

# Housing Data Modeling

Scott Breitbach

11/14/2021

## Load Dataframes

```
# Entire Dataset with Ordinal Variables converted
dfOrd <- read.csv("data/dfTrainC.csv")      # 1460 x 81
# Entire Dataset with dummy variables created
dfAn <- read.csv("data/dfAnalysis.csv")     # 1 1460 x 257 (with all dummies)

## FEATURE SELECTED DATA SUBSETS:
# Variables that highly correlate (>0.5) w/SalePrice
dfCorrSP <- read.csv("data/dfTrain1.csv")   # 2 1460 x 16 (correlate with SalePrice)
# Variables with high F-statistic value (top 30)
dfFstat <- read.csv("data/featFstatistic.csv") # 4 1460 x 31 (top F stat features)
# Variables with high LightBGM value (top 30)
dfLBGM <- read.csv("data/featLightBGM.csv")  # 5 1460 x 31 (top LightBGM features)
# Variables with high Logistic Regression value (top 30)
dfLogReg <- read.csv("data/featLogisticRegression.csv") # 6 1460 x 31 (top logistic regression features)
# Variables with high Mutual Information value (top 30)
dfMInf <- read.csv("data/featMutualInformation.csv") # 7 1460 x 31 (top mutual information features)
# Overall combined top variables from feature selections (top 30)
dfOverall <- read.csv("data/featOverall.csv") # 8 1460 x 31 (top overall features)
```

## Linear Regression Modeling

```
# Get summary of model for each data set
lmOrd = lm(formula = SalePrice~., data = dfOrd)
lmAn = lm(formula = SalePrice~., data = dfAn)
lmCorrSP = lm(formula = SalePrice~., data = dfCorrSP)
lmFstat = lm(formula = SalePrice~., data = dfFstat)
lmLBGM = lm(formula = SalePrice~., data = dfLBGM)
lmLogReg = lm(formula = SalePrice~., data = dfLogReg)
lmMInf = lm(formula = SalePrice~., data = dfMInf)
lmOverall = lm(formula = SalePrice~., data = dfOverall)

# Summary of one model
summary(lmFstat)
```

##

```
## Call:
## lm(formula = SalePrice ~ ., data = dfFstat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -380441  -17720     182   16110  266365
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.259e+05  6.779e+03 -18.570 < 2e-16 ***
## SaleType_Con     3.943e+04  2.519e+04  1.566 0.117646
## Condition2_RRAn  -1.821e+04  3.478e+04 -0.523 0.600732
## Heating_Floor     3.332e+04  3.502e+04  0.952 0.341440
## Exterior2nd_Other  4.943e+04  3.493e+04  1.415 0.157327
## SaleCondition_Alloca 7.366e+03  1.018e+04  0.724 0.469303
## LotArea          9.266e-01  9.547e-02  9.706 < 2e-16 ***
## Neighborhood_Veenker 3.028e+04  1.082e+04  2.799 0.005193 **
## OverallQual       1.450e+04  1.219e+03 11.892 < 2e-16 ***
## Neighborhood_NoRidge 5.504e+04  5.967e+03  9.224 < 2e-16 ***
## Neighborhood_NridgHt 4.006e+04  4.581e+03  8.746 < 2e-16 ***
## Heating_Grav      -2.521e+03  1.333e+04 -0.189 0.850045
## SaleCondition_Partial -5.575e+03  2.013e+04 -0.277 0.781838
## SaleType_New       2.542e+04  2.042e+04  1.245 0.213361
## ExterQual          9.285e+03  2.693e+03  3.448 0.000582 ***
## GarageCars         1.259e+04  1.614e+03  7.798 1.20e-14 ***
## Exterior2nd_CmentBd  1.480e+04  4.768e+03  3.105 0.001942 **
## KitchenQual        1.407e+04  2.117e+03  6.647 4.23e-11 ***
## BsmtQual           8.715e+03  1.401e+03  6.223 6.40e-10 ***
## Condition2_PosN    -1.663e+05  2.532e+04 -6.568 7.12e-11 ***
## GrLivArea          4.551e+01  2.337e+00 19.474 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34740 on 1439 degrees of freedom
## Multiple R-squared:  0.8114, Adjusted R-squared:  0.8088
## F-statistic: 309.6 on 20 and 1439 DF, p-value: < 2.2e-16
```

## Compare Linear Regression Models

Source

```
# Load libraries
library(performance)

# Compare models
compare_performance(lmOrd, lmAn, lmCorrSP, lmFstat,
                   lmLBGM, lmLogReg, lmMInf, lmOverall)
```

```
## # Comparison of Model Performance Indices
```

```
##
## Name      | Model |      AIC | AIC_wt |      BIC | BIC_wt |    R2 | R2 (adj.) |    RMSE |
## -----
## lmOrd     |   lm | 33825.292 | 0.500 | 34908.961 | < 0.001 | 0.919 | 0.906 | 22576.270 | 2434
```

## lmAn		lm		33825.292		0.500		34908.961		< 0.001		0.919		0.906		22576.270		2434
## lmCorrSP		lm		34833.905		< 0.001		34923.770		< 0.001		0.791		0.789		36273.295		3647
## lmFstat		lm		34696.338		< 0.001		34812.634		1.000		0.811		0.809		34485.714		3473
## lmLBGM		lm		35085.861		< 0.001		35196.871		< 0.001		0.753		0.750		39433.996		3970
## lmLogReg		lm		35790.347		< 0.001		35906.643		< 0.001		0.601		0.596		50159.396		5052
## lmMInf		lm		37048.850		< 0.001		37165.147		< 0.001		0.055		0.042		77184.615		7774
## lmOverall		lm		34739.914		< 0.001		34856.210		< 0.001		0.806		0.803		35004.212		3525

Using the entire data set provides the same results, whether or not dummy variables are included. Barring use of the entire data set for modeling, feature selection by F-statistic provides the lowest AIC and highest r-squared, followed closely by the Overall selected features and the highly correlated.

