Harvard Capstone Project - House Price Prediction

Leandro Rodrigues Carvalho

07/03/2021

Introduction

This is an R Markdown project aiming to develop a model to predict house selling prices, developed as part of Harvard University online Professional Certificate course on Data Science, lead by Professor Rafael A. Irizarry.

The *goal* of this project is to predict homes final selling prices using a data set from Kaggle's educational competition "House Prices - Advanced Regression Techniques". The data set includes 79 explanatory variables of residential homes in Ames, Iowa, USA.

The *key steps* that were performed included Exploratory Data Analysis (EDA), data visualization, data wrangling and training the machine learning algorithm using the inputs in one subset to predict selling prices in the test set.

Acknowledgments

(https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview)

"The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset"

The original study can be found on the link below, where Professor Dean De Cock, from Truman State University, explain the process of accessing the data and preparing it for further use.

http://jse.amstat.org/v19n3/decock.pdf

Data & Key Questions

Every home buyer asks some of the basic questions in relation to a property to asses if the sales price is fair. When was it built? How big is the lot? How many square feet of living space is in the dwelling? Is the basement finished? How many bathrooms are there? The variables contained in this data set contain the information that a typical home buyer would want to know when buying a house.

As mentioned initially, the data set describes the sale of individual residential property in Ames, Iowa, from 2006 to 2010, and contains 2930 observations and a large number of explanatory variables (23 nominal, 23 ordinal, 14 discrete, and 20 continuous) involved in assessing home values.

Missing packages are automatically installed with external link

This code creates a training set and a validation, or test, set (final hold-out test set)

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(gbm)) install.packages("gbm", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
```

Loading packages

```
library(tidyverse)
library(caret)
library(data.table)
library(randomForest)
library(gbm)
library(rpart)
library(knitr)
```

Retrieving train and test sets

```
df <- read.csv("/Users/leandrocarvalho/Documents/Education/HarvardX - Data Science/HarvardX9 - Capstone

row.names(df) <- df$Id
df <- df[,-1]
df[is.na(df)] <- 0
for(i in colnames(df[,sapply(df, is.character)])){
  df[,i] <- as.factor(df[,i])
}

### Test values
test.n <- sample(1:nrow(df), nrow(df)/3, replace = F)

### Test dataset
test <- df[test.n,]

### Train dataset
train <- df[-test.n,]

rm(test.n, df)

### Combined Data Frame
combined.df <- rbind(test, train)
```

Method and Analyses

Exploring multiple statistical methods

We used several packages to explore multiple statistical methods, computing the RMSE for the trained models: Linear Regression, Regression Tree, Random Forest and Gradient Boosting Machine.

Understanding of the data set

The combined Data Set is made of 80 variables and 1460 observations, including "Sales Price".

