.

Note that a factor being highly predictive of an outcome does not necessarily mean that the factor caused the outcome. For example, beverages that end up with higher than desired pH levels may also show high bowl setpoints. This does not mean that high bowl setpoints cause excessive pH values. They might both be similarly affected by an unknown cause, or large pH values may cause the high bowl setpoints. Our goal is simply to build a model that predicts pH levels. This process may lead to insights about possible causes of pH level problems, but that outcome is certainly not a given.

To construct this model, we had to first understand our data. We have more than 2800 production records of beverages that feature 33 data points, including Brand Code, Fill Ounces, PSC CO2, Temperature, etc. A full list of data elements will be conveyed in the next section. Some of the production records do not have values for all data elements, which is an obstacle we will had to overcome in building our model using established statistical principles. We also had to review

Data scientists use different methods to build models. No one approach gives the optimal approach for all data sets - and their underlying processes. To predict pH, we started by using linear regression, which constructs equations that look similar to you might have experienced in algebra at school. Next, we saw how effective nonlinear regression approaches such as neural networks and support vector machines (SVMs) are in predicting pH. Finally, we attempted to build models that use regression trees and rule-based models.

Don’t be overwhelmed by mathematical terminology. We - or, rather our computers - are just using different methods to take the 33 different data elements and trying to use them to figure out their relationship to pH levels in our beverages.

## Keywords and Phrases

* **Exploratory Data Analysis:** Getting to know your data – summarizing a dataset by its characteristics, often using visual methods. Mean, median, mode, looking for outliers, etc.
* **Missing Value Imputation**: Process of replacing missing values. Different methods of increasing levels of complexity can be used here.
* **Train-Test Split**: Splitting your data up so that, after training a predictive model, you can test your model on data has not previously seen for a (hopefully) honest take on its potential usefulness.
* **Linear Regression**: An approach to explaining the relationship between a dependent variable (pH, in our case) and independent variables (Fill Pressure, CO2) that will be used to predict the dependent variable.
* **Multicollinearity**: Phenomenon for which one or more independent (predictor) variables in a multiple regression model can be linearly predicted from the others in a reliable manner.
* **Ridge Regression**: A linear regression technique that improves predictions when multicollinearity exists.
* **Lasso Regression:** Like ridge regression, lasso attempts to reduce overfitting in linear regression by penalizing models with large numbers of variables
* **Elastic Net Model:** A regularized regression method that linearly combines the penalties of the lasso and ridge regression models.
* **MARS**: Multivariate Adaptive Regression Splines is a non-parametric regression technique that handles non-linear relationships between variables.
* **SVM:** Support Vector Machines attempt to create an optimal hyerplane for linearly separable patterns and then extend them to patterns that are not linearly separable by transformations of original data into a new space. It’s complicated! A supervised learning method that can be used for classification (predicting category A or category B) or regression (a numerical value prediction).
* **KNN:** K-Nearest Neighbors attempts to predict a value (regression) or category (classification) based on observations that are most similar to it.
* **Ensemble Learning Methods:** A modeling technique that combines predictions from multiple machine learning algorithms to (hopefully) make more accurate predictions than any individual model.
* **Random Forest Regression**: An ensemble method that uses multiple decision trees to predict an outcome – whether a classification or a regression.
* **Gradient Boosting Tree:** A classification or regression method that uses an ensemble of weak prediction methods such as decision trees. More complex than random forests.

## Approach

1. Exploratory Data Analysis
2. Data Processing
3. Linear Regression
4. Nonlinear Regression
5. Regression Trees and Rule-Based Models
6. Conclusion

## Exploratory Data Analysis

### Data Loading

bev\_model <- readxl::read\_excel('StudentData\_TO\_MODEL.xlsx',col\_names = TRUE, sheet = 'Subset')

bev\_score <- readxl::read\_excel('StudentEvaluation\_TO\_PREDICT.xlsx',col\_names = TRUE, sheet = 'Subset (2)')

For this predictive model project, we’ve been provided two data sets - a modeling data set we will use to train and test the predictive models and an evaluation data set which will be used to predict unknown pH values and be scored to assess our model performance.

str(bev\_model)

## tibble [2,571 × 33] (S3: tbl\_df/tbl/data.frame)

## $ Brand Code : chr [1:2571] "B" "A" "B" "A" ...

## $ Carb Volume : num [1:2571] 5.34 5.43 5.29 5.44 5.49 ...

## $ Fill Ounces : num [1:2571] 24 24 24.1 24 24.3 ...

## $ PC Volume : num [1:2571] 0.263 0.239 0.263 0.293 0.111 ...

## $ Carb Pressure : num [1:2571] 68.2 68.4 70.8 63 67.2 66.6 64.2 67.6 64.2 72 ...

## $ Carb Temp : num [1:2571] 141 140 145 133 137 ...

## $ PSC : num [1:2571] 0.104 0.124 0.09 NA 0.026 0.09 0.128 0.154 0.132 0.014 ...

## $ PSC Fill : num [1:2571] 0.26 0.22 0.34 0.42 0.16 ...

## $ PSC CO2 : num [1:2571] 0.04 0.04 0.16 0.04 0.12 ...

## $ Mnf Flow : num [1:2571] -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...

## $ Carb Pressure1 : num [1:2571] 119 122 120 115 118 ...

## $ Fill Pressure : num [1:2571] 46 46 46 46.4 45.8 45.6 51.8 46.8 46 45.2 ...

## $ Hyd Pressure1 : num [1:2571] 0 0 0 0 0 0 0 0 0 0 ...

## $ Hyd Pressure2 : num [1:2571] NA NA NA 0 0 0 0 0 0 0 ...

## $ Hyd Pressure3 : num [1:2571] NA NA NA 0 0 0 0 0 0 0 ...

## $ Hyd Pressure4 : num [1:2571] 118 106 82 92 92 116 124 132 90 108 ...

## $ Filler Level : num [1:2571] 121 119 120 118 119 ...

## $ Filler Speed : num [1:2571] 4002 3986 4020 4012 4010 ...

## $ Temperature : num [1:2571] 66 67.6 67 65.6 65.6 66.2 65.8 65.2 65.4 66.6 ...

## $ Usage cont : num [1:2571] 16.2 19.9 17.8 17.4 17.7 ...

## $ Carb Flow : num [1:2571] 2932 3144 2914 3062 3054 ...

## $ Density : num [1:2571] 0.88 0.92 1.58 1.54 1.54 1.52 0.84 0.84 0.9 0.9 ...

## $ MFR : num [1:2571] 725 727 735 731 723 ...

## $ Balling : num [1:2571] 1.4 1.5 3.14 3.04 3.04 ...

## $ Pressure Vacuum : num [1:2571] -4 -4 -3.8 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 ...

## $ PH : num [1:2571] 8.36 8.26 8.94 8.24 8.26 8.32 8.4 8.38 8.38 8.5 ...

## $ Oxygen Filler : num [1:2571] 0.022 0.026 0.024 0.03 0.03 0.024 0.066 0.046 0.064 0.022 ...

## $ Bowl Setpoint : num [1:2571] 120 120 120 120 120 120 120 120 120 120 ...

## $ Pressure Setpoint: num [1:2571] 46.4 46.8 46.6 46 46 46 46 46 46 46 ...

## $ Air Pressurer : num [1:2571] 143 143 142 146 146 ...

## $ Alch Rel : num [1:2571] 6.58 6.56 7.66 7.14 7.14 7.16 6.54 6.52 6.52 6.54 ...

## $ Carb Rel : num [1:2571] 5.32 5.3 5.84 5.42 5.44 5.44 5.38 5.34 5.34 5.34 ...

## $ Balling Lvl : num [1:2571] 1.48 1.56 3.28 3.04 3.04 3.02 1.44 1.44 1.44 1.38 ...

We have 33 data elements for 2571 records that will be used for training. We expect some missing or NULL values - here we see them reflected in the “NA” values in the “Hyd Pressure 2” measurements.

Here, we verify that our scoring data contains the same elements.

dim(bev\_score)

## [1] 267 33

As mentioned above, we will generate the model on using the training data set that covers 2571 rows and then evaluate its accuracy - and other metrics - using the 267-record test data set.

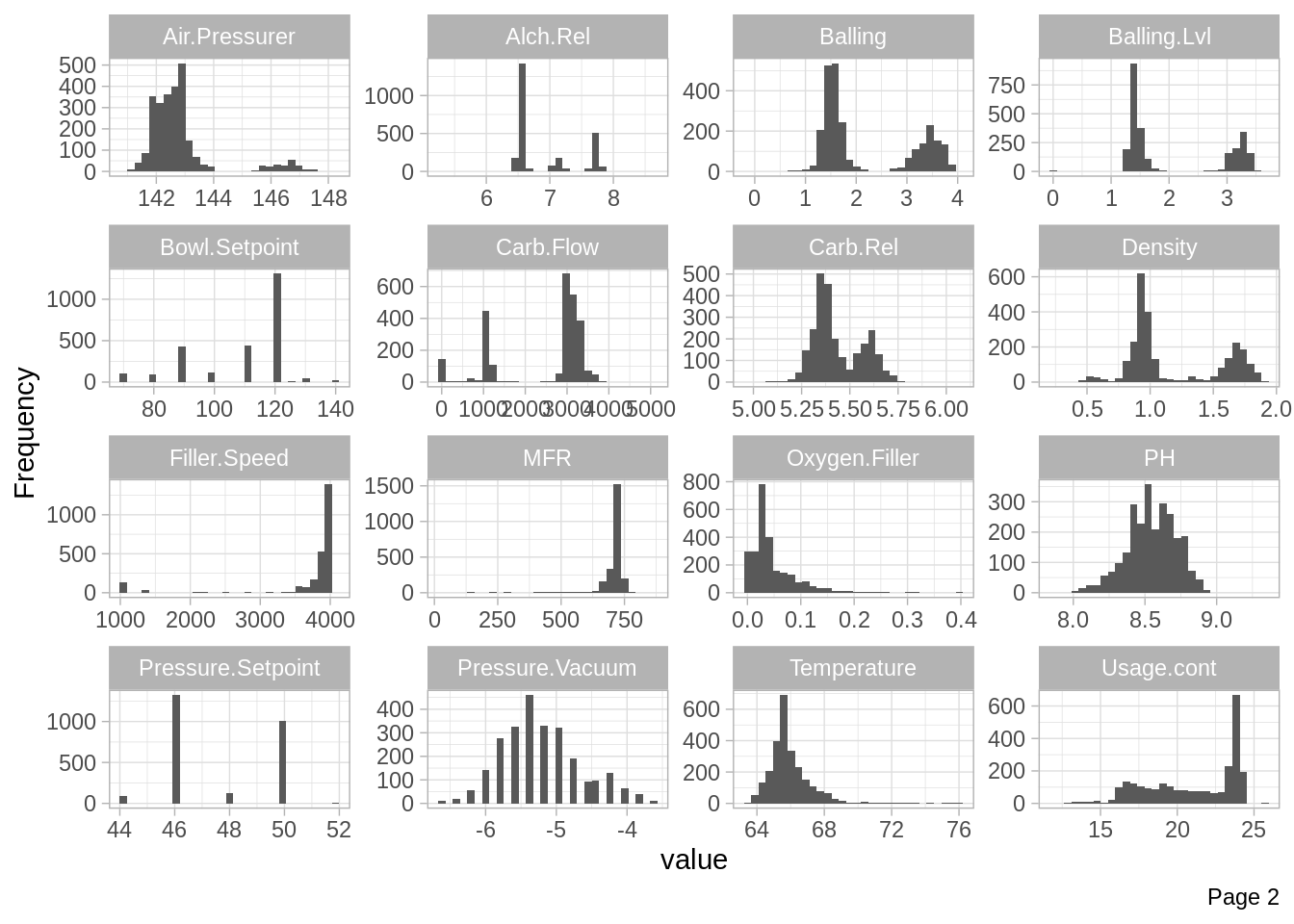
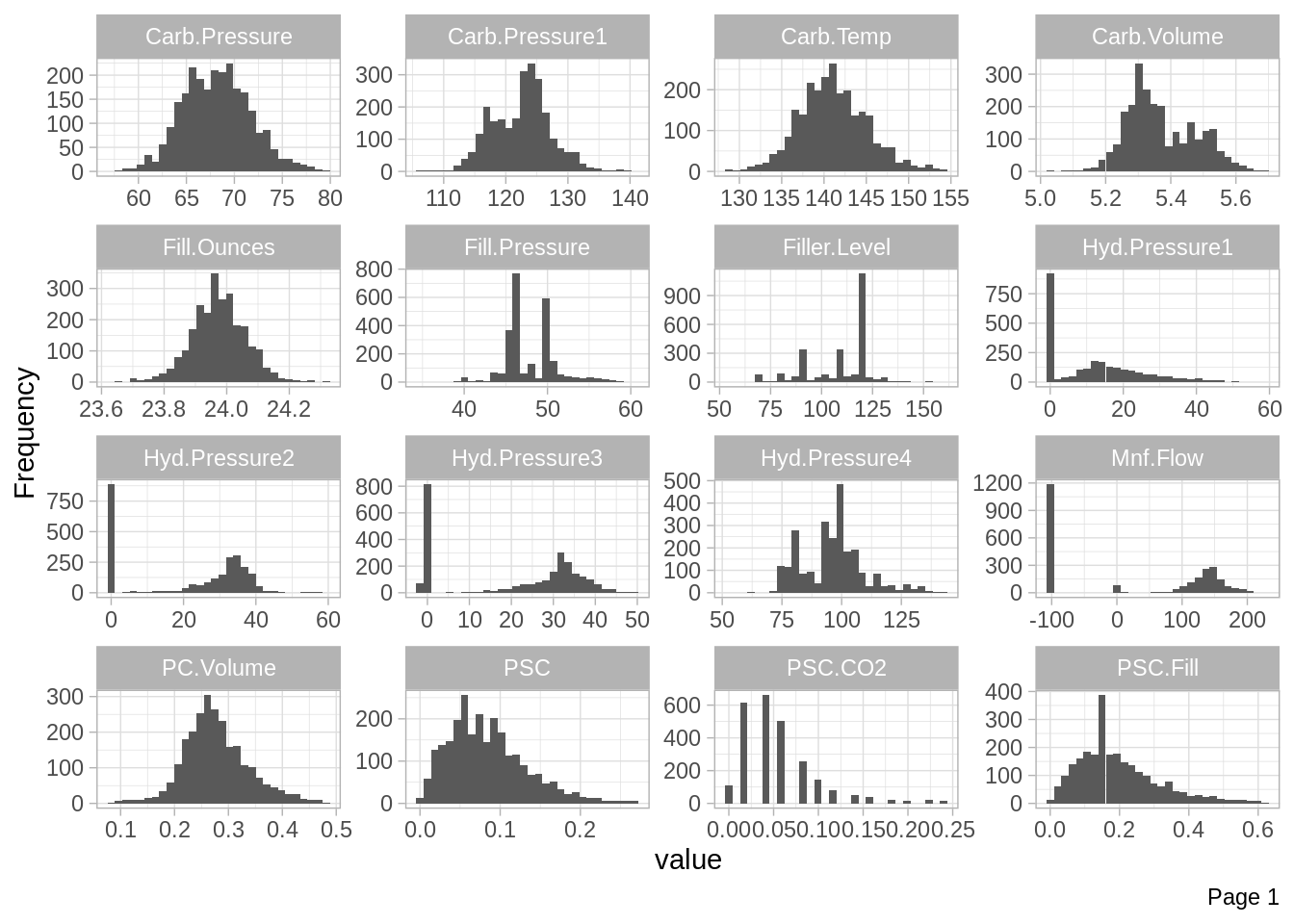
### Data Exploration

Next, we explore our data using summary statistics.

*#describe(bev\_model)*

Brand Code is a string (non-numeric) field, so its lack of a mean makes sense. We see a wide range of values. Some may be on different scales. Despite being mostly numeric fields, some of these might use temperatures in Celsius, whereas others like Carb[onation] Pressure use pounds per square inch. In order to build the most effective predictive model, we may apply data transformations to standardize these numeric data elements.

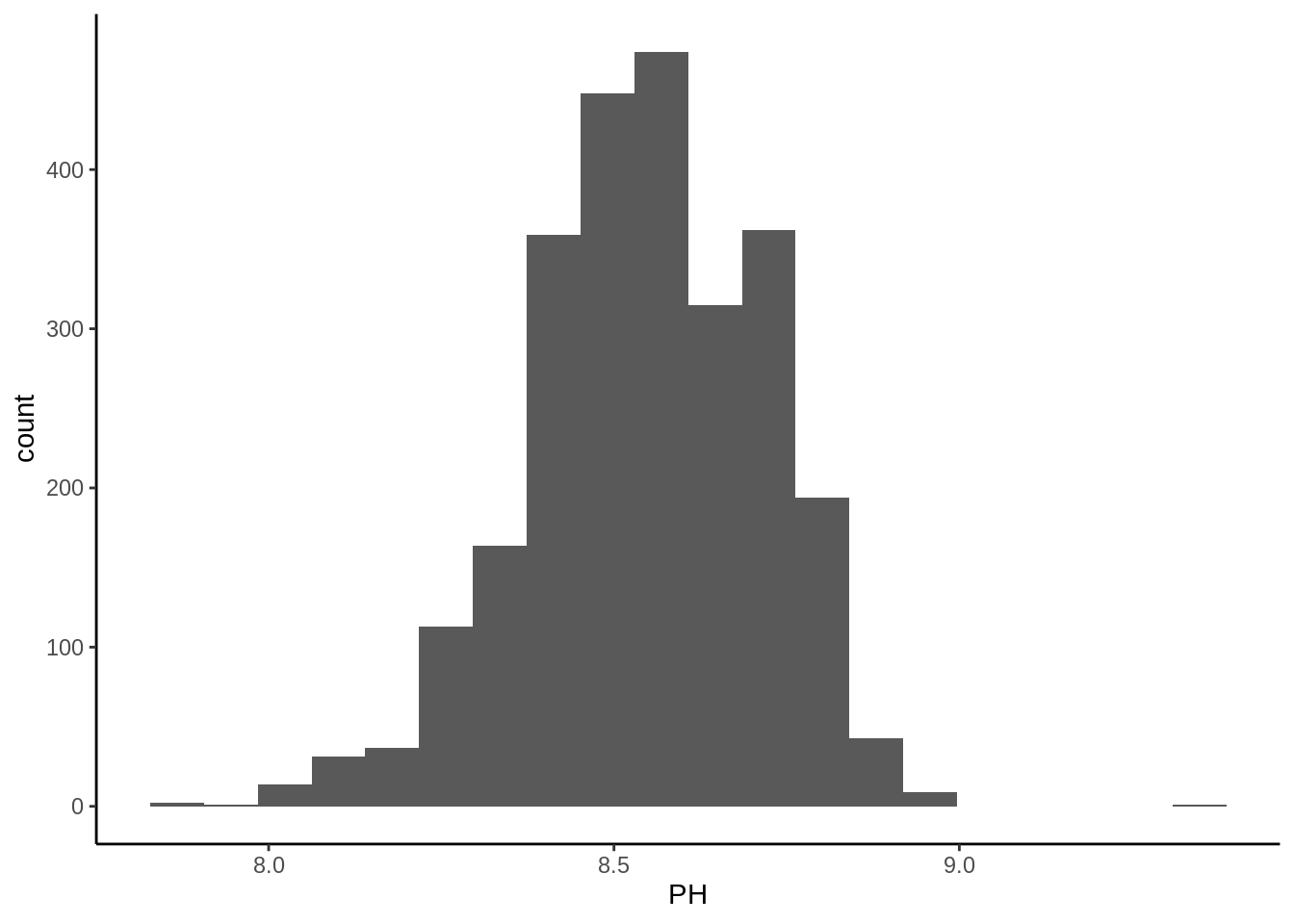
plot\_histogram(bev\_model, ggtheme=theme\_light())



Note that the scales of the x-axes vary for each element. Nonetheless, we see data elements with normal distributions and many with likely outliers. Some histograms reveal bimodal or even trimodal distributions.

Here, we take a closer look at the pH values present in our training data.

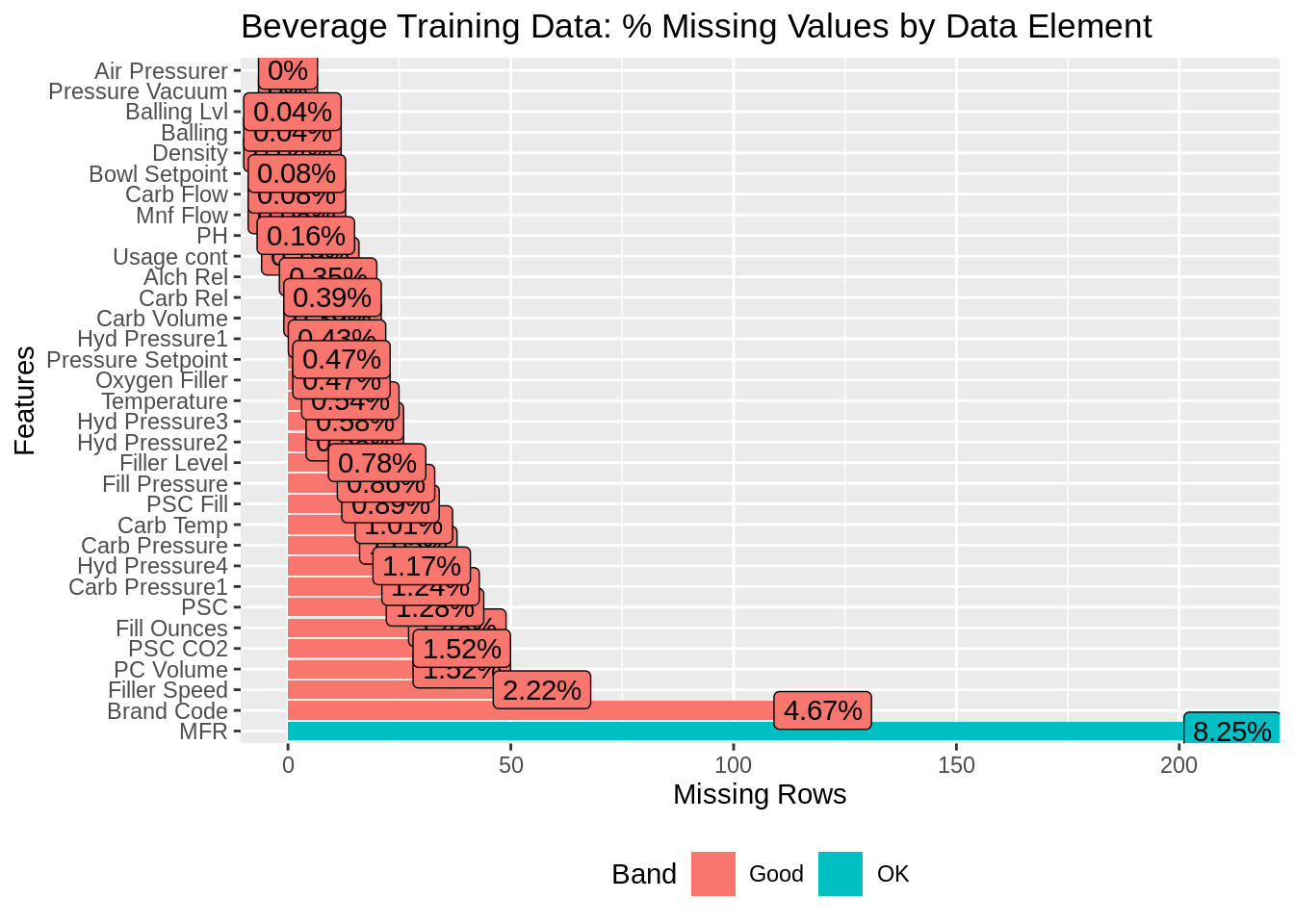
ggplot(bev\_model, aes(PH)) + geom\_histogram(bins=20) + theme\_classic()



A slight left skew is evident, as is a right potential outlier.

