Best Model to Predict House Prices in Ames, Iowa

4/27/2020

Jennifer Nguyen, Garrett Hastings, Yucong Hu

Contents

1. Introduction 1
   1.1 Dataset 1
   1.2 Problem 1
   1.3 Concerns 1
2. Data Description and Preprocessing 1
   2.1 Missing data 1
   2.2 Variable distribution assessment 1
   2.3 Dummy variable transformation 2
3. Methods 2
   3.1 Variable filtering 2
   3.2 Linear regression techniques 3
   3.3 Non-linear regression 6
4. Results 8
5. Conclusion and remarks 9
6. Appendix 11

1. Introduction

1.1 Dataset

The dataset consists of independent variables that may or may not affect the values of properties in Ames, Iowa. Some examples are roofing types, last remodeled date, etc. Our data source is from Kaggle.com's House Prices: Advanced Regression Techniques competition. The data set has 80 independent variables and 1 dependent variable, consisting of 1460 observations (450 kb). The dependent variable is the sale price of properties.

1.2 Problem

We need to find the best model to predict the final price of residential homes in Ames, Iowa. To real estate agencies, they could use the model to target certain locations in the city where housing has more potential for higher sale price. On the other hand, homeowners could use the same model to know what to do to raise their property value (remodel, put a new roof on, etc.), depending on which variables raise the value of their property the most. Moreover, home seekers/potential buyers could choose the features of the homes that they want to buy to estimate the necessary budget.

1.3 Concerns

There are a few concerns about data preprocessing. First is how to deal with missing data. We will look into the most reasonable interpretation of N/A for each of the variables, and make the appropriate changes to clean the data. Second is the computational speed of each model. If there is an issue with how long a certain analysis is taking, then we will have to reduce some variables with other methods and re-run. However, the data set is relatively small, so we would not expect these issues. Third is that the model can perform very well but is hard to interpret. We will try to opt for a simpler model if the test errors from the 2 models (complex vs. simple) are not drastically different.

2. Data Description and Preprocessing

2.1 Missing data

We found that we had N/A's present in our data for both categorical and numerical variables, which had to be handled separately and differently.

For categorical variables, we simply made the N/A's into its own category, called "Missing." For example, if the "Alley" variable had N/A's in it, it meant that there was no Alley present in the House, so a category for all the houses with Alley="Missing" is appropriate and makes sense.

For numerical variables, we used the median value to replace N/A's. When a numerical variable had an N/A, it usually implied the value was genuinely missing, or else the value would have been a 0. So instead of using 0 for all N/A's, we used the median to not affect our data as much. We debated using the mean as well, but with our large data set, the median would be sufficient, and would resist outliers.

2.2 Variable distribution assessment

Our next step of data cleaning involved analyzing the distribution of variables, including our response variable, SalePrice. We calculated skewness (using the e1071 library) to see which variables we cannot assume normality.

For any variable whose skewness >3, we did one of 3 things: converted it into a binary “0” or “More than 0” variable, a binary “1” or “More than 1” variable, or did a log transformation. We did the first two transformations on several variables, but we only did a log transformation on LotArea and our response variable, SalePrice. We decided to do Log transformation to lessen the effect of outliers and to make the data distribution more normal, as normal distribution is the underlying assumption for many linear regression models.

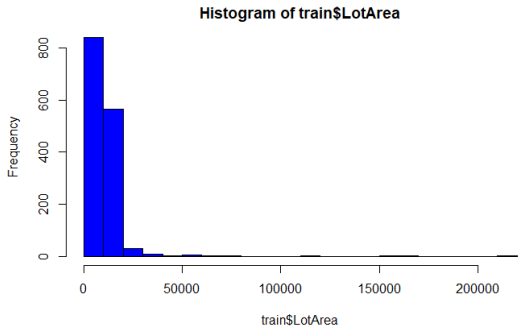


Figure 2.2.1 Histogram of train$Lot Area

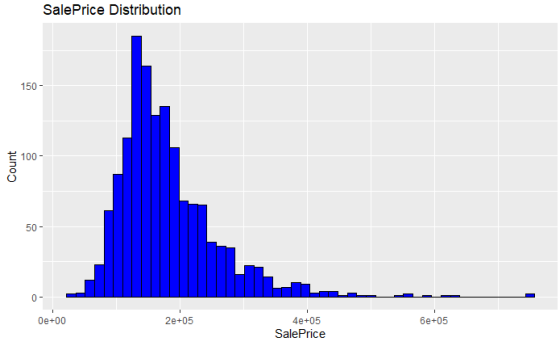


Figure 2.2.2 SalePrice Distribution

2.3 Dummy variable transformation

The last part of our data preparation and cleaning was for categorical variables. It was tough for us to do all of the modelling methods we wanted with categorical variables as factors, so we wanted each category to be binary - a piece of data is either in this category or it is not. And this applies to each category within a variable. For example, Basement Flooring Type is either hardwood or not hardwood, and also either carpet or not carpet. This is called dummy variable transformation. While it increased our explanatory variables from 80 to 314, this allowed us much more freedom to do more types of analyses, as you will see throughout our report.

3. Methods

3.1 Variable filtering

Next, we realized 314 variables was simply too many for several of the processes we wanted to implement. We thought that a good way to take out a large amount of the variables which do not relate to SalePrice would be to simply take the correlation between each variable and the SalePrice. We then took out any variable whose absolute correlation with SalePrice is less than 0.4. We are now only left with 28 predictors. This is much more manageable, from not only a computational perspective but also an interpretability perspective. Below is a heatmap of the correlations between the 29 variables, with SalePrice all the way to the right and top. Darker means more correlated.