## Check Model

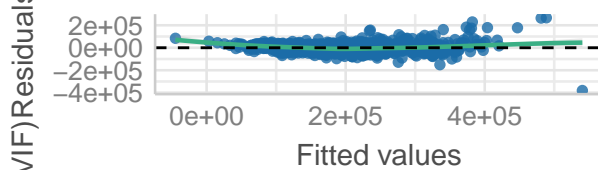
```
# Check Collinearity of Model Variables
check_collinearity(lmFstat)
```

```
## # Check for Multicollinearity
##
## Low Correlation
##
##          Term  VIF Increased SE Tolerance
##      SaleType_Con 1.05      1.02      0.95
##      Condition2_RRAn 1.00      1.00      1.00
##      Heating_Floor 1.02      1.01      0.98
##      Exterior2nd_Other 1.01      1.01      0.99
##      SaleCondition_Alloca 1.02      1.01      0.98
##      LotArea 1.10      1.05      0.91
##      Neighborhood_Veenker 1.06      1.03      0.94
##      OverallQual 3.44      1.85      0.29
##      Neighborhood_NoRidge 1.18      1.08      0.85
##      Neighborhood_NridgHt 1.27      1.13      0.79
##      Heating_Grav 1.03      1.01      0.97
##      ExterQual 2.89      1.70      0.35
##      GarageCars 1.76      1.33      0.57
##      Exterior2nd_CmentBd 1.08      1.04      0.92
##      KitchenQual 2.39      1.55      0.42
##      BsmtQual 1.82      1.35      0.55
##      Condition2_PosN 1.06      1.03      0.94
##      GrLivArea 1.82      1.35      0.55
##
## High Correlation
##
##          Term  VIF Increased SE Tolerance
##      SaleCondition_Partial 38.38      6.19      0.03
##      SaleType_New 38.63      6.22      0.03
```

```
# Visualization of multiple model checks
check_model(lmFstat)
```

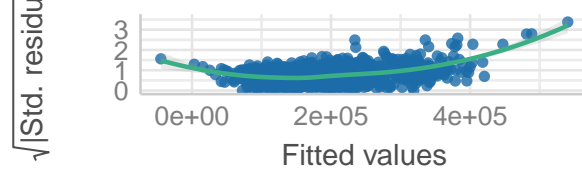
## Linearity

Reference line should be flat and horizontal



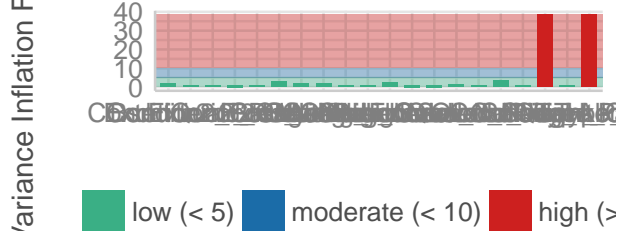
## Homogeneity of Variance

Reference line should be flat and horizontal



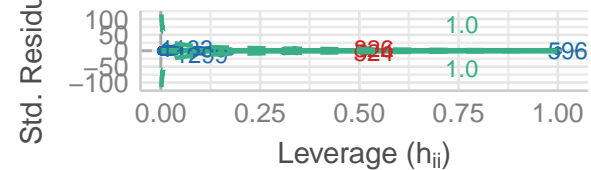
## Collinearity

Higher bars (>5) indicate potential collinearity issue



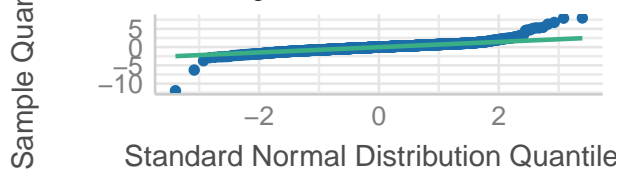
## Influential Observations

Points should be inside the contour lines



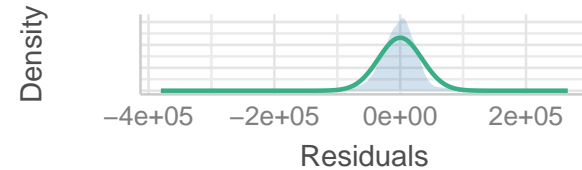
## Normality of Residuals

Dots should fall along the line



## Normality of Residuals

Distribution should be close to the normal curve



The dummy variables `SaleCondition_Partial` and `SaleType_New` show a high degree of correlation and one of these should probably be removed.

## Check Accuracy of Models

```
# Only included feature-selected data sets due to size
accCorrSP <- performance_accuracy(lmCorrSP, method = c("cv", "boot"), k = 5, n = 1000)
print("Accuracy lmCorrSP: ")
```

```
## [1] "Accuracy lmCorrSP: "
```

```
accCorrSP$Accuracy
```

```
## [1] 0.8858577
```

```
accFstat <- performance_accuracy(lmFstat, method = c("cv", "boot"), k = 5, n = 1000)
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a
## rank-deficient fit may be misleading
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a
## rank-deficient fit may be misleading
```

```
print("Accuracy lmFstat: ")
```

```
## [1] "Accuracy lmFstat: "
```

```
accFstat$Accuracy
```

```
## [1] 0.8880362
```

```
accLBGM <- performance_accuracy(lmLBGM, method = c("cv", "boot"), k = 5, n = 1000)
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a  
## rank-deficient fit may be misleading
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a  
## rank-deficient fit may be misleading
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a  
## rank-deficient fit may be misleading
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a  
## rank-deficient fit may be misleading
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a  
## rank-deficient fit may be misleading
```

```
print("Accuracy lmLBGM: ")
```

```
## [1] "Accuracy lmLBGM: "
```

```
accLBGM$Accuracy
```

```
## [1] 0.8648927
```

```
accLogReg <- performance_accuracy(lmLogReg, method = c("cv", "boot"), k = 5, n = 1000)  
print("Accuracy lmLogReg: ")
```

```
## [1] "Accuracy lmLogReg: "
```

```
accLogReg$Accuracy
```

```
## [1] 0.7671006
```

```
accMInf <- performance_accuracy(lmMInf, method = c("cv", "boot"), k = 5, n = 1000)
```

```
## Warning in predict.lm(.y, newdata = model_data[.x$test, ]): prediction from a  
## rank-deficient fit may be misleading
```

```
print("Accuracy lmMInf: ")
```

```
## [1] "Accuracy lmMInf: "
```

```
accMInf$Accuracy
```

```
## [1] 0.1874233
```

```
accOverall <- performance_accuracy(lmOverall, method = c("cv", "boot"), k = 5, n = 1000)
print("Accuracy lmOverall: ")
```

```
## [1] "Accuracy lmOverall: "
```

```
accOverall$Accuracy
```

```
## [1] 0.8898121
```

The Overall features data set shows the best accuracy here at 89%, narrowly beating out F-statistic and Correlation as feature selection methods for linear regression modeling.

## Hosmer-Lemeshow Goodness-of-Fit Test

```
# Requires glm instead of lm
glmOverall = glm(formula = SalePrice~., data = dfOverall)
performance_hosmer(glmOverall)
```

```
## # Hosmer-Lemeshow Goodness-of-Fit Test
##
##   Chi-squared: -3.840
##           df:   8
##   p-value:    1.000
```

```
## Summary: model seems to fit well.
```

Model seems to fit well.