Finally, a count of missing (NA) values by variable is conducted.

plot\_missing(bev\_model, title = "Beverage Training Data: % Missing Values by Data Element")



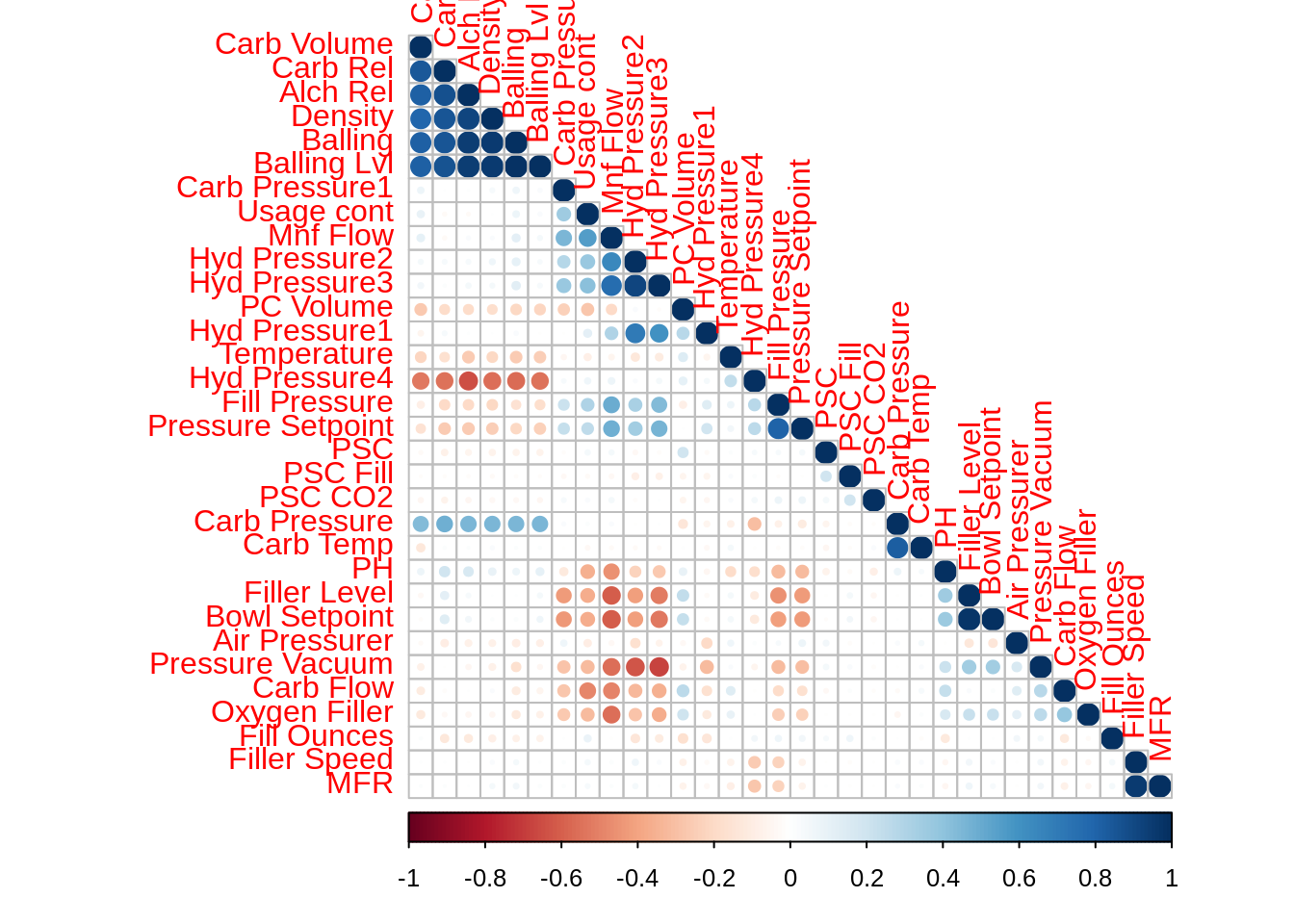
MFR is missing for more than eight percent of our training records, and Brand Code is NA for almost five percent of them. Those will need to be handled in our next section.

Perhaps the most conceptually difficult aspect of data exploration in predictive modeling is checking for correlation. If two different variables reliably occur together, they can negatively affect our model. They tend to change in unison, and it becomes very difficult for the model to estimate the relationship between the two (or more) correlated independent variables and the dependent variable. If this sounds tricky - don’t worry. In short, if we wanted to predict if it rained, we probably wouldn’t want to include both 1) is the road wet and 2) whether drivers were using windshield wipers. One would give us the information contained within the others.

The correlations between variables in our training dataset are below.

cor\_bev\_model <- cor(bev\_model[,-1], use = "na.or.complete")

corrplot(cor\_bev\_model, order = 'hclust', type = 'lower')



With so many dimensions (variables) in the training data, seeing individual correlations between variables is difficult here. Suffice to say, we have high correlation between certain variables that would negatively affect our predictive model if we use certain approaches.

### Data Processing

Before we get to modeling, we will do a little data processing - also known as cleansing. First, we will remove the handful of records (0.16%, as we saw before) from the training set that have missing pH values. This will remove four records. Next, we will label the small number of records that have missing Brand categories as “U” for unknown.

*#remove NA pH records*

bev\_model <- bev\_model[complete.cases(bev\_model[,26]),]

*#update NA Brand Code records to "U"*

bev\_model$`Brand Code`[is.na(bev\_model$`Brand Code`)] <- 'U'

Finally we will impute missing values for the training records that remain. There are a number of advanced methods used by data scientists to impute or “fill in” missing/NA values. Based on past experience and the fact that we don’t have “big data,” we will use k-nearest neighbor (KNN) imputation, which uses a distance function to essentially predict values for missing data elements based on other records that are similar to in it for the elements that are present. Simply, if everyone on your street has two cars, we’ll guess that you do two.

bev\_model\_i <- as.data.frame(bev\_model[, !names(bev\_model) %**in**% c("Brand Code", "PH")])

*#https://www.r-bloggers.com/missing-value-treatment/*

bev\_model\_impute <- knnImputation(bev\_model\_i, k = 10)

bev\_model\_impute$pH <- bev\_model$PH

bev\_model\_impute$'Brand Code' <- bev\_model$`Brand Code`

Before getting to modelling, we will split our modelling set into train and test sets.

set.seed(3456)

trainIndex <- createDataPartition(bev\_model\_impute$pH, p = .8,

list = FALSE,

times = 1)

bev\_model\_train <- bev\_model\_impute[ trainIndex,]

bev\_model\_test <- bev\_model\_impute[-trainIndex,]

As mentioned earlier, data processing frequently includes such steps as performing a train-test split, transforming variables so that they’re all on normalized scales, and removing variables that have correlation with others - or have no correlation with the dependent variables, pH in this case. As different modeling approaches handle these data issues in disparate ways, we will handle those data transformations and processing steps in the indivdual sections of model construction.

## Linear Regression

In this section, we will try to fit multiple Linear Regression Model and its cousins. We will especially try building : + Simple Linear Regression Model + Ridge Regression Model + Lasso Regression Model + Elastic Net Regression Model

We will be doing 10 fold cross-validation, we will repeat it 5 times as per our train control parameter.

### Linear Model

metric = 'RMSE'

head(bev\_model\_train )

|  |
| --- |
|  |
|  | | Carb Volume  <dbl> | Fill Ounces  <dbl> | PC Volume  <dbl> | Carb Pressure  <dbl> | Carb Temp  <dbl> | PSC  <dbl> | PSC Fill  <dbl> | PSC CO2  <dbl> | Mnf Flow  <dbl> |  |
| 1 | | 5.340000 | 23.96667 | 0.2633333 | 68.2 | 141.2 | 0.10400000 | 0.26 | 0.04 | -100 |  |
| 2 | | 5.426667 | 24.00667 | 0.2386667 | 68.4 | 139.6 | 0.12400000 | 0.22 | 0.04 | -100 |  |
| 4 | | 5.440000 | 24.00667 | 0.2933333 | 63.0 | 132.6 | 0.07957591 | 0.42 | 0.04 | -100 |  |
| 5 | | 5.486667 | 24.31333 | 0.1113333 | 67.2 | 136.8 | 0.02600000 | 0.16 | 0.12 | -100 |  |
| 6 | | 5.380000 | 23.92667 | 0.2693333 | 66.6 | 138.4 | 0.09000000 | 0.24 | 0.04 | -100 |  |
| 7 | | 5.313333 | 23.88667 | 0.2680000 | 64.2 | 136.8 | 0.12800000 | 0.40 | 0.04 | -100 |  |

6 rows | 1-10 of 34 columns

*# Train control*

customTrainControl <- trainControl(method = "repeatedcv",

number = 10 ,

repeats = 5 ,

verboseIter = F)

*#Linear Model*

lm <- train(pH ~ .,

bev\_model\_train,

method= 'lm',

trControl = customTrainControl

)

lm$results

|  |
| --- |
|  |
|  | | intercept  <lgl> | RMSE  <dbl> | Rsquared  <dbl> | MAE  <dbl> | RMSESD  <dbl> | RsquaredSD  <dbl> | MAESD  <dbl> |
| 1 | | TRUE | 0.1313413 | 0.4163949 | 0.1023655 | 0.005721587 | 0.04236108 | 0.004737473 |

1 row

lm *# 2055 , 32 predictors,*

## Linear Regression

##

## 2055 samples

## 32 predictor

##

## No pre-processing

## Resampling: Cross-Validated (10 fold, repeated 5 times)

## Summary of sample sizes: 1849, 1851, 1849, 1850, 1849, 1850, ...

## Resampling results:

##

## RMSE Rsquared MAE

## 0.1313413 0.4163949 0.1023655

##

## Tuning parameter 'intercept' was held constant at a value of TRUE

summary(lm)

##

## Call:

## lm(formula = .outcome ~ ., data = dat)

##

## Residuals:

## Min 1Q Median 3Q Max

## -0.50064 -0.07818 0.01034 0.08761 0.42805

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 1.081e+01 1.081e+00 9.997 < 2e-16 \*\*\*

## `\\`Carb Volume\\`` -7.194e-02 7.597e-02 -0.947 0.343800

## `\\`Fill Ounces\\`` -8.633e-02 3.565e-02 -2.421 0.015547 \*

## `\\`PC Volume\\`` -1.106e-01 5.927e-02 -1.866 0.062122 .

## `\\`Carb Pressure\\`` -1.098e-03 3.200e-03 -0.343 0.731532

## `\\`Carb Temp\\`` 1.940e-03 2.557e-03 0.759 0.448130

## PSC -9.354e-02 6.273e-02 -1.491 0.136072

## `\\`PSC Fill\\`` -4.709e-02 2.611e-02 -1.804 0.071396 .

## `\\`PSC CO2\\`` -1.063e-01 7.020e-02 -1.515 0.130036

## `\\`Mnf Flow\\`` -6.891e-04 5.048e-05 -13.652 < 2e-16 \*\*\*

## `\\`Carb Pressure1\\`` 7.019e-03 7.824e-04 8.971 < 2e-16 \*\*\*

## `\\`Fill Pressure\\`` 2.426e-03 1.340e-03 1.811 0.070229 .

## `\\`Hyd Pressure1\\`` -2.767e-04 3.984e-04 -0.695 0.487331

## `\\`Hyd Pressure2\\`` -8.644e-04 5.803e-04 -1.490 0.136484

## `\\`Hyd Pressure3\\`` 3.225e-03 6.434e-04 5.013 5.82e-07 \*\*\*

## `\\`Hyd Pressure4\\`` 1.596e-05 3.493e-04 0.046 0.963555

## `\\`Filler Level\\`` -1.190e-03 5.960e-04 -1.997 0.045936 \*

## `\\`Filler Speed\\`` 2.374e-06 8.319e-06 0.285 0.775384

## Temperature -1.220e-02 2.511e-03 -4.857 1.28e-06 \*\*\*

## `\\`Usage cont\\`` -6.680e-03 1.258e-03 -5.308 1.23e-07 \*\*\*

## `\\`Carb Flow\\`` 2.035e-05 4.023e-06 5.058 4.62e-07 \*\*\*

## Density -1.235e-01 3.090e-02 -3.996 6.69e-05 \*\*\*

## MFR -3.691e-05 6.574e-05 -0.561 0.574533

## Balling -1.063e-01 2.766e-02 -3.845 0.000124 \*\*\*

## `\\`Pressure Vacuum\\`` -2.687e-02 8.549e-03 -3.144 0.001693 \*\*

## `\\`Oxygen Filler\\`` -3.649e-01 7.948e-02 -4.591 4.69e-06 \*\*\*

## `\\`Bowl Setpoint\\`` 3.688e-03 6.248e-04 5.902 4.19e-09 \*\*\*

## `\\`Pressure Setpoint\\`` -8.756e-03 2.136e-03 -4.100 4.30e-05 \*\*\*

## `\\`Air Pressurer\\`` -3.586e-03 2.604e-03 -1.377 0.168633

## `\\`Alch Rel\\`` 3.951e-02 2.386e-02 1.656 0.097932 .

## `\\`Carb Rel\\`` 3.645e-02 5.219e-02 0.698 0.485018

## `\\`Balling Lvl\\`` 1.643e-01 2.786e-02 5.896 4.35e-09 \*\*\*

## `\\`Brand Code\\`B` 9.669e-02 2.432e-02 3.976 7.25e-05 \*\*\*

## `\\`Brand Code\\`C` -4.460e-02 2.406e-02 -1.854 0.063956 .

## `\\`Brand Code\\`D` 7.184e-02 1.698e-02 4.231 2.43e-05 \*\*\*

## `\\`Brand Code\\`U` 1.090e-02 2.722e-02 0.400 0.688949

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

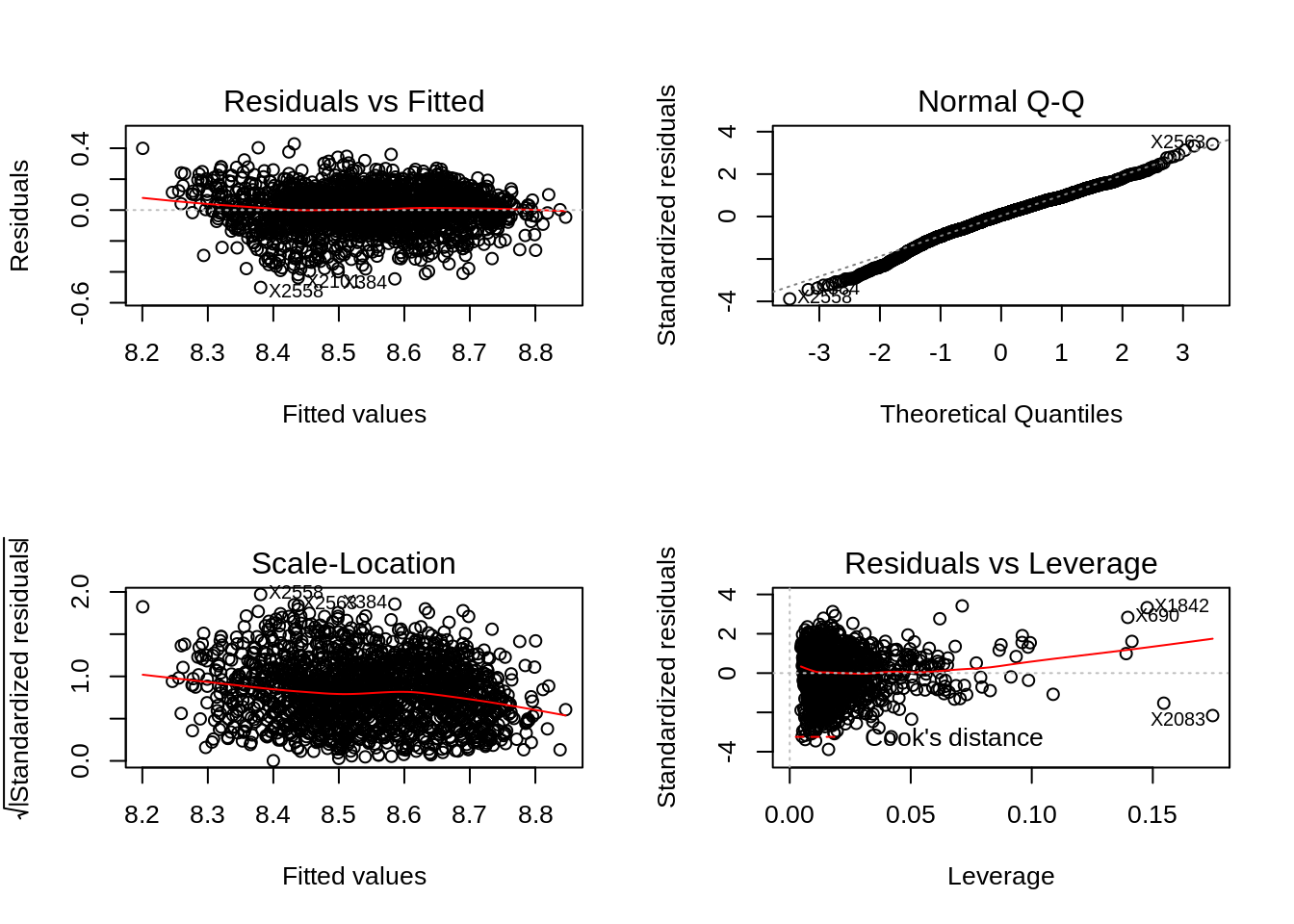
## Residual standard error: 0.13 on 2019 degrees of freedom

## Multiple R-squared: 0.4354, Adjusted R-squared: 0.4257

## F-statistic: 44.49 on 35 and 2019 DF, p-value: < 2.2e-16

par(mfrow=c(2,2))

plot(lm$finalModel)



We see the method ‘lm’ can explain 39 % of the data of the training set. As per the Linear Model, we see that we have 32 predictors, collected from 2055 samples. QQ plot suggests it has little variations in the beginning of the data. The residual plot shows some condensed around the zero mean lines.

### Ridge Regression

Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value.

With the tuning parameter, alfa= 0, and lambda is between 0 to 1, we can say that is a ridge model. This model is not sensitive to the variables that has multicollinearity.

*# lambda = 0.00621*

ridge2 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

tuneGrid = expand.grid(alpha= 0,

lambda= seq(0.1,0.001,length= 20)),

trControl = customTrainControl)

*# lambda = 0.00716*

ridge <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

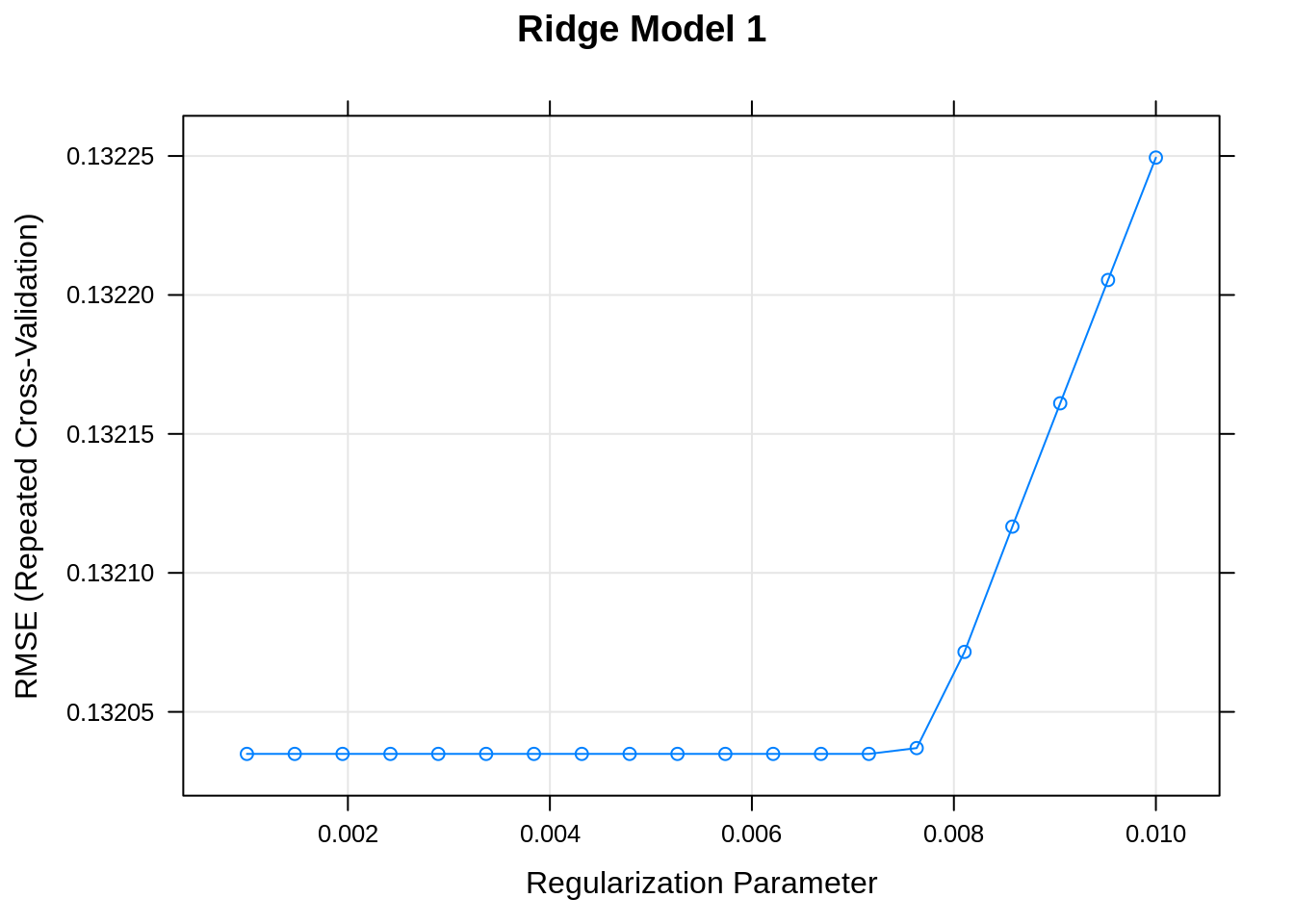
tuneGrid = expand.grid(alpha= 0,

lambda= seq(0.001,0.01,length= 20)),

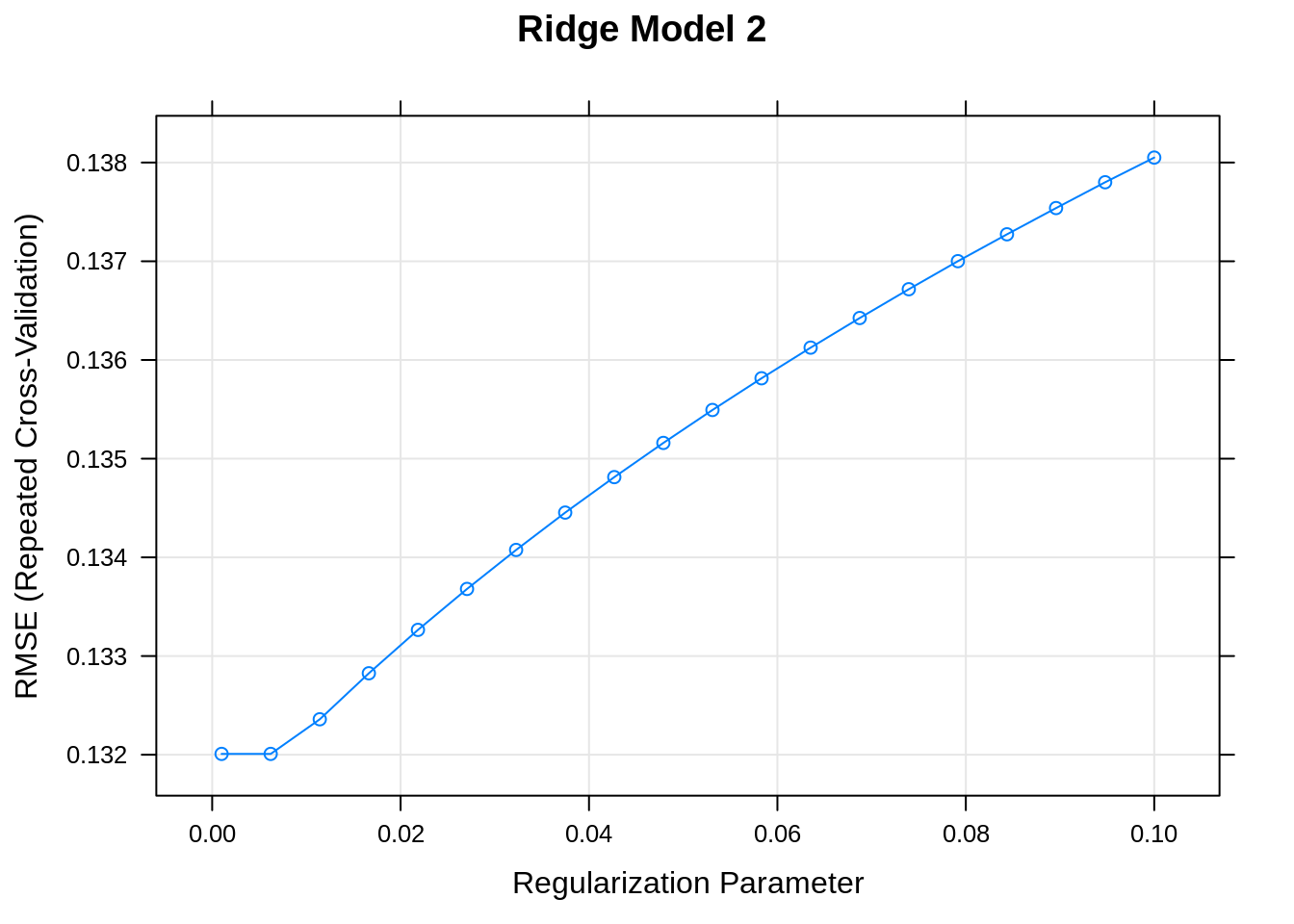
trControl = customTrainControl)

par(mfrow=c(1,2))

plot(ridge, main="Ridge Model 1 ") *# PLot suggest the*



plot(ridge2, main="Ridge Model 2 ")