```
glimpse(combined.df)
```

```
## Rows: 1,460
## Columns: 80
## $ MSSubClass    <int> 70, 20, 120, 20, 60, 50, 20, 20, 20, 20, 85, 20, 60, ...
## $ MSZoning      <fct> RL, RL, RL, RL, RL, RL, RL, RL, RL, RL, RL, RL, RL, R...
## $ LotFrontage   <dbl> 60, 75, 43, 103, 92, 0, 59, 60, 0, 182, 54, 79, 75, 6...
## $ LotArea       <int> 9550, 8250, 3182, 13472, 11764, 11250, 10593, 6600, 8...
## $ Street        <fct> Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pave,...
## $ Alley         <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ LotShape      <fct> IR1, Reg, Reg, Reg, IR1, Reg, IR1, Reg, IR1, IR3, Reg...
## $ LandContour   <fct> Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl...
## $ Utilities     <fct> AllPub, AllPub, AllPub, AllPub, AllPub, AllPub, AllPu...
## $ LotConfig     <fct> Corner, Inside, Inside, Inside, CulDSac, Inside, Insi...
## $ LandSlope     <fct> Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl...
## $ Neighborhood  <fct> Crawfor, NAmes, Blmngtn, NridgHt, NoRidge, ClearCr, N...
## $ Condition1    <fct> Norm, Norm, Norm, Norm, Norm, Norm, Norm, Norm, Norm,...
## $ Condition2    <fct> Norm, Norm, Norm, Norm, Norm, Norm, Norm, Norm, Norm,...
## $ BldgType      <fct> 1Fam, 1Fam, TwnhsE, 1Fam, 1Fam, 1Fam, 1Fam, 1Fam, 1Fa...
## $ HouseStyle    <fct> 2Story, 1Story, 1Story, 1Story, 2Story, 1.5Fin, 1Stor...
## $ OverallQual   <int> 7, 6, 7, 10, 8, 4, 7, 5, 4, 7, 5, 8, 7, 5, 5, 5, 4, 7...
## $ OverallCond   <int> 5, 7, 5, 5, 7, 5, 5, 9, 3, 5, 7, 5, 6, 5, 5, 7, 7, 5,...
## $ YearBuilt     <int> 1915, 1964, 2005, 2003, 1999, 1957, 1996, 1982, 1956,...
## $ YearRemodAdd  <int> 1970, 1964, 2006, 2003, 2007, 1989, 1996, 2008, 1956,...
## $ RoofStyle     <fct> Gable, Hip, Gable, Hip, Gable, Gable, Hip, Gable, Gab...
## $ RoofMatl      <fct> CompShg, CompShg, CompShg, CompShg, CompShg, CompShg,...
## $ Exterior1st   <fct> Wd Sdng, HdBoard, VinylSd, VinylSd, VinylSd, Wd Sdng,...
## $ Exterior2nd   <fct> Wd Shng, HdBoard, VinylSd, VinylSd, VinylSd, Wd Sdng,...
## $ MasVnrType    <fct> None, Stone, BrkFace, BrkFace, BrkFace, None, BrkFace...
## $ MasVnrArea    <dbl> 0, 260, 14, 922, 348, 0, 338, 0, 0, 0, 0, 336, 0, 0, ...
## $ ExterQual     <fct> TA, TA, Gd, Ex, Gd, TA, Gd, Gd, TA, Gd, TA, Gd, TA, T...
## $ ExterCond     <fct> TA, TA, TA, TA, TA, TA, TA, Gd, TA, TA, Gd, TA, TA, F...
## $ Foundation    <fct> BrkTil, CBlock, PConc, PConc, PConc, CBlock, PConc, C...
## $ BsmtQual      <fct> TA, Gd, Gd, Ex, Gd, TA, Gd, TA, TA, Gd, Gd, Ex, Gd, T...
## $ BsmtCond      <fct> Gd, TA, Gd, TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, T...
## $ BsmtExposure  <fct> No, No, No, Gd, No, Av, No, No, No, Av, Av, Gd, Av, N...
## $ BsmtFinType1  <fct> ALQ, Rec, GLQ, GLQ, GLQ, Unf, GLQ, ALQ, Unf, GLQ, ALQ...
## $ BsmtFinSF1    <int> 216, 787, 16, 56, 524, 0, 919, 641, 0, 1300, 619, 128...
## $ BsmtFinType2  <fct> Unf, Unf, Unf, Unf, Unf, Unf, Unf, Unf, Unf, Unf, Unf...
## $ BsmtFinSF2    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ BsmtUnfSF     <int> 540, 305, 1330, 2336, 628, 1104, 801, 175, 672, 230, ...
## $ TotalBsmtSF   <int> 756, 1092, 1346, 2392, 1152, 1104, 1720, 816, 672, 15...
## $ Heating       <fct> GasA, GasA, GasA, GasA, GasA, GasA, GasA, GasA, GasA,...
## $ HeatingQC     <fct> Gd, Ex, Ex, Ex, Ex, Ex, Ex, Ex, Ex, Ex, Ex, Ex, Ex, E...
## $ CentralAir    <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y,...
## $ Electrical    <fct> SBrkr, SBrkr, SBrkr, SBrkr, SBrkr, FuseA, SBrkr, SBrk...
## $ X1stFlrSF     <int> 961, 1092, 1504, 2392, 1164, 1104, 1720, 816, 960, 15...
## $ X2ndFlrSF     <int> 756, 0, 0, 0, 1106, 684, 0, 0, 0, 0, 0, 0, 989, 0, 0,...
## $ LowQualFinSF  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 514, 0, 0, 0, ...
## $ GrLivArea     <int> 1717, 1092, 1504, 2392, 2270, 1788, 1720, 816, 960, 1...
## $ BsmtFullBath  <int> 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,...
## $ BsmtHalfBath  <int> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ FullBath      <int> 1, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, 2,...
## $ HalfBath      <int> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,...
## $ BedroomAbvGr  <int> 3, 3, 2, 3, 4, 5, 3, 3, 3, 3, 2, 3, 3, 4, 2, 3, 3, 3,...
## $ KitchenAbvGr  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ KitchenQual   <fct> Gd, TA, Gd, Ex, Gd, TA, Gd, Gd, TA, Gd, TA, Gd, Gd, T...
## $ TotRmsAbvGrd  <int> 7, 6, 7, 8, 9, 8, 7, 5, 5, 7, 5, 7, 8, 7, 5, 5, 6, 7,...
## $ Functional    <fct> Typ, Typ, Typ, Typ, Typ, Min2, Typ, Typ, Typ, Typ, Ty...
## $ Fireplaces    <int> 1, 0, 1, 1, 1, 2, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,...
## $ FireplaceQu   <fct> Gd, 0, Gd, Ex, Gd, TA, TA, Ex, 0, Gd, 0, Gd, TA, 0, 0...
## $ GarageType    <fct> Detchd, Attchd, Attchd, Attchd, Attchd, Attchd, Attch...
## $ GarageYrBlt   <dbl> 1998, 1964, 2005, 2003, 1999, 1957, 1996, 1982, 1956,...
## $ GarageFinish  <fct> Unf, RFn, Fin, Fin, Fin, Unf, Fin, Unf, Unf, Fin, Unf...
## $ GarageCars    <int> 3, 2, 2, 3, 3, 1, 2, 1, 1, 3, 2, 3, 2, 0, 2, 1, 2, 2,...
## $ GarageArea    <int> 642, 504, 437, 968, 671, 304, 527, 264, 288, 630, 624...
## $ GarageQual    <fct> TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, 0...
## $ GarageCond    <fct> TA, Gd, TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, TA, 0...
## $ PavedDrive    <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, P, Y, Y, Y, Y,...
## $ WoodDeckSF    <int> 0, 0, 156, 248, 132, 120, 240, 0, 64, 144, 104, 208, ...
## $ OpenPorchSF   <int> 35, 0, 20, 105, 57, 0, 56, 0, 0, 36, 0, 44, 170, 0, 0...
## $ EnclosedPorch <int> 272, 0, 0, 0, 0, 0, 154, 0, 0, 0, 0, 0, 0, 138, 0, 12...
## $ X3SsnPorch    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ ScreenPorch   <int> 0, 0, 0, 0, 0, 0, 0, 0, 160, 0, 0, 0, 0, 0, 0, 0, 168...
## $ PoolArea      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ PoolQC        <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ Fence         <fct> 0, MnPrv, 0, 0, 0, 0, 0, MnPrv, MnPrv, 0, 0, 0, 0, 0,...
## $ MiscFeature   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ MiscVal       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ MoSold        <int> 2, 7, 5, 6, 4, 11, 3, 10, 10, 11, 4, 9, 6, 6, 10, 7, ...
## $ YrSold        <int> 2006, 2008, 2008, 2009, 2010, 2009, 2010, 2008, 2009,...
## $ SaleType      <fct> WD, WD, WD, WD, WD, WD, WD, WD, COD, WD, WD, WD, WD, ...
## $ SaleCondition <fct> Abnorml, Normal, Normal, Normal, Normal, Normal, Norm...
## $ SalePrice     <int> 140000, 145000, 191000, 386250, 290000, 161500, 26040...
```

Checking for duplicates

Property ID

In this data set, each row is a different property. The PID is the Parcel Identification Number assigned to each property within the Ames Assessors system. Checking for duplicates in relation to unique properties we see that there is none.