## Decision Trees

Source

## Random Forest

```
# Load Libraries
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
# Set up training set
train.overall = sample(1:nrow(dfOverall), 1060)

# Set up random forest model using training subset
rf.overall = randomForest(SalePrice~., data = dfOverall, subset = train.overall)
rf.overall
```

```
##
## Call:
## randomForest(formula = SalePrice ~ ., data = dfOverall, subset = train.overall)
##              Type of random forest: regression
##              Number of trees: 500
## No. of variables tried at each split: 6
##
##              Mean of squared residuals: 1165025567
##              % Var explained: 81.82
```

```
## TUNING
```

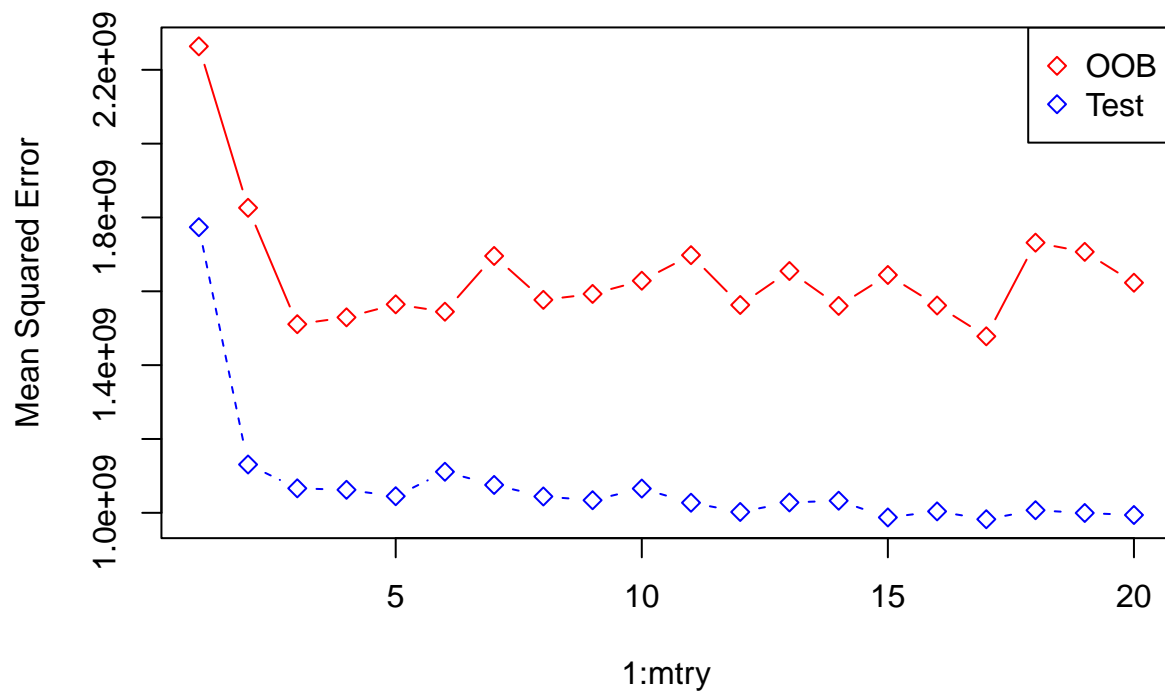
```
# Set up error variables
```

```
oob.err = double(20)
test.err = double(20)
```

```
# Determine optimal number of variables (mtry) to randomly select at each split
for (mtry in 1:20){
  fit = randomForest(SalePrice~., data = dfOverall, subset = train.overall,
                     mtry = mtry, ntree = 350)
  oob.err[mtry] = fit$mse[10]
  pred = predict(fit, dfOverall[-train.overall,])
  test.err[mtry] = with(dfOverall[-train.overall,], mean( (SalePrice-pred)^2 ))
}
```

```
# Plot results
```

```
matplot(1:mtry, cbind(oob.err, test.err), pch = 23, col = c("red", "blue"), type = "b", ylab = "Mean Squared Error")
legend("topright", legend = c("OOB", "Test"), pch = 23, col = c("red", "blue"))
```



Model may be improved by setting `mtry` to 5 or 6.

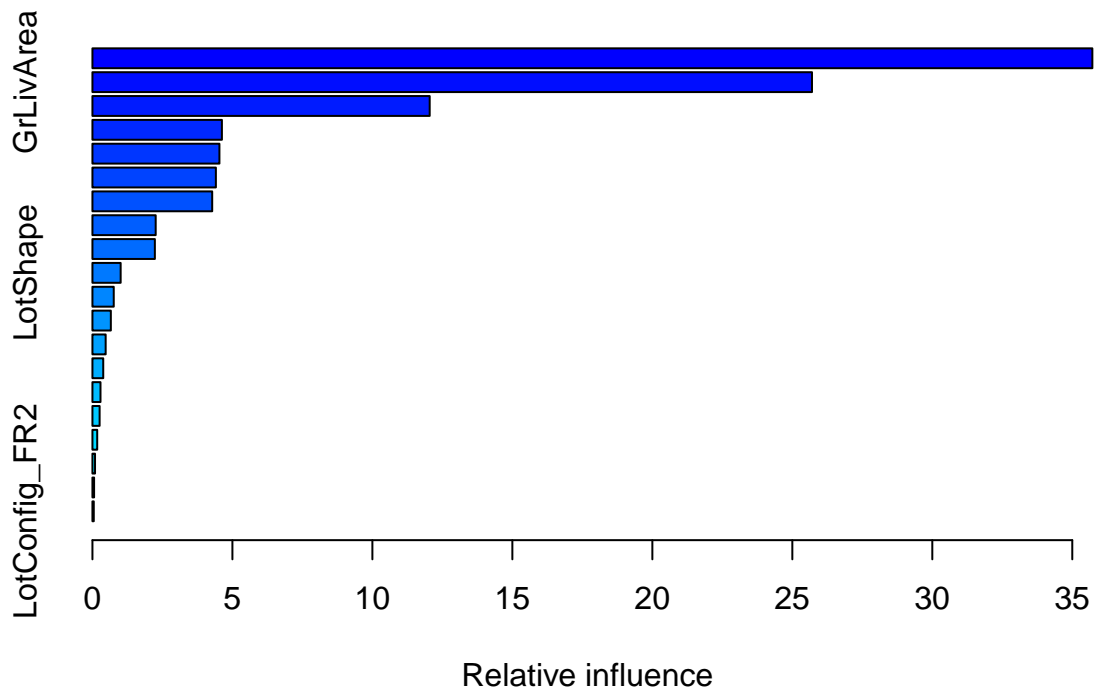
## Boosting Trees using Gradient Boosted Modeling

```
# Load Libraries
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
# Set up boosting model
boost.overall = gbm(SalePrice~., data = dfOverall[train.overall,],
  distribution = "gaussian", # continuous
  n.trees = 10000, shrinkage = 0.01, interaction.depth = 4)
# Show variable importance plot
summary(boost.overall)
```