 When we had lambda sequence between 0 to 1, it was close to 0.01 or less than that, we reduced the lambda to see its impact better in model 1. We can see the Ridge model start increase the RMSE with lambda = 0.007.

ridge *# SHows the best model my rmse and alfa as zero as we are ridge refression*

## glmnet

##

## 2055 samples

## 32 predictor

##

## No pre-processing

## Resampling: Cross-Validated (10 fold, repeated 5 times)

## Summary of sample sizes: 1849, 1849, 1850, 1849, 1850, 1850, ...

## Resampling results across tuning parameters:

##

## lambda RMSE Rsquared MAE

## 0.001000000 0.1320349 0.4103829 0.1037743

## 0.001473684 0.1320349 0.4103829 0.1037743

## 0.001947368 0.1320349 0.4103829 0.1037743

## 0.002421053 0.1320349 0.4103829 0.1037743

## 0.002894737 0.1320349 0.4103829 0.1037743

## 0.003368421 0.1320349 0.4103829 0.1037743

## 0.003842105 0.1320349 0.4103829 0.1037743

## 0.004315789 0.1320349 0.4103829 0.1037743

## 0.004789474 0.1320349 0.4103829 0.1037743

## 0.005263158 0.1320349 0.4103829 0.1037743

## 0.005736842 0.1320349 0.4103829 0.1037743

## 0.006210526 0.1320349 0.4103829 0.1037743

## 0.006684211 0.1320349 0.4103829 0.1037743

## 0.007157895 0.1320349 0.4103829 0.1037743

## 0.007631579 0.1320370 0.4103727 0.1037766

## 0.008105263 0.1320716 0.4101163 0.1038227

## 0.008578947 0.1321167 0.4097658 0.1038824

## 0.009052632 0.1321610 0.4094235 0.1039404

## 0.009526316 0.1322054 0.4090829 0.1039984

## 0.010000000 0.1322494 0.4087470 0.1040558

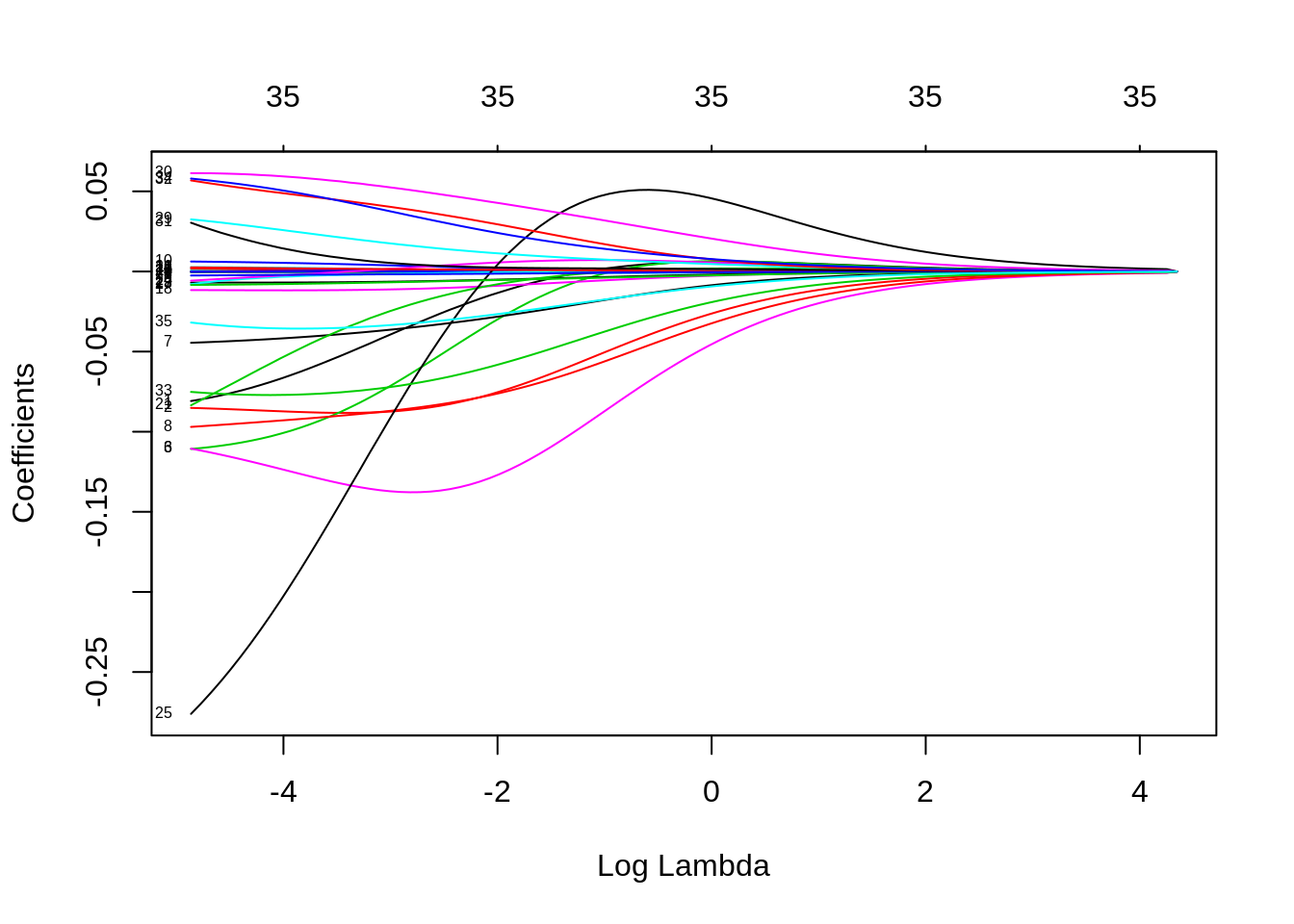
##

## Tuning parameter 'alpha' was held constant at a value of 0

## RMSE was used to select the optimal model using the smallest value.

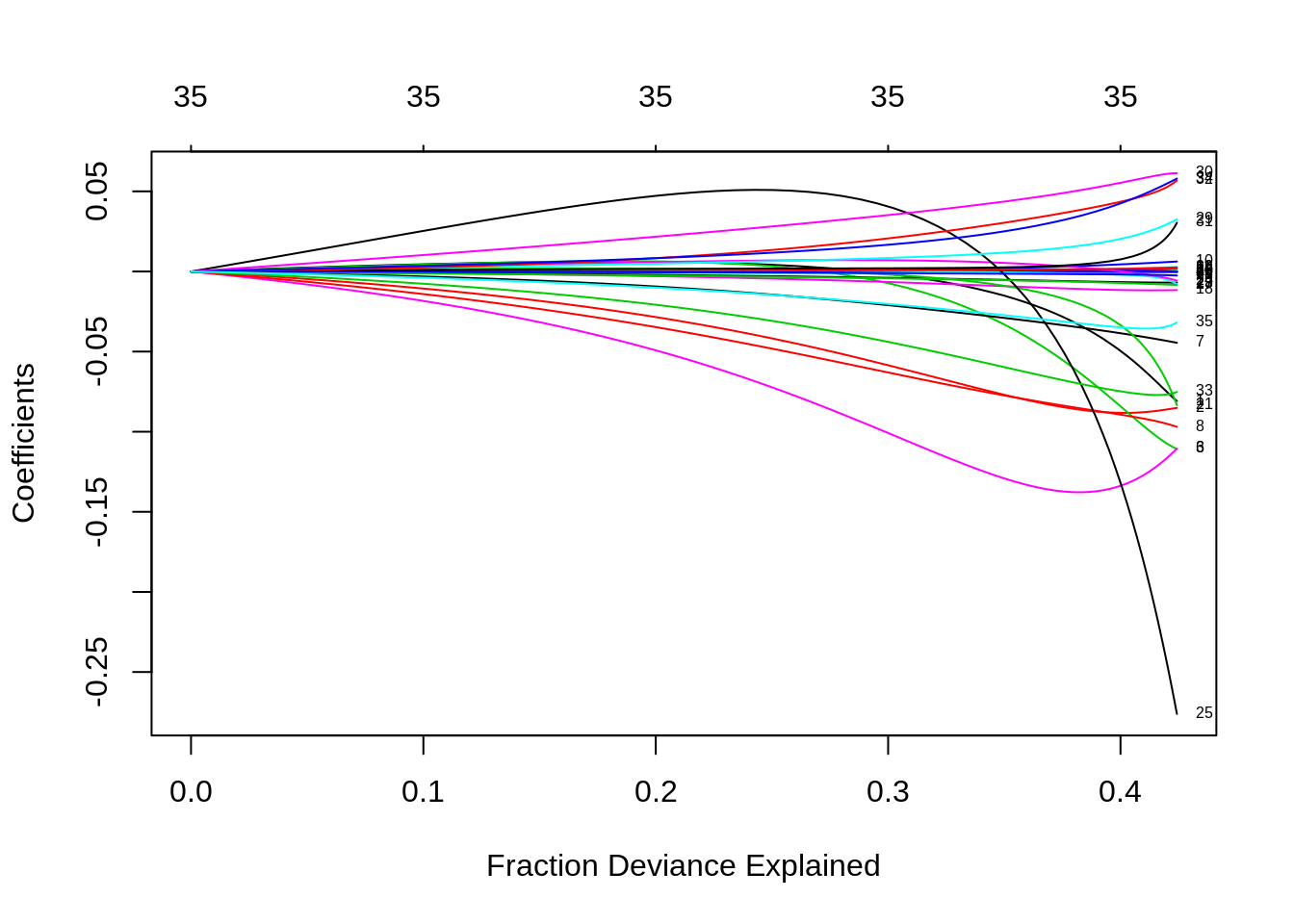
## The final values used for the model were alpha = 0 and lambda = 0.007157895.

plot(ridge$finalModel, xvar= "lambda",label= T)



When Log Lamda is big all coefficients are close to zero, but when we relax lambda, coefficients increase. As you can see we have all 35 variables used in the model coefficients even when all the coefficients are close to zero. So increasing the lamda is not helping us get away with multicollinearity attributes or any other nonsignificant variables.

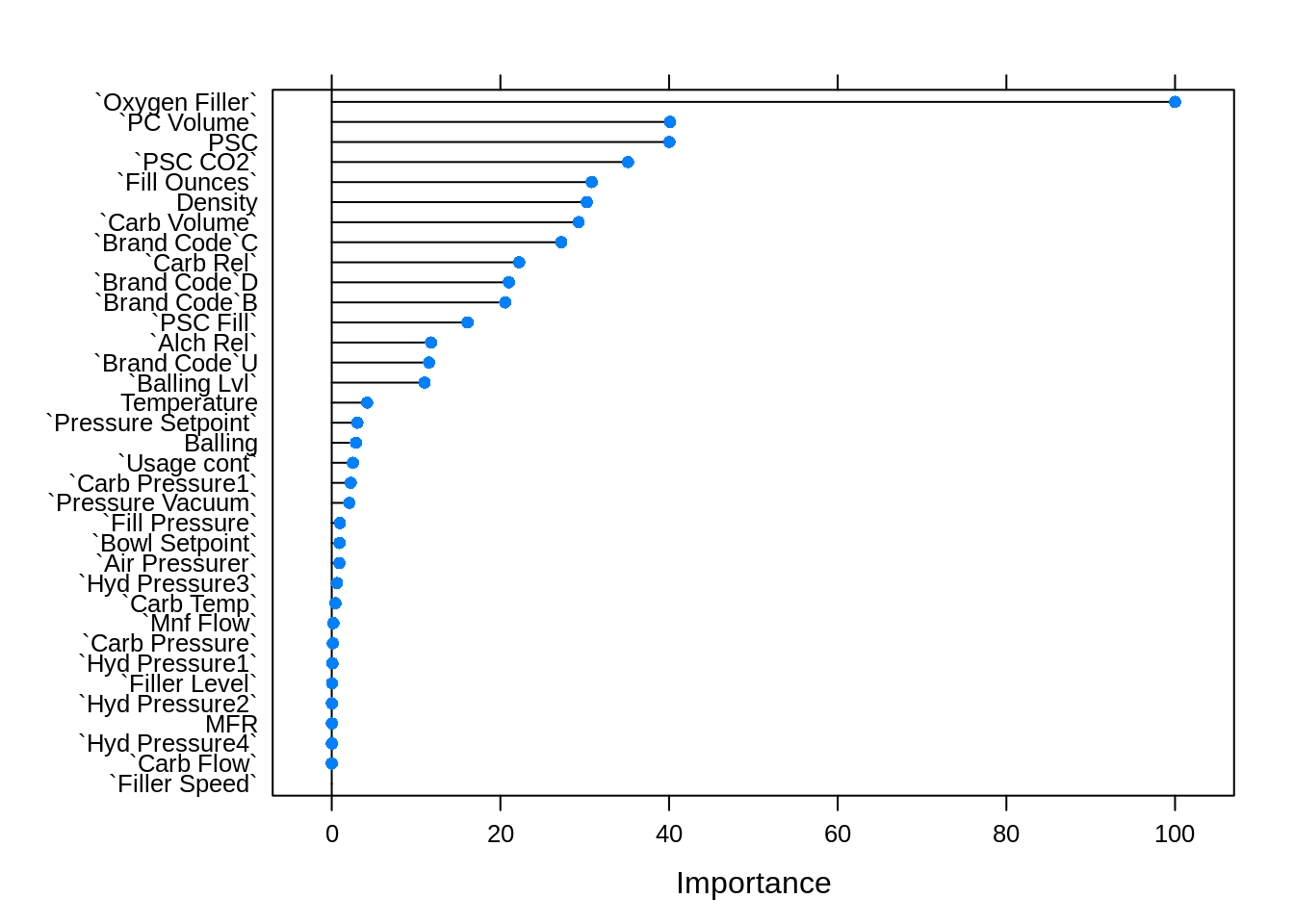
plot(ridge$finalModel,xvar= 'dev', label=T)



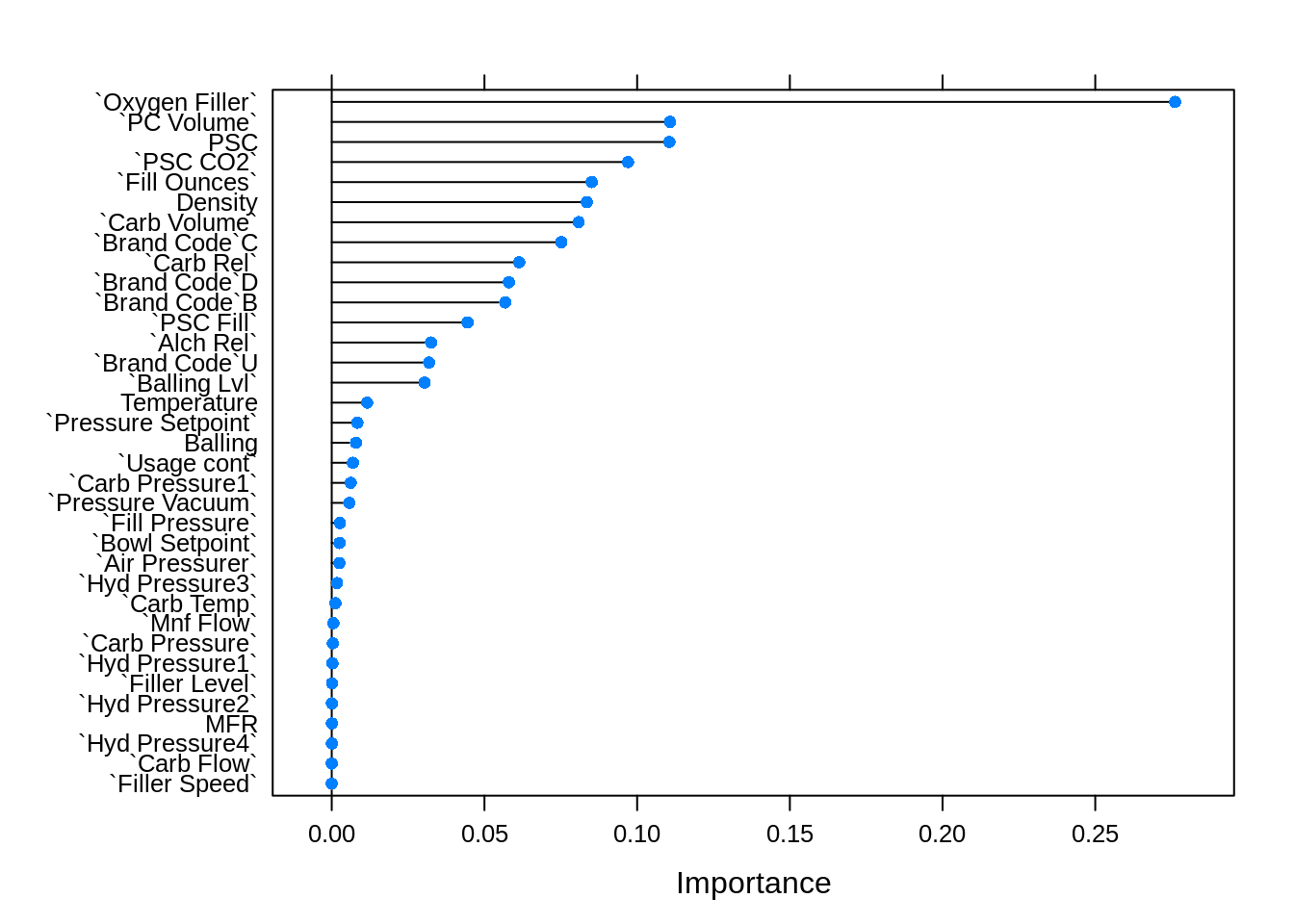
This plot very clearly says that this model only explains little more than 40% of the total data. In that too until 30%, we see a slight trend in the coefficients but after that, they have moved too much in different directions, and coefficients has increased a lot for variable 25. These variables may not be significant for the model.