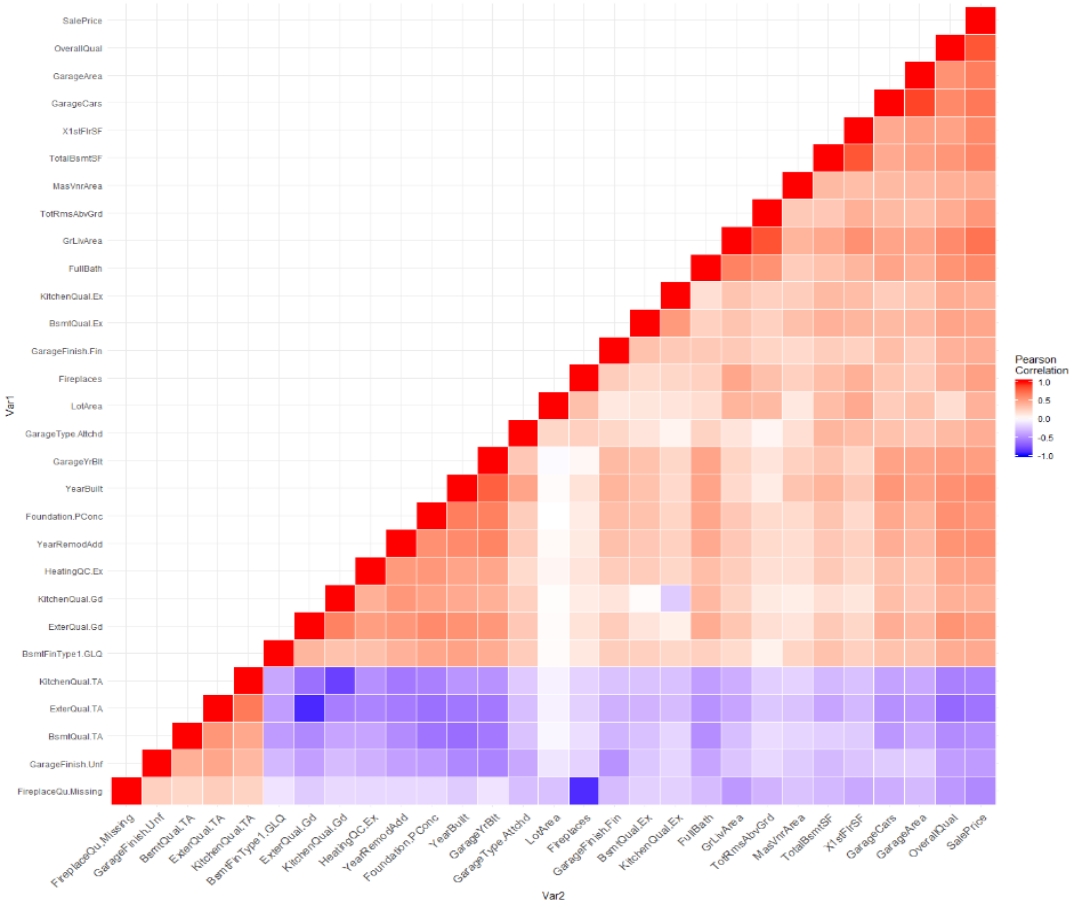


Figure 3.1 Heatmap of correlations

3.2 Linear regression techniques

3.2.1 Subset Selection Method: Best Subset Selection

To make the model more interpretable, we are especially interested in techniques that perform feature selection. Best subset selection not only can result in a sparse model but also has a simple fitting procedure. Best subset selection is exhaustive, which leads to better performance than forward and backward stepwise regression. However, due to its exhaustiveness, best subset selection would take a long time to process larger data sets.

We will use 10-fold Cross validation to choose the number of variables to include in our best subset selection model.

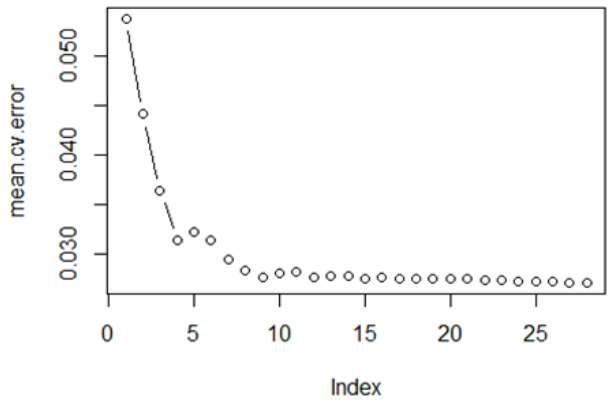


Figure 3.2.1 10-fold Cross validation

From the point of 9 variables, when we add another variable to the model, the mean cv error does not decrease substantially. Thus, we would use 9 variables in our final best subset selection model. We received a test MSE of 0.0188.

3.2.2 Regularization Methods

Lasso

In order to reduce the number of predictors in a model, we use LASSO regression. As opposed to Ridge Regression, LASSO performs feature selection. Its penalization method allows variables to be shrunken to 0 with an intermediate value of lambda (tuning parameter). Therefore, the resulting model will be sparser than the least square model. We use a 10-fold cv process to select the best lambda = 0.00253231. The final Lasso model results in a test MSE of 0.0184 and has 23 variables.

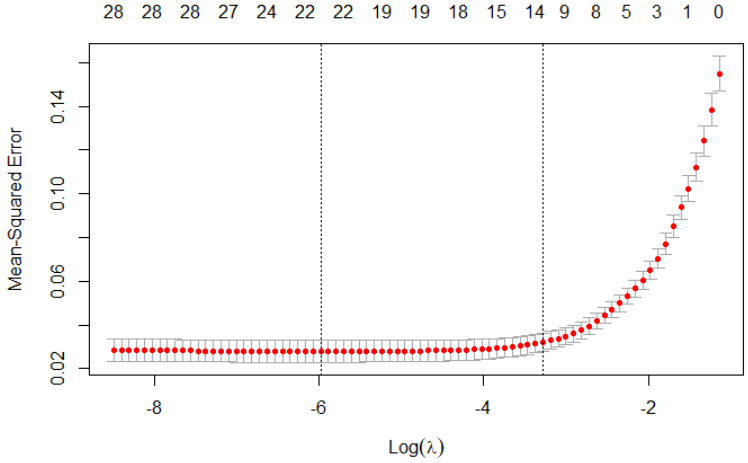


Figure 3.2.2-1 Lasso Test MSE

Ridge

We also try Ridge regression, to see if we get a lower MSE with it over LASSO. We also use 10-fold Cross validation to select the best lambda. Ridge, as opposed to LASSO, does not remove any variables, but otherwise is very similar in construction. Below is the graph of the log(lambda) vs. MSE. The lowest MSE is 0.019, which is slightly higher than the LASSO.