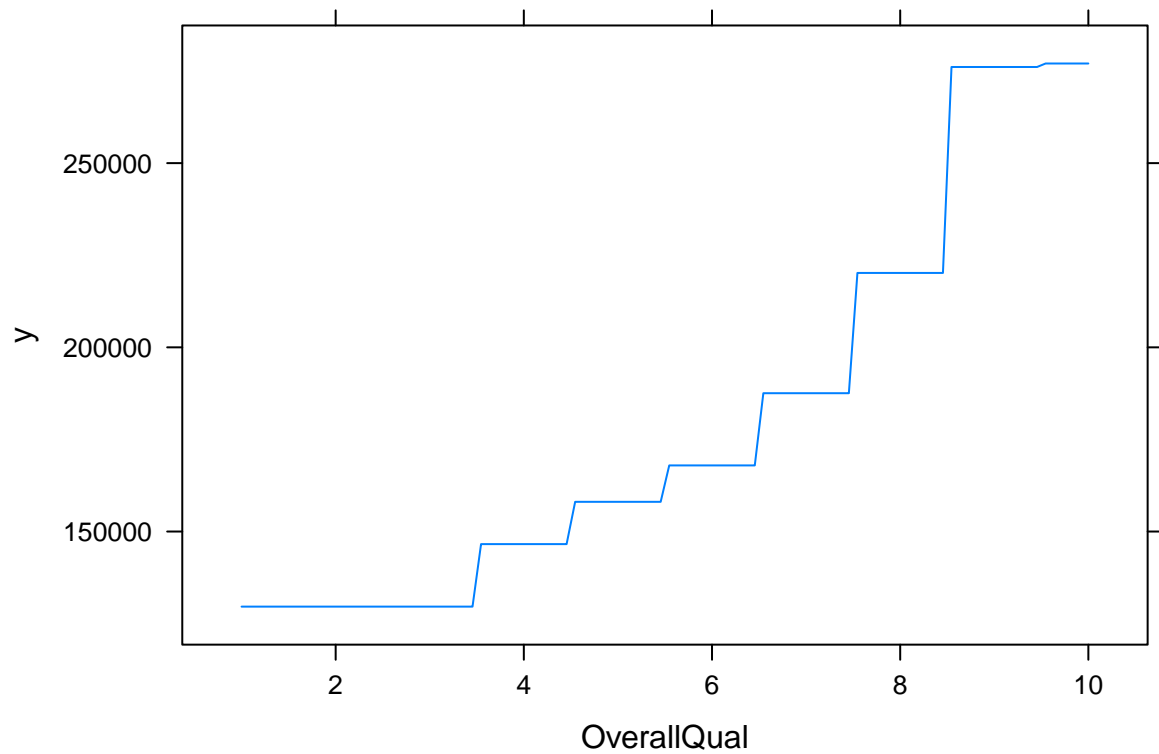




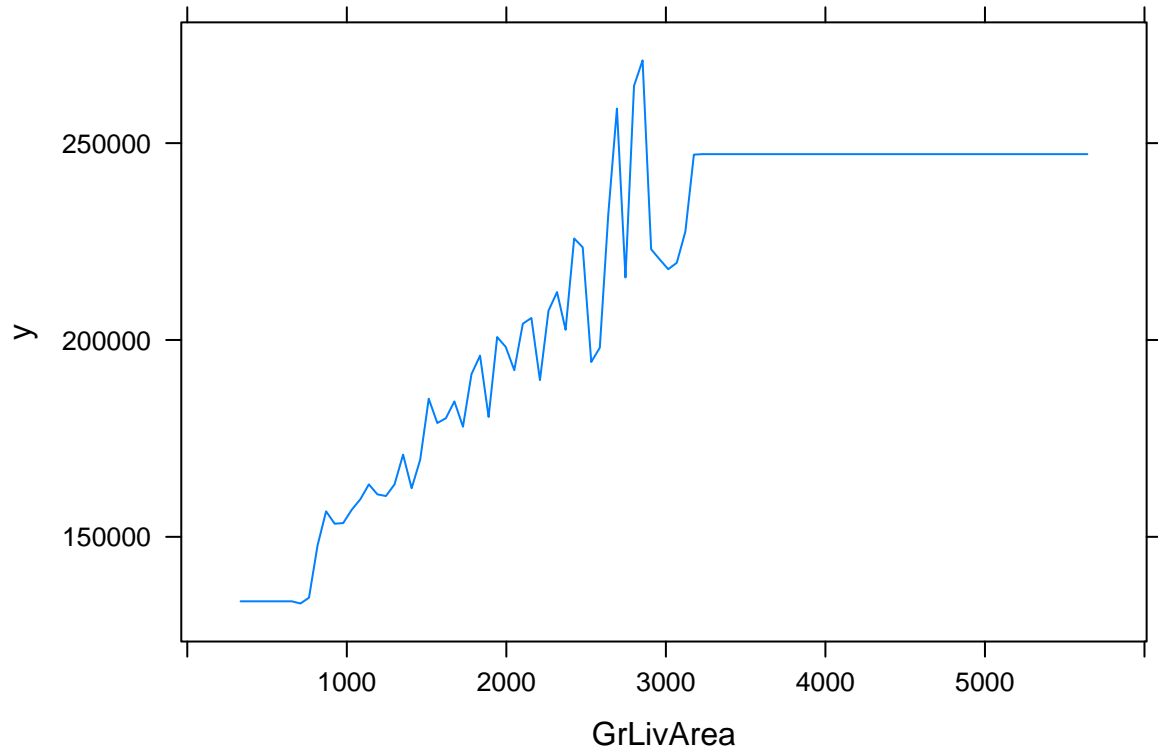
```
##               var      rel.inf
## OverallQual    OverallQual 35.71729846
## GrLivArea      GrLivArea  25.70208425
## GarageArea     GarageArea 12.04654108
## KitchenQual    KitchenQual  4.62446706
## ExterQual      ExterQual   4.53577966
## MasVnrArea     MasVnrArea   4.40933588
## WoodDeckSF     WoodDeckSF   4.27824540
## FullBath       FullBath    2.25929886
## FireplaceQu    FireplaceQu  2.22945815
## LotShape       LotShape     1.00766294
## Neighborhood_NridgHt Neighborhood_NridgHt 0.76084092
## Neighborhood_NoRidge Neighborhood_NoRidge 0.65705996
## LotConfig_CulDSac LotConfig_CulDSac 0.47240193
## LandContour_HLS LandContour_HLS 0.38552188
## Exterior1st_CemntBd Exterior1st_CemntBd 0.28800577
## Neighborhood_StoneBr Neighborhood_StoneBr 0.25665528
## MSZoning_FV     MSZoning_FV 0.17019440
## Neighborhood_CollgCr Neighborhood_CollgCr 0.09257300
## Neighborhood_Timber Neighborhood_Timber 0.05882538
## LotConfig_FR2   LotConfig_FR2 0.04774975
```

