HOUSE PRICES

12/08/2018

TIM 209 – Sravya Pulavarthi, Adit Patel

Kaggle competition on House prices dataset to apply machine leaning techniques and predict the price for test data.

CONTENTS

House Prices

TIM 209 – SRAVYA PULAVARTHI, ADIT PATEL

INTRODUCTION

House Prices: Advanced Regression Techniques is a great dataset to apply intermediate machine learning algorithms. The dataset basically aims to predict the house prices of a neighborhood located in Ames, Iowa. The final goal is to predict the final price of each home located in the neighborhood. This dataset was compiled by Dean De Cock for use in data science education.

Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this playground competition's dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges students to predict the final price of each home.

Exploring data

Before we began applying machine learning techniques and algorithms, we decided to look and understand the data first. We had at our disposal 79 explanatory variables, and thus one of the first task at hand was data preparation. Make sure there was no missing data, and no outliers which might potentially skew the analysis.

Correct Data Types

One of the first observation we made was the existence a few columns which despite having numerical information stored within them behaved as factors according to the data description. We performed table operation on the suspicious columns and confirmed our hypothesis that these columns should be of factor type. Some of the variables on which this treatment was performed are: MSSubClass, Fireplaces, OverallQual, etc.

With the use of *apply* function, we converted the data present in these columns to factor datatype.

Missing Data

The next task we wanted to focus on was dealing with missing values and making our dataset exhaustive. We first try to see how many rows are there with all the values available. For this we use the *complete.cases* function. At an overall level there is not a single row which has all the values available. We plot the columns and the count of missing values present in them. Below is the plot for the same:

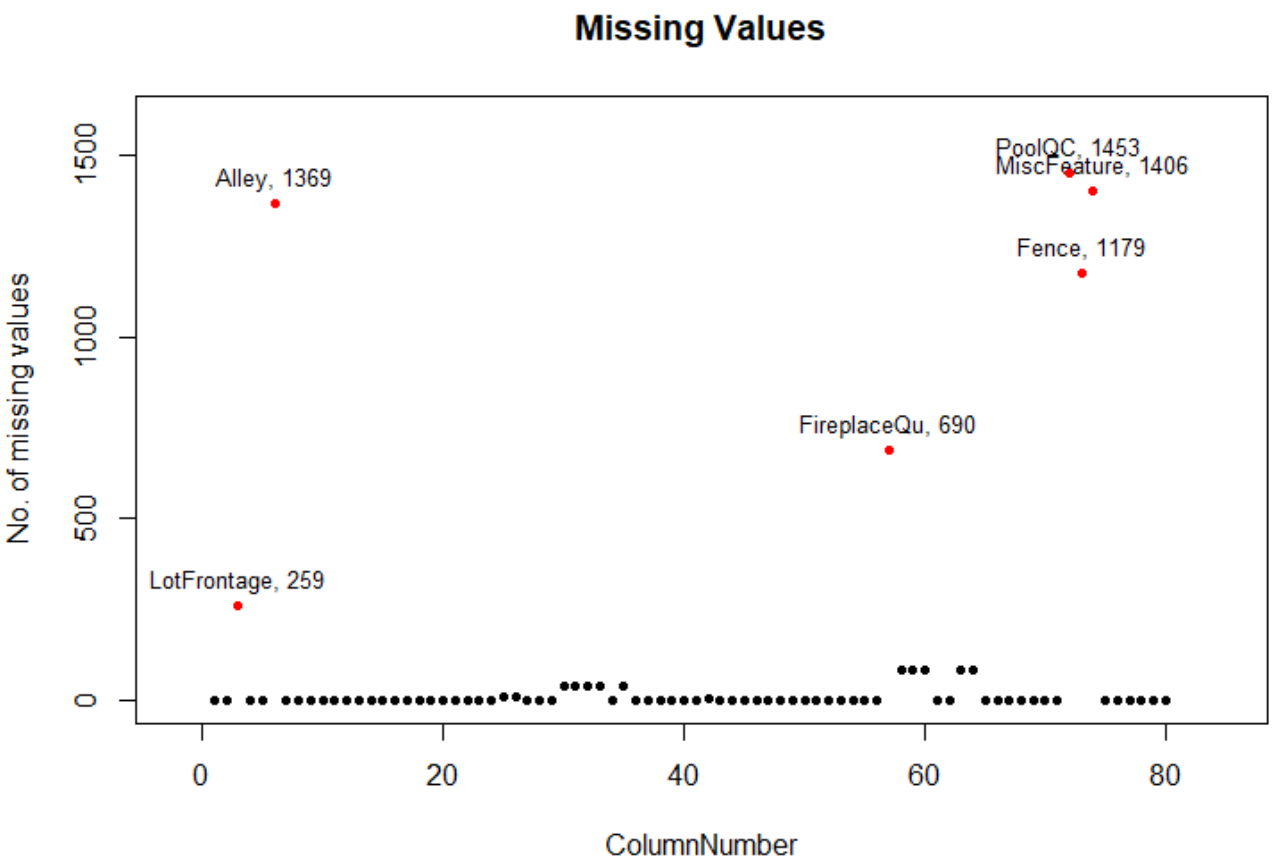


Figure 1: The above plot shows columns who have more than 100 missing values

Exploring on these columns further, we observe that these columns are not that significant in our analysis and we can move ahead by removing them completely because they do not have much information to add to our model anyways because of the number of missing values. So, after removing these 6 columns we are left with a total of 74 columns.

After this operation we run *complete.cases* function again to see how many rows are there with information present for all the remaining columns. We observe that there are 1338 such rows and 122 rows where one or more than one column has a missing value. For our analysis, we decide to remove these rows so that we can decide which columns are important and their coefficients can be calculated on equal data points across all the columns.

Figuring out significant columns

Next task was to determine which columns will be selected for methods which are computationally expensive as number of predictors increases. GAM modelling, Best Subset selection are some of the methods where a reduced number of columns can be useful. But how do we know which columns to choose. At the very least we will choose columns who have a linear relationship with the response. So, we first begin with applying OLS (Ordinary Least Squares) to all the predictors present in our dataset. We will only choose the columns which show some statistical significance (i.e. a small p-value), which means that they possess some linear relationship with the response variable. After going through this operation, the remaining data consists of 22 significant predictors. We can

further look at the relationship between these variables and the response variable to remove a few more columns which might not give any significant information towards building the model. Our final dataset contains 21 predictors which have some linear relationship with the response (SalePrice) variable. This dataset will only be used with techniques where analyzing all the variables is not an option. For methodologies such as Lasso, decision trees, etc. where we can tune parameters or prune the output to a more simplistic dataset, we will be using the complete dataset with all the 73 predictor variables.

Relationship between response and predictors

Using the above reduced dataset, we can calculate the correlation between each numerical variable. In our current reduced dataset, we have 7 numerical variables. The correlation plot between these variables is as below:

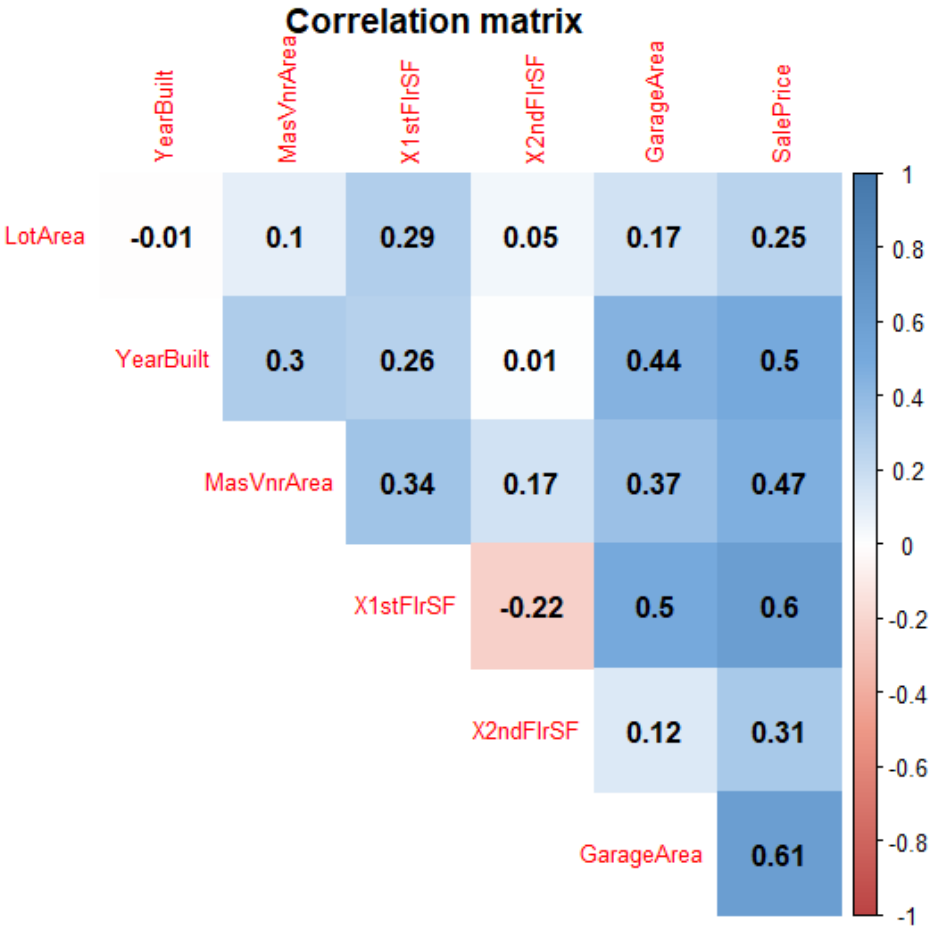


Figure 2: Correlation plot between numerical variables

MODELLING

After understanding and treating our data, we begin with modelling the data by applying statistical machine learning techniques. For our approach we will be mainly looking at techniques such as Lasso, Ridge, Splines, GAM and Decision trees. We will cover all the aspects revolving around these techniques. When enough information about all these techniques is available to us, our next approach will be to use Ensemble Modelling

in order to capture all the strengths of the above techniques and weed out the weaknesses which these messages are not able to capture.