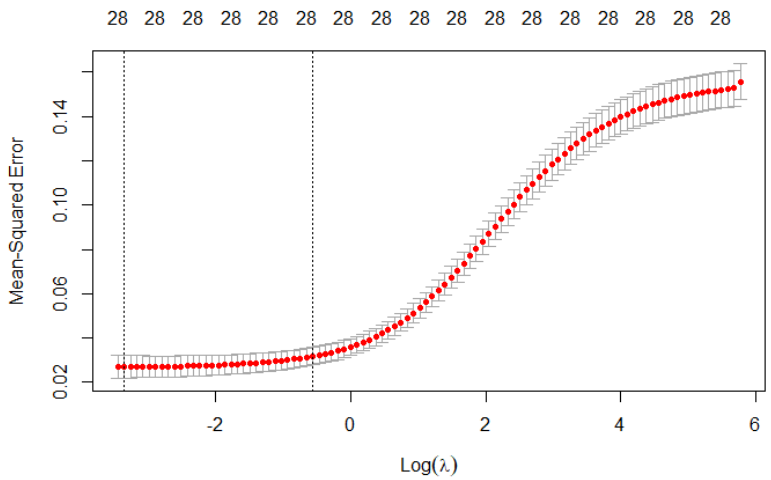


Figure 3.2.2-2 Ridge Test MSE

3.2.3 Dimension Reduction Methods

Dimension reduction methods are useful for high-dimensional data. Moreover, as we can see in the heatmap presented under the "Variable filtering" section, there are several pairs of predictors that are highly correlated such as *GarageCars* & *GarageArea* and *YearBuilt* & *GarageYrBlt*. Dimension reduction methods can help us avoid multicollinearity, since performing them on the raw data produces linear combinations of the predictors that are uncorrelated. We will get into more details below with Principal Component Regression (PCR) and Partial Least Squares (PLS).

Principal Component Regression (PCR)

PCR creates M new components from linear combinations of all original variables such that they capture as much variability in the predictors as possible. Next, the principal components create a low-dimensional subspace of the data and then fit a model on that subspace using Least Squares. When M is small, we can mitigate overfitting. However, PCR has a few weaknesses. First, it does not yield feature selection. Second, the first M principal components, though may best explain the predictors, are not necessarily predictive of the response.

Based on the result of the 10-fold cv process that you can see below, we've decided to include 10 components in our final PCR model. The resulting test MSE is 0.0216.

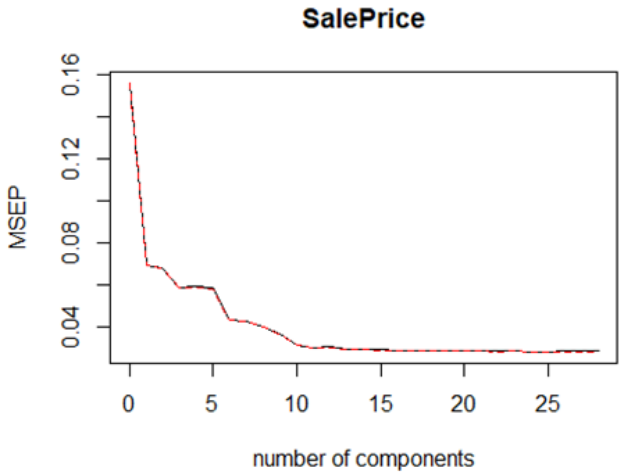


Figure 3.2.3-1 PCR Test MSE

Partial Least Squares (PLS)

PLS is the “supervised” alternative to PCR. The reason is that the PLS approach attempts to find linear combinations, i.e. directions, that help explain both the predictors AND the response. PLS bears all of PCR’s advantages and disadvantages. Nonetheless, there is one difference. Because PLS takes the response variable into account, it will reduce bias. However, the increased variance will wash out this benefit, making PLS model performance not much better than that of PCR.

After a 10-fold CV process (results below), we’ve decided to include 9 components in our PLS model. We achieved a test MSE of 0.0201.

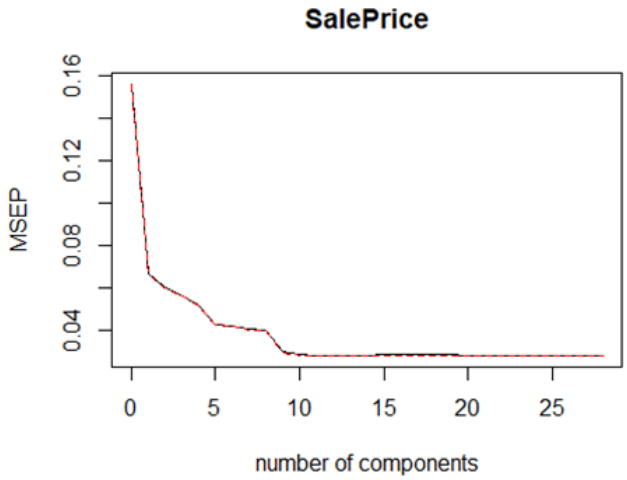


Figure 3.2.3-2 PLS Test MSE

3.3 Non-linear regression

3.3.1 KNN

As we know, KNN, or K-nearest neighbors, is a non-linear method that applies to both classification and regression. In this regression case, it is taking the estimate of an observation based on the values of its closest neighbors. We find the optimal k, then calculate MSE. With k= 16, the final KNN model has a test MSE of 0.026. The following graph is #Neighbors (or k) vs. RMSE.

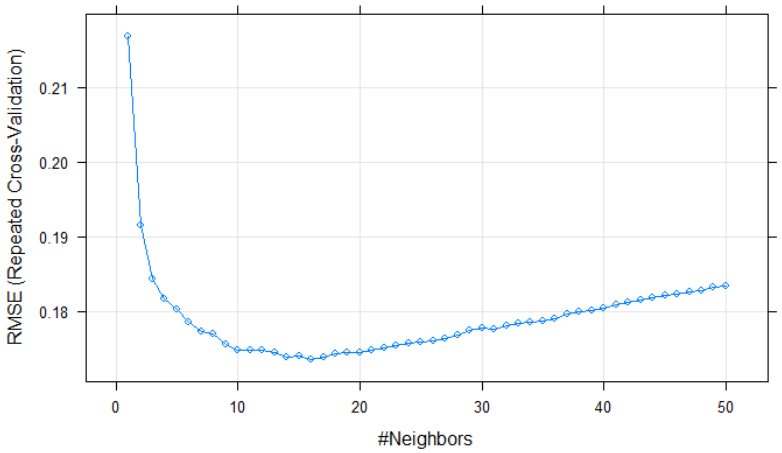


Figure 3.3.1 KNN Test MSE

3.3.2 Regression Tree

Decision Trees (DTs) where the target variable can take continuous values are called regression trees. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules.

We first fit a regression tree using training data and got a .0443 test MSE whereas the tree is complicated to interpret. For more interpretability and overfitting concerns we did pruning using terminal nodes from 6 to 9. We picked 6 according to comparison of cross-validations from 6 to 9 and got a 0.0553 test MSE.