Let’s see how the other imp. variables stand in this model.

plot(varImp(ridge,scale = T))



plot(varImp(ridge,scale = F))



### Lasso Regression

*#Lasso Regression*

*# IT slects feature and also drops varaibles that has multicolinerlty*

set.seed(3456)

customTrainControl <- trainControl(method = "repeatedcv",

number = 10 ,

repeats = 5 ,

verboseIter = F)

lasso1 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

metric = metric,

tuneGrid = expand.grid(alpha= 1,

lambda= seq(0.0001,0.001,length= 20)),

trControl = customTrainControl)

lasso2 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

metric = metric,

tuneGrid = expand.grid(alpha= 1,

lambda= seq(0.001,0.0001,length= 20)),

trControl = customTrainControl)

lasso3 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

metric = metric,

tuneGrid = expand.grid(alpha= 1,

lambda= seq(0.001,0.00005,length= 20)),

trControl = customTrainControl)

lasso4 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

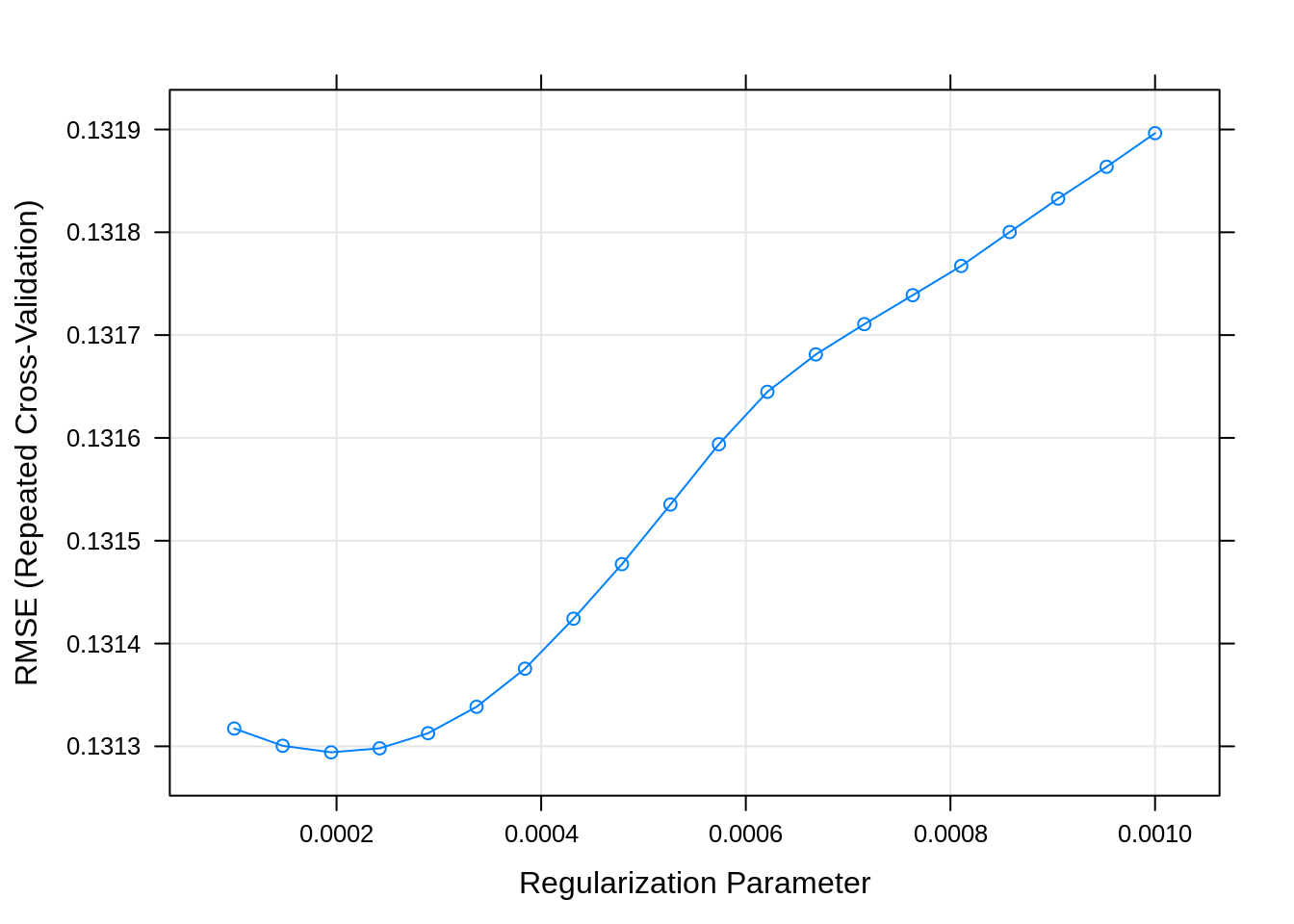
metric = metric,

tuneGrid = expand.grid(alpha= 1,

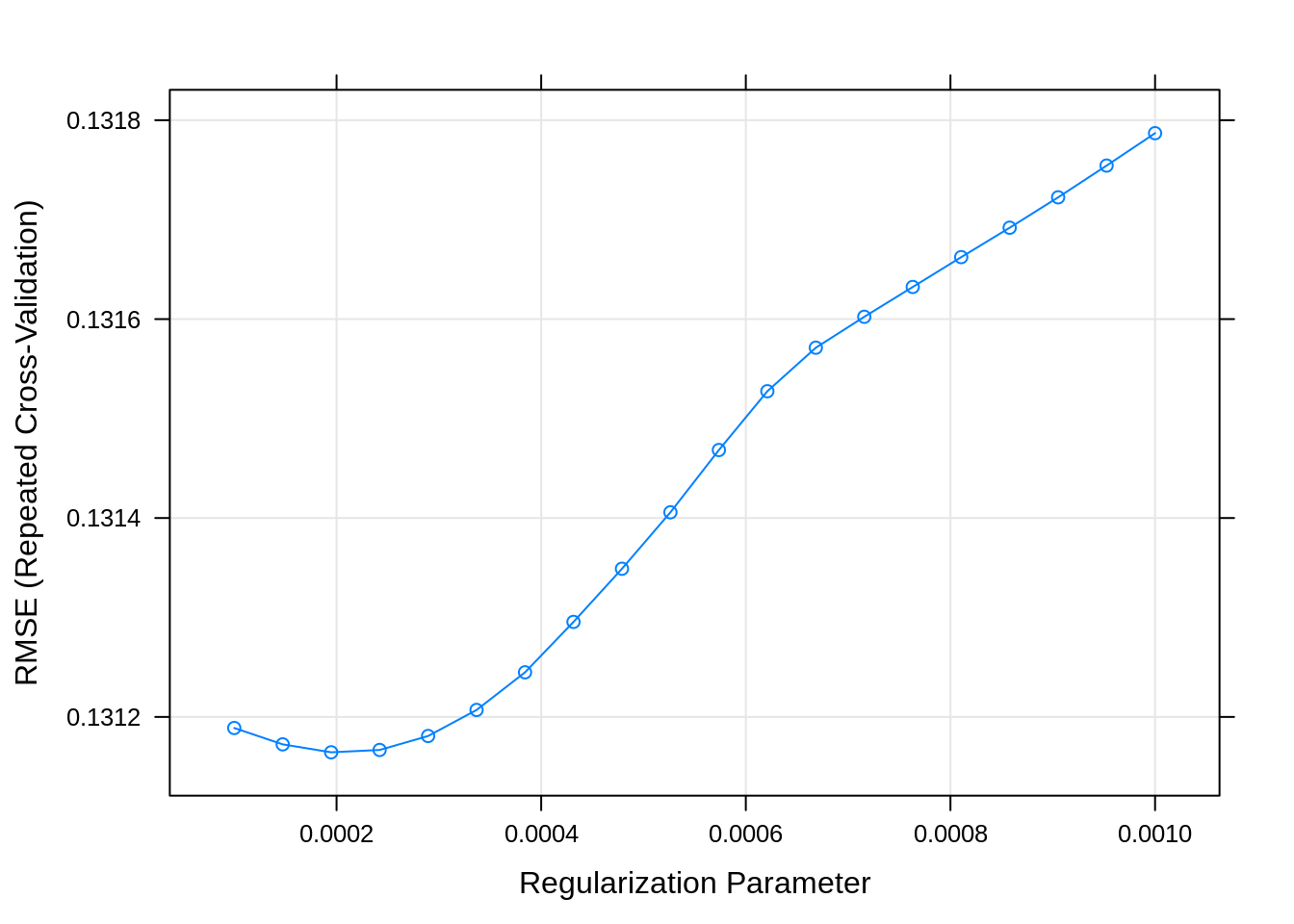
lambda= seq(0.00001,0.00003,length= 20)),

trControl = customTrainControl)

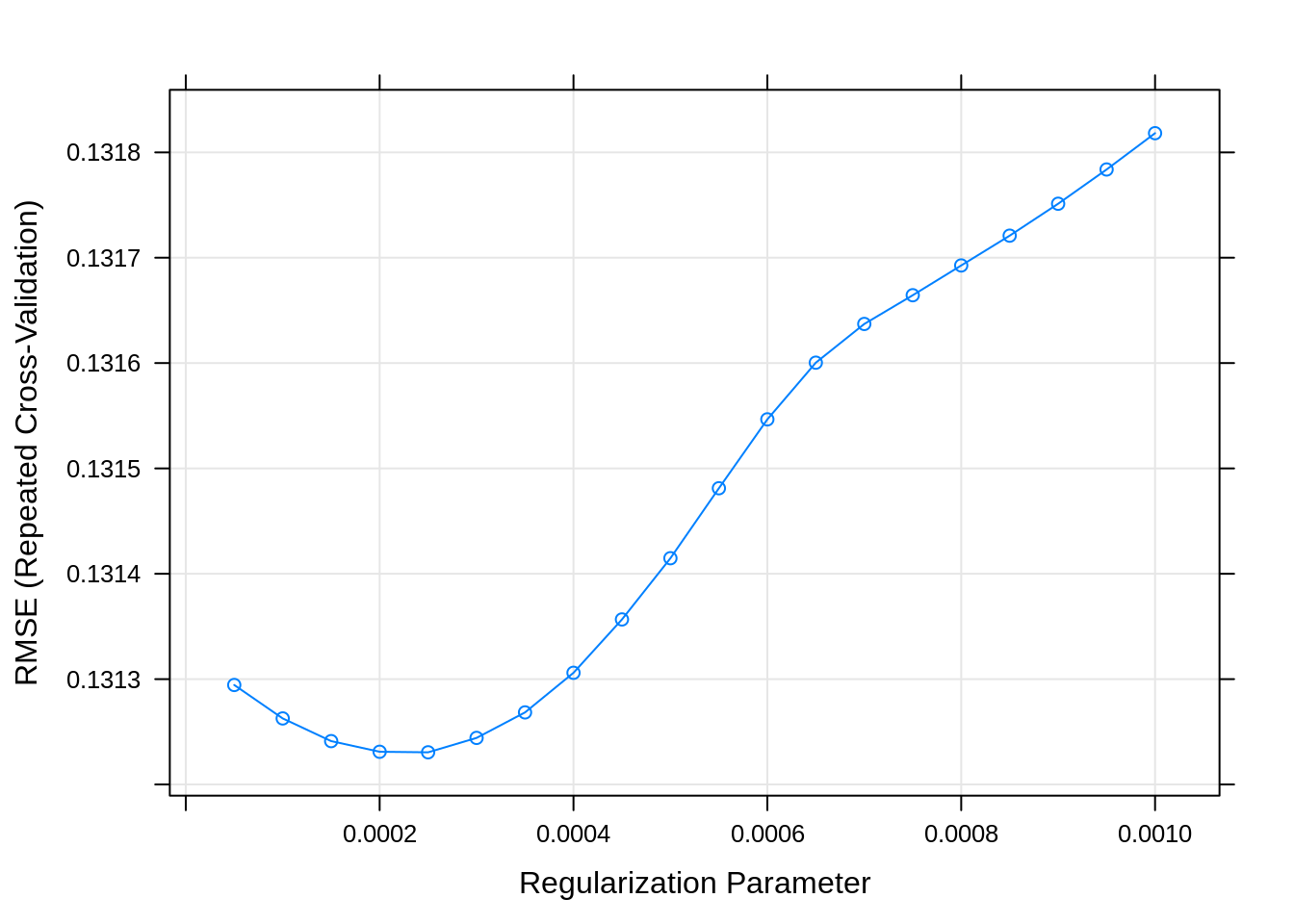
plot(lasso1)



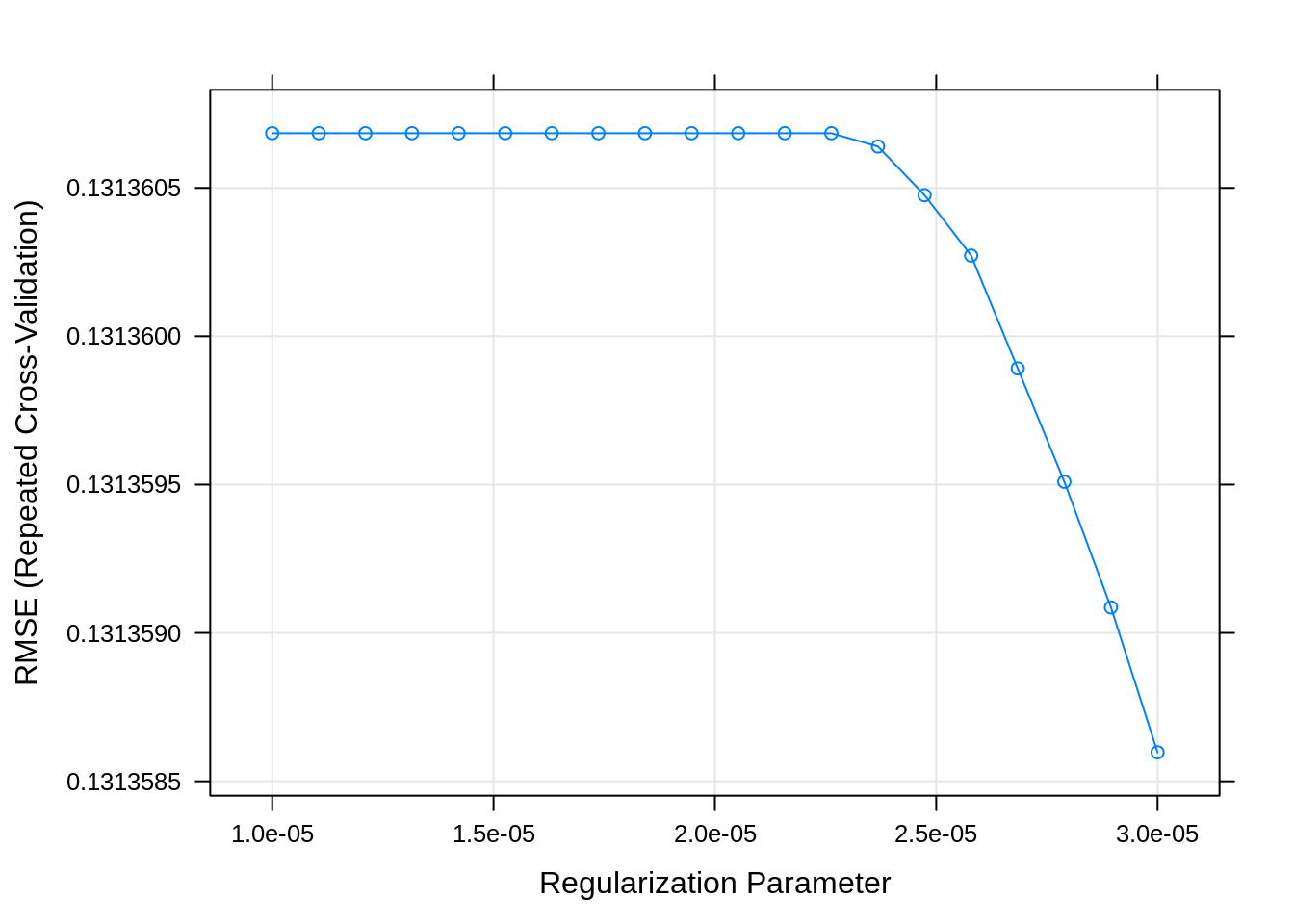
plot(lasso2)



plot(lasso3) *# PLot suggest the*



plot(lasso4)



lasso1$bestTune

|  |
| --- |
|  |
|  | | alpha  <dbl> | lambda  <dbl> |
| 3 | | 1 | 0.0001947368 |

1 row

lasso4$bestTune

|  |
| --- |
|  |
|  | | alpha  <dbl> | lambda  <dbl> |
| 20 | | 1 | 3e-05 |

1 row

lasso<- lasso4

lasso *# Shows the best model my rmse and alfa as zero as we are ridge regression*

## glmnet

##

## 2055 samples

## 32 predictor

##

## No pre-processing

## Resampling: Cross-Validated (10 fold, repeated 5 times)

## Summary of sample sizes: 1850, 1849, 1849, 1850, 1849, 1850, ...

## Resampling results across tuning parameters:

##

## lambda RMSE Rsquared MAE

## 1.000000e-05 0.1313607 0.4153846 0.1024867

## 1.105263e-05 0.1313607 0.4153846 0.1024867

## 1.210526e-05 0.1313607 0.4153846 0.1024867

## 1.315789e-05 0.1313607 0.4153846 0.1024867

## 1.421053e-05 0.1313607 0.4153846 0.1024867

## 1.526316e-05 0.1313607 0.4153846 0.1024867

## 1.631579e-05 0.1313607 0.4153846 0.1024867

## 1.736842e-05 0.1313607 0.4153846 0.1024867

## 1.842105e-05 0.1313607 0.4153846 0.1024867

## 1.947368e-05 0.1313607 0.4153846 0.1024867

## 2.052632e-05 0.1313607 0.4153846 0.1024867

## 2.157895e-05 0.1313607 0.4153846 0.1024867

## 2.263158e-05 0.1313607 0.4153846 0.1024867

## 2.368421e-05 0.1313606 0.4153850 0.1024866

## 2.473684e-05 0.1313605 0.4153863 0.1024866

## 2.578947e-05 0.1313603 0.4153880 0.1024866

## 2.684211e-05 0.1313599 0.4153911 0.1024864

## 2.789474e-05 0.1313595 0.4153943 0.1024863

## 2.894737e-05 0.1313591 0.4153978 0.1024862

## 3.000000e-05 0.1313586 0.4154018 0.1024860

##

## Tuning parameter 'alpha' was held constant at a value of 1

## RMSE was used to select the optimal model using the smallest value.

## The final values used for the model were alpha = 1 and lambda = 3e-05.

With alpha = 1 and lambda = 3e-05 , we selected lasso model, after multiple model tuning .

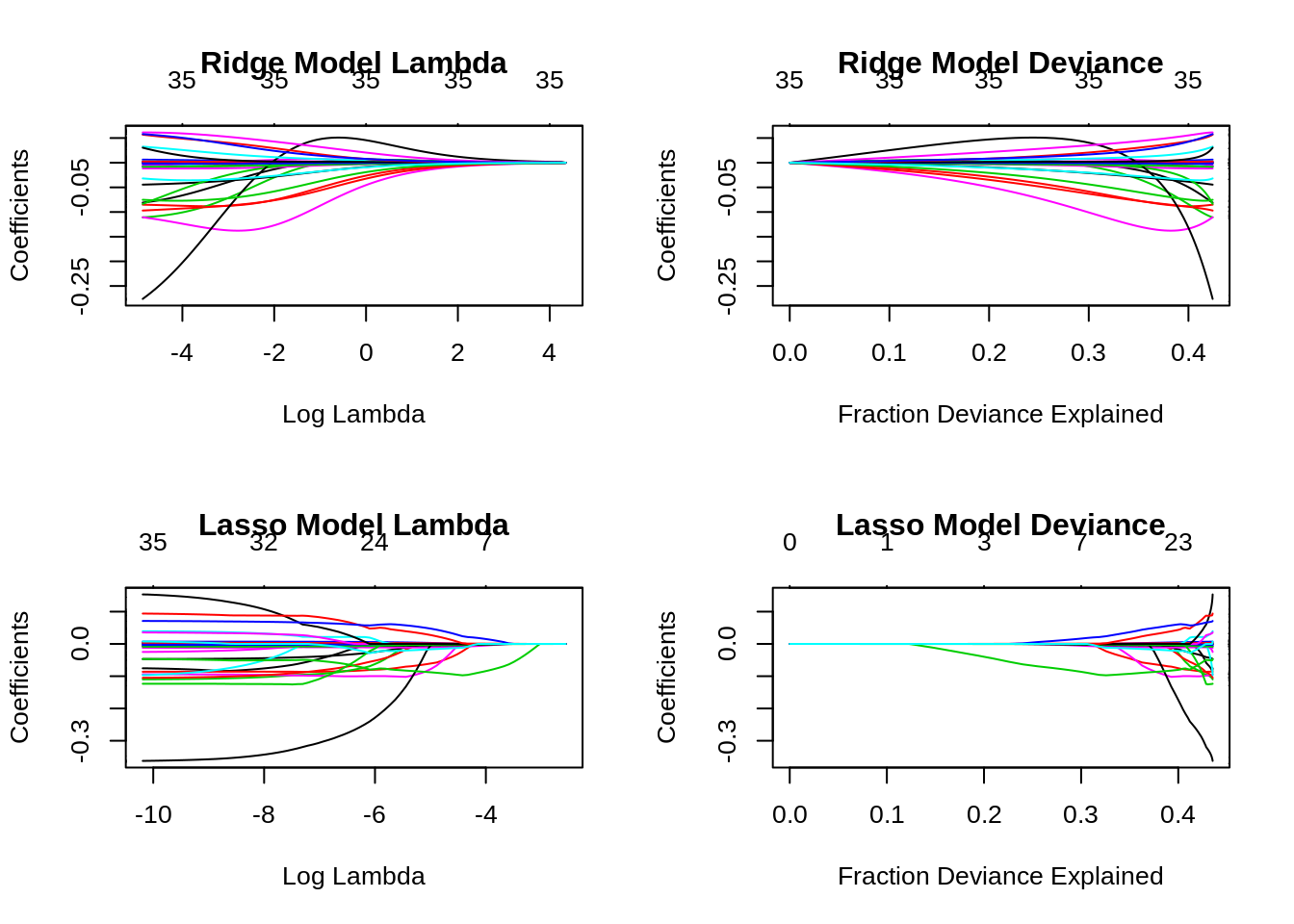
par(mfrow=c(2,2))

plot(ridge$finalModel,xvar= 'lambda', label=T,main="Ridge Model Lambda")

plot(ridge$finalModel,xvar= 'dev', label=T,main="Ridge Model Deviance")

plot(lasso$finalModel, xvar= "lambda",label= T,main="Lasso Model Lambda")

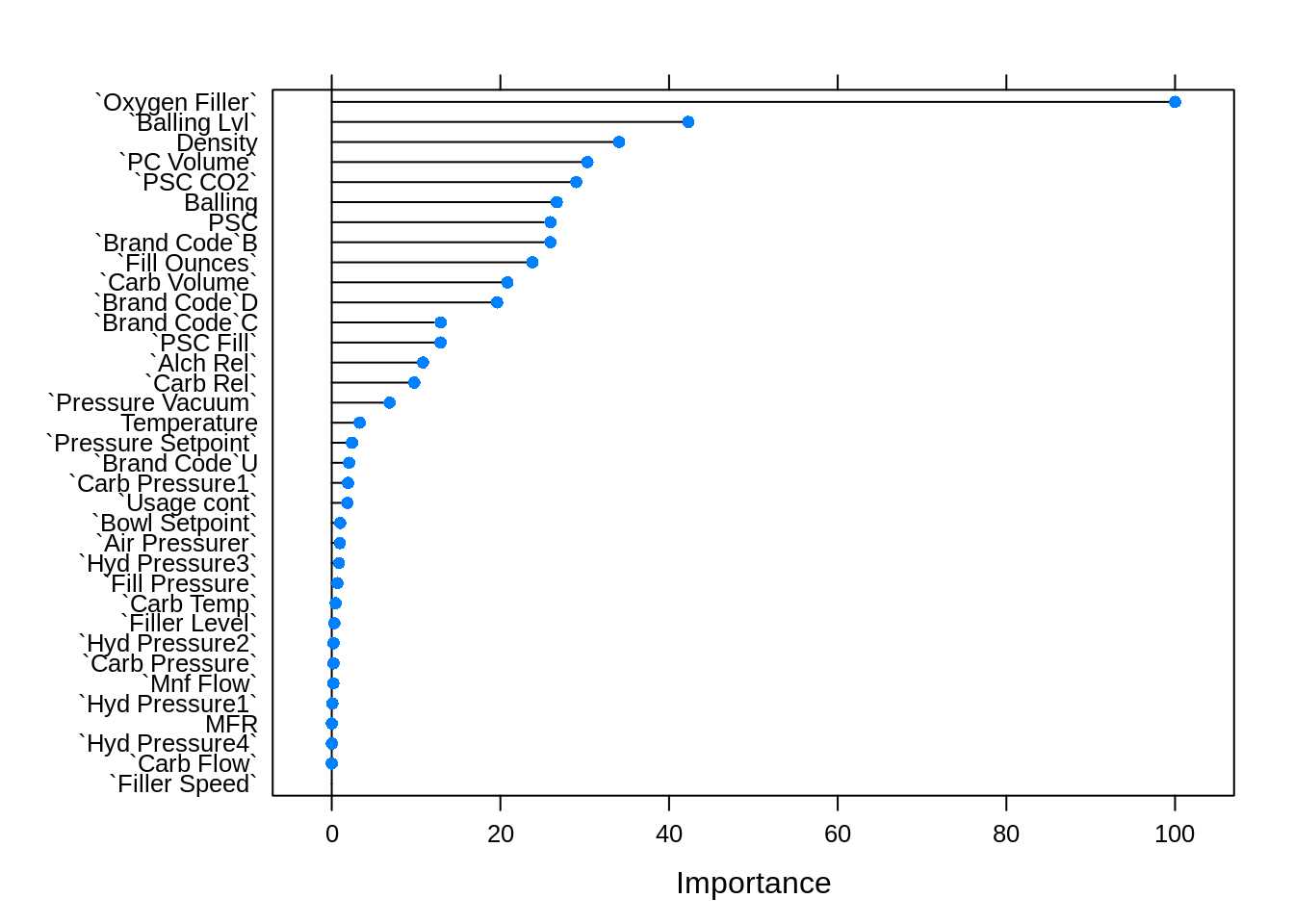
plot(lasso$finalModel,xvar= 'dev', label=T,main="Lasso Model Deviance")



