House Prices

Gabriel Lapointe

September 18, 2016

Contents

Data Acquisition

In this section, we specify the business problem to solve for this project. From the data source, we will ask questions on the dataset and establish a methodology to solve the problem.

Objective

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

Data Source

The data is provided by Kaggle at https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data.

Dataset Questions

Before we start the exploration of the dataset, we need to write a list of questions about this dataset considering the problem we have to solve.

- How big is the dataset?
- Does the dataset contains `NA` or missing values? Can we replace them by a value? Why?
- Does the data is coherent (date with same format, no out of bound values, no misspelled words, etc.)?
- What does the data look like and what are the relationships between features if they exist?
- What are the measures used?
- Can we solve the problem with this dataset?

Evaluation Metrics

Submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

Methodology

In this document, we start by exploring the dataset and build the data story behind it. This will give us important insights which will answer our questions on this dataset. The next step is to proceed to feature engineering which consists to create, remove or replace features regarding insights we got when exploring the dataset. Then, we will peoceed to a features selection to know which features are strongly correlated to the outcome. We will ensure our new dataset is a valid input for each of our prediction models. We will fine-tune the model's parameters by cross-validating the model with the train set to get the optimal parameters. After applying our model to the test set, we will visualize the predictions calculated and explain the results. Finally, we will give our recommandations to fulfill the objective of this project.

Data Exploratory

In this section, we explore the dataset and we test hypotheses on features. The objective is to visualize and understand the relationships between features in the dataset we have to solve the problem. We will also compare changes we will make to this dataset to validate if they have significant influance on the outcome or not.

A house buyer could be interested to know the following features about the house:

- Area of the basement and the floors
- Area of the garage and number of cars that can enter
- The area of the house's land
- The overall quality

Loading Dataset

We load 'train.csv' and 'test.csv'. Then, we merge them to proceed to the exploration of the entire dataset.

```
library(data.table)
library(dplyr)
library(scales)
library(gridExtra)
library(ggplot2)
library(caret)
library(corrplot)
library(randomForest)
library(xgboost)

setwd("/home/gabriel/Documents/Projects/HousePrices")
#source("Visualisation.R")
#source("Dataset.R")

## Remove scientific notation (e.g. E-005).
options(scipen = 999)

set.seed(1234)

na.strings <- c("NA")
train <- fread(input = "train.csv",
               showProgress = FALSE,
               stringsAsFactors = FALSE,
               na.strings = na.strings,
               header = TRUE)

test <- fread(input = "test.csv",
              showProgress = FALSE,
              stringsAsFactors = FALSE,
              na.strings = na.strings,
              header = TRUE)

test$SalePrice <- -1
dataset <- rbind(train, test)
```