Ridge Regression

Instead of performing ridge regression for an automatically selected range of λ values, we chose to implement the function over a grid of values ranging from λ = 0.01 to λ = 100 billion, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit. We also standardized the variables so that they are on the same scale. To choose the best value of the tuning parameter lambda, we use cross-validation and choose the model with the lowest CV error.

Dataset with all Predictors

When we apply ridge regression on the entire dataset with 73 predictors, the best value of the tuning parameter λ according to CV is 32985.87.

We compute the error corresponding to this value of the tuning parameter to be 749953166.

Dataset with Significant Predictors

Applying ridge regression on the dataset with significant predictors, the best value of the tuning parameter λ according to CV is 8388.824, which is lower than that obtained previously. The error corresponding to this value of the tuning parameter is 887416149 and is slightly higher than the model with all the predictors.

The plot shown below corresponds to the change in mean-squared error with respect to the λ values with standard deviation.

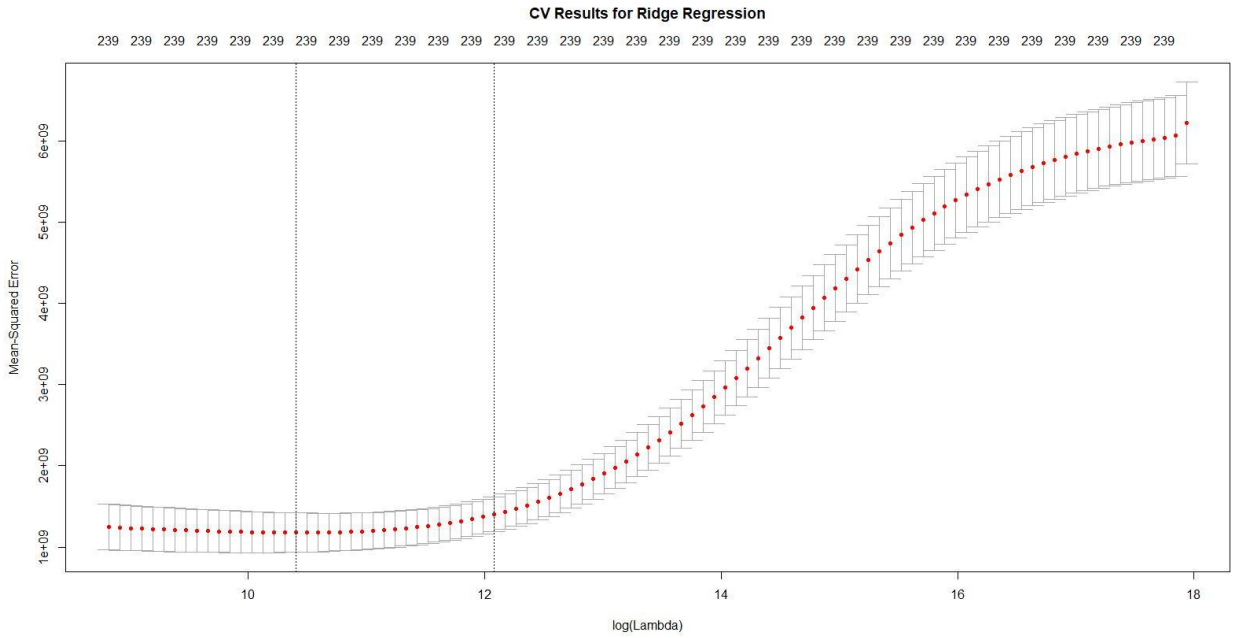


Figure 4: MSE vs. λ values for Ridge Regression

The Lasso

Like ridge regression, we chose to implement the function over a grid of values ranging from $\lambda = 0.01$ to $\lambda =$ 100 billion, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit. To choose the best value of the tuning parameter lambda, we use cross-validation and choose the model with the lowest CV error. We now build the model on the dataset with all predictors and the dataset with only significant variables and compare the results.

Dataset with all Predictors

When we apply the lasso to the entire dataset with 73 predictors, we can see from the coefficient plot that depending on the choice of tuning parameter, some of the coefficients will be exactly equal to zero when the Lasso is applied. The best value of the tuning parameter λ according to CV is 779.9473.

We compute the error corresponding to this value of the tuning parameter to be 707544889.

Dataset with Significant Predictors

Applying the Lasso to the dataset with only the significant predictors, the best value of the tuning parameter λ according to CV is 664.8, which is lower than that obtained previously.

We observe that a few coefficients will be assigned a value of 0 when the Lasso is applied and the MSE increases slightly compared to that obtained by applying Ridge Regression. The corresponding MSE for the best value of λ chosen is 901835183.

The plot shown below corresponds to the change in mean-squared error with respect to the λ values, along with the standard deviation.

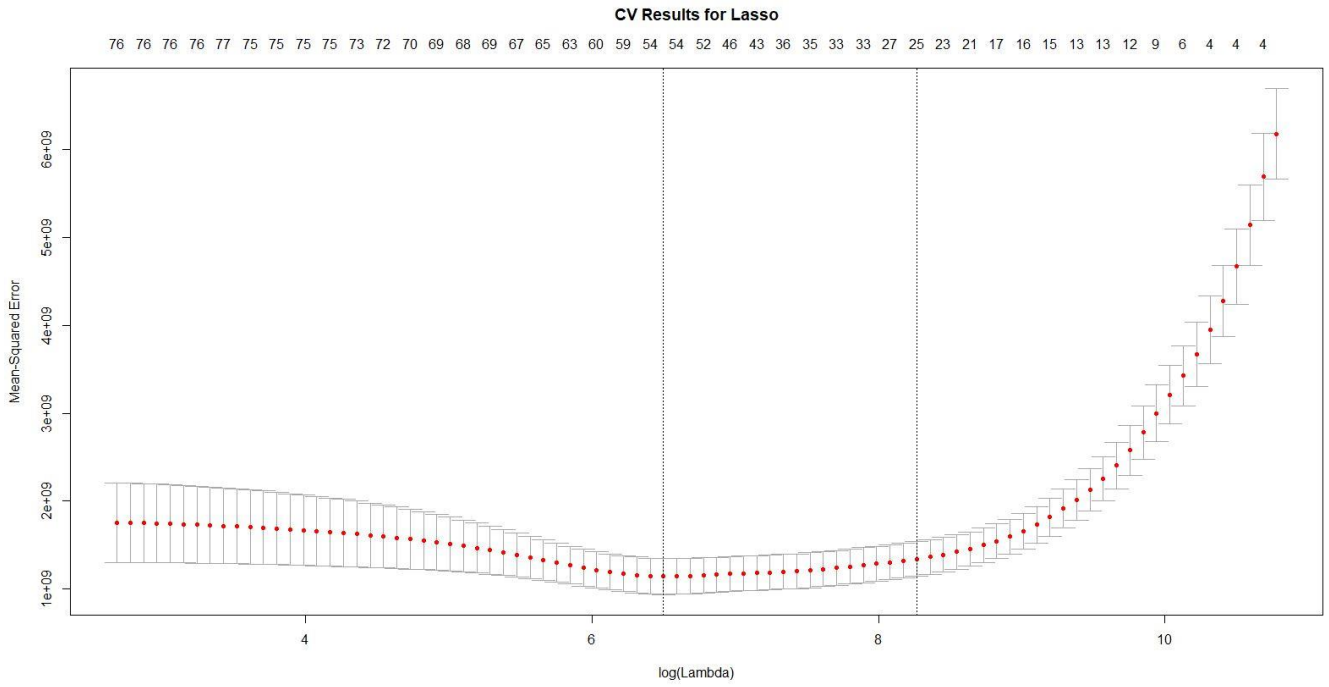


Figure 5: MSE vs. λ values for Lasso

We summarize the results obtained by applying Ridge Regression and the Lasso on all the predictors and the significant predictors below.

Methodology	Reduced Predictors	
	Ridge Regression	Lasso
Training Error	887,416,149	901,835,183
Validation Test Error	810,823,946	957,045,433
Cross-Validation Error	1,185,049,832	1,172,463,095

Methodology	All Predictors	
	Ridge Regression	Lasso
Training Error	749,953,166	707,544,889
Validation Test Error	685,854,427	651,191,331
Cross-Validation Error	1,227,455,049	1,271,173,219

It is observed that applying the regularization techniques to the dataset with more predictors generates a smaller error compared to the models obtained using only the significant predictors. However, this error is acceptable as there is no huge improvement in the error when we use all the predictors. Our chosen model with fewer variables is satisfactory and accurate.

Generalized Additive Models and Splines

The first method we will discuss here is Generalized additive models. Generalized additive model is used to capture non-linear relationships between the response and the individual predictors. It can fit higher degree polynomials while simultaneously dividing the data into different parts (at knots) all the while maintaining constraints so that the relationship is smooth. Iterating to find a non-linear relationship with the entire list of variables available to us is not feasible, hence we will be using the reduced dataset and learn the relationships they share with the response. To do this, we begin with plotting the response variable with these predictors and then deciding which non-linear method best estimates the predictor values but also takes care that no overfitting occurs.

Quantitative Variables

The quantitative variables present in our dataset are: LotArea, YearBuilt, MasVnrArea, 1stFlrSF, 2ndFlrSF, and GarageArea. We will try to build a non-linear relationship for each of these variables.