```
sum(duplicated(combined.df))
```

```
## [1] 0
```

```
cat("The number of duplicated rows are", nrow(combined.df) - nrow(unique(combined.df)))
```

```
## The number of duplicated rows are 0
```

Missing Values

The percentage of data missing in train

```
sum(is.na(train)) / (nrow(train) *ncol(train))
```

```
## [1] 0
```

The percentage of data missing in test

```
sum(is.na(test)) / (nrow(test) * ncol(test))
```

```
## [1] 0
```

The percentage of data missing in combined data set

```
sum(is.na(combined.df)) / (nrow(combined.df) * ncol(combined.df))
```

```
## [1] 0
```

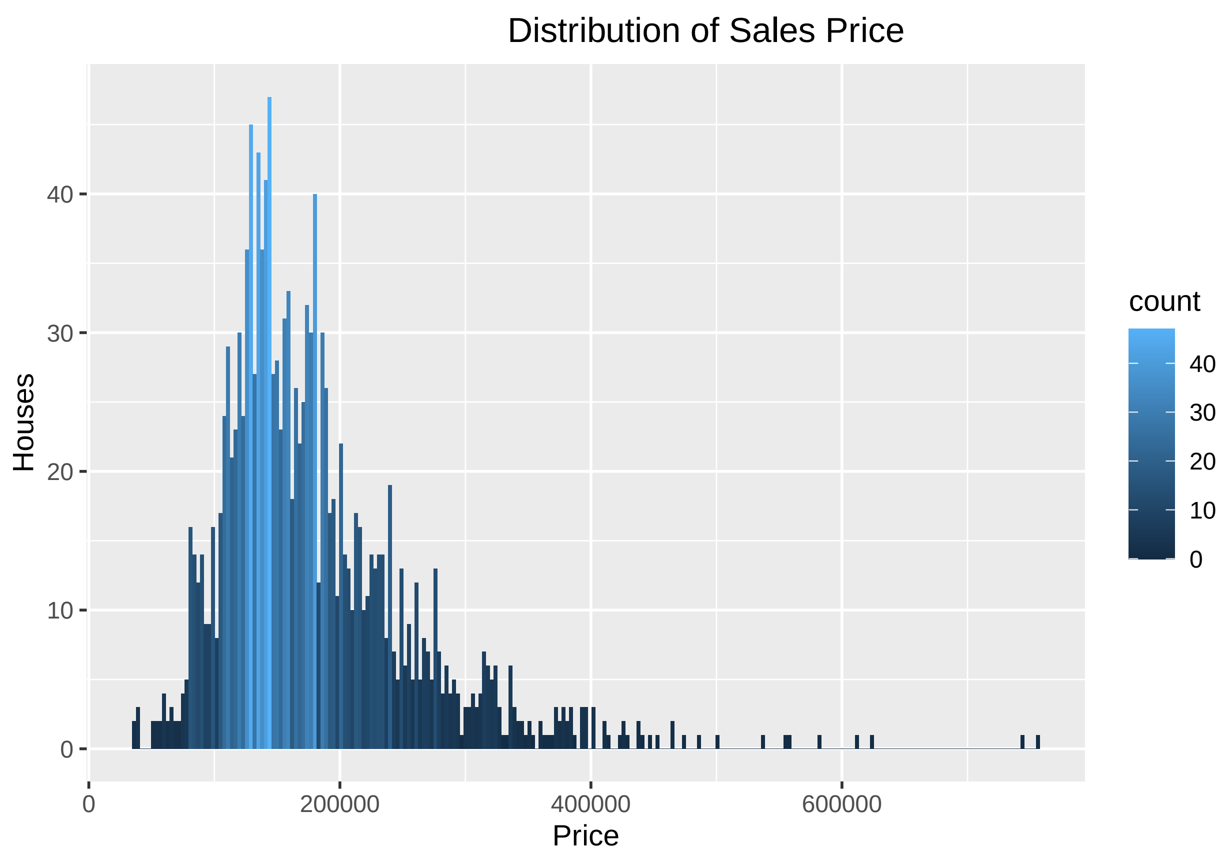
Summary and Distribution of the variable "SalePrice"

Sale Price is our target variable and also the dependent variable for prediction. According to the assumptions of Linear Regression, data should be normally distributed. By checking the distribution of SalePrice, we can decide if we need non-linear transformation, like log term, to make a better prediction.