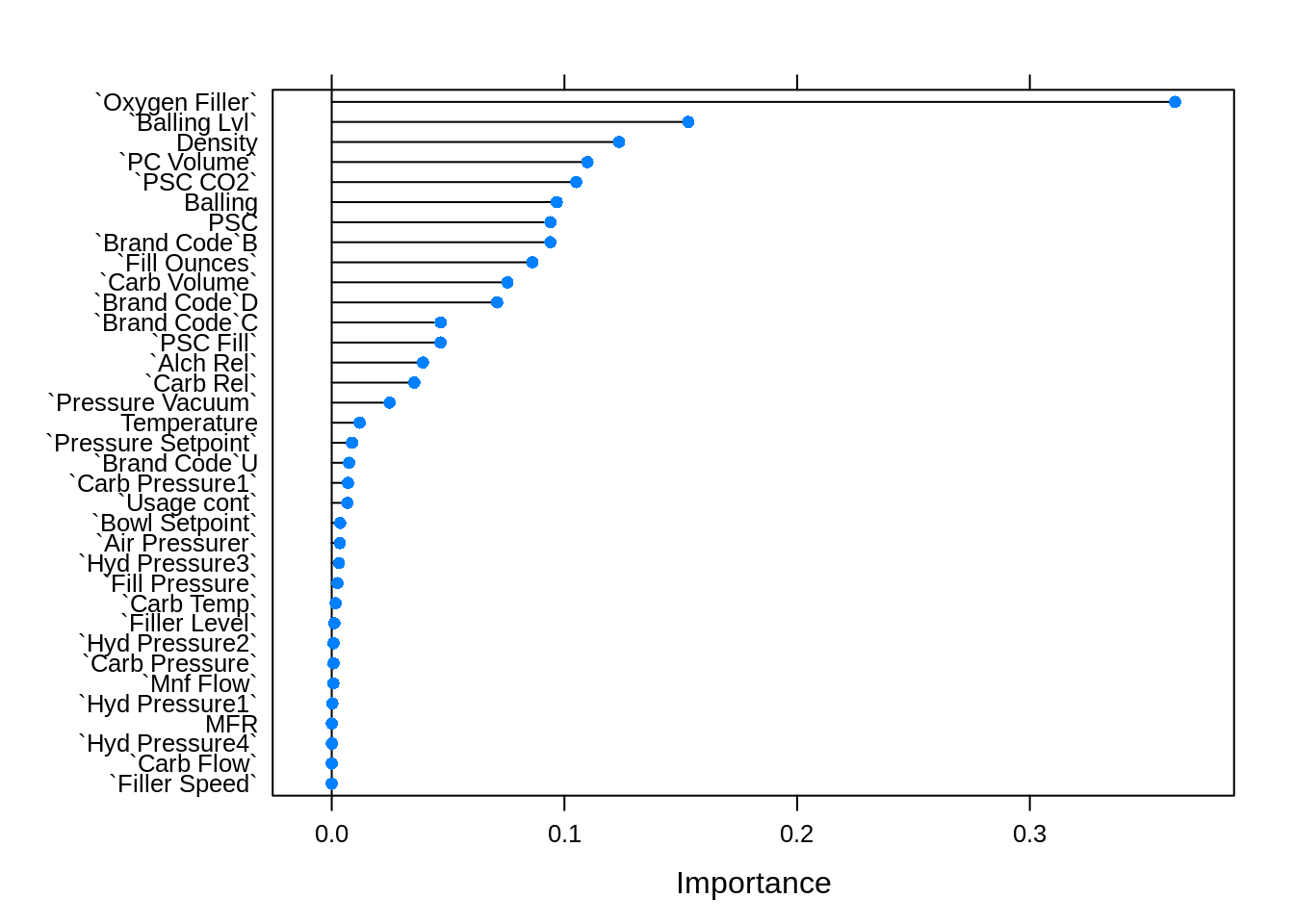
We can see that we have 25 variables when log lambda is -6, and it increases fast to log lambda go further down. Similarly, fraction deviance shows both models can explain 40% of the data but the number of coefficients increases at the end but max it goes to 27 variables. With 9 variables we can explain 30% of the data with Lasso model.

Imp variables from this model:

plot(varImp(lasso,scale = T))



plot(varImp(lasso,scale = F))



### Elastic Net Model

Let’s apply the Elastic Net model where we can tune both the parameter i.e. Alpha and Lambda. So our model can give us a hybrid model and best performance.

en1 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

tuneGrid = expand.grid(alpha= seq(0.001,0.01,length=10),

lambda= seq(0.00001,0.00003,length= 10)),

trControl = customTrainControl)

en2 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

preProcess = c("center", "scale"),

tuneGrid = expand.grid(alpha= seq(0.001,0.01,length=10),

lambda= seq(0.00001,0.00003,length= 10)),

trControl = customTrainControl)

en3 <- train(pH ~ .,

bev\_model\_train,

method= 'glmnet',

metric = metric,

tuneGrid = expand.grid(alpha= seq(0,0.1,length=10),

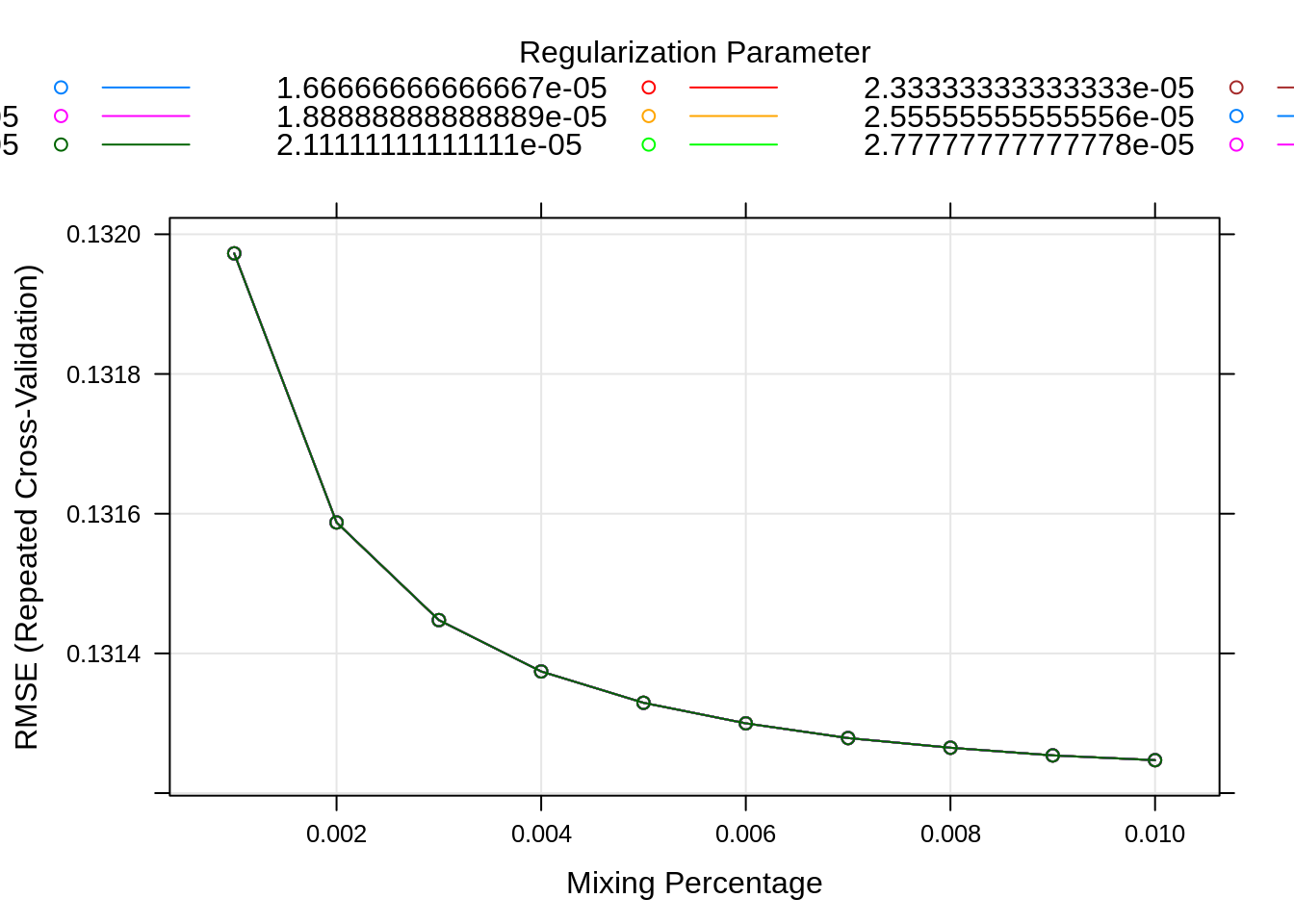
lambda= seq(0.00001,0.00003,length= 10)),

trControl = customTrainControl)

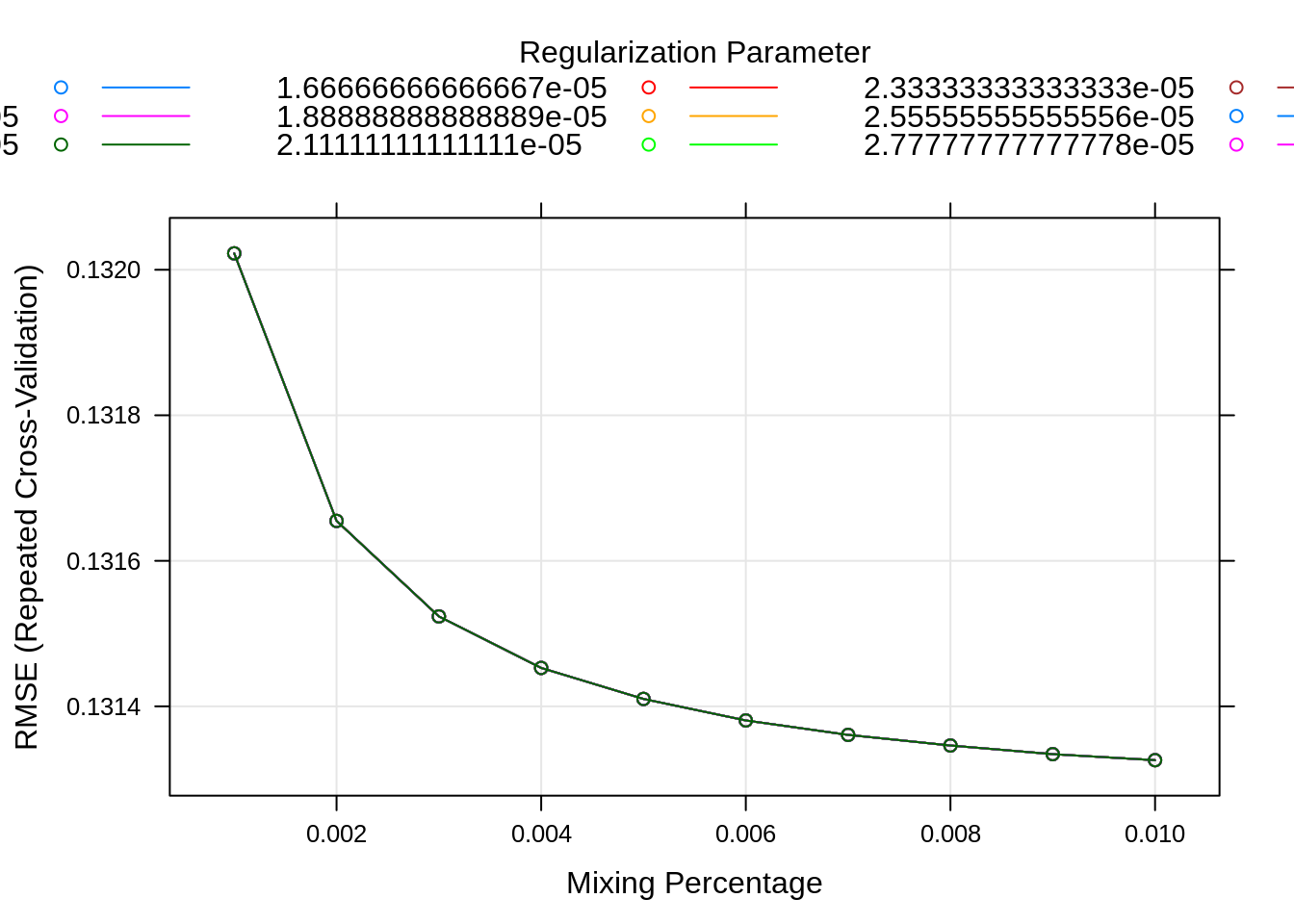
en <- en1

par(mfrow=c(2,2))

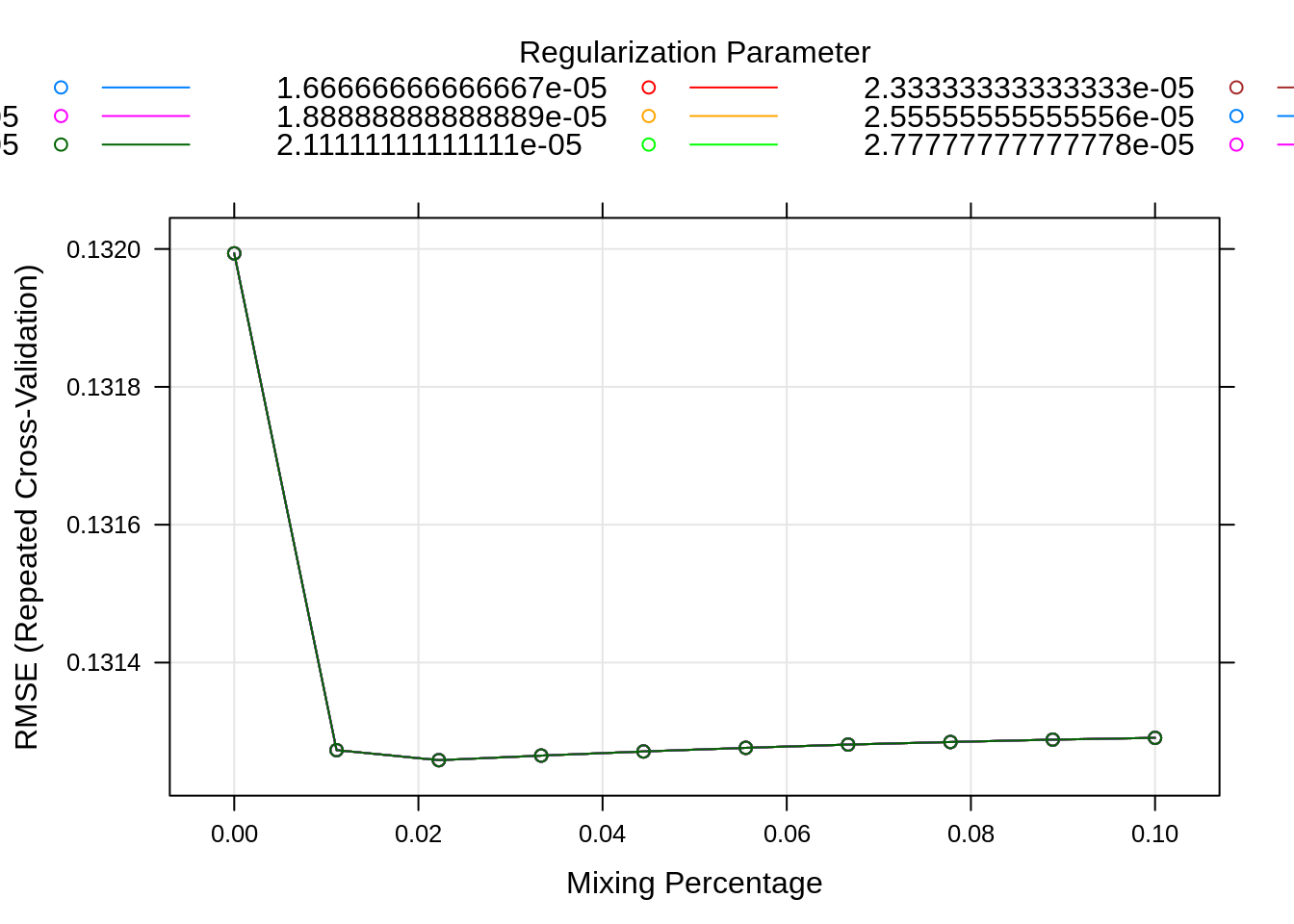
plot(en1)



plot(en2)



plot(en3) *# Plot suggest the*



en1$bestTune

|  |
| --- |
|  |
|  | | alpha  <dbl> | lambda  <dbl> |
| 100 | | 0.01 | 3e-05 |

1 row

en2$bestTune

|  |
| --- |
|  |
|  | | alpha  <dbl> | lambda  <dbl> |
| 100 | | 0.01 | 3e-05 |

1 row

en3$bestTune

|  |
| --- |
|  |
|  | | alpha  <dbl> | lambda  <dbl> |
| 30 | | 0.02222222 | 3e-05 |

1 row

*# As we can see the best tune is close to 0.00003 for lambda and alpha is 0.009 , We choose to see how these models look.*

The above plot suggests how the mixing parameter is changing for each regularization parameter and its impact on RMSE. We note from the last plot that alpha of 0.009 and lambda = 3e-05 is the best-tuned parameter of the elastic net model.

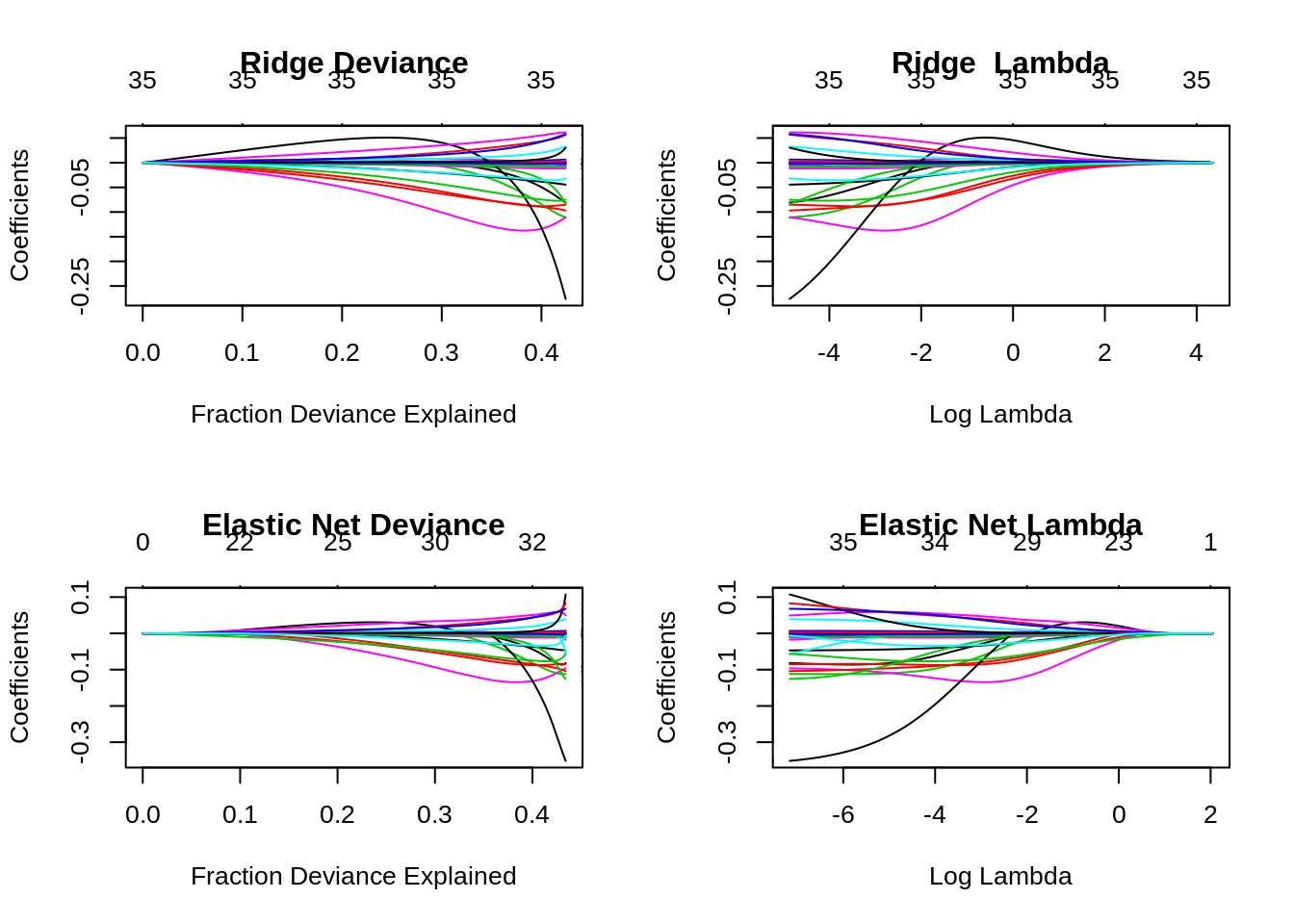
par(mfrow=c(2,2))

plot(ridge$finalModel,xvar= 'dev', label=T,main="Ridge Deviance")

plot(ridge$finalModel, xvar= "lambda",label= T,main="Ridge Lambda")

plot(en1$finalModel,xvar= 'dev', label=T,main="Elastic Net Deviance")

plot(en1$finalModel, xvar= "lambda",label= T,main="Elastic Net Lambda")

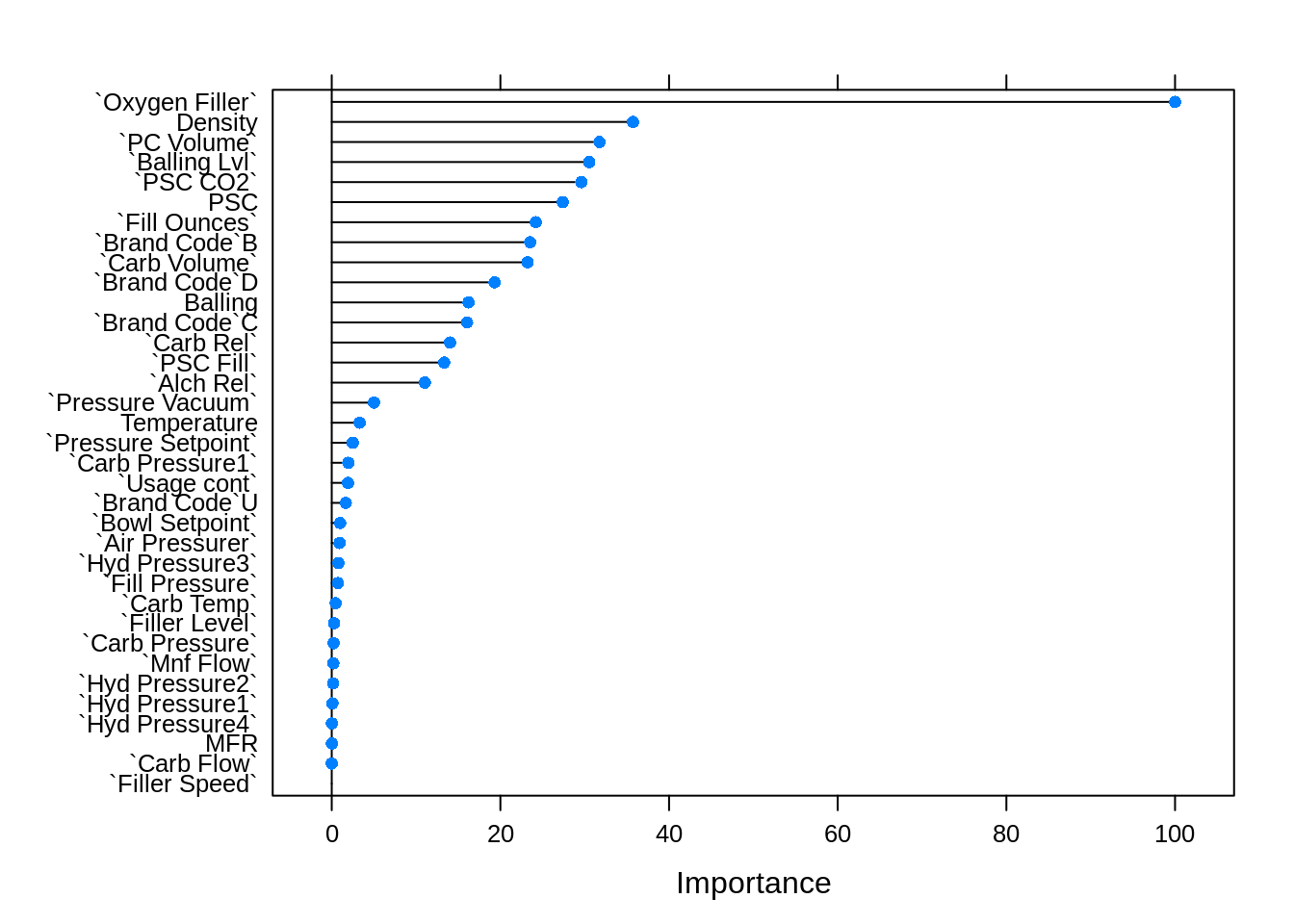


For elastic net model when log lambda =2 coefficients is close to zero, when it’s close to -2 it is having 29 variables and coefficients starts increasing rapidly. after log lambda crosses -4. Variable 25 which would have less impact on the model as it’s increasing very fast after log lambda is equal to -2.3.