1. LOTAREA

The first variable is LotArea. We try building non-linear models using basis splines, natural splines, smooth splines and local regression. We tweak the parameters such that the predicted model makes sense to use. We do not want the model to overfit on the training data and we want to make sure that with increase in lot area, the sale price should increase as well. Below is a representation of the output we received while applying the above models:

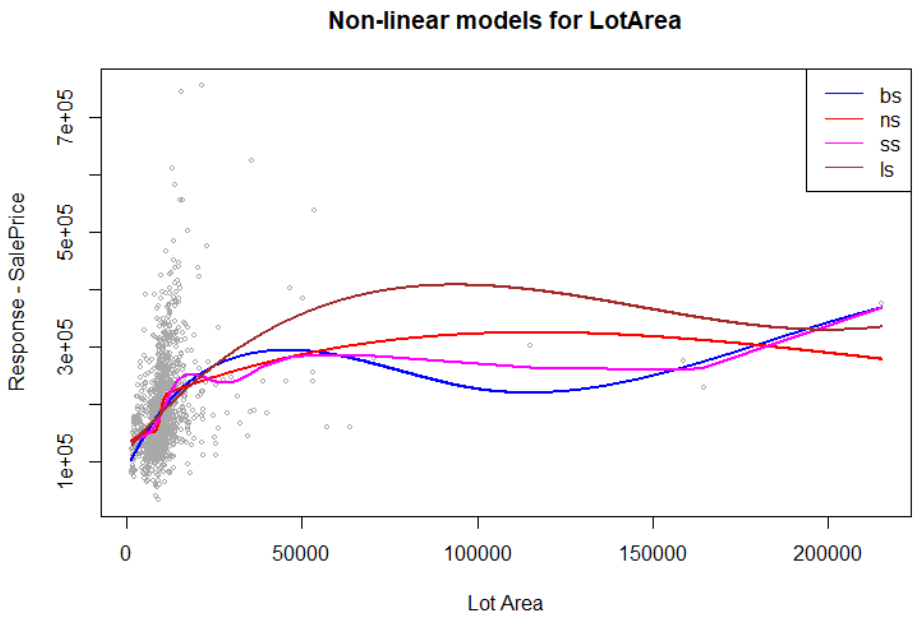


Figure 5: Non-linear models for LotArea. Selected: Local Regression with span = 1

By Looking at the diagram above, we found that local regression with a span of 1 is our best bet because it doesn't overfit and ensures that SalePrice increases with an increase in the value of LotArea.

2. YEARBUILT

For YearBuilt, we had an expectation that more recent the houses, costlier they should be. We wanted to build a model which will basically preserve that information but also try to generalize the information we have about each individual year. The non-linear relationships with the best parameters are shown below:

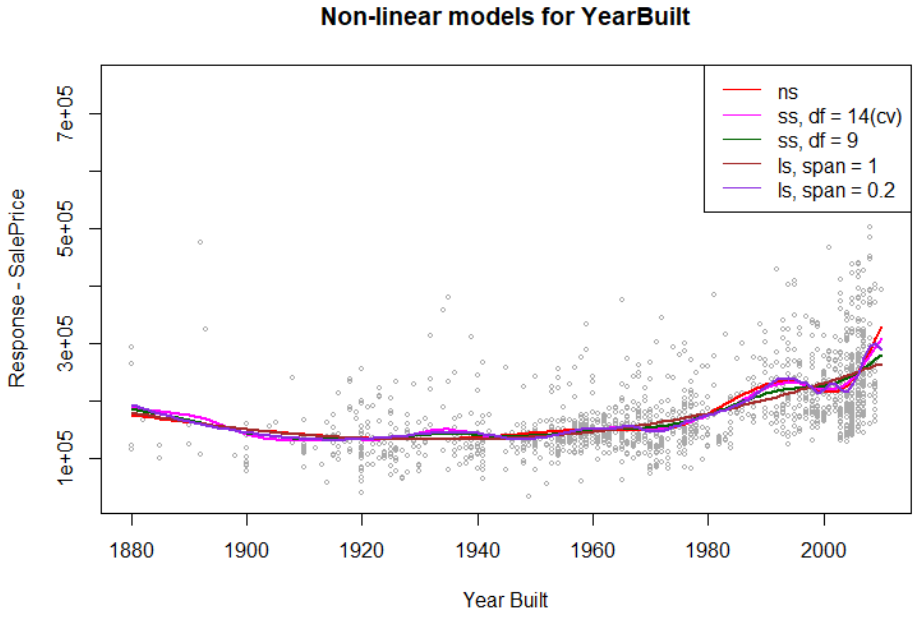


Figure 6: Non-linear models for YearBuilt. Selected: Smooth spline with df = 14 (using CV)

We choose the model containing smooth spline with degrees of freedom chosen with the help of Cross Validation which is 14.16. This prediction is smooth, and it captures the bumps of certain years and the slump around the 2008 housing crisis.

3. MASVNRAREA

Masonry Veneer Area is a variable which normally is not taken into consideration. For our dataset however, we find out that this predictor is significant enough to influence the response variable. The variable on its own however does not give much information pertaining to prediction, but when we combine this variable with MasVnrType (Masonry Veneer Type) we see some patterns which start to emerge. Below is the plot of the data, when we plot it in conjunction with MasVnrType in Figure 8.

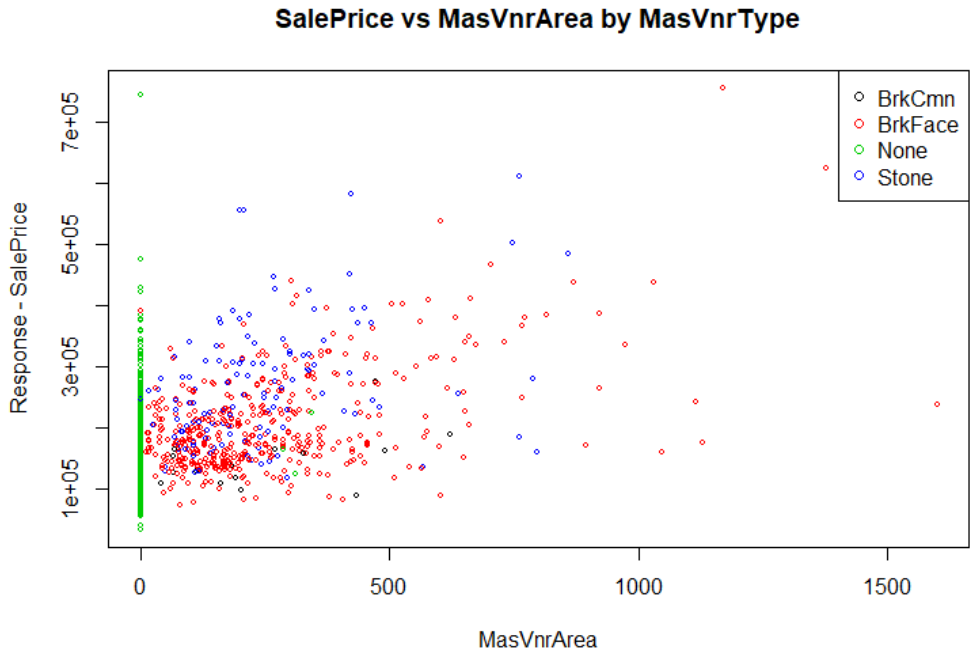


Figure 7: Plotting SalePrice on Masonry Veneer Area by Masonry Veneer type

Because different Masonry Veneer Types have different values, we decided to go with an interaction of the terms MasVnrArea and MasVnrType. In Figure 9, we plot the nonlinear relationships between the response and the interaction.

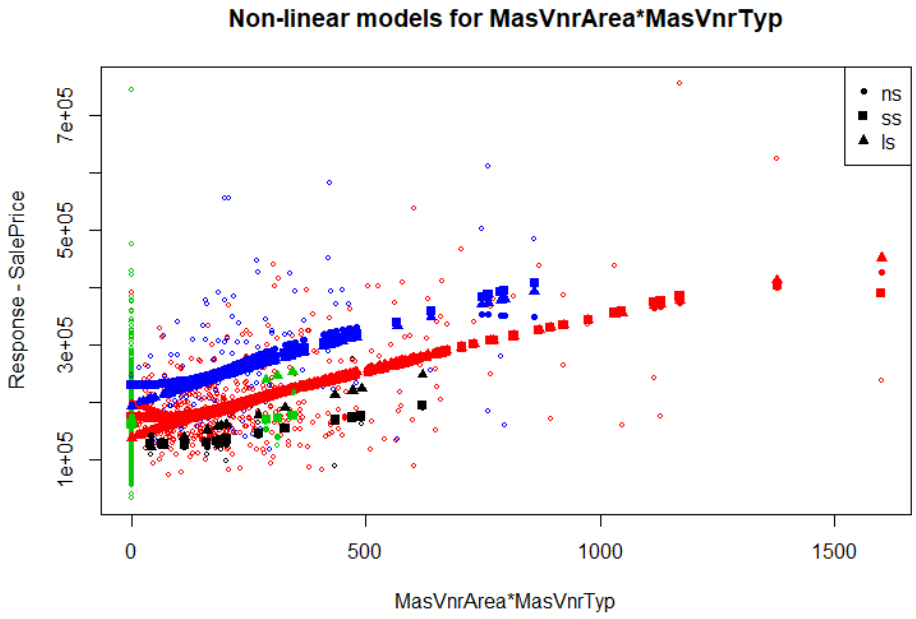


Figure 8: Non-linear models for MasVnrArea and Type. Selected: Smooth spline, degree 5 with interaction

4. 1STFLRSF

The next variable is footage of first floor. This is a quantitative variable which usually is given the most importance while deciding on the price for a house, so naturally this was bound to end up on our shortlist of variables. This variable is expected to follow a very simple and intuitive pattern, wherein the response i.e. SalePrice increases with increase in the square footage of the first floor. We also have other variables such as basement square footage, second floor square footage, but these values are not necessarily the primary column. The other values might be less than or equal to the square footage of first floor, so we decided to treat this variable independently. We do expect some outliers while exploring this variable, so our goal is to obtain a function which overlooks the outliers and is able to capture the essence of the underlying relationship by using most of the perfect data points. We consider different iterations of natural and smooth splines along with local regression with different values of span. The relationships are plotted below in figure 19.