```
summary(combined.df$SalePrice)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  129975  163000  180921  214000  755000
```

```
options(scipen=20000)
ggplot(combined.df, aes(x = SalePrice, fill = ..count..)) +
  geom_histogram(binwidth = 3000) +
  ggtitle("Distribution of Sales Price") +
  ylab("Houses") +
  xlab("Price") +
  theme(plot.title = element_text(hjust = 0.7))
```



The target variable Sales Price is skewed to the right. To make it normally distributed we apply a log term to this variable.

Log term of SalePrice

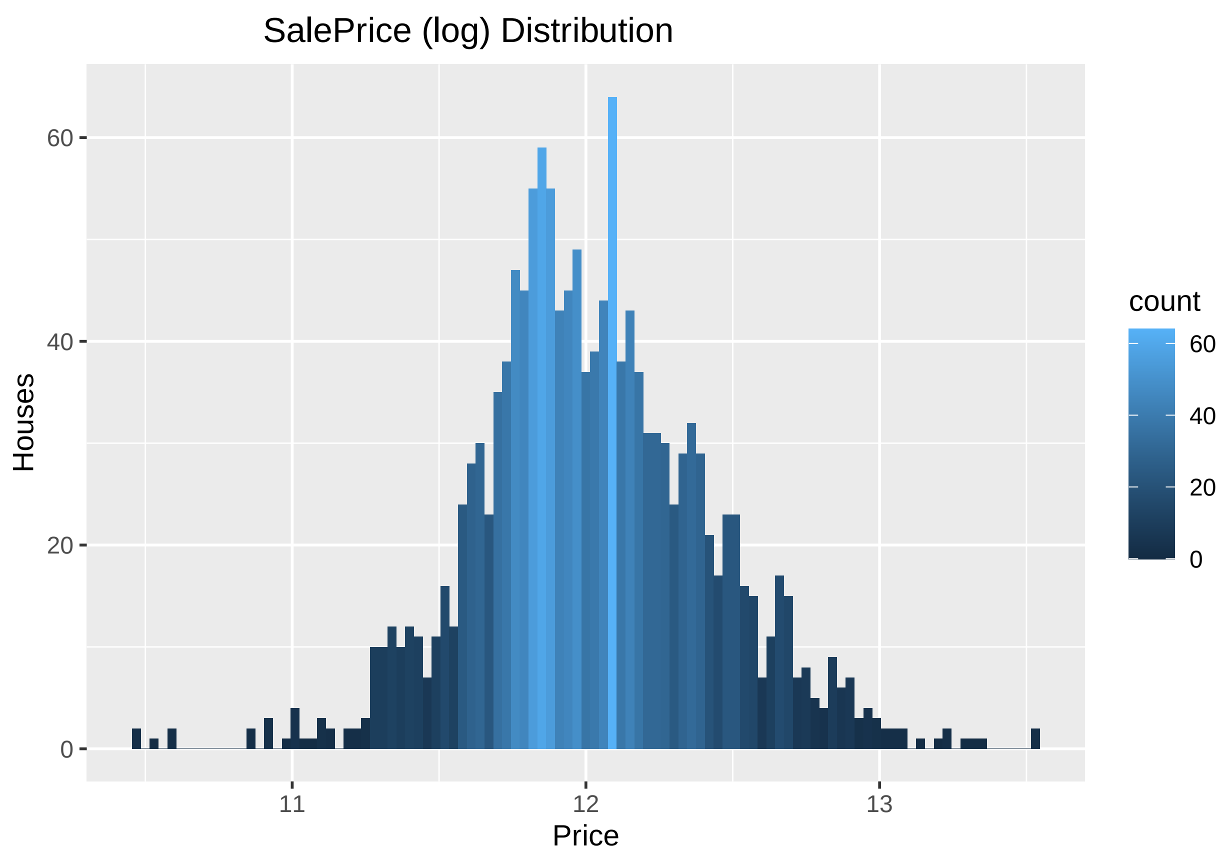
```
combined.df$lSalePrice <- log(combined.df$SalePrice)

test$lSalePrice <- log(test$SalePrice)

train$lSalePrice <- log(train$SalePrice)
```

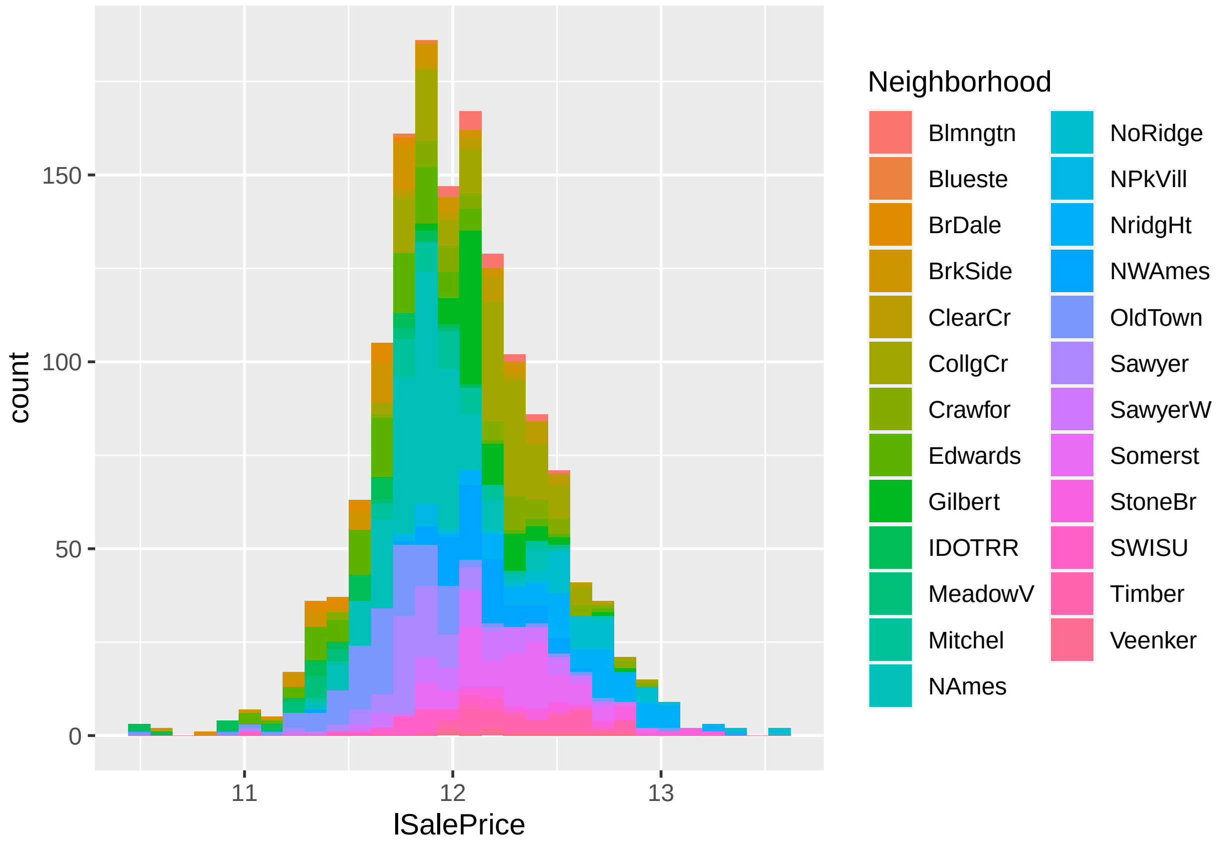
Distribution of log SalePrice