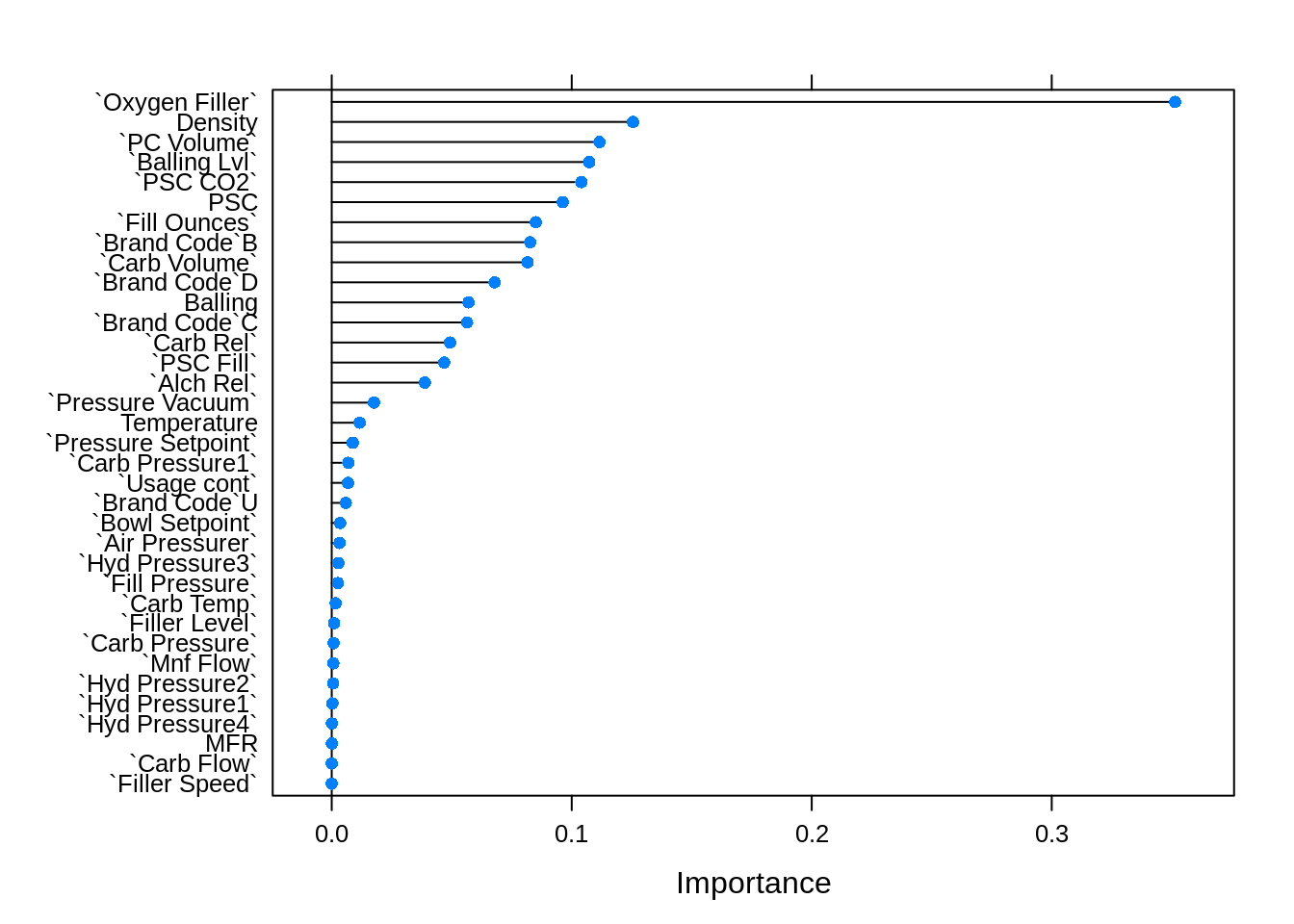
We don’t’ see many variations in the r-squared, this model seems to explain little more than 40% of the data, which is the same trend we have seen with the Lasso and Ridge model.

Let’s see impo variables:

plot(varImp(en1,scale = T))



plot(varImp(en1,scale = F))



model\_list <- list(Linearmodel = lm, Ridge = ridge , Lasso = lasso4,

ElasticNet= en1,

ElasticNet2 = en2,

ElasticNet3= en3 )

res <- resamples(model\_list)

summary(res)

##

## Call:

## summary.resamples(object = res)

##

## Models: Linearmodel, Ridge, Lasso, ElasticNet, ElasticNet2, ElasticNet3

## Number of resamples: 50

##

## MAE

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

## Linearmodel 0.09186350 0.09972553 0.1023034 0.1023655 0.1060179 0.1121602 0

## Ridge 0.09270570 0.09984595 0.1031833 0.1037743 0.1076150 0.1163017 0

## Lasso 0.08897654 0.09832794 0.1024924 0.1024860 0.1059790 0.1133664 0

## ElasticNet 0.09144848 0.09968102 0.1018118 0.1025915 0.1054057 0.1130878 0

## ElasticNet2 0.08824268 0.09988356 0.1020942 0.1026920 0.1070472 0.1146301 0

## ElasticNet3 0.09301287 0.09875131 0.1021940 0.1025994 0.1058778 0.1125598 0

##

## RMSE

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

## Linearmodel 0.1164416 0.1276706 0.1316864 0.1313413 0.1358476 0.1421800 0

## Ridge 0.1176747 0.1270073 0.1321035 0.1320349 0.1370879 0.1465547 0

## Lasso 0.1122011 0.1261659 0.1319352 0.1313586 0.1362078 0.1444531 0

## ElasticNet 0.1147645 0.1272717 0.1309882 0.1312472 0.1347260 0.1484133 0

## ElasticNet2 0.1149102 0.1266663 0.1307006 0.1313260 0.1376951 0.1454193 0

## ElasticNet3 0.1167273 0.1266896 0.1302771 0.1312584 0.1353114 0.1455213 0

##

## Rsquared

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

## Linearmodel 0.3376874 0.3872186 0.4173057 0.4163949 0.4430922 0.4958844 0

## Ridge 0.2897285 0.3838107 0.4119380 0.4103829 0.4409249 0.5111793 0

## Lasso 0.3222525 0.3882618 0.4121616 0.4154018 0.4383940 0.5266727 0

## ElasticNet 0.3253614 0.3834487 0.4164049 0.4166440 0.4408288 0.5300486 0

## ElasticNet2 0.3114430 0.3940444 0.4181120 0.4156182 0.4450998 0.4879624 0

## ElasticNet3 0.3099016 0.3842531 0.4240033 0.4178741 0.4479655 0.5275398 0

We can very clearly see that R-squared is better with close to 39.75 with the en1 model, but its Mean RMSE is not the winner but we would choose the en1 model due to its better Mean R-squared value.

### Best model

en1$bestTune *#$ 0.009 can be said to be more of close to zero , and its more of ridgw modwl.*

|  |
| --- |
|  |
|  | | alpha  <dbl> | lambda  <dbl> |
| 100 | | 0.01 | 3e-05 |

1 row

finalLMe<- en1$finalModel

coef(finalLMe,s= en1$bestTune$lambda)

## 36 x 1 sparse Matrix of class "dgCMatrix"

## 1

## (Intercept) 1.078813e+01

## `Carb Volume` -8.165601e-02

## `Fill Ounces` -8.505360e-02

## `PC Volume` -1.115321e-01

## `Carb Pressure` -7.884060e-04

## `Carb Temp` 1.642959e-03

## PSC -9.628233e-02

## `PSC Fill` -4.686808e-02

## `PSC CO2` -1.040152e-01

## `Mnf Flow` -6.712918e-04

## `Carb Pressure1` 6.934425e-03

## `Fill Pressure` 2.600573e-03

## `Hyd Pressure1` -2.818124e-04

## `Hyd Pressure2` -5.875973e-04

## `Hyd Pressure3` 2.827433e-03

## `Hyd Pressure4` 9.088797e-05

## `Filler Level` -1.007688e-03

## `Filler Speed` 7.438542e-07

## Temperature -1.165132e-02

## `Usage cont` -6.768409e-03

## `Carb Flow` 2.040281e-05

## Density -1.255158e-01

## MFR -4.645567e-05

## Balling -5.701340e-02

## `Pressure Vacuum` -1.762016e-02

## `Oxygen Filler` -3.513832e-01

## `Bowl Setpoint` 3.490909e-03

## `Pressure Setpoint` -8.756079e-03

## `Air Pressurer` -3.288605e-03

## `Alch Rel` 3.886897e-02

## `Carb Rel` 4.927516e-02

## `Balling Lvl` 1.073054e-01

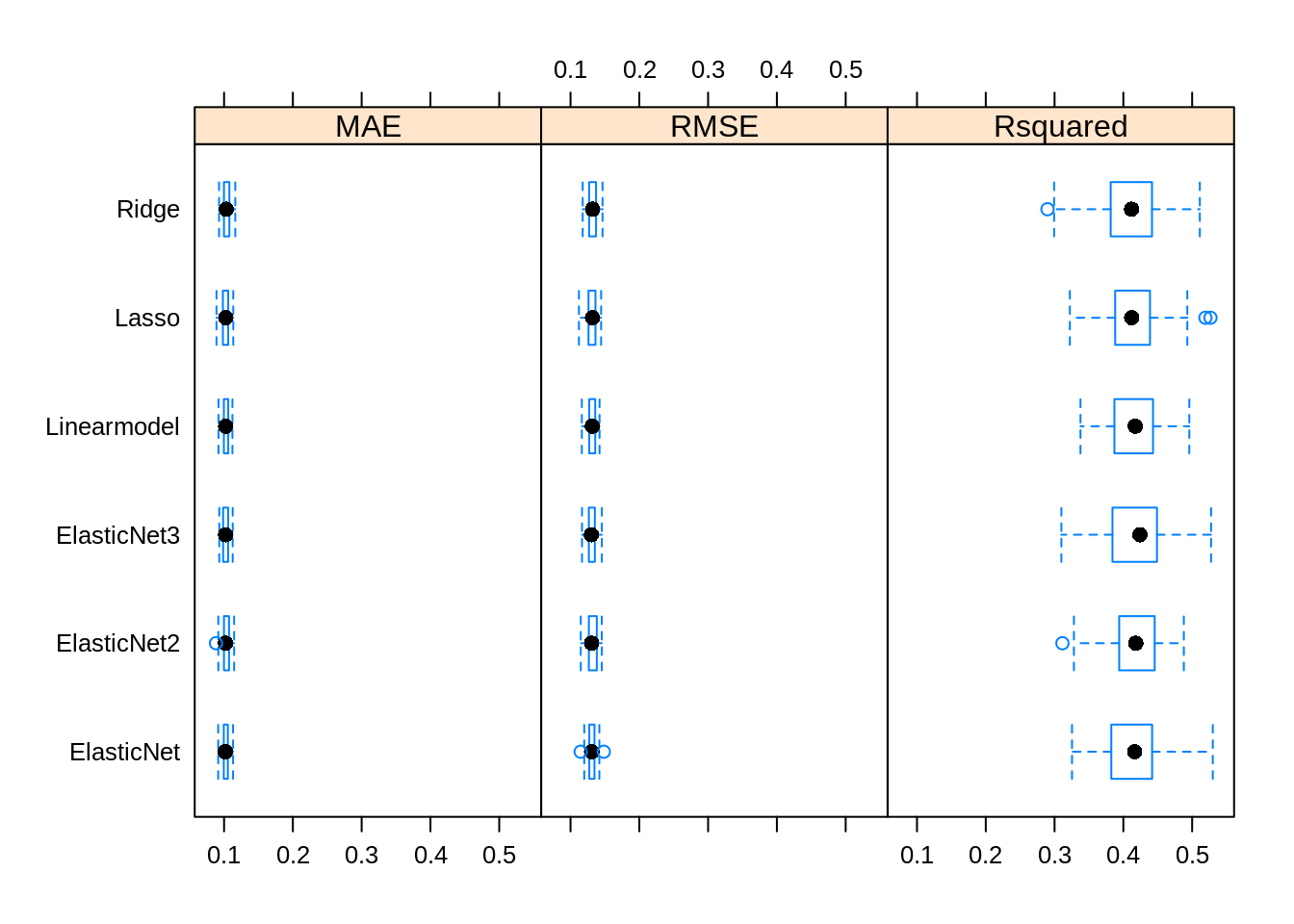
## `Brand Code`B 8.268195e-02

## `Brand Code`C -5.640632e-02

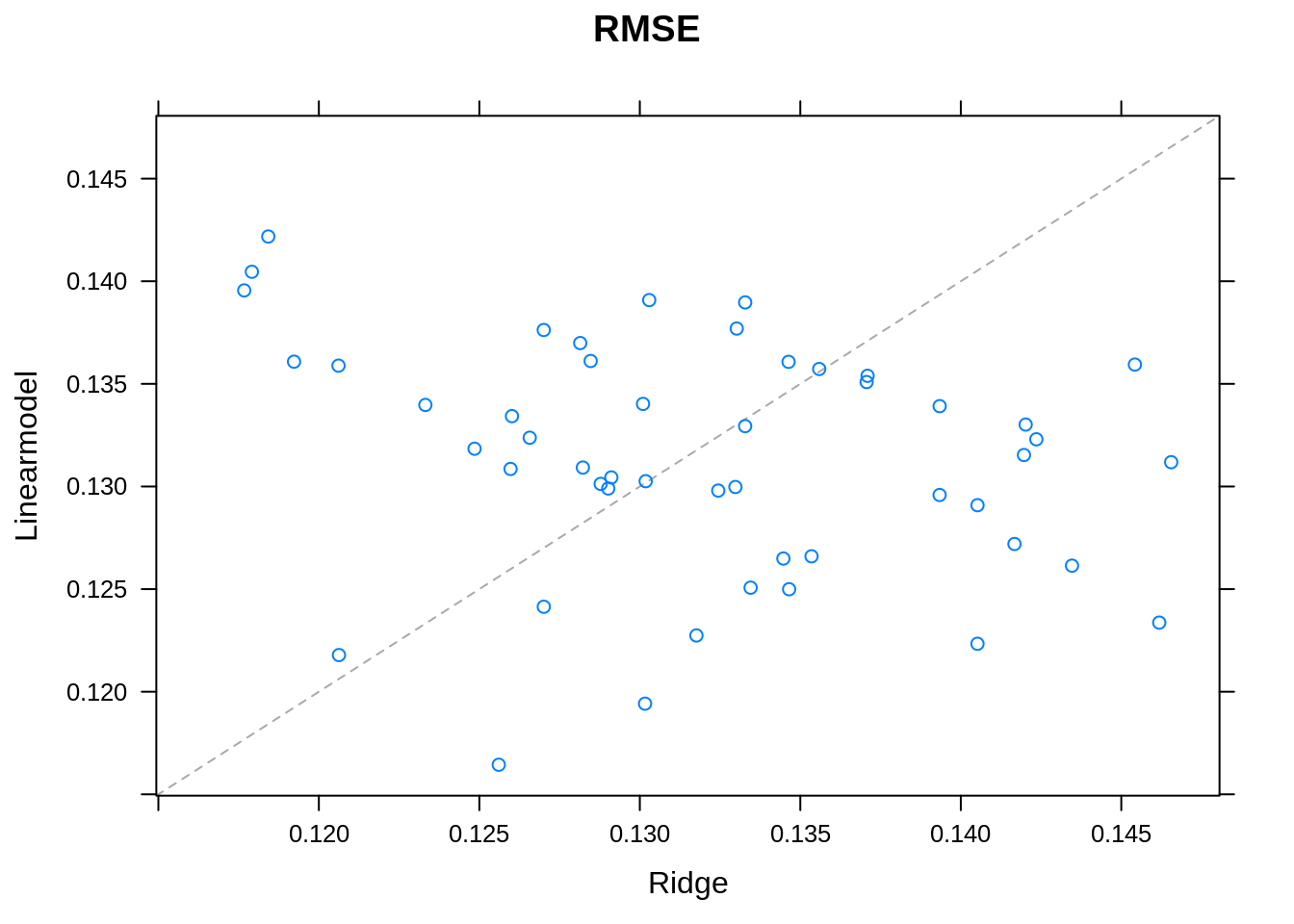
## `Brand Code`D 6.788007e-02

## `Brand Code`U -5.863891e-03

bwplot(res)



xyplot(res,metric = "RMSE")



There is an equal number of datapoint above and below the line which suggests that RMSE for LM and Ridge model is almost the same.

### Top 5 Variables

getRank <- **function**(trainObjects,n=5){

cn = 0

temp <- c()

methods <- c()

**for**(object **in** trainObjects){

methods <- c(methods, object$method)

varimp <- varImp(object)[[1]]

varimp$variables <- row.names(varimp)

rank <- varimp[order(varimp$Overall, decreasing = T),] %>% row.names()

temp <- cbind(temp[1:5], rank[1:5])

}

temp <- as.data.frame(temp)

names(temp) <- methods

temp$Rank <- c(1:dim(temp)[1])

temp <- select(temp, Rank, everything())

**return**(temp)

}

getRank(list(lm))

|  |
| --- |
|  |
| Rank  <int> | | lm  <fctr> |
| 1 | | `\\`Mnf Flow\\`` |
| 2 | | `\\`Carb Pressure1\\`` |
| 3 | | `\\`Bowl Setpoint\\`` |
| 4 | | `\\`Balling Lvl\\`` |
| 5 | | `\\`Usage cont\\`` |

5 rows

getRank(list(lasso))

|  |
| --- |
|  |
| Rank  <int> | | glmnet  <fctr> |
| 1 | | `Oxygen Filler` |
| 2 | | `Balling Lvl` |
| 3 | | Density |
| 4 | | `PC Volume` |
| 5 | | `PSC CO2` |

5 rows

getRank(list(ridge))

|  |
| --- |
|  |
| Rank  <int> | | glmnet  <fctr> |
| 1 | | `Oxygen Filler` |
| 2 | | `PC Volume` |
| 3 | | PSC |
| 4 | | `PSC CO2` |
| 5 | | `Fill Ounces` |

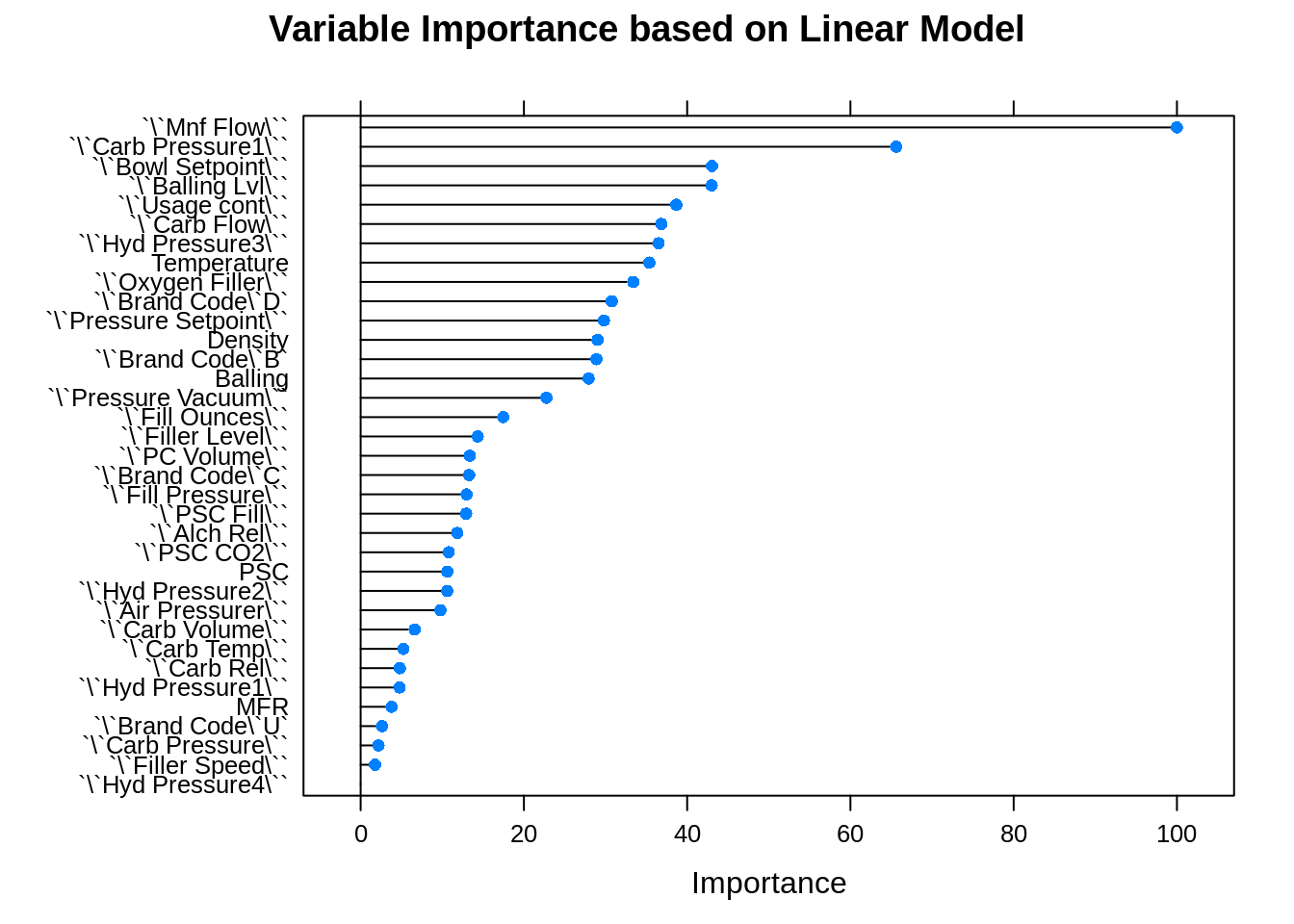
5 rows

getRank(list(en1))

|  |
| --- |
|  |
| Rank  <int> | | glmnet  <fctr> |
| 1 | | `Oxygen Filler` |
| 2 | | Density |
| 3 | | `PC Volume` |
| 4 | | `Balling Lvl` |
| 5 | | `PSC CO2` |