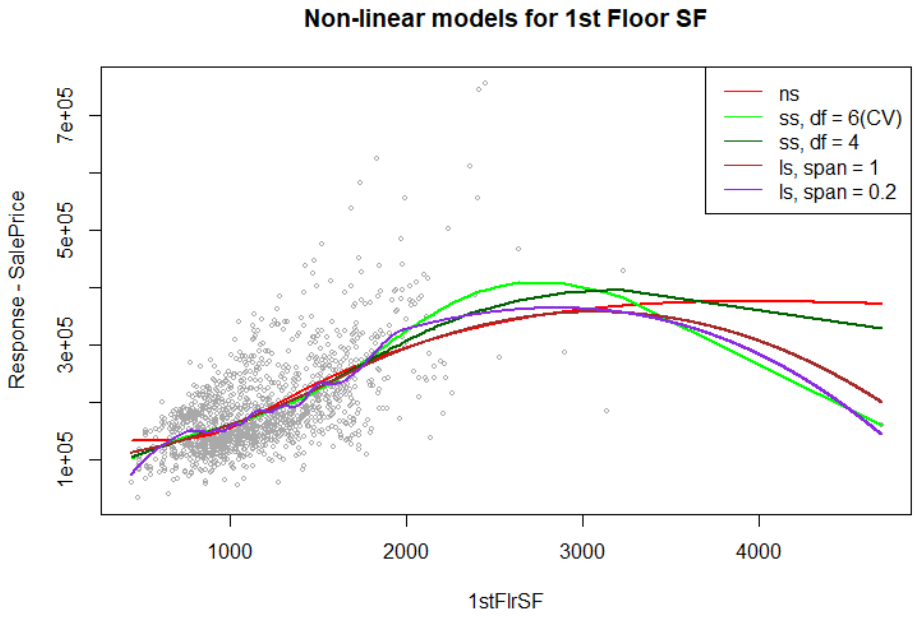


Figure 9: Non-linear models for 1stFlrSF.

We choose the model which models with the help of natural splines with degree 3. All the other models overfit on the training data and predict that the Sale Price will go down after a point with increase in the square footage, which is something we want to avoid. So, we choose the model which increases in the beginning and stabilizes at a value for higher values of 1st floor square footage.

5. 2NDFLRSF

This variable refers to the square footage of second floor of the house. This variable is expected to perform in a similar fashion as 1stflrsf variable we discussed above. One important thing we observed here was that a lot of the training values were 0, and this could possibly throw our model off while training. In order to further use this variable, we observe that an interaction with the variable MSSubClass perfectly explains the 0 values. Below is a plot of the 2nd floor square footage values in conjunction with the variable MSSubClass, which stands for the type of dwelling being sold.

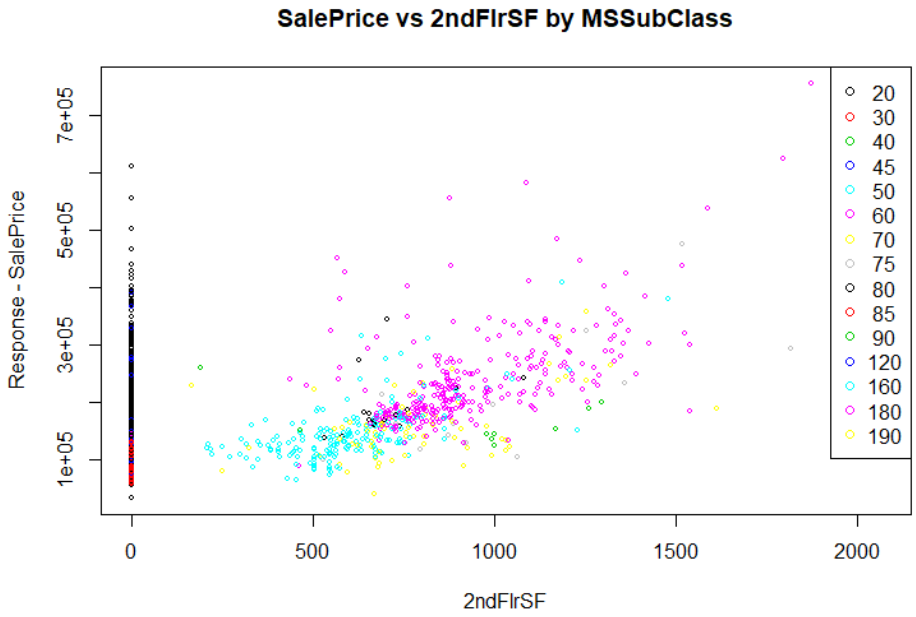


Figure 10: Plotting SalePrice on 2nd Floor SF by MS Sub Class

So here we will again explore non-linear relationship on this variable while considering an interaction.

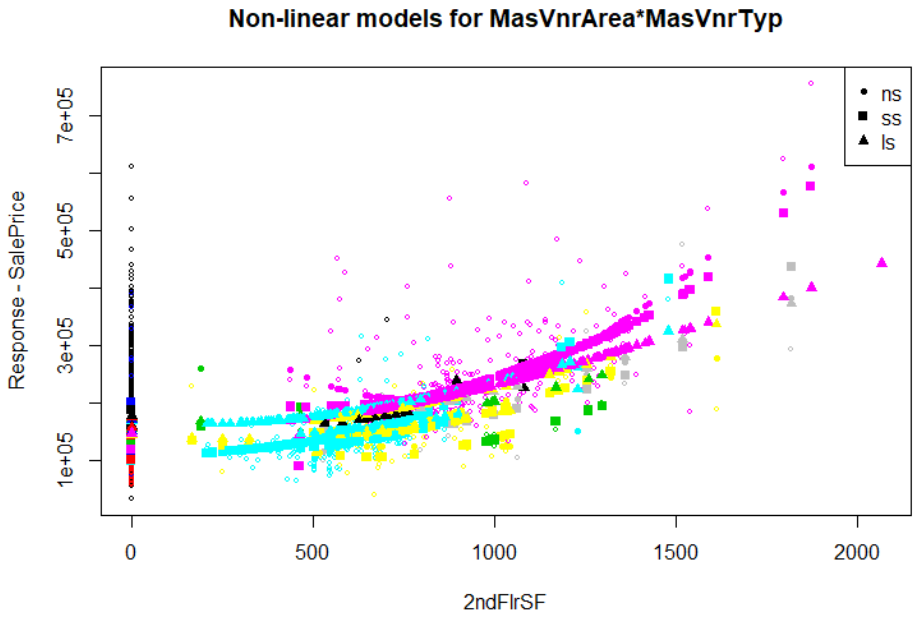


Figure 11: Non-linear models for 2ndFlrSF. Selected: Smooth spline, degree 5 with interaction

Here we again base the selection on the model which gives a smooth and intuitive prediction. The smooth spline fits our requirements perfectly and hence we finally choose that model to predict the square footage of second floor.

6. GARAGEAREA

The final quantitative variable we have with us is Garage Area. We consider an interaction with this variable as well, because depending on different types of garage, the garage area adds different amount of value to the house. The garagetype variable has values to indicate if the garage is attached, in basement, built in, more than 1, etc. This information combined with GarageArea can have a significant impact on our predictor. We first plot the data by garage types and then try to model the relationship with the predictor.

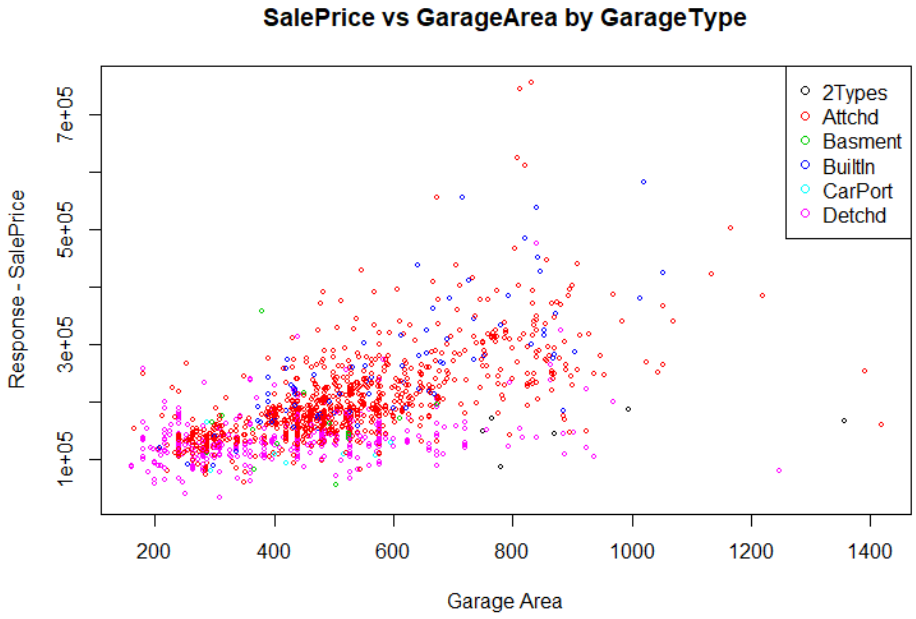


Figure 12: Plotting SalePrice on GarageArea by Garagetype

The non-linear relationships plotted on this interaction are shown in figure 14 below. Much of the training dataset consists of garagetype attached, so we need a non-linear curve that definitely gives a smooth fit.

In the figure we do not see much of a difference between different non-linear methods, so we go ahead with smooth spline with 4 degrees of freedom and with an interaction.

Qualitative Variables

In our reduced dataset also, we have 15 qualitative variables. Non-linear modelling only applies to quantitative data. For qualitative data, we can supply them as arguments, and an additive model is created while keeping their information in mind. The qualitative columns are as follows:

1. LandSlope 2. OverallQual 3. RoofMatl 4. MSSubClass 5. MasVnrType 6. ExterQual

7. BsmtQual 8. FullBath 9. BedroomAbvGr 10. KitchenAbvGr 11. KitchenQual

12. Fireplaces 13. GarageType 14. GarageCars 15. OverallCond

From these variables, we are using MSSubClass, MasVnrType and GarageType as interaction variables with our quantitative variables. For the remaining variables, we first plot them individually with the response variable and see if there is enough value being added by their inclusion.