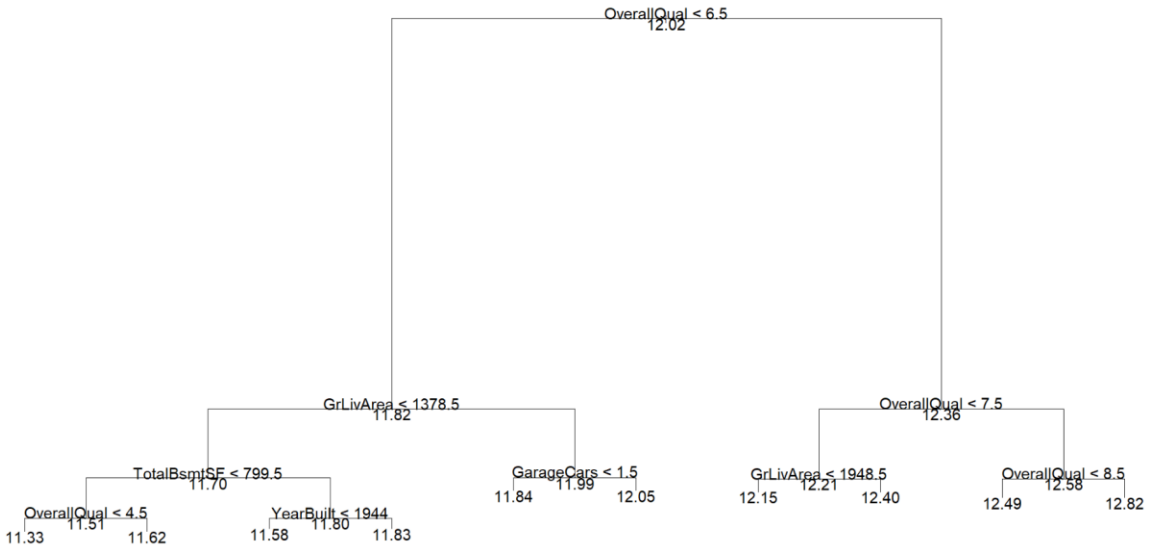


Figure 3.3.2-1 Regression Tree before Pruning

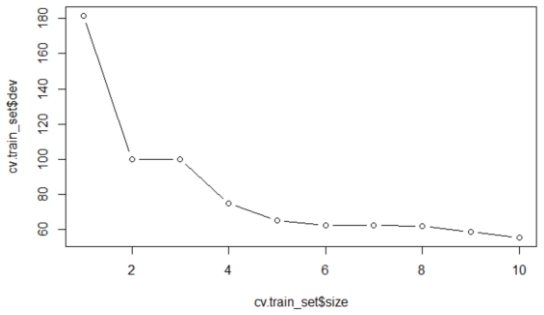


Figure 3.3.2-2 CV for Regression Tree

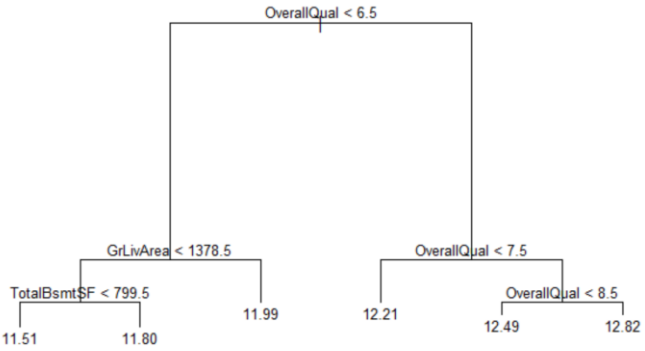


Figure 3.3.2-2 Regression Tree after Pruning

Decision trees are in general easy to understand and interpret. Unimportant features will not influence the result. The presence of features that depend on each other (multicollinearity) also doesn't affect the quality. However, they are inflexible in the sense that it is hard to incorporate new data. With these disadvantages, using an ensemble of predictive models can improve upon a single model's performance. We will discuss two common methods: bagging and random forests.

3.3.3 Bagging and Random Forest

The fundamental difference between bagging and random forest is that in Random forests, only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node. We got three different test MSEs by tuning parameters (number of trees and mtry) with different values as shown below.

# of Tree	mtry	Test MSE
500	28	0.0194
25	28	0.0207
25	20	0.0200

Figure 3.3.3-1 Test MSE for Bagging and Random Forest

Whether you have a regression or classification task, random forest is an applicable model for your needs. They are parallelizable, meaning that we can split the process to multiple machines to run. This results in faster computation time. But because we average all the trees in random forest, we are averaging the variance as well so that we have a low bias and moderate variance model. Random Forest creates a lot of trees and combines their outputs (unlike only one tree in case of decision tree), requiring much more computational power and resources.

3.3.4 Boosting

Boosting aims at fitting a new predictor in the residuals committed by the preceding predictor. There are four parameters: minimum observations in each tree measures the size of a tree, interaction depth measures the complexity of the boosted ensemble, shrinkage controls the learning rate and number of trees. When tuning parameters, we first consider minimum observations in each tree and interaction depth, then think about learning rate - shrinkage and aggregated number of trees. After a few times of trying, we get a relatively best set of four parameters with 0.0173 test MSE which will be talked about later is also our best model among all models.

n.minobsinnode	interaction.depth	n.tree	shrinkage	Test MSE
10	2	900	0.05	0.0173

Figure 3.3.3-2 Best Model Parameters

As an ensemble model, boosting comes with an easy to read and interpret algorithm, making its prediction interpretations easy to handle. The prediction capability is efficient through the use of its clone methods, such as bagging or random forest, and decision trees. Boosting is a resilient method that curbs over-fitting easily. One disadvantage of boosting is that it’s sensitive to outliers since every classifier is obliged to fix the errors in the predecessors. Another disadvantage is that because every estimator bases its correctness on the previous predictors, thus making the procedure difficult to streamline.

4. Results

Among all models we discussed and tested above, the gradient boosting model has the lowest test MSE at 0.0173. Therefore, the gradient boosting model is our best performing model.