Our model here is largely influenced by only a handful of variables: `OverallQual` accounting for about a third, `GrLivArea` about a fourth, followed by `GarageArea` and then some others. Let's look at a few of them:

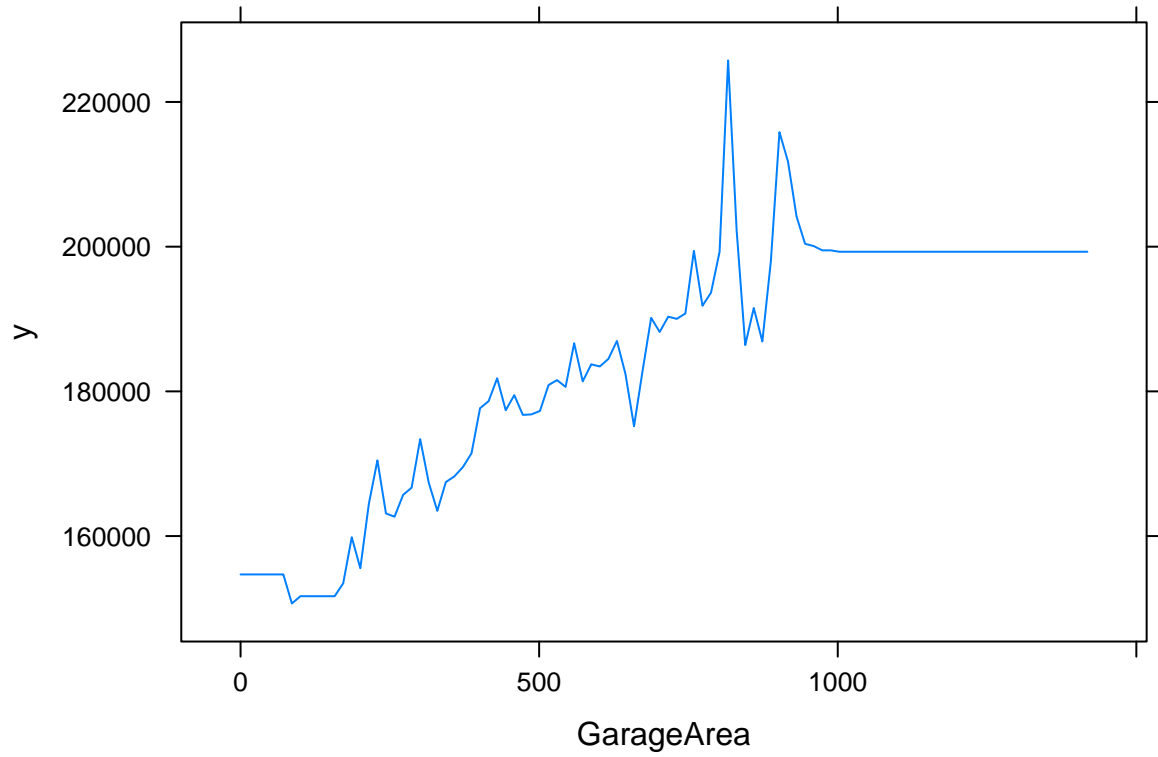
```
# Plot variables  
plot(boost.overall,i="OverallQual")
```



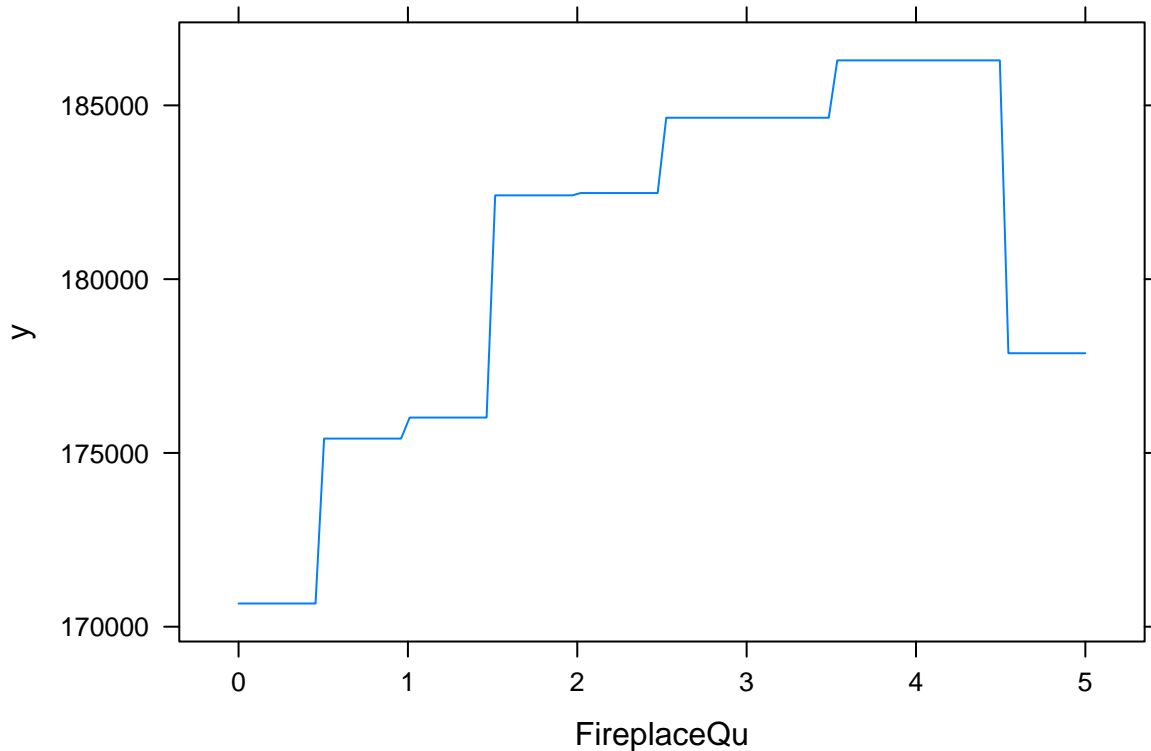
```
plot(boost.overall,i="GrLivArea")
```



```
plot(boost.overall,i="GarageArea")
```



```
plot(boost.overall,i="FireplaceQu")
```



Most of these variables appear to show a roughly linear increase in correlation with `SalePrice`, but notice that `FirePlaceQu` has a huge jump between 0/1 rating compared to 2+ rating. In the future, this variable could probably be binned into a binary good/bad rating.