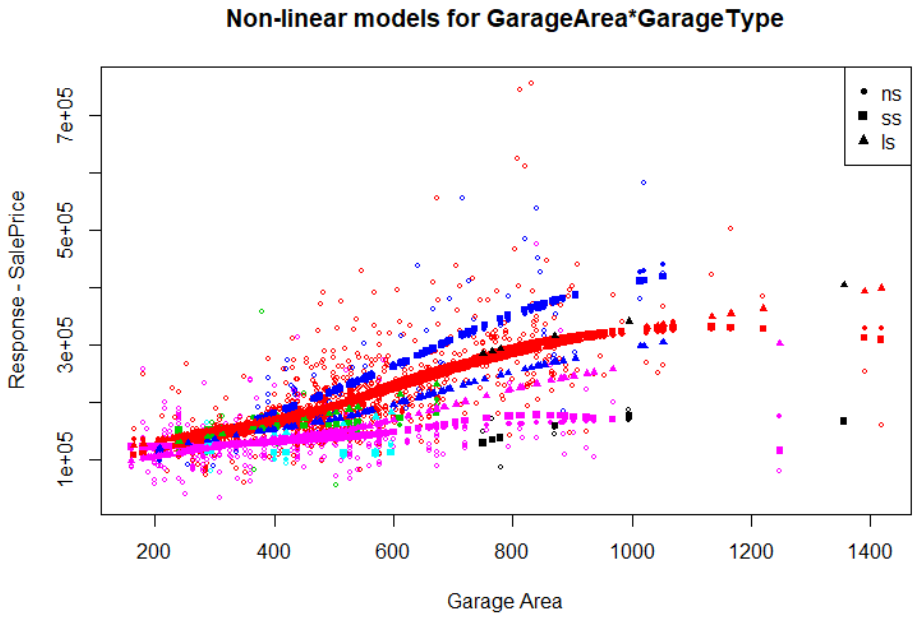
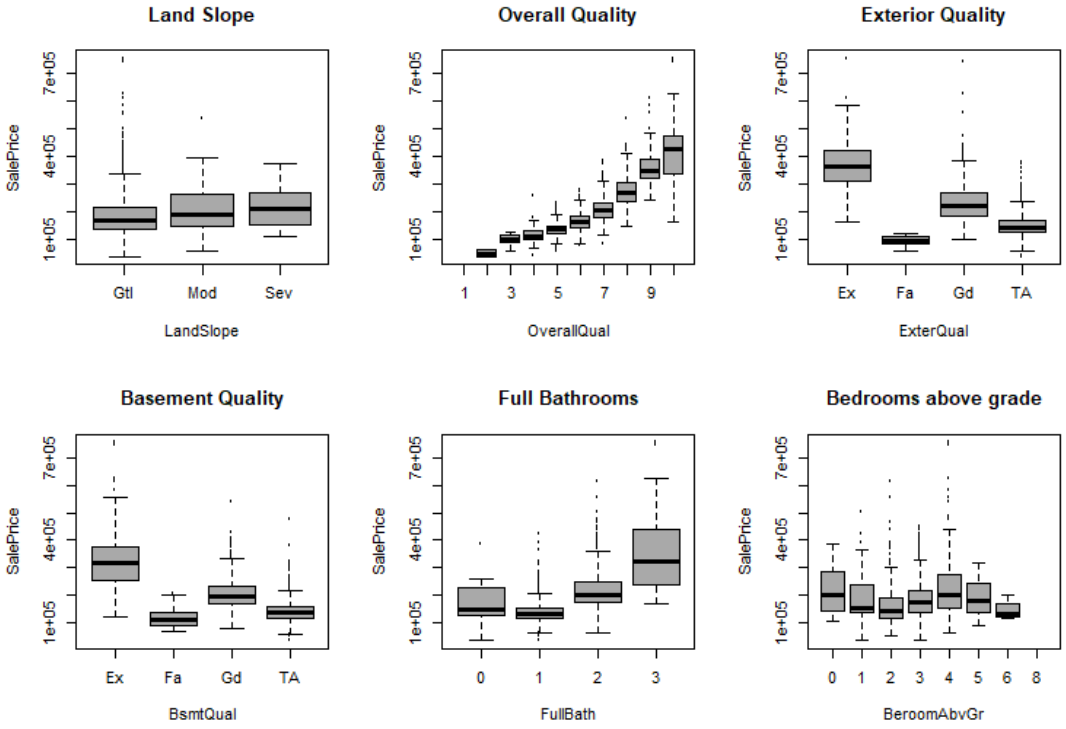


Figure 13: Non-linear models for GarageArea. Selected: Smooth spline, degree 4 with interaction

While performing basic plots for the qualitative (factor) variables, we find that there are some variables who do not have sufficient distribution across factors and hence if we build a model, we will not gain any particular advantage by including them. These variables are RoofMatl and KitchenAbvGr. Because of this reason we remove these variables from further analysis. Below is the qualitative relationship with the response.



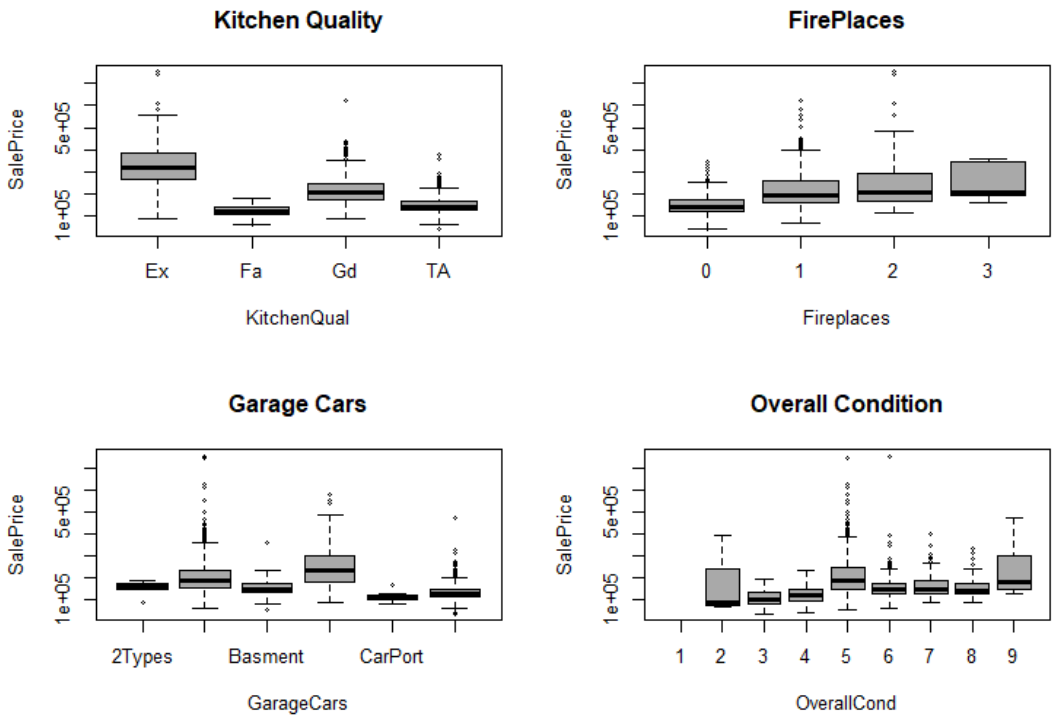


Figure 14 & 15: BoxPlot for the qualitative variables with the response column.

The final GAM model fit on our data looks as below:

```
FinalGAMFit <- gam(SalePrice ~ lo(LotArea, span = 1) + LandSlope + OverallQual + s(YearBuilt, df = 14.16361) + s(MasVnrArea,5)*MasVnrType + ExterQual + BsmtQual + ns(X1stFlrSF, df = 3) + s(X2ndFlrSF,5)*MSSubClass + FullBath + BedroomAbvGr + KitchenQual + Fireplaces + s(GarageArea,4)*GarageType + GarageCars + OverallCond, data = FinalTrain)
```

Upon training the above model, we again generate a few standard error metrics, to measure and compare the performance of this method with other methods that we have taken into consideration. We first look at training error. The mean squared training error on full data is *700,066,002*.

The next step we perform is validation set error testing. We divide our data into a 70:30 split of rows. We train on the 70% data and test our error on the remaining 30% hold out data. The training error in this case now was *740,542,861*. The test error in this case was *858,939,192*. We also applied 10-fold cross validation on the above formula. The cross-validation error calculated was *1,184,401,598*.

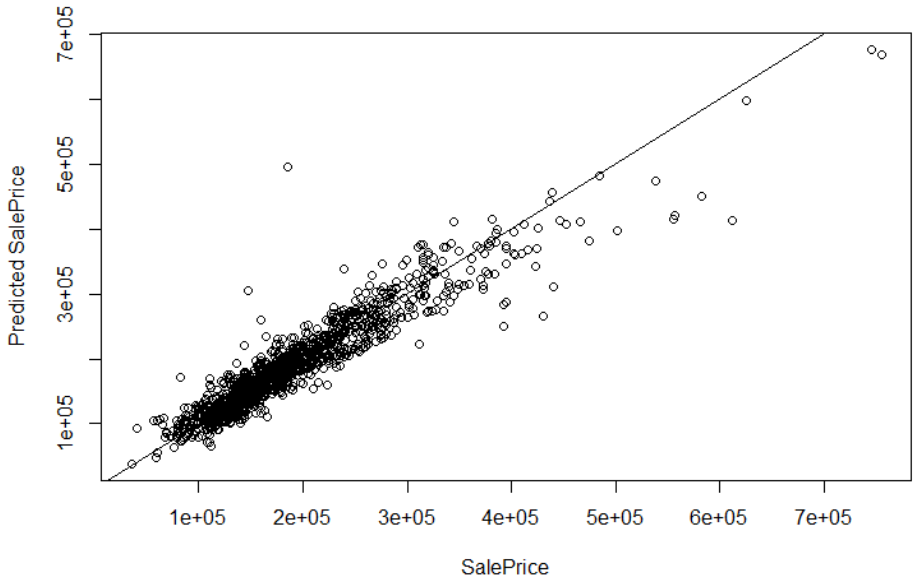


Figure 16: Response in the dataset vs predicted response

Methodoogy	Generalized Additive Models
Training Error	700,066,002
Validation Test Error	858,939,192
Cross Validation Error	1,184,401,598

Decision Trees

The next technique we apply to our dataset includes the use of decision trees algorithm. Decision trees recognizes the boundaries generated by the variables in order to predict the response. The output of this algorithm is a set of rules for the test dataset to follow in order to generate it's prediction. We do not need to restrict the number of variables for this algorithm, however just for comparison sake, we will run the codes on both the complete as well as the reduced dataset. Decision trees are often generalized in the terms of usage of variables, and hence using the complete dataset is possible in this algorithm.