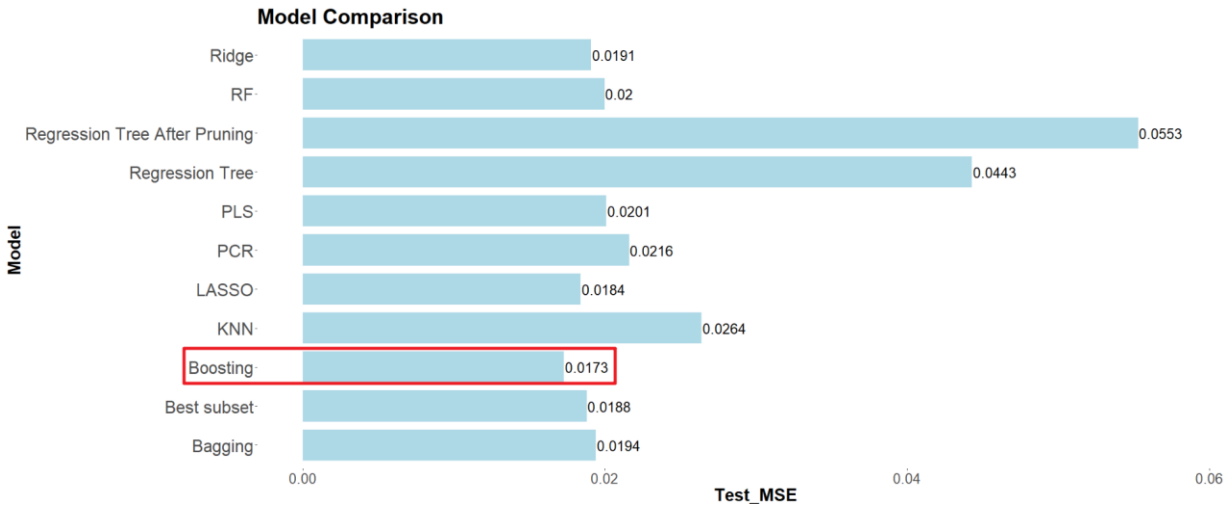


Figure 4.1 Model Comparison by Test MSE

The bar chart below shows the relative influence of the top 15 most important variables selected by gradient boosting. According to the chart, *OverallQual* (Overall material and finish quality) and *GrLivArea* (Above grade (ground) living area square feet) are much more significant in predicting house prices in Ames, IA, than other predictors.

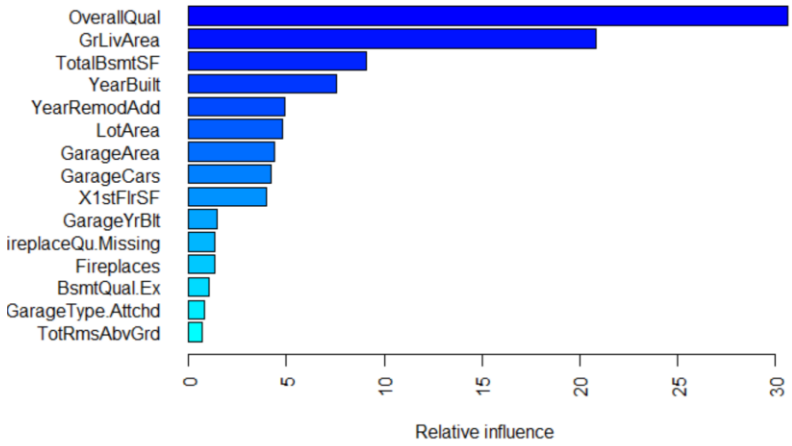


Figure 4.2 Relative influence by Gradient Boosting

5. Conclusion and remarks

In this project, we have applied 11 algorithms to predict the selling price of houses in Ames, IA, from linear techniques such as Lasso to non-linear such as Regression Tree, from parametric techniques such as Best subset selection to non-parametric such as KNN. After discussing the advantages and disadvantages of each algorithm, as well as comparing the test MSE's across models, we conclude that the best regression model is the Gradient Boosting model with a test MSE of 0.0173. The parameters of this model are as follows: n.tree = 900, interaction.depth = 2, shrinkage = 0.05, and n.minobsinnode = 10.

The Gradient Boosting model reveals the top 4 most important predictors. They are *OverallQual* (Overall material and finish quality), *GrLivArea* (Above grade (ground) living area square feet), *TotalBsmtSF* (Total square feet of basement area), and *YearBuilt* (Original construction date). These factors are reasonable since we have expected that when pricing a house, people tend to care about the size and the year it was built. However, we are not too happy that *OverallQual* is the most significant variable because we don't know how the score was calculated, i.e., which specific housing features were included in *OverallQual*.

In addition, there are a few surprises. First, among the top 15 predictors, 4 variables are garage-related. This may be a special case for Ames residents. Second, we found no location-related predictors in the top 15 variables. Our explanation is that since the only location indicator in our data is *Neighborhood* and since Ames is a small city, the neighborhoods will not have much influence on the prices of otherwise 2 similar houses. Another explanation is that by using variable filtering (keeping only variables whose absolute correlations with our dependent variable are more than .4), we have removed *Neighborhood*, which could have been an important variable if included. For future improvement of the project, we look forward to applying these same 11 techniques as well as more advanced techniques such as XGBoost and Neural Network, without variable filtering at the beginning, to predict house selling price and to achieve better performance. We also plan to train our models in a cloud environment such as AWS so we can tune parameters with a wider range more efficiently. After this project, we believe that we are more prepared to deal with high-dimensional data.

6. Appendix

1. We check for missing values during the data pre-processing step.

```{r, echo = F}
summary(train$Alley)
```

```
 Grvl  Pave  NA's
   50    41  1369
```

```{r, echo=FALSE}
summary(train$PoolQC)
```

```
  Ex   Fa   Gd NA's
   2    2    3 1453
```

```{r, echo = FALSE}
summary(train$Fence)
```

```
 GdPrv  GdWo MnPrv  MnWw  NA's
    59    54   157    11  1179
```