Coherence of the Dataset

We have to check if the dataset is valid with the possible values given in the code book. Thus, we need to ensure that there are no mispelled words or no values that are not in the code book. Also, all numerical values should be coherent with their description meaning that their bounds have to be logiclly correct. Regarding the code book, none of the categorical features have over 25 features. We display the unique values for each features where the number of unique values is less or equal than 25. Then, we will compare the values mentioned in the code book with the values we have in the dataset.

```
getUniqueValues <- function(feature)
{
    feature.values <- unique(feature)
    if(length(feature.values) <= 25)
    {
        paste(sort(feature.values, na.last = TRUE), collapse = ", ")
    }
}

sapply(dataset, getUniqueValues)
```

```
## $Id
## NULL
##
## $MSSubClass
## [1] "20, 30, 40, 45, 50, 60, 70, 75, 80, 85, 90, 120, 150, 160, 180, 190"
##
## $MSZoning
## [1] "C (all), FV, RH, RL, RM, NA"
##
## $LotFrontage
## NULL
##
## $LotArea
## NULL
##
## $Street
## [1] "Grvl, Pave"
##
## $Alley
## [1] ", Grvl, Pave, NA"
##
## $LotShape
## [1] "IR1, IR2, IR3, Reg"
##
## $LandContour
## [1] "Bnk, HLS, Low, Lvl"
##
## $Utilities
## [1] "AllPub, NoSeWa, NA"
##
## $LotConfig
## [1] "Corner, CulDSac, FR2, FR3, Inside"
##
## $LandSlope
```

```
## [1] "Gtl, Mod, Sev"
##
## $Neighborhood
## [1] "Blmngtn, Blueste, BrDale, BrkSide, ClearCr, CollgCr, Crawfor, Edwards, Gilbert, IDOTRR, MeadowV
##
## $Condition1
## [1] "Artery, Feedr, Norm, PosA, PosN, RRAe, RRAn, RRNe, RRNn"
##
## $Condition2
## [1] "Artery, Feedr, Norm, PosA, PosN, RRAe, RRAn, RRNn"
##
## $BldgType
## [1] "1Fam, 2fmCon, Duplex, Twnhs, TwnhsE"
##
## $HouseStyle
## [1] "1.5Fin, 1.5Unf, 1Story, 2.5Fin, 2.5Unf, 2Story, SFoyer, SLvl"
##
## $OverallQual
## [1] "1, 2, 3, 4, 5, 6, 7, 8, 9, 10"
##
## $OverallCond
## [1] "1, 2, 3, 4, 5, 6, 7, 8, 9"
##
## $YearBuilt
## NULL
##
## $YearRemodAdd
## NULL
##
## $RoofStyle
## [1] "Flat, Gable, Gambrel, Hip, Mansard, Shed"
##
## $RoofMatl
## [1] "ClyTile, CompShg, Membran, Metal, Roll, Tar&Grv, WdShake, WdShngl"
##
## $Exterior1st
## [1] "AsbShng, AsphShn, BrkComm, BrkFace, CBlock, CemntBd, HdBoard, ImStucc, MetalSd, Plywood, Stone,
##
## $Exterior2nd
## [1] "AsbShng, AsphShn, Brk Cmn, BrkFace, CBlock, CmentBd, HdBoard, ImStucc, MetalSd, Other, Plywood,
##
## $MasVnrType
## [1] "BrkCmn, BrkFace, None, Stone, NA"
##
## $MasVnrArea
## NULL
##
## $ExterQual
## [1] "Ex, Fa, Gd, TA"
##
## $ExterCond
## [1] "Ex, Fa, Gd, Po, TA"
##
## $Foundation
```

```
## [1] "BrkTil, CBlock, PConc, Slab, Stone, Wood"
##
## $BsmtQual
## [1] "Ex, Fa, Gd, TA, NA"
##
## $BsmtCond
## [1] "Fa, Gd, Po, TA, NA"
##
## $BsmtExposure
## [1] "Av, Gd, Mn, No, NA"
##
## $BsmtFinType1
## [1] "ALQ, BLQ, GLQ, LwQ, Rec, Unf, NA"
##
## $BsmtFinSF1
## NULL
##
## $BsmtFinType2
## [1] "ALQ, BLQ, GLQ, LwQ, Rec, Unf, NA"
##
## $BsmtFinSF2
## NULL
##
## $BsmtUnfSF
## NULL
##
## $TotalBsmtSF
## NULL
##
## $Heating
## [1] "Floor, GasA, GasW, Grav, OthW, Wall"
##
## $HeatingQC
## [1] "Ex, Fa, Gd, Po, TA"
##
## $CentralAir
## [1] "N, Y"
##
## $Electrical
## [1] "FuseA, FuseF, FuseP, Mix, SBrkr, NA"
##
## $`1stFlrSF`
## NULL
##
## $`2ndFlrSF`
## NULL
##
## $LowQualFinSF
## NULL
##
## $GrLivArea
## NULL
##
## $BsmtFullBath
```

```
## [1] "0, 1, 2, 3, NA"
##
## $BsmtHalfBath
## [1] "0, 1, 2, NA"
##
## $FullBath
## [1] "0, 1, 2, 3, 4"
##
## $HalfBath
## [1] "0, 1, 2"
##
## $BedroomAbvGr
## [1] "0, 1, 2, 3, 4, 5, 6, 8"
##
## $KitchenAbvGr
## [1] "0, 1, 2, 3"
##
## $KitchenQual
## [1] "Ex, Fa, Gd, TA, NA"
##
## $TotRmsAbvGrd
## [1] "2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15"
##
## $Functional
## [1] "Maj1, Maj2, Min1, Min2, Mod, Sev, Typ, NA"
##
## $Fireplaces
## [1] "0, 1, 2, 3, 4"
##
## $FireplaceQu
## [1] "Ex, Fa, Gd, Po, TA, NA"
##
## $GarageType
## [1] "2Types, Attchd, Basment, BuiltIn, CarPort, Detchd, NA"
##
## $GarageYrBlt
## NULL
##
## $GarageFinish
## [1] "Fin, RFn, Unf, NA"
##
## $GarageCars
## [1] "0, 1, 2, 3, 4, 5, NA"
##
## $GarageArea
## NULL
##
## $GarageQual
## [1] "Ex, Fa, Gd, Po, TA, NA"
##
## $GarageCond
## [1] "Ex, Fa, Gd, Po, TA, NA"
##
## $PavedDrive
```

```
## [1] "N, P, Y"
##
## $WoodDeckSF
## NULL
##
## $OpenPorchSF
## NULL
##
## $EnclosedPorch
## NULL
##
## $`3SsnPorch`
## NULL
##
## $ScreenPorch
## NULL
##
## $PoolArea
## [1] "0, 144, 228, 368, 444, 480, 512, 519, 555, 561, 576, 648, 738, 800"
##
## $PoolQC
## [1] ", Ex, Fa, Gd, NA"
##
## $Fence
## [1] "GdPrv, GdWo, MnPrv, MnWw, NA"
##
## $MiscFeature
## [1] ", Gar2, Othr, Shed, TenC, NA"
##
## $MiscVal
## NULL
##
## $MoSold
## [1] "1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12"
##
## $YrSold
## [1] "2006, 2007, 2008, 2009, 2010"
##
## $SaleType
## [1] "COD, Con, ConLD, ConLI, ConLw, CWD, New, Oth, WD, NA"
##
## $SaleCondition
## [1] "Abnorml, AdjLand, Alloca, Family, Normal, Partial"
##
## $SalePrice
## NULL
```