**Predict the boosted model on the data set**

```
# Make a grid of number of trees
n.trees = seq(from = 100, to = 10000, by = 100)
# Run predict on boosted model
predmat = predict(boost.overall, newdata = dfOverall[-train.overall,], n.trees = n.trees)
# dim(predmat)

# Compute the test error
boost.err = with(dfOverall[-train.overall,],
                 apply( (predmat - SalePrice)^2, 2, mean) )

# Best error from random forest
print("Random forest best error:")
```

```
## [1] "Random forest best error:"
```

```
min(test.err)
```

```
## [1] 982582164
```

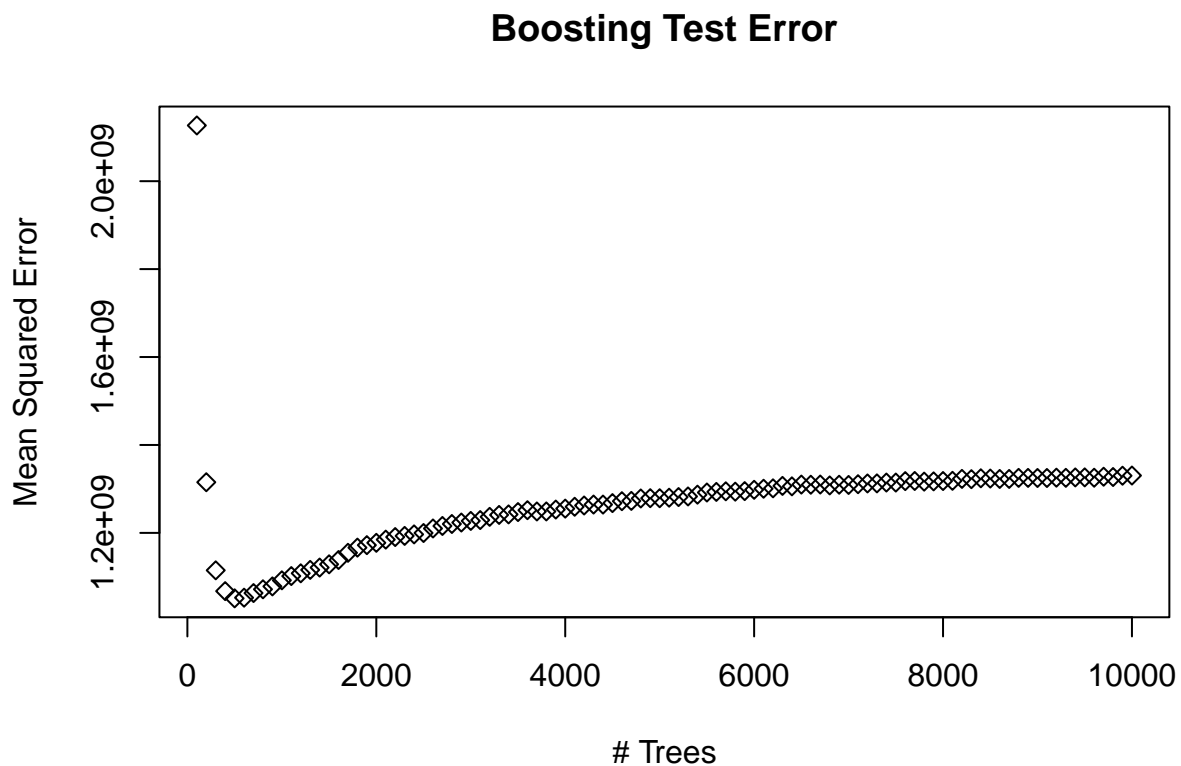
```
# Best error from boosting
print("Boosting best error:")
```

```
## [1] "Boosting best error:"
```

```
min(boost.err)
```

```
## [1] 1051126050
```

```
# Plot results
plot(n.trees, boost.err, pch = 23, ylab = "Mean Squared Error",
     xlab = "# Trees", main = "Boosting Test Error")
abline(h = min(test.err), col = "red")
```



Boosting manages to drop the error below the best error from the Random Forest model (red line). NOTE: the first time I ran this it looked great; not sure what happened and hopefully it fixes itself when I run it again. Nope, it did not. idk

## Ensemble Modeling

Using `SuperLearner()`

Sources:

\* Ensembles \* SuperLearner

```

# Load libraries
library("SuperLearner")

## Loading required package: nnls

## Loading required package: gam

## Loading required package: splines

## Loading required package: foreach

## Loaded gam 1.20

## Super Learner

## Version: 2.0-28

## Package created on 2021-05-04

# Split data
train_idx <- sample(nrow(dfOverall), 2/3 * nrow(dfOverall))
df_train <- dfOverall[train_idx,]
df_test <- dfOverall[-train_idx,]

# Spilt out target variable
y <- df_train[,1]
ytest <- df_test[,1]

# Split out predictor variables
x <- df_train[,2:21]
xtest <- df_test[,2:21]

# Return the model
sl <- SuperLearner(y, x, family = gaussian(), # gaussian for continuous var
                  SL.library = list("SL.speedglm", "SL.svm", "SL.gbm",
                                    "SL.extraTrees"))

## Loading required namespace: e1071

## Loading required namespace: speedglm

## Loading required namespace: extraTrees

# # Models removed from ensemble due to low coefficient:
# list("SL.bayesglm", "SL.nnet", "SL.speedglm", "SL.caret.rpart", "SL.glmnet",
#      "SL.randomForest", "SL.biglasso", "SL.ranger", "SL.step.forward",
#      "SL.earth", "SL.ipredbag", "SL.polymars", "SL.step", "SL.rpart",
#      "SL.cforest", "SL.ksvm", "SL.stepAIC", # removed first pass
#      "SL.xgboost", "SL.bartMachine",      # removed second pass
#      "SL.rpartPrune",                     # removed third pass
#      "SL.loess")                          # removed due to high added risk

# Return the model
sl

```

```
##
## Call:
## SuperLearner(Y = y, X = x, family = gaussian(), SL.library = list("SL.speedlm",
##   "SL.svm", "SL.gbm", "SL.extraTrees"))
##
##
##
##           Risk      Coef
## SL.speedlm_All    1375651628 0.3183292
## SL.svm_All        1833595838 0.0000000
## SL.gbm_All        1373445073 0.1342865
## SL.extraTrees_All 1275673942 0.5473843
```

```
# Look at modeling time
sl$times
```

```
## $everything
##   user  system elapsed
## 63.65    1.90  170.06
##
## $train
##   user  system elapsed
## 57.13    1.68  148.75
##
## $predict
##   user  system elapsed
##  6.50    0.22   21.17
```

The `extraTrees` model in the ensemble has the highest coefficient, indicating it is weighted highly in the ensemble. Following this, `speedlm` and `gbm` are the next highest weighted models.