We will explore all the techniques associated with Decision trees, such as Bagging, RandomForest as well as Boosting. The results are better than the accuracy obtained using splines or Lasso algorithm. Here we showcase the flow of application of this algorithm and our learnings along the way.

Classical Decision Tree

To compare the accuracy of decision trees with the methods previously explored in this document, we run this algorithm on the reduced as well as the complete dataset.

REDUCED DATASET

Upon running the decision tree on the reduced dataset, we obtain the first optimal tree as below:

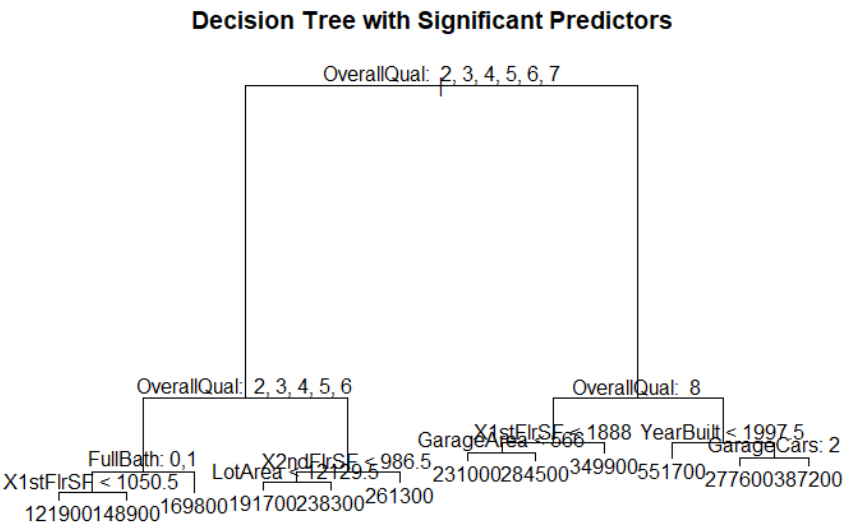


Figure 17: Decision tree with the reduced predictors and leaf nodes = 12

This tree is easy to interpret, but currently has a lot of leaf nodes making it a bit complicated to use and it also tends to overfit. In order to determine the optimal number of leaves, we use cross validation. We use Cross validation on this tree to look at the deviance values for varying size of the tree. Below is the plot for the size of the tree and cross validation error.

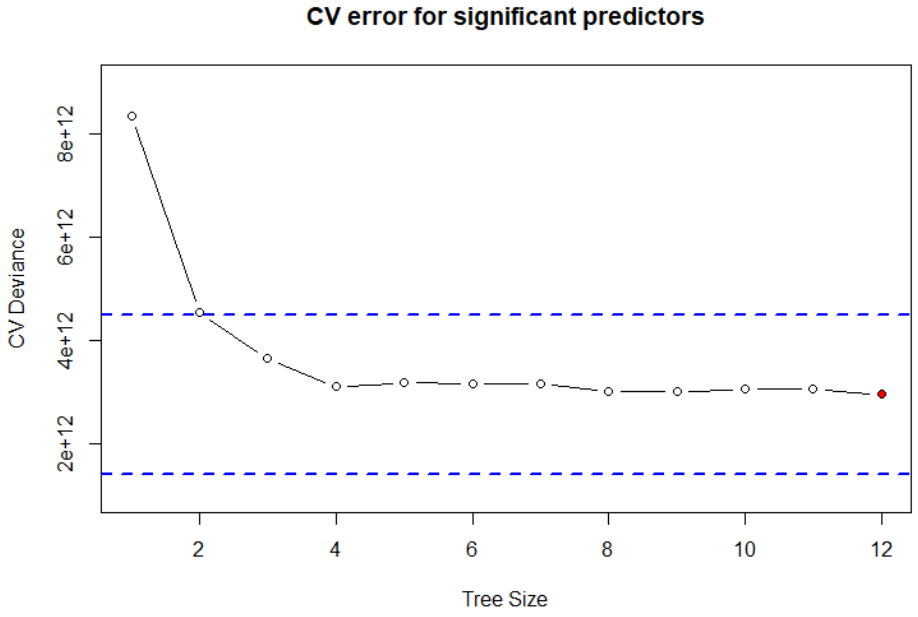


Figure 18: Cross Validation errors according to the size of the tree on significant predictors

We have previously learnt that it is better to have a simpler model with higher training error, rather than choosing a model with higher complexity and lower bias which might overfit on the training data. In the above case, we follow the 1 standard deviation rule. The blue lines represent the standard deviation for the deviance values. We can observe that there is not much of a difference between the deviance of the tree with 12 leaf nodes, the tree with 4 leaf nodes and the tree with 8 leaf nodes. We do not want to choose the tree with 4 leaf nodes because, the tree in that case will be very simplistic, and by choosing that tree, we are confirming that most of the variables made available to us have no use whatsoever. Thus, in this case we prune the tree to having 8 leaf nodes, and then predict using the tree. The pruned tree looks as below:

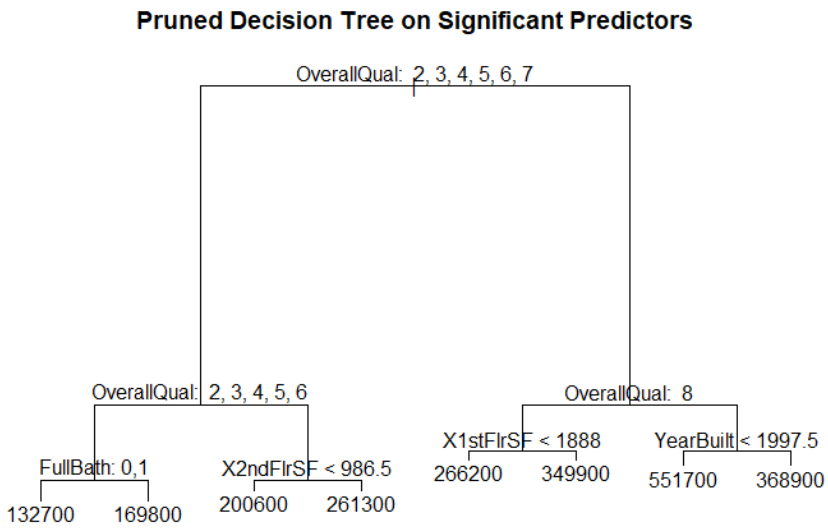


Figure 19: Decision tree with the reduced predictors and leaf nodes = 8

The training error when we predict using the same data is equal to *1,793,892,669*. We also calculated the cross-validation error after distributing this dataset into 10 different parts. The cross-validation error we got is equal to *2,414,375,872*.

Next, we aim to calculate the error using the validation set methodology. For this we will separate out 70% of the rows similar to what we did for Splines and Generalized additive models. Train the data on 70% of the rows and evaluate the performance of the algorithm and the best pruned tree on the remaining 30% of the data. The error obtained by using the validation set method is *2,165,425,664*. We wanted to point out that only the size of the tree was kept constant, whereas the variables and their influence have changed because of a new set of training rows.

COMPLETE DATASET

Upon running the decision tree for all the available variables, we find the below first optimal tree.

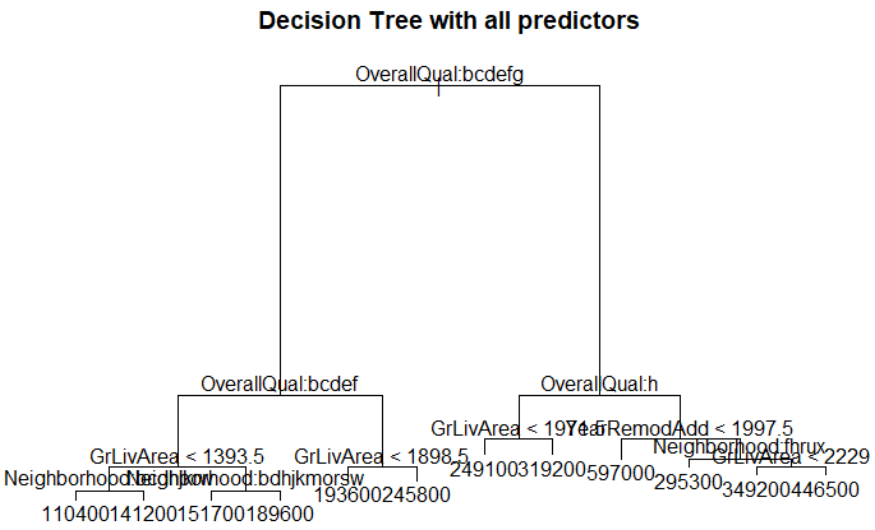


Figure 20: Decision tree with all predictors and leaf nodes = 12

We again use Cross-Validation to figure out the optimal tree size. Below we plot the tree size vs Cross-Validation deviance values.