2. Missing value treatments

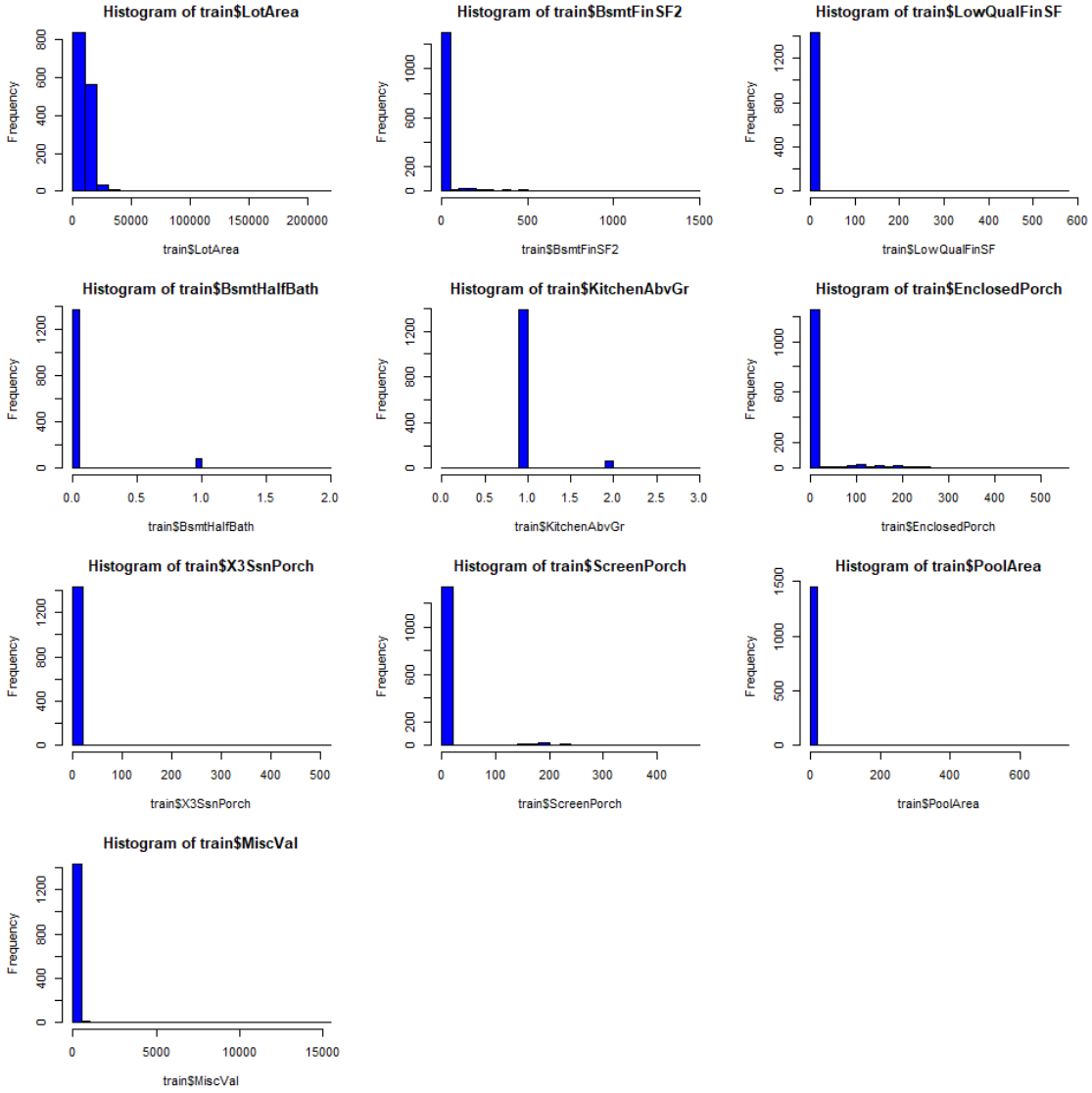
```{r}
# Creating a new level in categorical variables to store NA values

train[] <- lapply(train, function(x){
  # check if you have a factor first:
  if(!is.factor(x)) return(x)
  # otherwise include NAs into factor levels and change factor levels:
  x <- factor(x, exclude=NULL)
  levels(x)[is.na(levels(x))] <- "Missing"
  return(x)
  })
```

```{r}
summary(train$Alley)
```

```
   Grvl    Pave Missing
     50      41    1369
```

```{r}
# replacing NA's in numerical variables with the respective column median
for (i in 1:ncol(train)) {
  if(is.numeric(train[,i])) {
  train[,i][is.na(train[,i])] <- median(train[,i], na.rm=TRUE)
  }
}
```

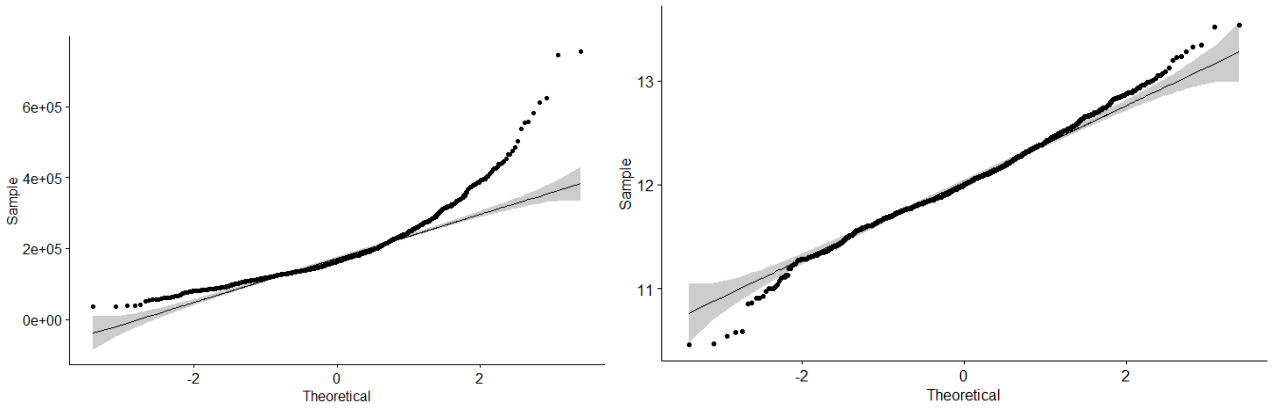
3. Identify highly-skewed variables and transformation




```{r, echo = FALSE}
# create a function to turn all highly-skewed numerical variables with many
# zeros into categorical variables with 2 levels "More than 0" and "0".

categorize <- function(x) {
  for (i in 1:length(x)) {
    if (x[i] > 0) {
      x[i] <- "More than 0"
  }
}

  x <- as.factor(x)
}

```

4. QQplot of SalePrice before and after Log transformation

```{r, echo = FALSE}
train_off$SalePrice <- log(train_off$SalePrice)
ggqqplot(train_off$SalePrice)
```