```
ggplot(combined.df, aes(x = lSalePrice, fill = ..count..)) +
  geom_histogram(binwidth = 0.03) +
  ggtitle("SalePrice (log) Distribution") +
  ylab("Houses") +
  xlab("Price") +
  theme(plot.title = element_text(hjust = 0.3))
```



Visualizing Sales Price in relation to Neighborhood.

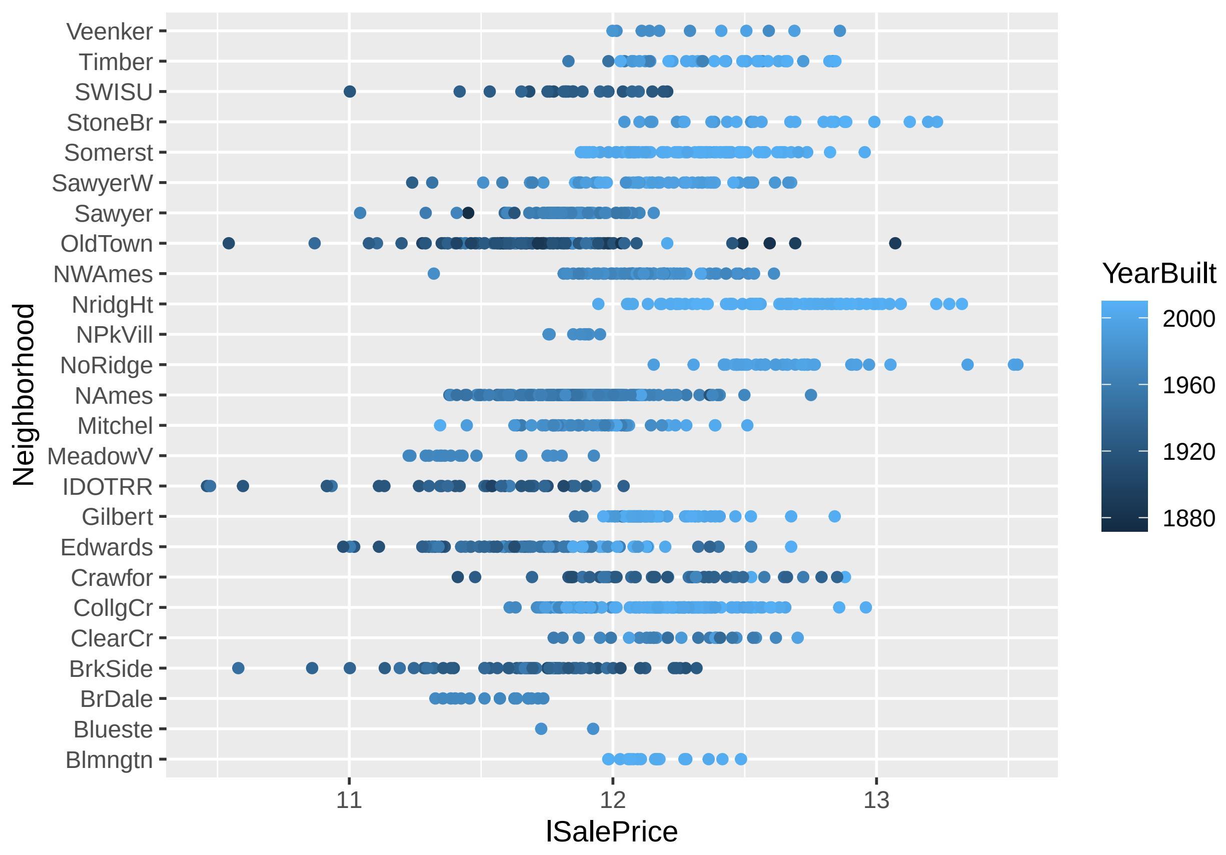
```
p <- ggplot(combined.df, aes(x = lSalePrice, fill = Neighborhood))
p + stat_bin()
```



Different perspectives

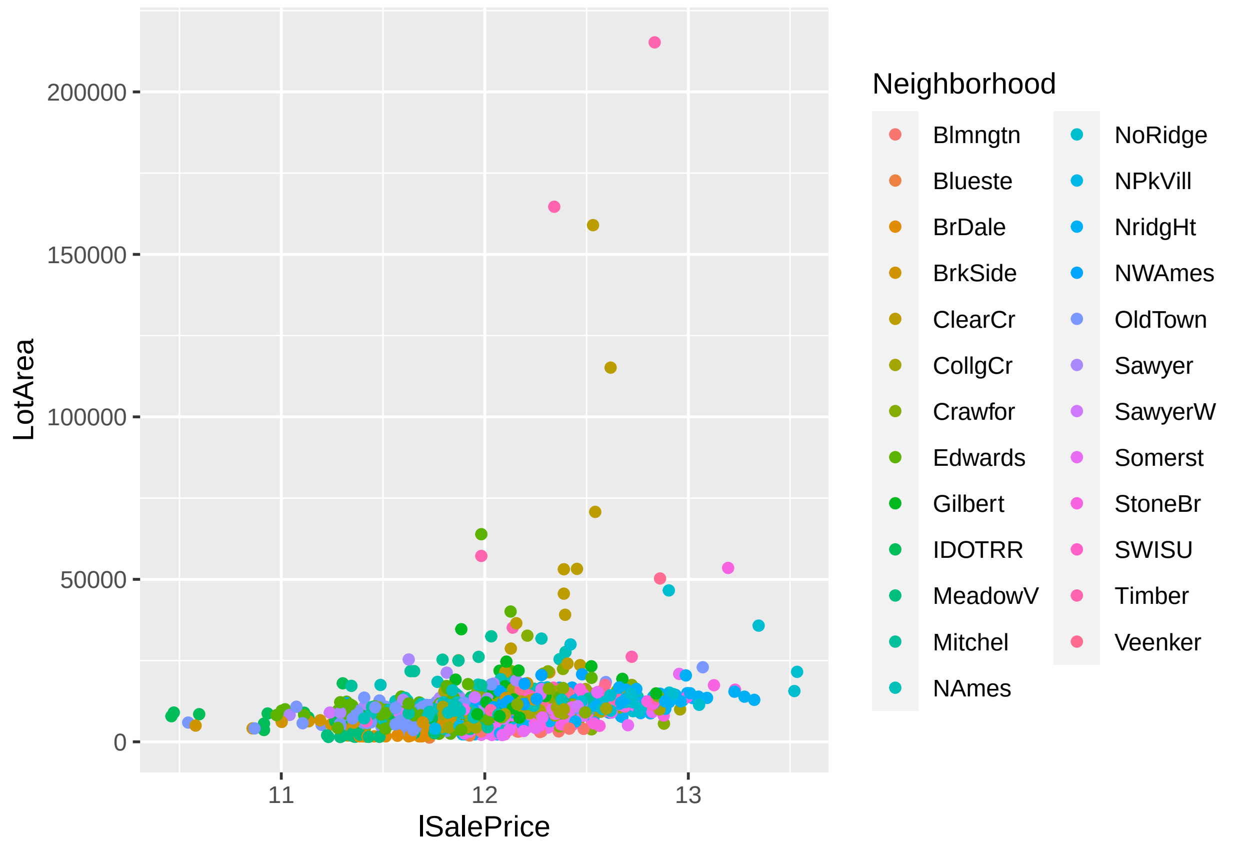
The plot below shows Sales Price in relation to Neighborhood, colored by the Year it was built.