## Cross-Validation of Ensemble

```
# Get V-fold cross-validated risk estimate
cv.sl <- CV.SuperLearner(y, x, V=5, family = gaussian(), # gaussian for continuous var
                        SL.library = list("SL.speedlm", "SL.svm", "SL.gbm",
                        "SL.extraTrees"))
```

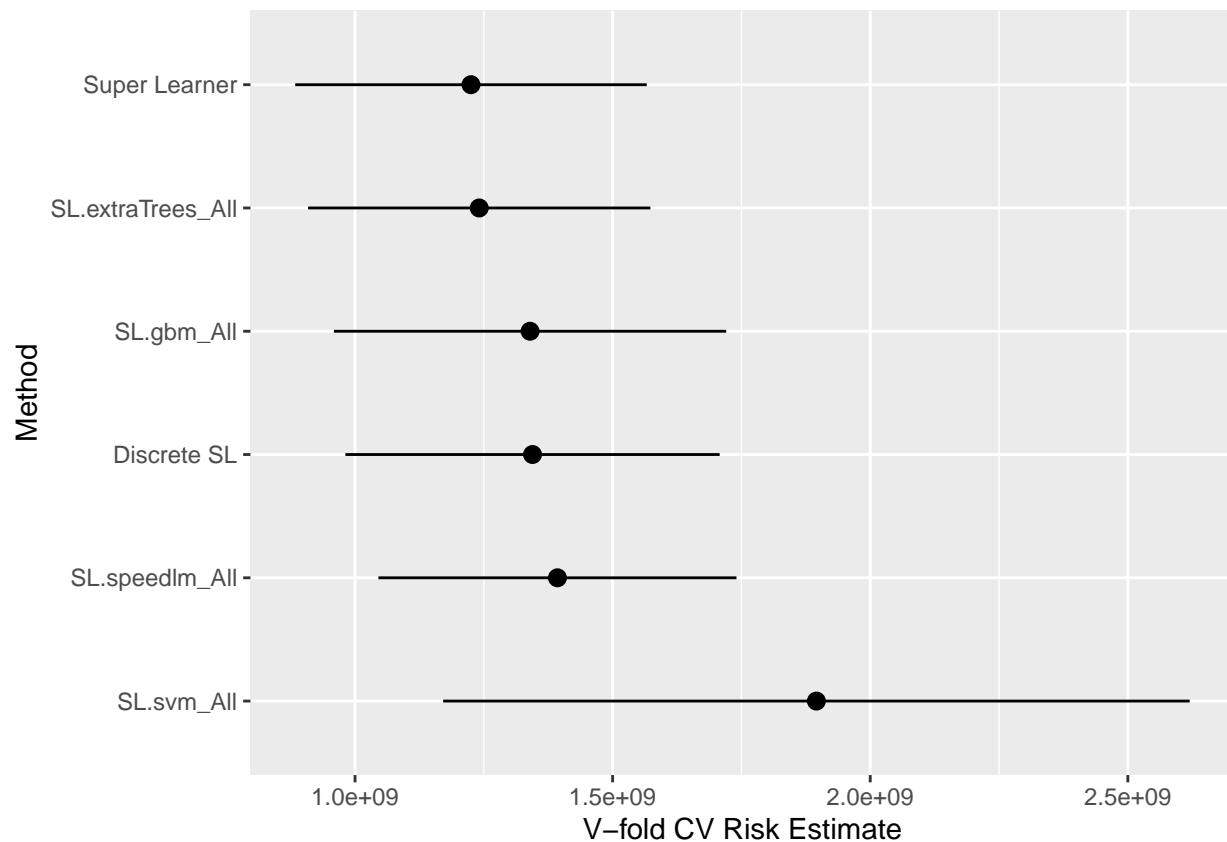
```
# Print out the summary statistics
summary(cv.sl)
```

```
##
## Call:
## CV.SuperLearner(Y = y, X = x, V = 5, family = gaussian(), SL.library = list("SL.speedlm",
##   "SL.svm", "SL.gbm", "SL.extraTrees"))
##
## Risk is based on: Mean Squared Error
##
## All risk estimates are based on V = 5
##
##           Algorithm      Ave      se      Min      Max
##   Super Learner 1225075362 174077187 795139225 1818539901
```



```
##      Discrete SL 1344515445 185330289 972639528 1873179525
##      SL.speedlm_All 1392922738 177293836 984039657 1810157453
##      SL.svm_All 1895509109 369641008 910536807 2605249598
##      SL.gbm_All 1339755793 194247170 869984675 2000156766
##      SL.extraTrees_All 1241077851 169502345 824813415 1873179525
```

```
# Plot models used and their variation
plot(cv.sl)
```



Make Predictions using SuperLearner()

```
# Make Predictions with SuperLearner
predictions <- predict.SuperLearner(sl, newdata=xtest)

# Return ensemble predictions
head(predictions$pred)
```

```
##      [,1]
## [1,] 159856.5
## [2,] 222815.0
## [3,] 249073.5
## [4,] 193046.6
## [5,] 117269.0
## [6,] 118625.0
```

```
# Return individual library predictions
head(predictions$library.predict)
```

```
##      SL.speedlm_All SL.svm_All SL.gbm_All SL.extraTrees_All
## [1,]      150511.6    172271.4    158512.6          165620.7
## [2,]      229544.5    221887.7    213074.5          221291.0
## [3,]      239833.8    238418.0    247640.5          254798.4
## [4,]      186094.5    186684.4    191304.4          197516.9
## [5,]      108575.4    111368.2    120206.3          121604.1
## [6,]      108596.9    120246.1    119904.1          124142.9
```

Predict using test data set

```
# Predict back on the holdout data set
# Note: onlySL=TRUE includes only models weighted > 0
pred = predict(sl, xtest, onlySL = TRUE)
```

```
# Check the structure of this prediction object.
str(pred)
```

```
## List of 2
## $ pred          : num [1:487, 1] 159857 222815 249074 193047 117269 ...
## $ library.predict: num [1:487, 1:4] 150512 229544 239834 186095 108575 ...
```

```
# Review the columns of $library.predict.
summary(pred$library.predict)
```

```
##      V1          V2          V3          V4
## Min.   : 15355    Min.   :0     Min.   : 76557    Min.   : 59984
## 1st Qu.:132050    1st Qu.:0     1st Qu.:134079    1st Qu.:135545
## Median :172052    Median :0     Median :164578    Median :167914
## Mean   :184577    Mean   :0     Mean   :182401    Mean   :182394
## 3rd Qu.:220955    3rd Qu.:0     3rd Qu.:207825    3rd Qu.:211327
## Max.   :561791    Max.   :0     Max.   :517006    Max.   :527005
```