5. Variable filtering code:

```r
# dataset train_X only contain predictors
train_X <- train_off[,-which(colnames(train_off)=="SalePrice")]
num_list <- c() # a vector that only contain numerical variables from the train_off dataset
for (i in 1:ncol(train_X)) {
  a <- is.numeric(train_X[,i])
  if (a == TRUE) {
    num_list <- c(num_list,i)
  }
}

# train_X_num is a dataset that only contains numerical variables
train_X_num <- train_X[,num_list]
dim(train_X_num)
```

```
[1] 1459  314
```

```r
cors <- cor(train_X_num,train_off$SalePrice)
whichers <- which(abs(cors)>.4)
whichers
```

```
  [1]   9  88  90  91 142 145 146 154 158 161 175 191 198 211 215 219 224 226 227 228 236 242 244 250 251 253 255 256
```

```r
colnames(train_X_num[88])
colnames(train_X_num[191])
colnames(train_X_num[215])
colnames(train_X_num[255])
colnames(train_X_num[256])
```

```
[1] "OverallQual"
[1] "TotalBsmtSF"
[1] "GrLivArea"
[1] "GarageCars"
[1] "GarageArea"
```

```r
train_X <- train_X_num[, c(9,88,90,91,142,145,146,154,158,161,175,191,198,211,215,219,224,226,227,
                           228,236,242,244,250,251,253,255,256)]
train_off_new <- train_X
train_off_new$SalePrice <- train_off$SalePrice
train_off <- train_off_new
```

train_off has 29 variables including 28 predictors and 1 response variable, SalePrice.

```r
dim(train_off)
```

```
[1] 1459   29
```

6. Best subset selection code

```r
library(leaps)
```

```{r, echo = FALSE}
#Let's use a 10 fold CV to select the best number of variables to include in the model

k=10
set.seed(10)
folds=sample(1:10,nrow(train_set),replace=TRUE) #construct 10 different subsets of the training data
cv.error=matrix(NA,k,28) #construct a matrix to store validation error for each fold across all models
```

```{r, echo = FALSE}
for (j in 1:k){
  best.fit=regsubsets(SalePrice~., data=train_set[folds!=j,],nvmax=28) #we fit the model using all data except for fold j
  for (i in 1:28){
    coefi=coef(best.fit,id=i) #extract coefficients for the best model with i number of regressors
    cvi.mat=model.matrix(SalePrice~.,data=train_set[folds==j,]) #convert data from fold j to a matrix
    pred=cvi.mat[,names(coefi)]%*%coefi #calculate prediction
    cv.error[j,i]=mean((train_set[folds==j,]$SalePrice-pred)^2) #calculate squared errors
  }
}
```

```{r, echo = FALSE}
mean.cv.error=apply(cv.error,2,mean) #calculate mean squared error for each model by averaging the MSE across all folds
par(mfrow=c(1,1))
plot(mean.cv.error,type="b")
```

```{r, echo = F}
#calculate the test MSE

reg.best=regsubsets(SalePrice~.,data=train_set,nvmax=28)
coef.test=coef(reg.best,id=9) # variables in the best-fit model

test.mat=model.matrix(SalePrice~.,data=test_set)
pred=test.mat[,names(coef.test)]%*%coef.test
best_sub_mse=mean((test_set$SalePrice-pred)^2)
best_sub_mse
```

7. Nine variables in the final best subset selection model along with their respective coefficient estimates. LotArea and OverallQuall are the most important predictors.

```
##      (Intercept)          LotArea      OverallQual        YearBuilt
##     1.500238e+00     1.070885e-01     9.018400e-02     1.785542e-03
##     YearRemodAdd BsmtFinType1.GLQ      TotalBsmtSF        GrLivArea
##     2.503247e-03     5.221221e-02     6.667633e-05     1.691430e-04
##       Fireplaces       GarageCars
##     6.618327e-02     7.028058e-02
```

8. Lasso model 23 non-zero variables, and its code:

```{r, include = F}
#LASSO regression
set.seed(3)
lasso.mod=glmnet(x.train,y.train,alpha=1) #build a LASSO regression
cv.out=cv.glmnet(x.train,y.train,alpha=1) # use 10 fold cv to select shrinkage parameter
plot(cv.out)
bestlam_l=cv.out$lambda.min #find the best shrinkage parameter
bestlam_l
lasso.pred=predict(lasso.mod,s=bestlam_l,newx=x.test) #making prediction using the best shrinkage parameter
lasso_mse=mean((lasso.pred-y.test)^2) #calculate test MSE
lasso_mse
out=glmnet(x.train,y.train,alpha=1)
lasso.coef = predict(out,type="coefficients",s=bestlam_l)[1:29,]
# show only coefficients that are not zero
lasso.coef[lasso.coef!=0]
```

```
##         (Intercept)             LotArea         OverallQual
YearBuilt
##        3.562722e+00        9.875456e-02        7.935398e-02
1.513384e-03
##        YearRemodAdd          MasVnrArea         BsmtQual.Ex
BsmtQual.TA
##        2.009279e-03        2.332795e-05        4.690704e-02
8.213336e-03
##    BsmtFinType1.GLQ         TotalBsmtSF        HeatingQC.Ex
GrLivArea
##        4.624970e-02        4.771747e-05        3.259321e-02
1.460163e-04
##            FullBath      KitchenQual.Ex      KitchenQual.Gd
TotRmsAbvGrd
##        7.591947e-03        6.760217e-02        3.002766e-02
7.079510e-03
##          Fireplaces FireplaceQu.Missing   GarageType.Attchd
GarageYrBlt
##        4.380824e-02       -2.547013e-02        3.549729e-02
2.213093e-04
##    GarageFinish.Unf          GarageCars          GarageArea
##       -2.940296e-03        5.851941e-02        3.573118e-05
```

9. Ridge model and its code (plot is in the report):

```
```{r, echo = F}
#Ridge regression
set.seed(53)
ridge.mod=glmnet(x.train,y.train,alpha=0) #build a ridge regression
cv.out=cv.glmnet(x.train,y.train,alpha=0) # use 10 fold cv to select shrinkage parameter
plot(cv.out)
bestlam_r=cv.out$lambda.min #find the best shrinkage parameter
bestlam_r
ridge.pred=predict(ridge.mod,s=bestlam_r,newx=x.test) #making prediction using the best shrinkage parameter
ridge_mse = mean((ridge.pred-y.test)^2) #calculate test MSE
ridge_mse
```
```

10. PCR code (model fit and cv graph)

```
```{r, echo = F}
#install.packages("pls")
library(pls)
set.seed(2)
pcr.fit=pcr(SalePrice~., data = train_set, validation="CV") #run principal component regression using 10 fold CV to select the best number of component
#summary(pcr.fit)
```
```

```
```{r, echo = F}
validationplot(pcr.fit,val.type = "MSEP") #show the CV MSE for different number of components
```
```

```
```{r, include = T}
pcr.pred=predict(pcr.fit,test_set,ncomp=10) #make predictions using 10 components
pcr_mse = mean((test_set$SalePrice-pcr.pred)^2) #calculate mse using the test set
pcr_mse
```
```

```
[1] 0.02155774
```