```
q <- ggplot(combined.df, aes(x = lSalePrice, y = Neighborhood, color = YearBuilt))
q + geom_point()
```



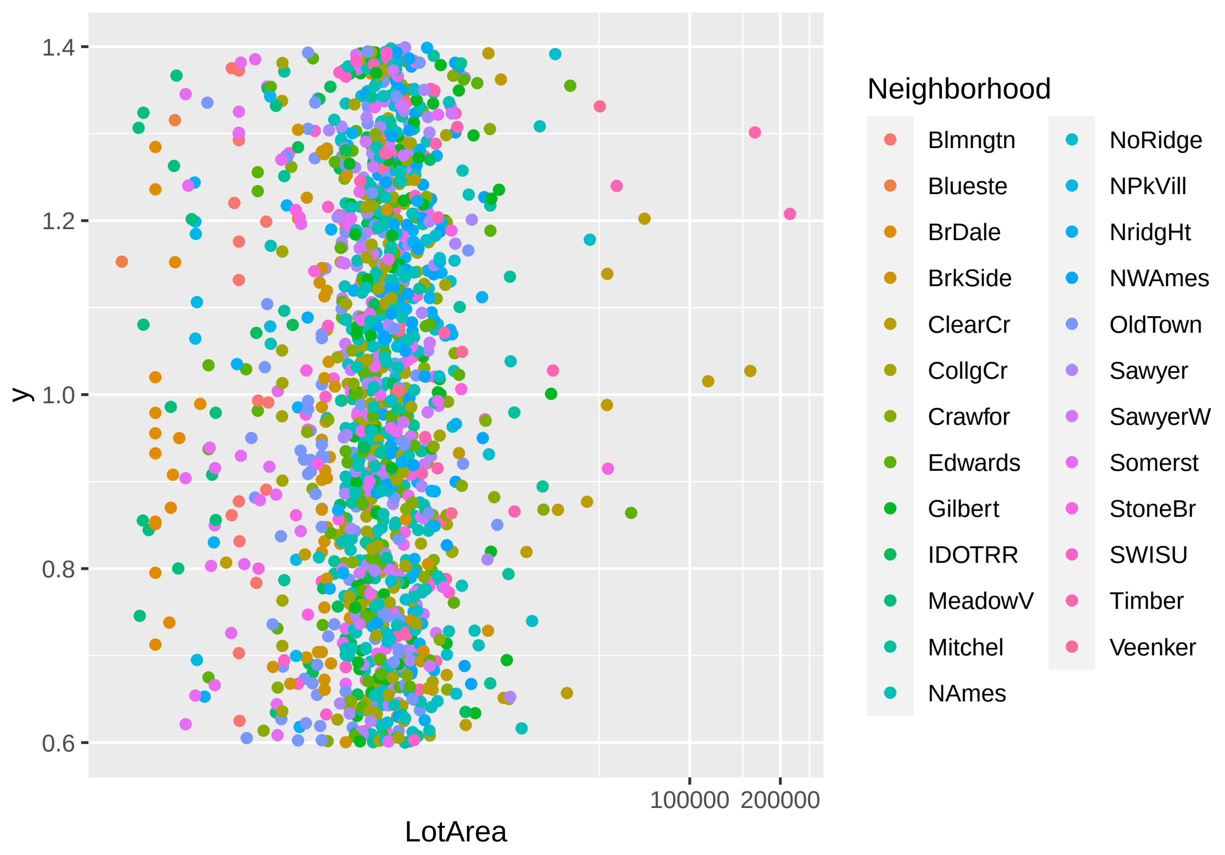
Exploring Sales Price variable, this time in relation to Lot Area and colored by Neighborhood. We can see that the outliars are located in the same neighborhood.