Comparing with the code book's possible codes manually, the followings have difference:

Feature	Dataset	CodeBook
MSZoning	C (all)	C
MSZoning	NA	No corresponding value
Alley	Empty string	No corresponding value

Feature	Dataset	CodeBook
Utilities	NA	No corresponding value
Neighborhood	NAmes	Names (should be NAmes)
BldgType	2fmCon	2FmCon
BldgType	Duplex	Duplx
BldgType	Twnhs	TwnhsI
Exterior1st	NA	No corresponding value
Exterior2nd	NA	No corresponding value
Exterior2nd	Wd Shng	WdShing
MasVnrType	NA	No corresponding value
Electrical	NA	No corresponding value
KitchenQual	NA	No corresponding value
Functional	NA	No corresponding value
MiscFeature	Empty string	No corresponding value
SaleType	NA	No corresponding value
Bedroom	Named 'BedroomAbvGr'	

To be coherent with the code book (assuming the code book is the truth), we will replace mispelled categories in the dataset by their corresponding one from the code book. Also, the empty strings and spaces will be replaced by NA. Note that we will assume that the string 'Twnhs' corresponds to the string 'TwnhsI' in the code book.

```
feature.emptystring <- c("Alley", "MiscFeature")
dataset[, feature.emptystring] <- dataset %>%
    select(Alley, MiscFeature) %>%
    sapply(function(feature) gsub("^$|^ $", NA, feature))

dataset$MSZoning[dataset$MSZoning == "C (all)"] <- "C"

dataset$BldgType[dataset$BldgType == "2fmCon"] <- "2FmCon"
dataset$BldgType[dataset$BldgType == "Duplex"] <- "Duplx"
dataset$BldgType[dataset$BldgType == "Twnhs"] <- "TwnhsI"

dataset$Exterior2nd[dataset$Exterior2nd == "Wd Shng"] <- "WdShing"

unique(dataset$Alley)
```

```
## [1] NA     "Grvl" "Pave"
```

```
unique(dataset$MSZoning)
```

```
## [1] "RL" "RM" "C"  "FV" "RH" NA
```

```
unique(dataset$BldgType)
```

```
## [1] "1Fam"   "2FmCon" "Duplx"  "TwnhsE" "TwnhsI"
```

```
unique(dataset$Exterior2nd)
```

```
##  [1] "VinylSd" "MetalSd" "WdShing" "HdBoard" "Plywood" "Wd Sdng" "CmentBd"
##  [8] "BrkFace" "Stucco"  "AsbShng" "Brk Cmn" "ImStucc" "AsphShn" "Stone"
## [15] "Other"   "CBlock"  NA
```

Since we have feature names starting by a digit which is not allowed in programming language, we will rename them with their full name.

- 1stFlrSF renamed to FirstFlrSF
- 2ndFlrSF renamed to SecondFlrSF
- 3SsnPorch renamed to ThreeSsnPorch

```
colnames(dataset)[colnames(dataset) == '1stFlrSF'] <- 'FirstFlrSF'
colnames(dataset)[colnames(dataset) == '2ndFlrSF'] <- 'SecondFlrSF'
colnames(dataset)[colnames(dataset) == '3SsnPorch'] <- 'ThreeSsnPorch'

train <- dataset[dataset$SalePrice > -1, ]
test <- dataset[dataset$SalePrice == -1, ]
```

We also need to check the logic in the dataset to make sure the data make sense. Let's enumerate hypotheses and prove them to ensure the coherence of the data.