5 rows

plot(varImp(lm, scale=T), main='Variable Importance based on Linear Model')



### Test from the training set

*# Train data Test set*

X\_test <- bev\_model\_test[,-32] *# Dropped PH*

Y\_test <- bev\_model\_test[,32] *# Only PH*

test\_model <- **function**(modelName,predData){

options(warn=-1) *#turn off warnings*

predicted\_result <- predict(modelName, predData)

options(warn=1)

*#We can collect the observed and predicted values into a data frame, then use*

*# the caret function defaultSummary to estimate the test set performance*

DT\_model\_lm\_pred <- data.frame(obs=Y\_test,pred=predicted\_result)

res\_sum <- defaultSummary(DT\_model\_lm\_pred)

mape\_score <- MLmetrics::MAPE(predicted\_result,Y\_test)

**return**(cbind(res\_sum,mape\_score))

}

*#*

*# kable(list(test\_model(lm,X\_test),*

*# test\_model(ridge,X\_test),*

*# test\_model(lasso,X\_test),*

*# test\_model(en,X\_test)*

*# ))*

data.frame("LM Model"= defaultSummary(data.frame(obs=Y\_test,pred=predict(lm, X\_test))),"MAPE" = MLmetrics::MAPE(predict(lm, X\_test),Y\_test))

|  |
| --- |
|  |
|  | | LM.Model  <dbl> | MAPE  <dbl> |
| RMSE | | 0.1406472 | 0.01234998 |
| Rsquared | | 0.3642538 | 0.01234998 |
| MAE | | 0.1052219 | 0.01234998 |

3 rows

data.frame("Ridge Model"= defaultSummary(data.frame(obs=Y\_test,pred=predict(ridge, X\_test))),"MAPE" = MLmetrics::MAPE(predict(ridge, X\_test),Y\_test))

|  |
| --- |
|  |
|  | | Ridge.Model  <dbl> | MAPE  <dbl> |
| RMSE | | 0.1400393 | 0.01240247 |
| Rsquared | | 0.3709997 | 0.01240247 |
| MAE | | 0.1056628 | 0.01240247 |

3 rows

data.frame("Lasso Model"= defaultSummary(data.frame(obs=Y\_test,pred=predict(lasso, X\_test))),"MAPE" = MLmetrics::MAPE(predict(lasso, X\_test),Y\_test))

|  |
| --- |
|  |
|  | | Lasso.Model  <dbl> | MAPE  <dbl> |
| RMSE | | 0.1404304 | 0.01234 |
| Rsquared | | 0.3660780 | 0.01234 |
| MAE | | 0.1051343 | 0.01234 |

3 rows

data.frame("Elastic Net"= defaultSummary(data.frame(obs=Y\_test,pred=predict(en, X\_test))),"MAPE" = MLmetrics::MAPE(predict(en, X\_test),Y\_test))

|  |
| --- |
|  |
|  | | Elastic.Net  <dbl> | MAPE  <dbl> |
| RMSE | | 0.1398070 | 0.01232178 |
| Rsquared | | 0.3715865 | 0.01232178 |
| MAE | | 0.1049725 | 0.01232178 |

3 rows

Elastic net model seems to be having a better MAPE score of 0.0116. Let’s use the en model to check the test set.

### Test with Given Test set

*# Score data*

X\_Stest <- bev\_score[,-26] *# Dropped PH*

Y\_Stest <- bev\_score[,26] *# Only PH*

en\_lm\_result<- predict(en,X\_Stest)

*# Plots to show that how*

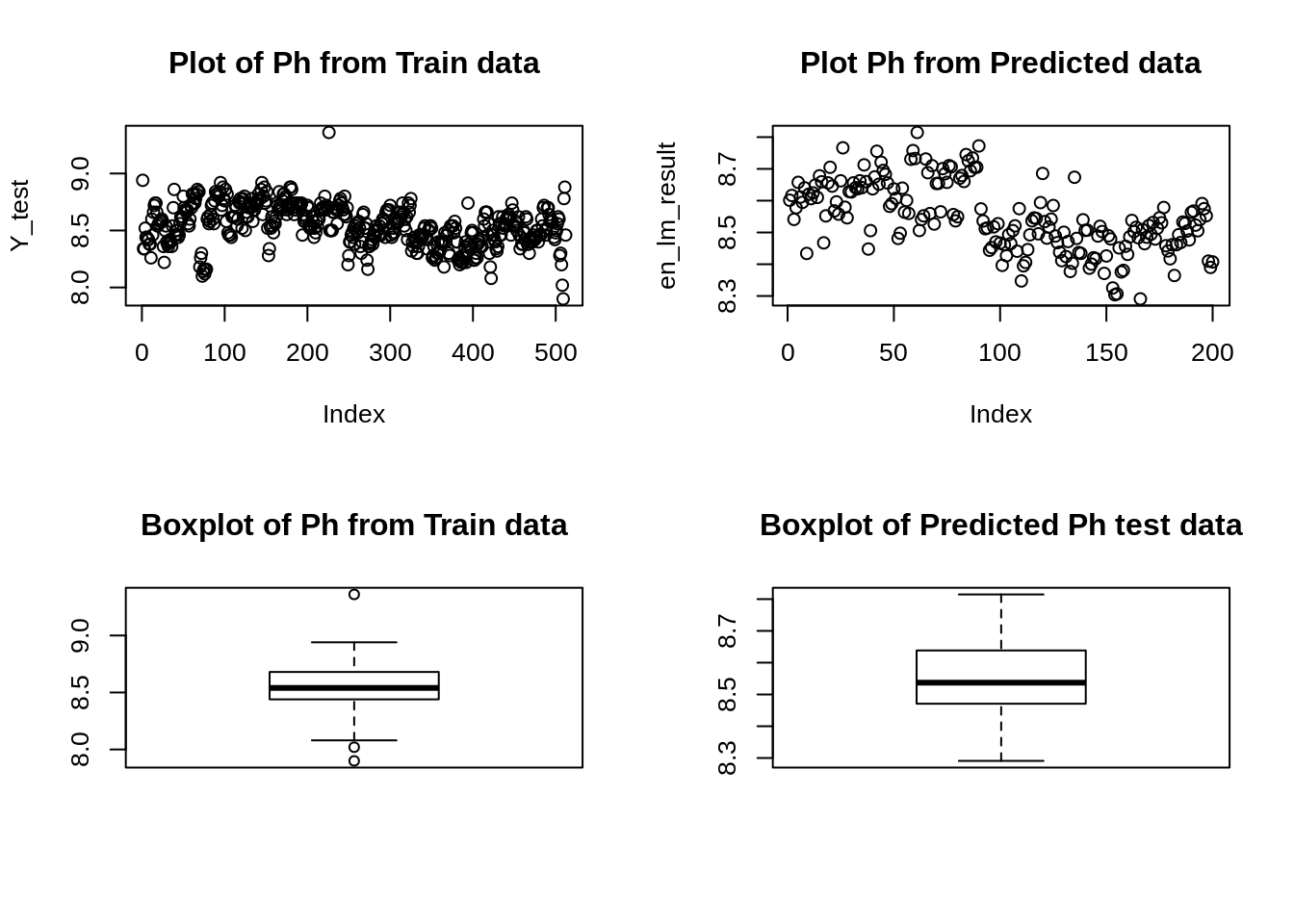
par(mfrow=c(2,2))

plot(Y\_test,main='Plot of Ph from Train data')

plot(en\_lm\_result,main='Plot Ph from Predicted data')

boxplot(Y\_test,label="sad",main='Boxplot of Ph from Train data')

boxplot(as.data.frame(en\_lm\_result)[,c(1)],main='Boxplot of Predicted Ph test data')



*# Write to file*

*# xlsx::write.xlsx(as.data.frame(en\_lm\_result)[,c(1)], file = "Project2\_lm.xlsx", col.name = T, row.names = T, append = T)*

## Nonlinear Regression

*# detect, use multicores*

numCores <- parallel::detectCores()

*# create a simple cluster on the local machine using all available threads*

cl <- parallel::makeCluster(detectCores(), methods = FALSE)

*# register our cluster*

doParallel::registerDoParallel(cl)

### MARS

*# model*

set.seed(1234)

marsModel <- caret::train(x = bev\_model\_train %>% dplyr::select(-pH),

y = bev\_model\_train %>% dplyr::select(pH) %>% .$pH,

method = "earth",

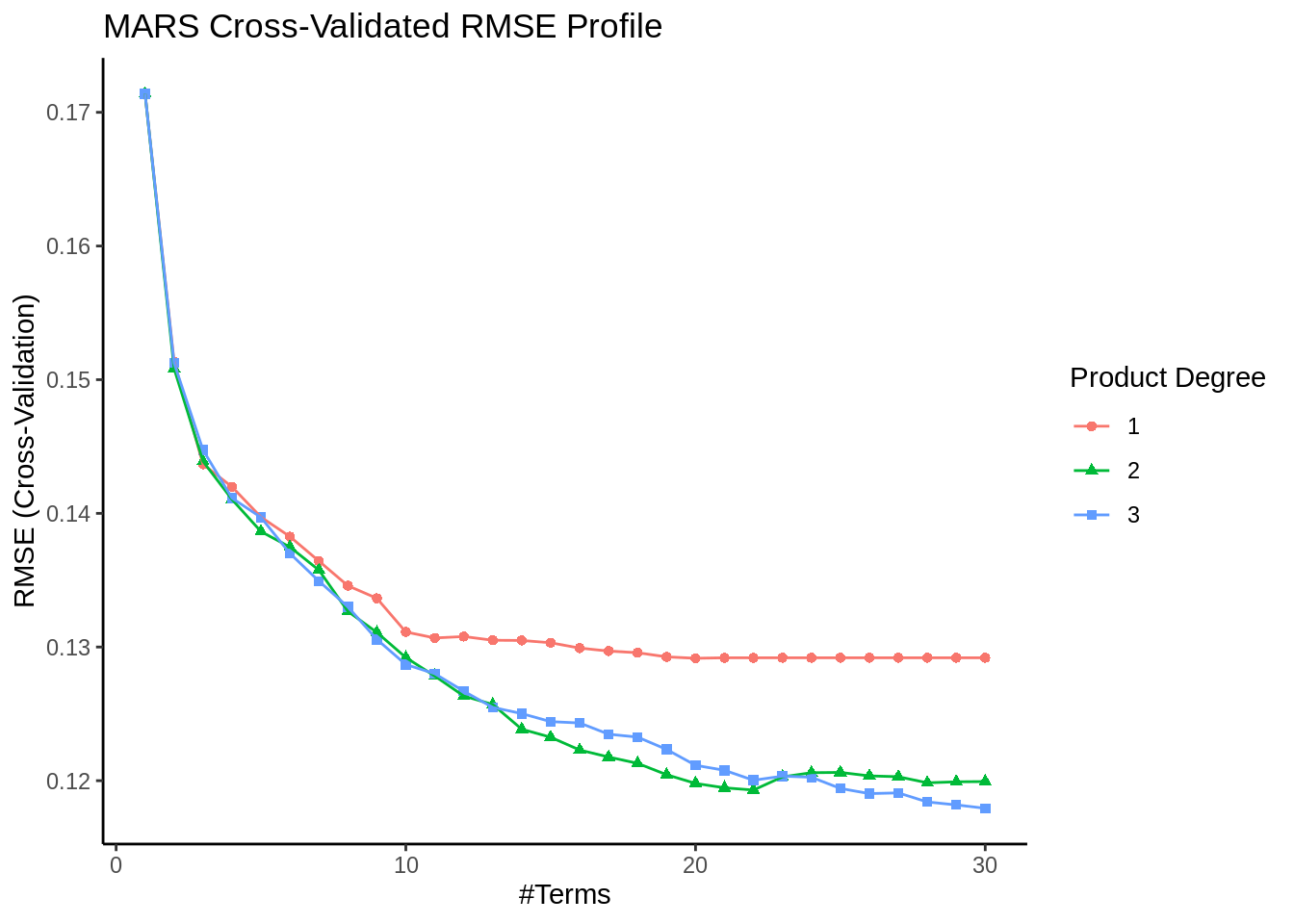
preProcess = c("center", "scale"),

tuneGrid = expand.grid(degree = 1:3, nprune = 1:30),

trControl = trainControl(method = "cv"))

*# ggplot*

ggplot(marsModel) + labs(title = "MARS Cross-Validated RMSE Profile") + theme\_classic()



*# importance (top 5)*

marsPerformance <- caret::varImp(marsModel)

marsPerformance$importance %>%

as.data.frame() %>%

tibble::rownames\_to\_column() %>%

dplyr::mutate(name = forcats::fct\_inorder(rowname)) %>%

arrange(desc(Overall)) %>%

head(5) %>%

ggplot(., aes(x = reorder(name, Overall), y = Overall)) +

geom\_point() +

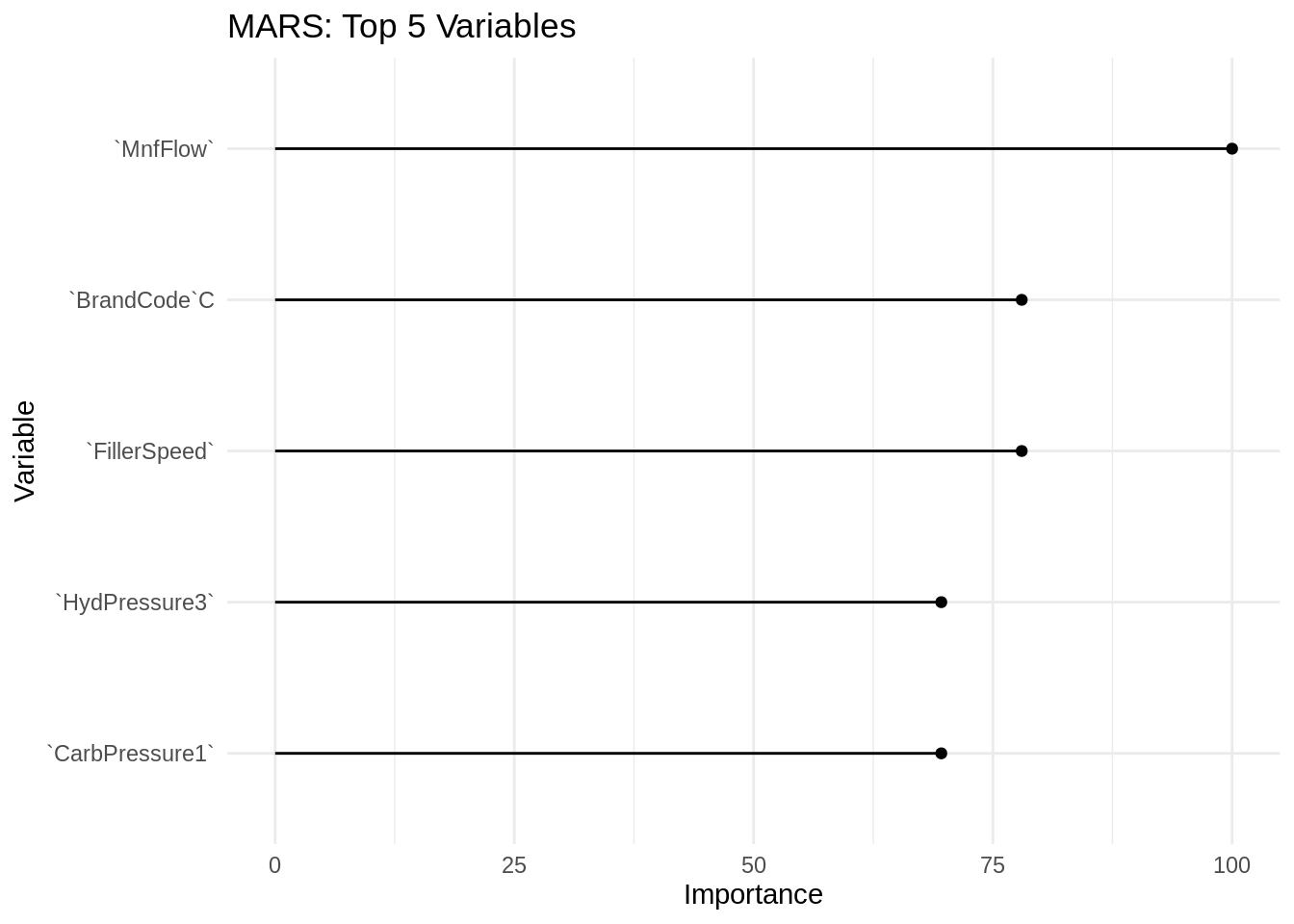
geom\_segment(aes(x = name, xend = name, y = 0, yend = Overall)) +

ggtitle("MARS: Top 5 Variables") +

labs(x = "Variable", y = "Importance") +

coord\_flip() +

theme\_minimal()



*# Validation on the hold-out set*

marsPred <- predict(marsModel, newdata = bev\_model\_test)

marsKPI <- postResample(pred = marsPred, obs = Y\_test)

marsMAPE <- MLmetrics::MAPE(predict(marsModel, X\_test), Y\_test)

### SVM

*# model*

set.seed(1234)

svmModel <- caret::train(pH ~., bev\_model\_train,

method = "svmRadial",

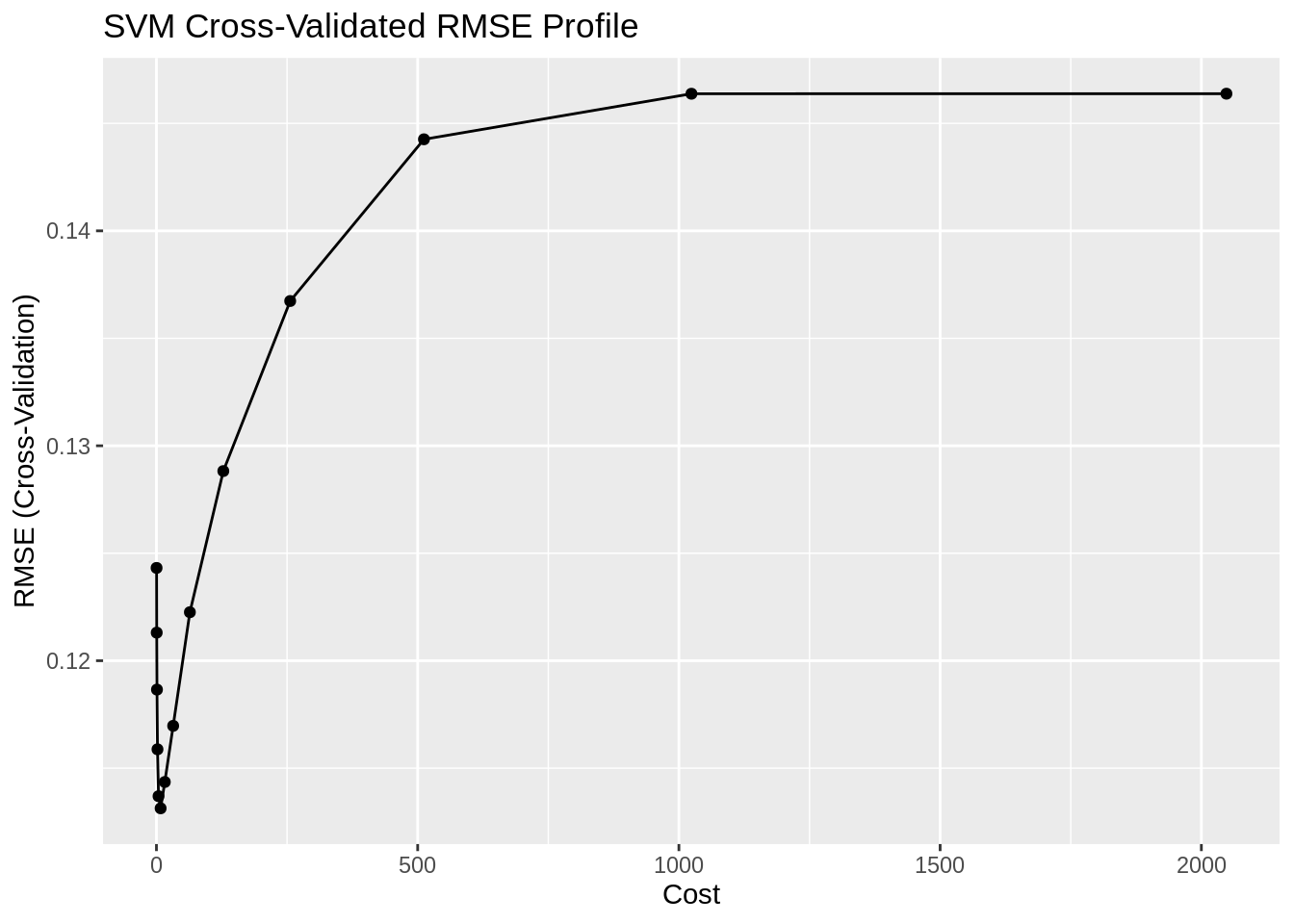
preProcess = c("center", "scale"),

tuneLength = 14,

trControl = trainControl(method = "cv"))

*# ggplot*

ggplot(svmModel) + labs(title = "SVM Cross-Validated RMSE Profile") + theme\_gray()