```
sales_lot <- ggplot(combined.df, aes(x = lSalePrice, y = LotArea, color = Neighborhood))
sales_lot + geom_point()
```



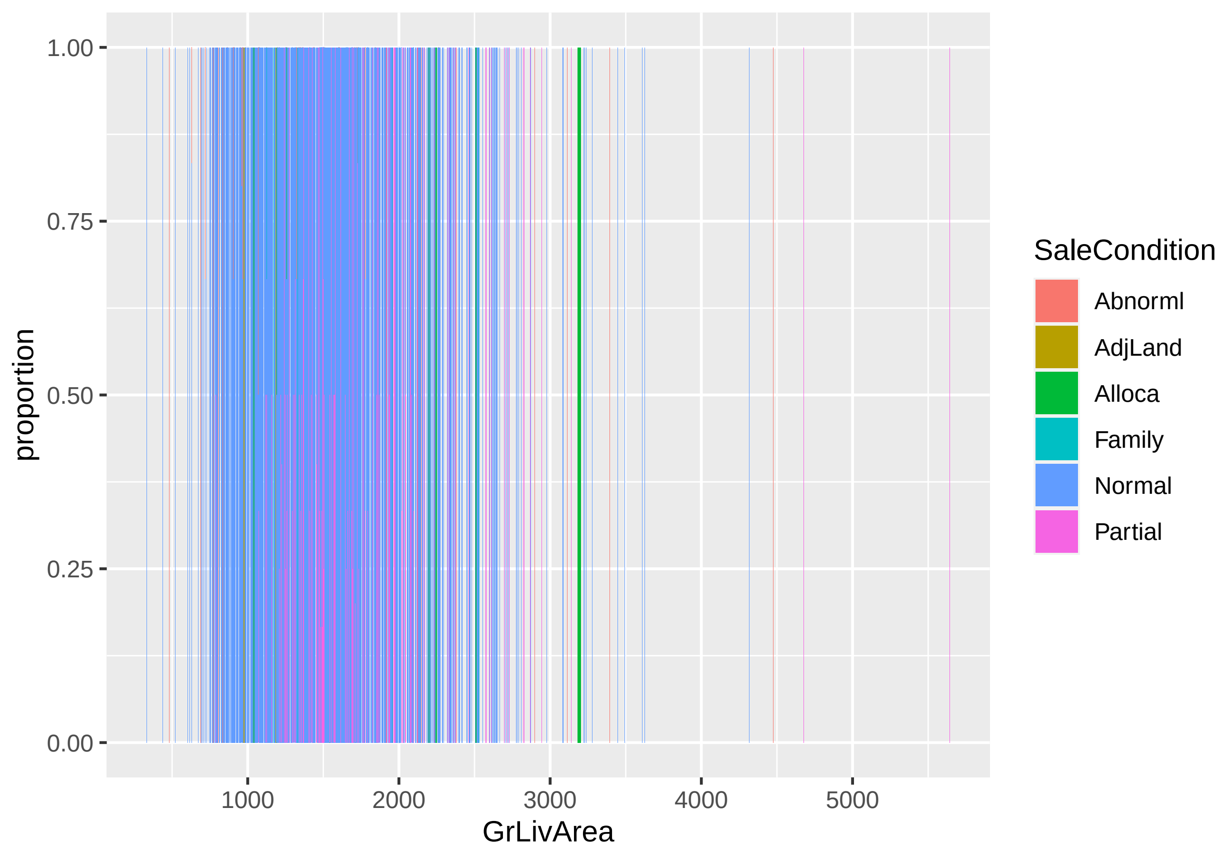
Log transformed x axis

```
sales_lot2 <- ggplot(combined.df, aes(LotArea, y = 1, color = Neighborhood))
sales_lot2 + geom_jitter() +
coord_trans(x = "log10")
```



Ground Living Area filled with Sale Condition

```
combined.df %>%
  ggplot(aes(x = GrLivArea, fill = SaleCondition)) +
geom_bar(position = "fill") +
ylab("proportion")
```



Residual Mean Squared Error (RMSE)

As in the MovieLens report, part important of this project is to provide the RMSE using only the training set, and experimenting with multiple parameters.

The first thing to do is to develop a function to compute the RMSE.

```r
RMSE <- function(x,y){
  a <- sqrt(sum((log(x)-log(y))^2)/length(y))
  return(a)
}
```

Regression

A Regression predicts a numerical outcome ("dependent variable") from a set of inputs ("independent variables").

Linear Regression

```r
model <- lm(lSalePrice ~ GrLivArea, data = train)
predict <- predict(model, test)
```

```r
# RMSE
RMSE0 <- RMSE(predict, test$lSalePrice)
RMSE0 <- round(RMSE0, digits = 3)
rmse_project_results <- data_frame(Method = "Linear Regression", RMSE = RMSE0)
rmse_project_results %>% knitr::kable()
```

Method	RMSE
Linear Regression	0.025

Regression Trees

A decision tree is a structure in which each internal node represents a test on a feature, each leaf node represents a class label, and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules.