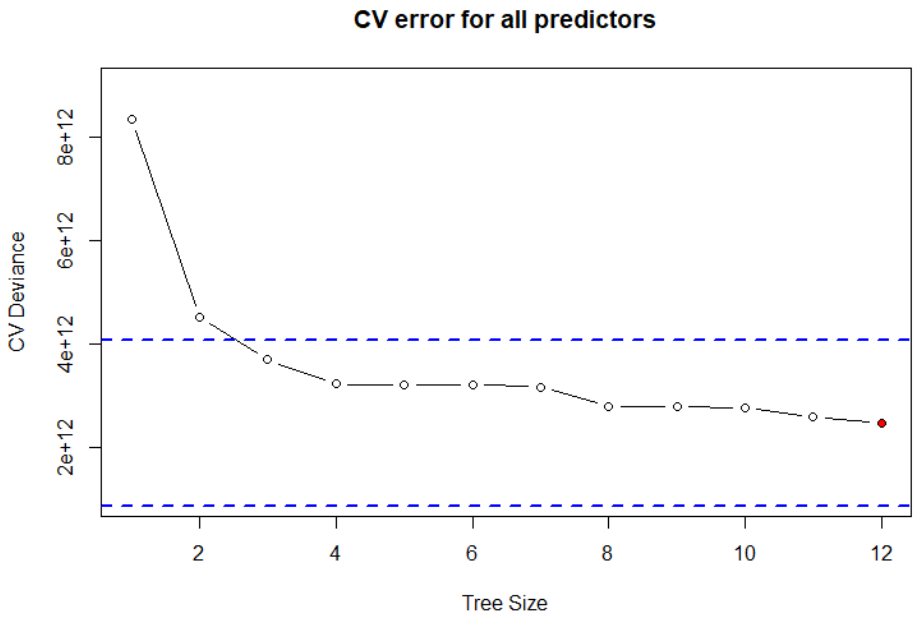


Figure 21: Cross Validation errors according to the size of the tree on all predictors

For this case again, we choose the decision tree with 8 leaf nodes, because we want to trade the interpretability with prediction accuracy. So, we prune the tree to 8 leaf nodes, and the pruned tree looks like below:

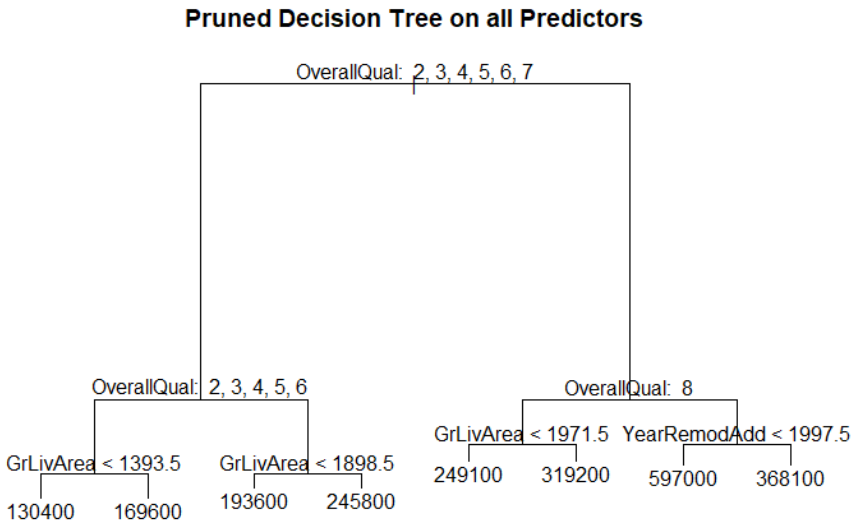


Figure 22: Decision tree with the all predictors and leaf nodes = 8

The training error on the pruned decision tree for all the predictors is *1,628,906,465*. The cross-validation error for the tree size 8 on all predictors is *2,069,860,181*. We applied the validation set methodology on the tree with all predictors as well and the test error obtained is *1,961,010,863*.

Methodology	**Significant Predictors**	**All Predictors**
Training error	1,793,892,669	1,628,906,465
Validation Set	2,165,425,664	1,961,010,863
Cross-Validation	2,414,375,872	2,069,860,181

By looking at the dataset above we can see that with regards to decision trees, building it using all the predictors is better than limiting the number of variables in the first place.

Bagging Decision Trees

We noticed above the predictive power of a single decision tree. Bagging technique basically combines and averages out the decision tree obtained by using bootstrap to select the dataset for each iteration. So, for this method, we generate a new dataset for each iteration, fit a decision tree on the new data and then for prediction use a combined version of this decision tree. Because of the combination of a lot of decision trees, some interpretation power is lost, but more accurate predictions are obtained. In Bagging we include all the variables while building the clones of the decision tree. Continuing the trend above, we will apply it on both the reduced as well as the dataset with all the predictors.

REDUCED DATASET

The training error obtained after applying bagging functionality on our decision tree is *171,024,871*. This value is very less than what we have obtained below, but this is also the training error, which is expected to be low in value. We do not verify the results of bagging with cross-validation, because choosing different number of rows for each model building replicates the methodology of Cross-Validation. Although we do apply the validation set methodology, and the test error obtained in that case is *787,851,033*. The 10-fold cross

validation error for bagging on reduced dataset is *1,057,711,165*. The predicted SalePrice vs the original SalePrice is shown below:

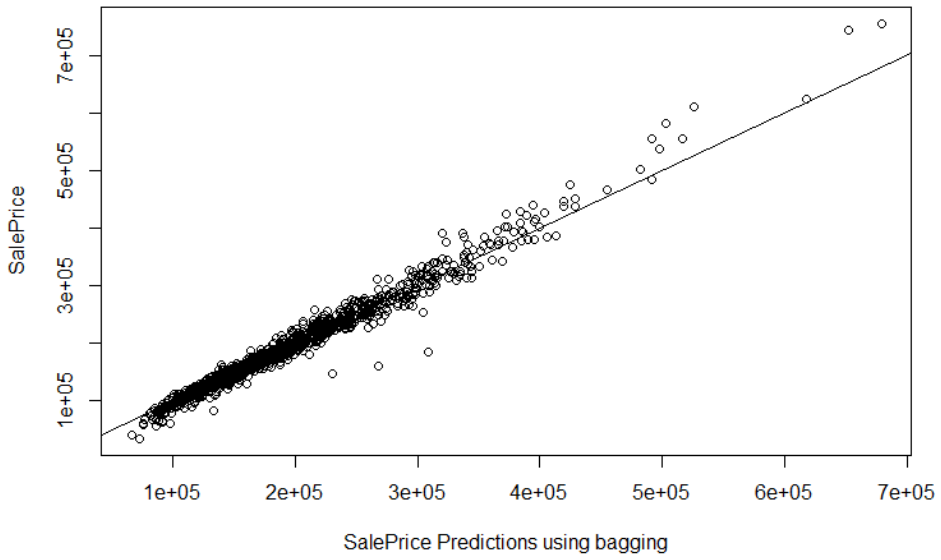


Figure 23: SalePrice vs Predicted SalePrice for bagging on significant predictors

COMPLETE DATASET

Next, we apply bagging to our decision tree built on the complete dataset. The training error in this case is *137,083,663*. The test error obtained by passing the bagged decision tree with all variables through the validation set method is *684,342,897*. The plot of predicted SalePrice after bagging on the dataset with all predictors vs the SalePrice in the test dataset is very similar to the plot shown above. The cross validation error for bagging on complete dataset is *902,190,216*.

Random Forests

Random forest is an improvement on the bagging methodology. The difference between these 2 techniques is that random forest not only selects different rows to build a decision tree, but also selects different number of reduced columns. This makes the output tree further independent of the training data, and if the number of trees is high then we can get a good approximation of the prediction. The typical number of columns to select for each iteration is equal to square root of number of predictors or a third of the number of predictors. We will illustrate the use of both the different numbers of predictors selected.

REDUCED DATASET

We first apply random forest to the dataset with significant predictors. The training error obtained when mtry = *5* (sqrt(21)) is *205,753,737*. The training error for the random forest model with mtry = *7* (21/3) is *186,230,644*. In case of RandomForest we can look at the importance of each variables considered in the modelling. The first metric is %IncMSE which is equal to the increase in MSE if the said variable is not included in building the decision tree in the random forest. The second metric is IncNodePurity which means that how much node purity is increased by involving the said variable. The Variable importance plot for the dataset with reduced predictors is as below:

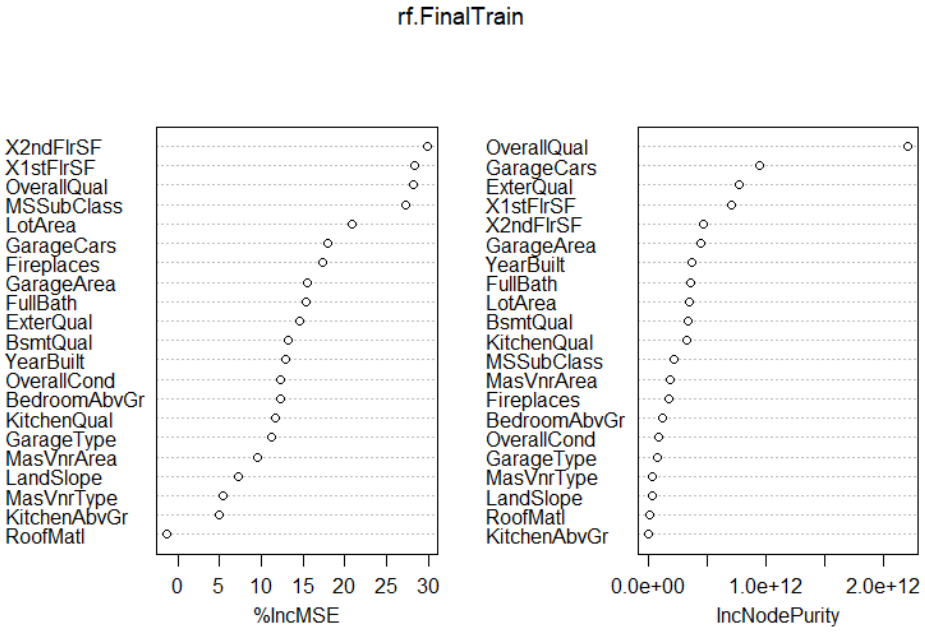


Figure 24: Variable Importance plot for reduced predictors

Here we can see that the square footage of first and second floor are really important variables. The test error calculated by dividing the dataset into training and testing dataset is *683,053,956* when mtry = *5* and *681,696,365* when mtry = *7*. We also calculated the 10-fold cross validation error for using random forest with mtry = *7*. The mean error in that case is *1,016,217,302*.