*# importance (top 5)*

svmPerformance <- caret::varImp(svmModel)

svmPerformance$importance %>%

as.data.frame() %>%

tibble::rownames\_to\_column() %>%

dplyr::mutate(name = forcats::fct\_inorder(rowname)) %>%

arrange(desc(Overall)) %>%

head(5) %>%

ggplot(., aes(x = reorder(name, Overall), y = Overall)) +

geom\_point() +

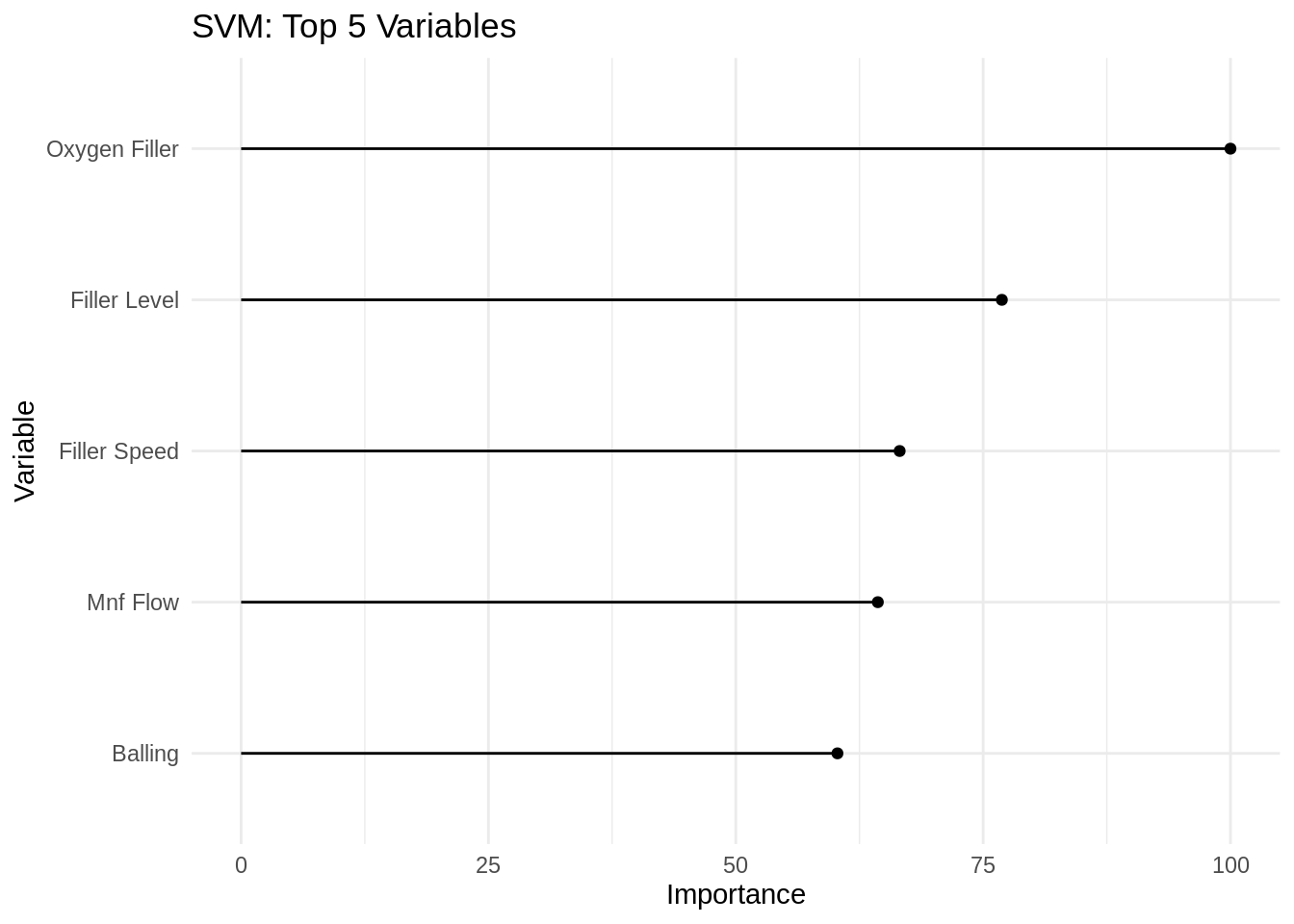
geom\_segment(aes(x = name, xend = name, y = 0, yend = Overall)) +

ggtitle("SVM: Top 5 Variables") +

labs(x = "Variable", y = "Importance") +

coord\_flip() +

theme\_minimal()



*# Validation on the hold-out set*

svmPred <- predict(svmModel, newdata = bev\_model\_test)

svmKPI <- postResample(pred = svmPred, obs = Y\_test)

svmMAPE <- MLmetrics::MAPE(predict(svmModel, X\_test), Y\_test)

### KNN

*# model*

set.seed(1234)

knnModel <- caret::train(pH ~ ., bev\_model\_train,

method = "knn",

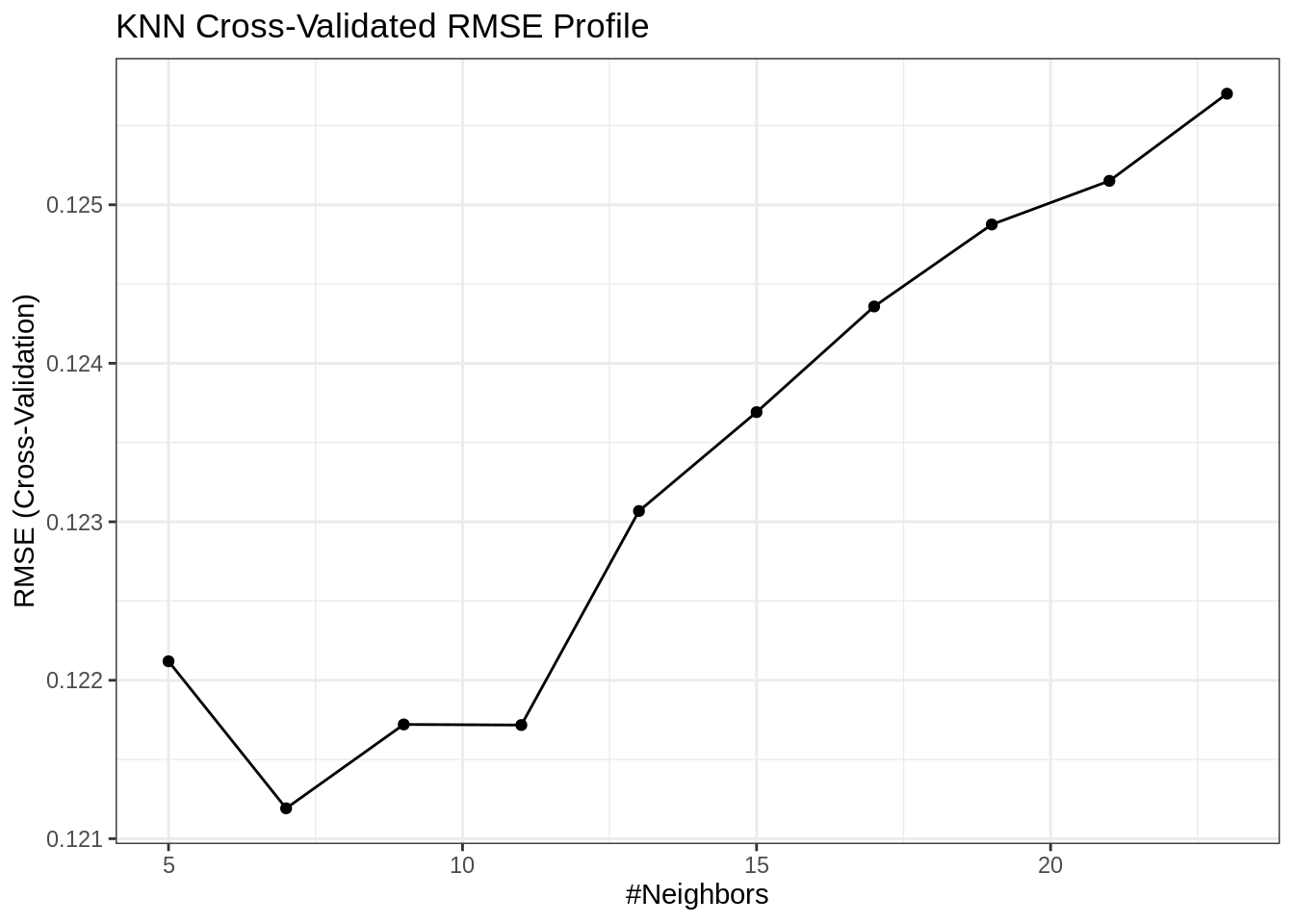
preProcess = c('center', 'scale'),

tuneLength = 10,

trControl = trainControl(method = "cv"))

*# ggplot*

ggplot(knnModel) + labs(title = "KNN Cross-Validated RMSE Profile") + theme\_bw()



*# importance (top 5)*

knnPerformance <- caret::varImp(knnModel)

knnPerformance$importance %>%

as.data.frame() %>%

tibble::rownames\_to\_column() %>%

dplyr::mutate(name = forcats::fct\_inorder(rowname)) %>%

arrange(desc(Overall)) %>%

head(5) %>%

ggplot(., aes(x = reorder(name, Overall), y = Overall)) +

geom\_point() +

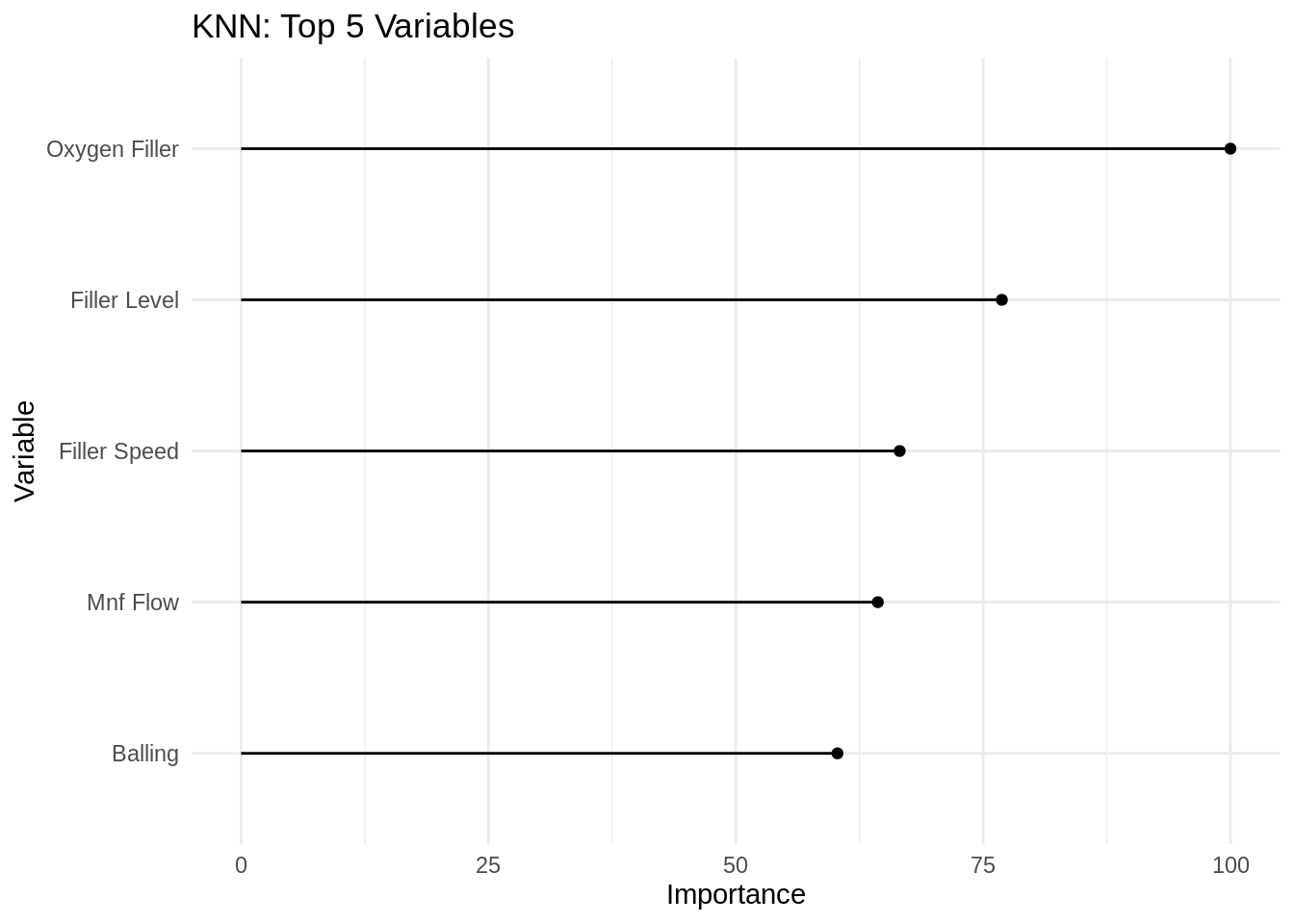
geom\_segment(aes(x = name, xend = name, y = 0, yend = Overall)) +

ggtitle("KNN: Top 5 Variables") +

labs(x = "Variable", y = "Importance") +

coord\_flip() +

theme\_minimal()



*# Validation on the hold-out set*

knnPred <- predict(knnModel, newdata = bev\_model\_test)

knnKPI <- postResample(pred = knnPred, obs = Y\_test)

knnMAPE <- MLmetrics::MAPE(predict(knnModel, X\_test), Y\_test)

### MARS model summary

| KPI | MARS.Model | MAPE |
| --- | --- | --- |
| RMSE | 0.1182389 | 0.0103305 |
| Rsquared | 0.5516399 | 0.0103305 |
| MAE | 0.0880410 | 0.0103305 |

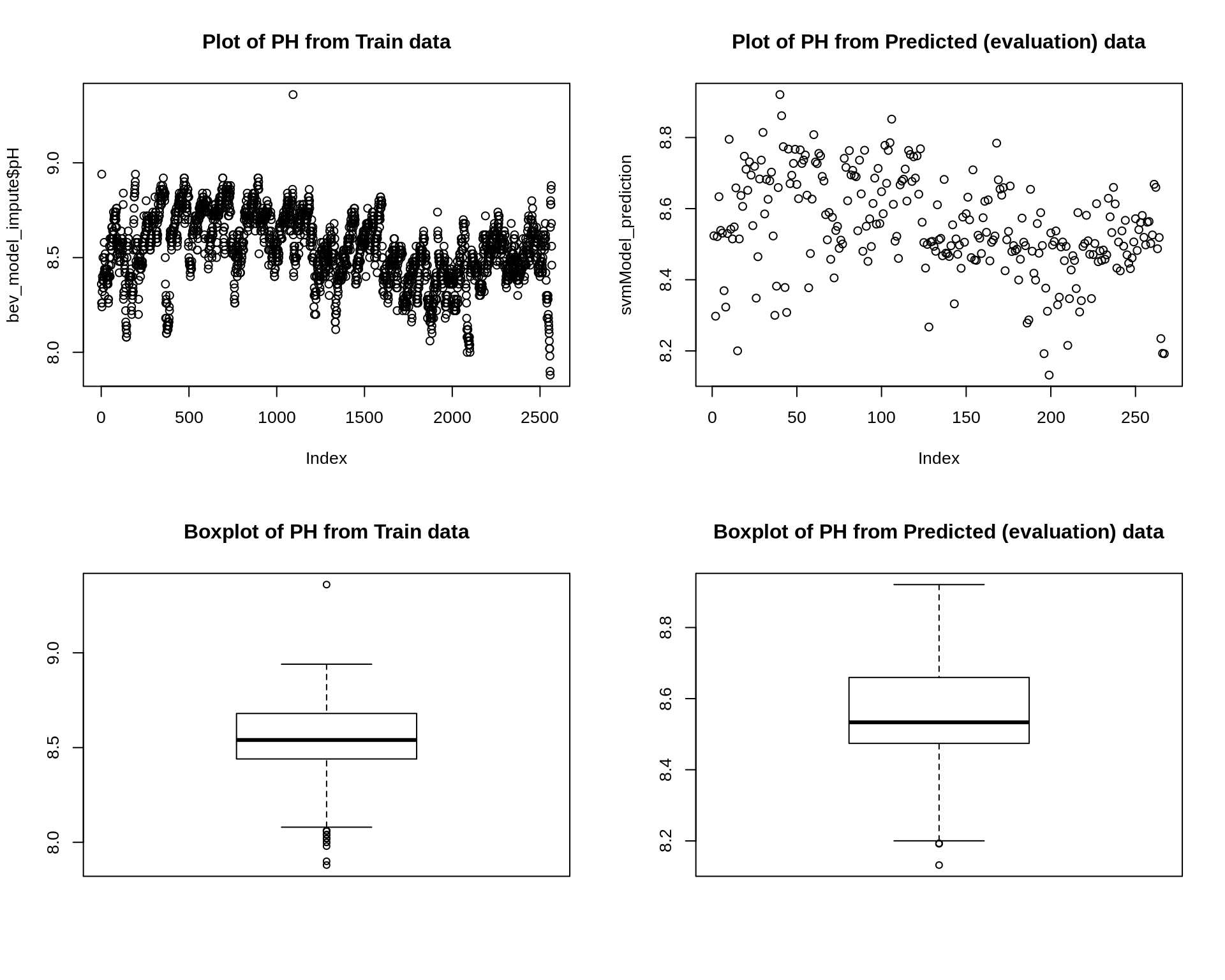
### SVM model summary

| KPI | SVM.Model | MAPE |
| --- | --- | --- |
| RMSE | 0.1200991 | 0.0100186 |
| Rsquared | 0.5402065 | 0.0100186 |
| MAE | 0.0853700 | 0.0100186 |

### KNN model summary

| KPI | KNN.Model | MAPE |
| --- | --- | --- |
| RMSE | 0.1255216 | 0.0105526 |
| Rsquared | 0.5003098 | 0.0105526 |
| MAE | 0.0898410 | 0.0105526 |

The performance of these models are very similar, but SVM seems to do slightly better than MARS and KNN. We can apply the model to the evaluation set.



*# stop the cluster*

parallel::stopCluster(cl)

# Regression Trees and Rule-Based Models

In this section, we will try to fit regression trees and rule-based models :

* Random forest tree-based model Ranger is a fast implementation of random forests (Breiman 2001) or recursive partitioning, particularly suited for high dimensional data. Classification, regression, and survival forests are supported.
* Gradient Boosting Tree Extreme is built on the principles of gradient boosting framework and designed to “push the extreme of the computation limits of machines to provide a scalable, portable and accurate library.

### Random Forest Tree-Based Model

model <- readxl::read\_excel('StudentData\_TO\_MODEL.xlsx',col\_names = TRUE, sheet = 'Subset')

score <- readxl::read\_excel('StudentEvaluation\_TO\_PREDICT.xlsx',col\_names = TRUE, sheet = 'Subset (2)')

set.seed(3456)

setDT(model)

setDT(score)

*# dependent variable*

trainY <- model$PH

*# Data processing*

trainDummy <- dummyVars(PH ~ ., data = model, fullRank = TRUE)

trainX <- as.data.table(predict(trainDummy, newdata = model))

predX <- as.data.table(predict(trainDummy, newdata = score))

*# Find missing values*

missingDat <- model[, lapply(.SD, **function**(x) sum(is.na(x)))]

missingTgt <- missingDat$PH

missingTgtLoc <- which(is.na(trainY))

*# Remove the missing values from the training and test sets*

trainX <- trainX[-missingTgtLoc, ]

trainY <- trainY[-missingTgtLoc]

tSplits <- createDataPartition(trainY, p = 0.75)

X\_Train <- trainX[tSplits$Resample1, ]

X\_test <- trainX[-tSplits$Resample1, ]

Y\_Train <- trainY[tSplits$Resample1]

Y\_test <- trainY[-tSplits$Resample1]

fitControl <- trainControl(method = "repeatedcv", number = 5L, repeats = 3L,

allowParallel = TRUE)

*# detect number of cores for clusters for parallel processing*

no\_cores <- detectCores() \* 0.75

no\_cluster <- makePSOCKcluster(no\_cores)

registerDoParallel(no\_cluster)

rfGrid <- expand.grid(mtry = 31:34,

splitrule = 'extratrees',

min.node.size = 5)

set.seed(3456)

rfmodel <- train(x = X\_Train, y = Y\_Train, method = 'ranger',

preProcess = c('nzv', 'knnImpute'), trControl = fitControl,

tuneGrid = rfGrid, importance = 'impurity')

pred\_RF <- predict(rfmodel, newdata = X\_test)

stopCluster(no\_cluster)

### Extreme Gradient Boosting Tree

no\_cluster <- makePSOCKcluster(no\_cores)

registerDoParallel(no\_cluster)

x\_gbTGrid <- expand.grid(max\_depth = 8:9, nrounds = 430,

eta = 0.082,

colsample\_bytree = 0.826,

min\_child\_weight = 1,

subsample = 1,

gamma = 0)

set.seed(3456)

xgbTFit <- train(x = X\_Train, y = Y\_Train, method = 'xgbTree',

preProcess = c('nzv', 'knnImpute'), trControl = fitControl,

tuneGrid = x\_gbTGrid)

## [18:13:03] WARNING: amalgamation/../src/objective/regression\_obj.cu:170: reg:linear is now deprecated in favor of reg:squarederror.

pred\_XGB <- predict(xgbTFit, newdata = X\_test)

stopCluster(no\_cluster)

### Model Comparison

data.frame("Extreme Gradient Boosting"= defaultSummary(data.frame(obs=Y\_test,pred=predict(xgbTFit, X\_test))),"MAPE" = MLmetrics::MAPE(predict(xgbTFit, X\_test),Y\_test))

|  |
| --- |
|  |

|  | **Extreme.Gradient.Boosting**  <dbl> | **MAPE**  <dbl> |
| --- | --- | --- |
| RMSE | 0.10546235 | 0.00835225 |
| Rsquared | 0.63206125 | 0.00835225 |
| MAE | 0.07124188 | 0.00835225 |

3 rows

data.frame("Random Forest"= defaultSummary(data.frame(obs=Y\_test,pred=predict(rfmodel, X\_test))),"MAPE" = MLmetrics::MAPE(predict(rfmodel, X\_test),Y\_test))

|  |
| --- |
|  |

|  | **Random.Forest**  <dbl> | **MAPE**  <dbl> |
| --- | --- | --- |
| RMSE | 0.10201509 | 0.008215002 |
| Rsquared | 0.66276669 | 0.008215002 |
| MAE | 0.07007316 | 0.008215002 |

3 rows

The performance of these models are very similar, but Random forest seems to do slightly better than Extreme Gradient Boosting. We can apply the model to the evaluation set.

# Conclusion

We found that our support vector machine (SVM) model performed best at predicting pH levels. A summary of the best model from each approach is below:

|  |  |  |  |
| --- | --- | --- | --- |
| Model Type | Best Model | Mape | Most Important Predictors |
| Linear Regression | Elastic Net | 0.0123 | Oxygen Filler, Density, PC Volume, Balling LVI, PSC CO2 |
| NonLinear Regression | SVM | 0.01002 | Oxygen Filler, Filler Level, Filler Speed, Mnt Flow, Balling |
| Regression Trees and Ruled Based | Random Forest | 0.0701 | N/A |

We see that Oxygen Filler, Density, and PC Volume had the most impact on pH levels. Remember that being a highly predictive value does not necessarily mean a factor causes changes in pH level. It could be that an unknown attribute effects both the predictor and the predicted value. We’re not performing causal analysis.

Nonetheless, we have developed a solid basis for predicted pH values based on other attributes involved in brewing. The most important predictors would certainly be a good place to start in examining how we could refine our production processes and better control pH values.

## References

* 1. <https://readxl.tidyverse.org/>
  2. <https://www.rdocumentation.org/packages/DataExplorer/versions/0.8.1/topics/plot_missing>
  3. <https://medium.com/coinmonks/dealing-with-missing-data-using-r-3ae428da2d17>
  4. <https://www.r-bloggers.com/missing-value-treatment/>
  5. <https://stackoverflow.com/questions/51548255/caret-there-were-missing-values-in-resampled-performance-measures> < Why we get NA for predict R squared for lasso regression.
  6. <https://blog.aylien.com/support-vector-machines-for-dummies-a-simple-explanation/>
  7. Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling.* New York: Springer.