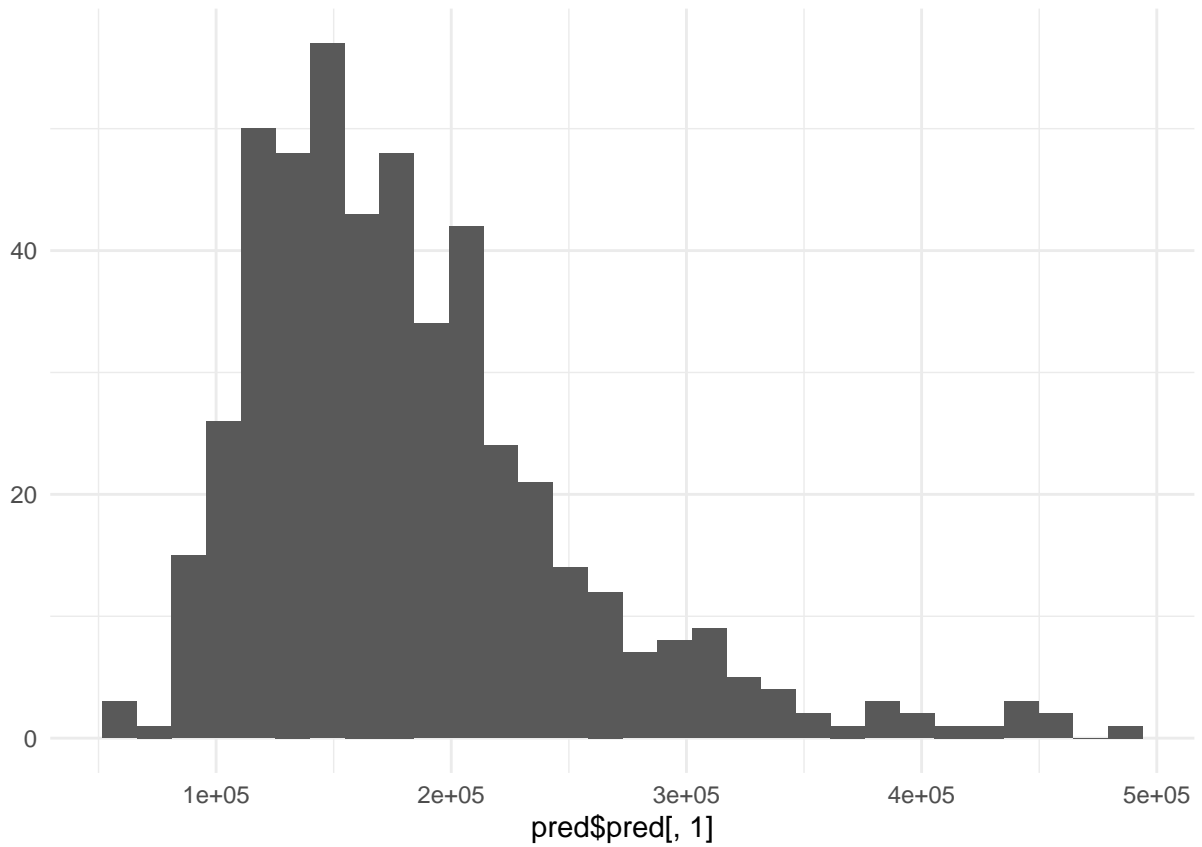
```
# Histogram of our predicted values.
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

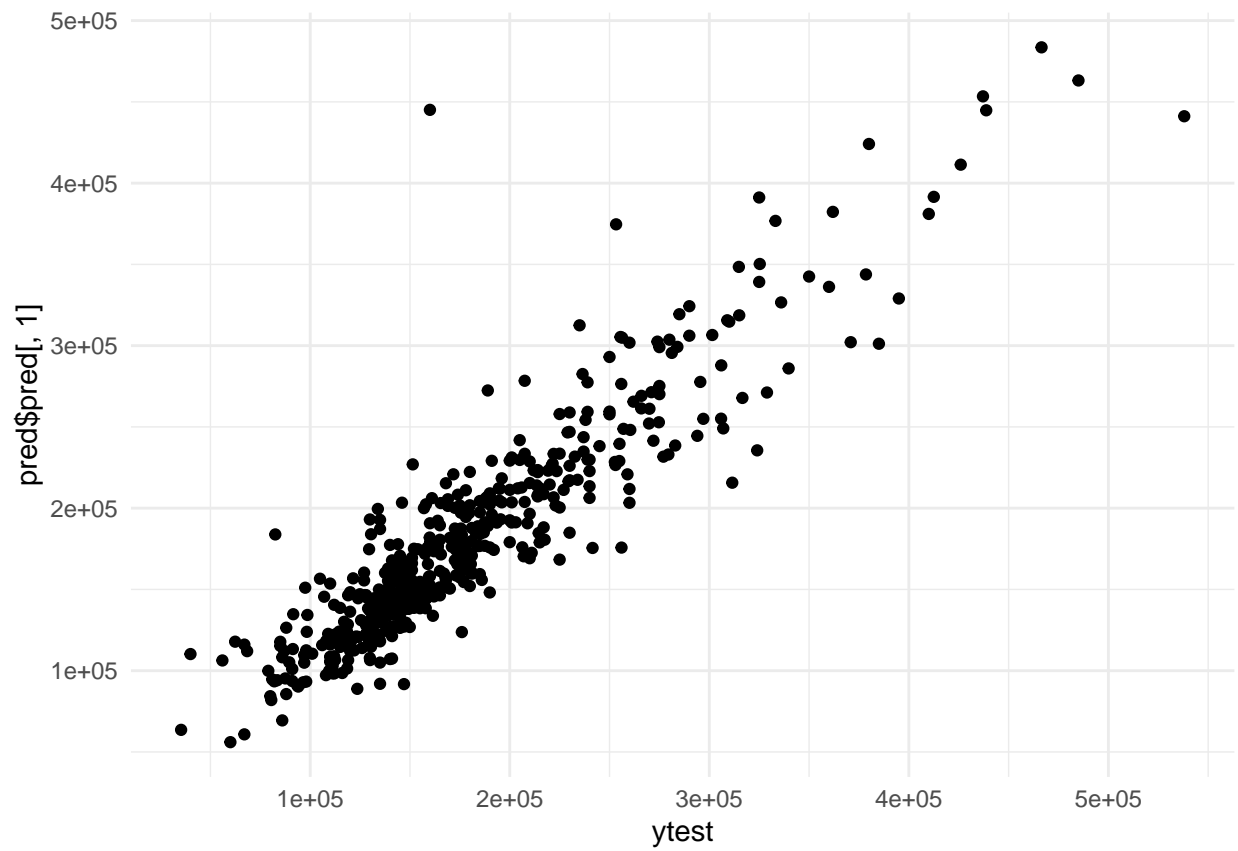
```
## The following object is masked from 'package:randomForest':
##
##      margin
```

```
qplot(pred$pred[, 1]) + theme_minimal()
```

```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



```
# Scatterplot of original values (0, 1) and predicted values.  
qplot(ytest, pred$pred[, 1]) + theme_minimal()
```



Note: there's some hyperparameter tuning available in SuperLearner as well.