COMPLETE DATASET

We again apply RandomForest algorithm to our complete dataset. By plotting the variable importance plot, we can also verify if the initial selection of significant variables was correct or not. The training error obtained for mtry = 25 (73/3) is *130,125,800* and for mtry = 9 (sqrt(73)) is *158,442,607*. The variable importance plot for all predictors is as below in figure 25. We can observe that GrLivingArea which captures the information about area of the establishment and Neighborhood are significant predictors to predict the Sale Price of a house. The test error obtained after dividing the data into train and test data is *616,505,553* for mtry = 25 and is *660,517,956* for mtry = 9. A surprising finding was Garage Cars. Including them in our model leads to a decent increase in node purity. The 10-fold cross validation error was also within range at *825,501,021*.

Gradient Boosting Decision Trees

Boosting decision trees is a similar approach to random forests, where we select a different number of rows and columns to build multiple decision trees. But in the case of boosting, the selection process is not random. It is incremental in nature, such that each new decision tree tries to model the residual left behind by the previous tree. Gradient boosting decision trees also involve the usage of shrinkage factor, which measures the rate of growth towards the response values. Each decision tree tries to model the residual while trying to reach the response value, but after the modelling is done, it is toned down by the shrinkage factor, so that the process slowly learns and adapts for maximum prediction accuracy.

REDUCED DATASET

We first apply boosting trees to our reduced dataset. Boosting produces a plot for relative influence of each variables. The relative influence in this case can be observed in figure 26. The training error for a boosted tree on the dataset with reduced predictors is *6,889,754*. We should not be influenced by such a small training error, because even if we use the validation set approach, we get the test error as *1,449,635,414*. Thus, Gradient Boosting trees might be overfitting on the data. The Cross Validation error of *2,680,720,720* also points to the same.

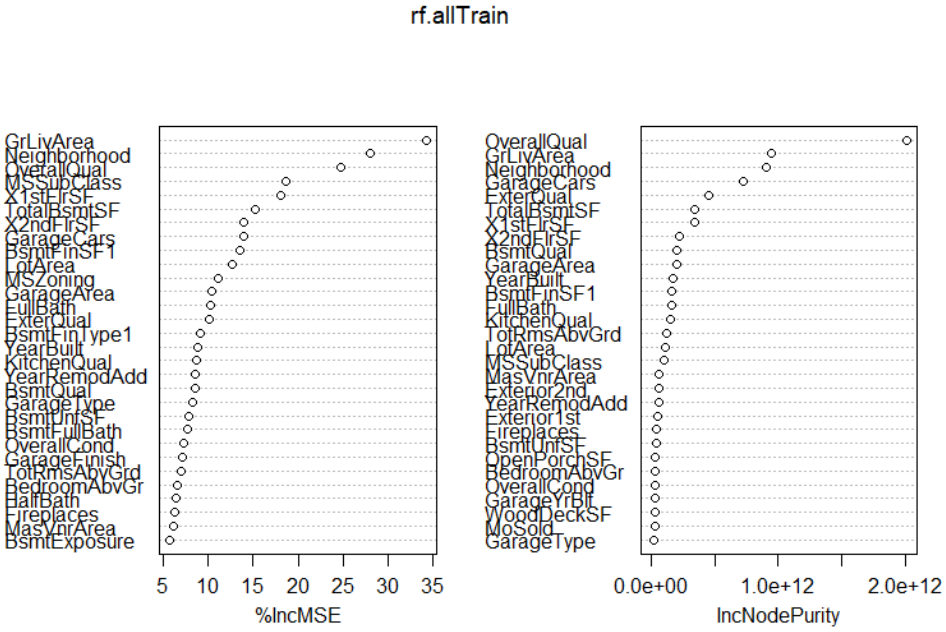


Figure 25: Variable Importance plot for all predictors

```
> summary(boost.FinalTrain)
                    var     rel.inf
OverallQual   OverallQual 35.69764258
X1stFlrSF       X1stFlrSF 14.74060580
LotArea           LotArea  6.82985557
GarageCars     GarageCars  4.51316403
MSSubClass     MSSubClass  4.51036409
X2ndFlrSF       X2ndFlrSF  4.49767281
GarageArea     GarageArea  4.44602923
FullBath         FullBath  4.13091792
MasVnrArea     MasVnrArea  3.60911563
BedroomAbvGr BedroomAbvGr  3.26373701
YearBuilt       YearBuilt  3.17374332
BsmtQual         BsmtQual  2.19361384
ExterQual       ExterQual  1.71871847
KitchenQual   KitchenQual  1.60968849
OverallCond   OverallCond  1.57810972
Fireplaces     Fireplaces  1.53380830
GarageType     GarageType  0.80457177
MasVnrType     MasVnrType  0.45360120
LandSlope       LandSlope  0.39360200
RoofMatl         RoofMatl  0.24088645
KitchenAbvGr KitchenAbvGr  0.06055174
```

Figure 26: Relative influence of reduced predictors by boosted trees

COMPLETE DATASET

Next, we use boosted trees to predict for the entire dataset. The training error in this case is *404,101.2*. This number is also similarly small as we found previously, but similar to the last case when we put this data through validation set method, the test error obtained is *1,383,488,104*. The 10-fold cross validation error for gradient boosted trees is *2,206,554,073*. The values with relative influence with all variables is shown below:

```
> summary(boost.AllTrain)
                               var      rel.inf
OverallQual         OverallQual 3.349495e+01
GrLivArea             GrLivArea 1.512929e+01
Neighborhood       Neighborhood 1.279340e+01
TotalBsmtSF         TotalBsmtSF 5.139476e+00
X1stFlrSF             X1stFlrSF 3.637836e+00
BsmtFinSF1           BsmtFinSF1 3.621845e+00
GarageCars           GarageCars 3.533636e+00
BsmtQual               BsmtQual 2.039720e+00
LotArea                 LotArea 1.912788e+00
MSSubClass           MSSubClass 1.467279e+00
X2ndFlrSF             X2ndFlrSF 1.219457e+00
KitchenQual         KitchenQual 1.104001e+00
FullBath               FullBath 9.759511e-01
GarageArea           GarageArea 9.030120e-01
Exterior2nd         Exterior2nd 8.387198e-01
BsmtExposure       BsmtExposure 7.661903e-01
MasVnrArea           MasVnrArea 7.320783e-01
SaleCondition     SaleCondition 7.280069e-01
TotRmsAbvGrd       TotRmsAbvGrd 7.253795e-01
GarageFinish       GarageFinish 7.251603e-01
OverallCond         OverallCond 7.103859e-01
BsmtUnfSF             BsmtUnfSF 6.415963e-01
```

Figure 26: Relative influence of reduced predictors by boosted trees

Methodology	Reduced Predictors			
	Decision Tree	Bagging	Random Forest	Boosting Tree
Training Error	1,793,892,669	171,024,871	186,230,644	6,889,754
Validation Test Error	2,165,425,664	787,851,033	681,696,365	1,449,635,414
Cross Validation Error	2,414,375,872	1,057,711,165	1,016,217,302	2,680,720,720

Methodology	All Predictors			
	Decision Tree	Bagging	Random Forest	Boosting Tree
Training Error	1,628,906,465	137,083,663	130,125,800	404,101
Validation Test Error	1,961,010,863	684,342,897	616,505,553	1,383,488,104

Cross Validation Error	2,069,860,181	902,190,216	825,501,021	2,206,554,073

Summary

The purpose of this Kaggle exercise was to predict the house prices for the test dataset, but because we did not have the response values for the test dataset, we could not post the actual error estimates here. But as discussed above, we do have training error, Validation set test error on a 70:30 constant split across the data and the 10-fold cross-validation error for all the above methods. We have collected all the information together and presented it below here.

We first present the results for the reduced dataset, because that dataset is constant throughout all the methodologies.

Reduced Predictors (1338 rows and 22 Columns)							
Methodology	Ridge	Lasso	GAM	DT	Bagging	RF	Boosting
Train Error	887 M	901 M	700 M	1,793 M	171 M	186 M	6 M
Validation Test Error	810 M	957 M	858 M	2,165 M	787 M	681 M	1,449 M
Cross-Validation Error	1,185 M	1,172 M	1,184 M	2,414 M	1,057 M	1,016 M	2,680 M

Similarly, if we compare the results for all the predictors, the values are shown as here.

All Predictors (1338 rows and 74 Columns)							
Methodology	Ridge	Lasso	GAM	DT	Bagging	RF	Boosting
Train Error	749 M	707 M	700 M	1,628 M	137 M	130 M	0.404 M
Validation Test Error	685 M	651 M	858 M	1,961 M	684 M	616 M	1,383 M
Cross-Validation Error	1,227 M	1,271 M	1,184 M	2,069 M	902 M	825 M	2,206 M

Because the numbers are too large and difficult to comprehend, we have scaled the values down using the scale function. The updated error estimates are:

Reduced Predictors (1338 rows and 22 Columns)

Methodology	Ridge	Lasso	GAM	DT	Bagging	RF	Boosting
Train	-0.23	-0.21	-0.52	1.17	-1.34	-1.32	-1.59
Validation Test Error	-0.35	-0.13	-0.28	1.74	-0.39	-0.55	0.63
Cross-Validation Error	0.22	0.21	0.22	2.12	0.03	-0.04	2.53

All Predictors (1338 rows and 74 Columns)

Methodology	Ridge	Lasso	GAM	DT	Bagging	RF	Boosting
Train	-0.45	-0.51	-0.52	0.91	-1.39	-1.40	-1.60
Validation Test Error	-0.55	-0.60	-0.28	1.42	-0.55	-0.65	0.53
Cross-Validation Error	0.29	0.36	0.22	1.59	-0.21	-0.33	1.80