```r
model <- rpart(lSalePrice ~., data = train, method = "anova")
predict <- predict(model, test)
RMSE1 <- RMSE(predict, test$lSalePrice)
RMSE1 <- round(RMSE1, digits = 3)
rmse_project_results <- bind_rows(rmse_project_results, data_frame(Method = "Regression Tree", RMSE = R
rmse_project_results%>% knitr::kable()
```

Method	RMSE
Linear Regression	0.025
Regression Tree	0.007

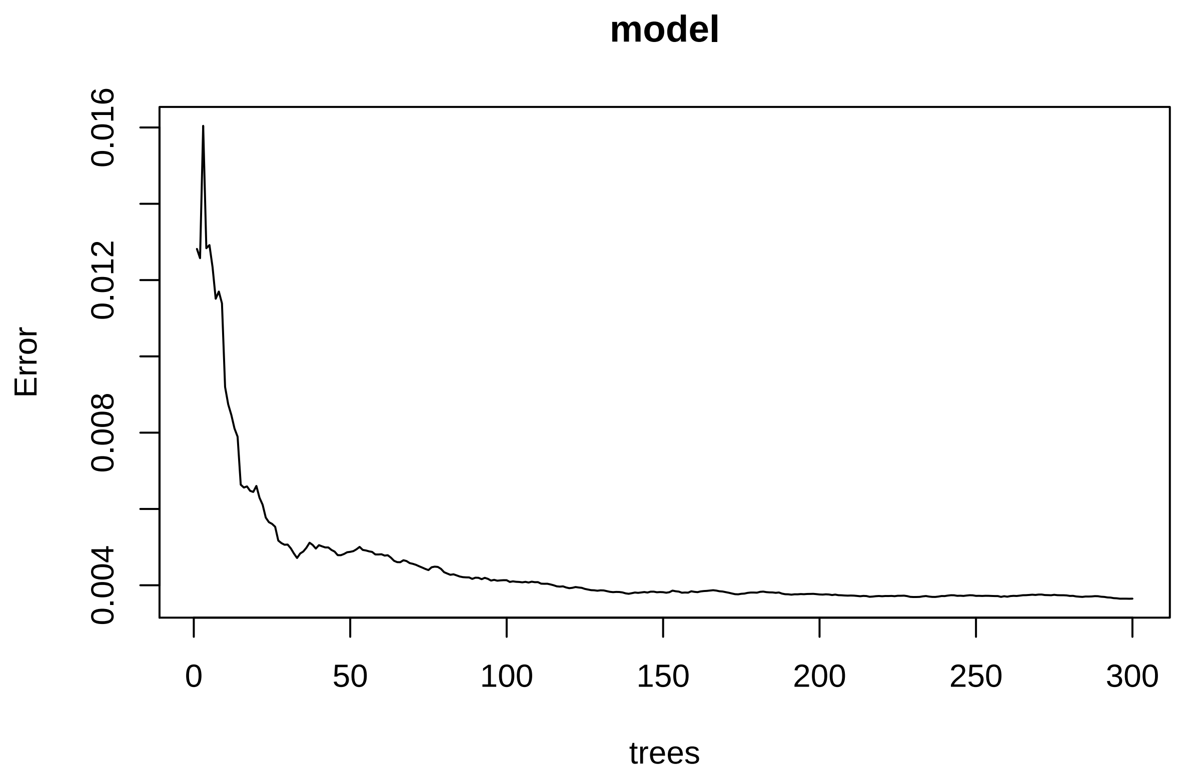
Random Forest

This method is also "Tree" based and it uses the qualities features of multiple Decision Trees for decision making. The term 'Random' is due to the fact that this method is a "forest" of random created [decision] 'trees'.

```r
model <- randomForest(lSalePrice ~., data = train, method = "anova",
                      ntree = 300,
                      mtry = 26,
                      replace = F,
                      nodesize = 1,
                      importance = T)
```

Visualizing the number of trees

```r
plot(model)
```



```
predict <- predict(model, test)
RMSE2 <- RMSE(predict, test$lSalePrice)
RMSE2 <- round(RMSE2, digits = 3)
rmse_project_results <- bind_rows(rmse_project_results, data_frame(Method = "Random Forest", RMSE = RMS
rmse_project_results %>% knitr::kable()
```

Method	RMSE
Linear Regression	0.025
Regression Tree	0.007
Random Forest	0.005

Gradient Boosting Machine

Gradient Boosting Machine is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set.

```
model <- gbm(lSalePrice ~., data = train, distribution = "laplace",
             shrinkage = 0.05,
             interaction.depth = 5,
             bag.fraction = 0.66,
             n.minobsinnode = 1,
             cv.folds = 100,
             keep.data = F,
             verbose = F,
             n.trees = 300)

predict <- predict(model, test, n.trees = 300)

RMSE3 <- RMSE(predict, test$lSalePrice)
RMSE3 <- round(RMSE3, digits = 3)
rmse_project_results <- bind_rows(rmse_project_results, data_frame(Method = "Gradient Boosting Machine"
rmse_project_results %>% knitr::kable()
```

Method	RMSE
Linear Regression	0.025
Regression Tree	0.007
Random Forest	0.005
Gradient Boosting Machine	0.003

Results and Conclusion

RMSE Project Results

Decision Trees, Random Forests and Boosting similar machine learning methods with some overlap.In the case analyzed here, using the House Price Prediction, the best performing model was the Gradient Boosting Machine, as it combines decision trees at the beginning, instead of at the end, as Random Forest does.

```
rmse_project_results %>% knitr::kable()
```

Method	RMSE
Linear Regression	0.025
Regression Tree	0.007
Random Forest	0.005
Gradient Boosting Machine	0.003