11. PLS code (model fit and cv graph)

```
```{r, echo = F}
set.seed(20)
pls.fit=plsr(SalePrice~.,data=train_set, validation="CV") #run partial least square using 10
fold CV to select the best number of component
#summary(pls.fit)
```
```

```
```{r, echo = F}
validationplot(pls.fit,val.type = "MSEP")
```
```

12. KNN Code (plot is in the report):

```r
trControl <- trainControl(method = 'repeatedcv',
                          number = 10,#number of resampling iterations
                          repeats = 3)# repeat the whole thing three times
set.seed(13)
knnFit <- train(SalePrice ~.,
                data=train_set,
                tuneGrid = expand.grid(k=1:50),
                method='knn',
                trControl = trControl,
                preProc = c('center','scale') #Used to normalize our data - "center" is used to subtract against the mean, and "scale" is used to divide by standard deviation, giving a Z-score, essentially.
                )
knnFit #k=16 Gives lowest value for RMSE (0.2075071) and a very close to highest value of R^2 of 0.7262369
plot(knnFit)
knn.pred=predict(knnFit,test_set,k=16) #make predictions using 9 components

knn_mse = mean((test_set$SalePrice-knn.pred)^2) #calculate mse using the test set
knn_mse
```

13. Codes for Regression Tree (Figure 3.3.2-1 Regression Tree before Pruning)

```r
# install.packages("tree")
library(tree)
```

```r
tree.train_set=tree(SalePrice~.,data = train_set) #fit a regression tree using the training data
summary(tree.train_set)
```

```
Regression tree:
tree(formula = SalePrice ~ ., data = train_set)
Variables actually used in tree construction:
[1] "OverallQual" "GrLivArea"   "TotalBsmtSF" "YearBuilt"   "GarageCars"
Number of terminal nodes:  10
Residual mean deviance:  0.04157 = 48.1 / 1157
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.869100 -0.101200  0.009439  0.000000  0.117300  0.715100
```

```r
plot(tree.train_set) # plot the tree
text(tree.train_set, all = TRUE, cex = 1.8, pretty = 0)
```

14. Codes for CV for Regression Tree (Figure 3.3.2-2 CV for Regression Tree)

```r
set.seed(4)
cv.train_set=cv.tree(tree.train_set) #find optimal num of terminal node using cross validation
```

```r
plot(cv.train_set$size, cv.train_set$dev, type="b")
```

15. Codes for Plot Regression Tree after Pruning (Figure 3.3.2-2 Regression Tree after Pruning)

```r
# for more interpretability and overfiting concerns do some pruning within 6-9 terminal nodes
prune.train_set = prune.tree(tree.train_set, best = 6)
plot(prune.train_set)
text(prune.train_set, cex = 0.8, pretty = 0)
```

16. Codes for Bagging and Random Forest Model

```{r, include = F}
library(gbm)
```

```{r, include = F}
library(MASS)
```

```{r}
set.seed(6)
bag.price=randomForest(SalePrice~.,data=train_set, mtry=28,importance=TRUE) #apply bagging on decision tree
bag.price
```

```
Call:
 randomForest(formula = SalePrice ~ ., data = train_set, mtry = 28,      importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 28

          Mean of squared residuals: 0.02546461
                    % Var explained: 83.63
```

17. Codes for Gradient boosting Tuning (Figure 3.3.3-2 Best Model Parameters)

```{r, include = FALSE}
# hyper-parameter tuning of gradient boosting
set.seed(7)
grid <- expand.grid(n.trees = 900, interaction.depth=2, shrinkage=c(0.05,0.1),n.minobsinnode=c(10,15))
ctrl <- trainControl(method  = "cv",number  = 10)
unwantedoutput <- capture.output(GBMModel <- train(SalePrice~.,data = train_set,
                  method = "gbm", trControl = ctrl, tuneGrid = grid))
```

18. Codes for Tuning Best Set of Parameters of Gradient Boosting

```{r}
print(GBMModel)
```

```
Stochastic Gradient Boosting

1167 samples
  28 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1049, 1051, 1050, 1050, 1050, 1050, ...
Resampling results across tuning parameters:

  shrinkage  n.minobsinnode  RMSE       Rsquared   MAE
  0.05       10              0.1537730  0.8508429  0.1037881
  0.05       15              0.1538543  0.8506796  0.1048401
  0.10       10              0.1558227  0.8469313  0.1052454
  0.10       15              0.1541082  0.8513729  0.1049824

Tuning parameter 'n.trees' was held constant at a value of 900
Tuning parameter
 'interaction.depth' was held constant at a value of 2
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were n.trees = 900, interaction.depth = 2, shrinkage = 0.05
 and n.minobsinnode = 10.
```

18. Codes for Gradient Boosting Test MSE

```r
# Test MSE
yhat.boost=predict(GBMModel,newdata=test_set,n.trees=900)
boost_mse = mean((yhat.boost-price.test)^2)
boost_mse
```

19. Codes for Figure 4.2 Relative influence by Gradient Boosting

```r
par(mar = c(5, 8, 1, 1))
summary(
  GBMModel,
  cBars = 15,
  method = relative.influence, # also can use permutation.test.gbm
  las = 2
  )
```

20. Codes for model comparison graph (Figure 4.1 Model Comparison by Test MSE)

```r
library(tidyverse)
library(ggpubr)
# Bar plot of results
Regression_MSE <- tibble(Model = c("Best subset", "LASSO", "Ridge", "PCR", "PLS",
                                   "KNN", "Regression Tree", "Regression Tree After Pruning",
                                   "Bagging","RF", "Boosting"),
                         Test_MSE = c(round(best_sub_mse,4), round(lasso_mse,4),
                                      round(ridge_mse,4), round(pcr_mse,4),
                                      round(pls_mse,4), round(knn_mse,4), round(reg_tree_MSE,4),
                                      round(prune_MSE,4),
                                      round(bag_mse,4), round(rf_mse,4), round(boost_mse,4)
                                      ))

p = Regression_MSE %>%
  ggplot(aes(Model, Test_MSE, ymin = 0.00, ymax = 0.06)) +
  geom_col(position = "dodge", width = 0.8, fill = "lightblue") +
  ggtitle("Model Comparison")

p + theme(plot.title = element_text(size = 30, face = "bold"),
axis.title=element_text(size=25,face="bold"), panel.grid.major = element_blank(), panel.grid.minor =
element_blank(),axis.text.x = element_text(angle = 0, size =20), axis.text.y = element_text(angle = 0,
size = 25)) + geom_text(aes(label=Test_MSE),position=position_dodge(width=0.9), hjust=-0.02, size = 7) +
  coord_flip() + bgcolor("white")
```