1. The feature '1stFlrSF' does not have an area of 0 ft². Otherwise, there would not have a first floor, thus no stories at all and then, no house.
2. It is possible to have a second floor area of 0 ft². This means that there is no second floor. Therefore, the number of stories must be 1 or 1½.
3. The number of stories should always be greater or equal to 1.
4. The HouseStyle feature values match with the values of the feature MSSubClass. Thus, the feature HouseStyle can be replaced by MSSubClass since this is a one-to-many mapping.
5. If the second floor area is not 0, then the stories are greater or equal to 1.5. Note that a 1.5 story house means that the main bedroom is on the first floor and the other bedrooms are on the second floors.
6. Values of MSSubClass for 1 and 2 stories match with the YearBuilt.
7. If there is no garage with the house, then GarageType = NA, GarageYrBlt = NA, GarageFinish = NA, GarageCars = 0, GarageArea = 0, GarageQual = NA and GarageCond = NA.
8. If there is no basement in the house, then TotalBsmtSF = 0, BsmtUnfSF = 0, BsmtFinSF2 = 0, BsmtHalfBath = 0, BsmtFullBath = 0, BsmtQual = NA and BsmtCond = NA, BsmtExposure = NA, BsmtFinType1 = NA, BsmtFinSF1 = 0, BsmtFinType2 = NA.
9. If there are no fireplaces, then FireplaceQu = NA.
10. If there are no Pool, then PoolArea = 0 and PoolQC = NA.

The minimum area of the first floor is 334 ft² which is greater than 0. Looking at features 'HouseStyle' and 'MSSubClass' in the code book, there is neither NA value nor another value indicating that there is no story in the house. Indeed, we have 0 NA values for 'HouseStyle' and 0 NA values for 'MSSubClass'. Thus, the first hypothesis is correct.

The minimum area of the second floor is 0 ft². Looking at the feature 'MSSubClass' in the code book, the codes 60, 70, 75 and 160 must not be used. Indeed, the codes used in the dataset are 20, 30, 40, 45, 50, 80, 85, 90, 120, 180, 190. For the feature 'HouseStyle', the codes '2Story', '2.5Fin' and '2.5Unf' must not be used. Indeed, the codes used in the dataset are 1.5Fin, 1.5Unf, 1Story, SFoyer, SLvl. Therefore, the second hypothesis is correct.

The third hypothesis is correct looking at the possible values of the features 'HouseStyle' and 'MSSubClass' in the code book. Also, we have seen that there is no NA value for those features in the dataset.

To prove the fourth hypothesis, we have to do a mapping between values of 'HouseStyle' and 'MSSubClass'. The mapping to check is the following:

HouseStyle	MSSubClass
1Story	20
1Story	30
1Story	40
1Story	90
1Story	120
1Story	190
1.5Fin	50
1.5Fin	150
1.5Unf	45
1.5Unf	150
2Story	60
2Story	70
2Story	90
2Story	160
2Story	190
2.5Fin	75
2.5Unf	75
SFoyer	85
SFoyer	180
SLvl	80
SLvl	180

```
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "1Story"]))
```

```
## [1]  20  30  40  90 120 190
```

```
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "1.5Fin"]))
```

```
## [1]  30  40  45  50  60  70  90 150 190
```

```
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "1.5Unf"]))
```

```
## [1]  30  45  50 190
```

```
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "2Story"]))
```

```
## [1]  20  50  60  70  75  90 160 190
```

```
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "2.5Fin"]))
```

```
## [1]  70  75 190
```

```
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "2.5Unf"]))
```

```
## [1]  60  70  75  90 190
```

```r
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "SFoyer"]))
```

```
## [1]  85  90 120 180 190
```

```r
sort(unique(dataset$MSSubClass[dataset$HouseStyle == "SLvl"]))
```

```
## [1]  20  60  80  90 160 180 190
```

```r
dataset$Id[dataset$SecondFlrSF > 0 & dataset$MSSubClass %in% c(20, 30, 40)]
```

```
## [1]  165  608 1271 2197 2455 2555
```

```r
dataset$Id[dataset$SecondFlrSF > 0 & dataset$HouseStyle == "1Story"]
```

```
## [1]  165 1271 2455
```

For houses having a second floor, we have to check that the values of 'HouseStyle' are not '1Story'. We have 3 houses with a second floor and having one story which is invalid. This is enough to prove that the fifth hypothesis is false. However, we need to identify these incoherences and check how many of them we will find to calculate the percentage of incoherent houses in this dataset.

```r
dataset$Id[dataset$SecondFlrSF > 0 & dataset$MSSubClass %in% c(20, 30, 40)]
```

```
## [1]  165  608 1271 2197 2455 2555
```

```r
dataset$Id[dataset$SecondFlrSF > 0 & dataset$HouseStyle == "1Story"]
```

```
## [1]  165 1271 2455
```

To verify the sixth hypothesis, we need to compare values of 'MSSubClass' with the 'YearBuilt' values. The hypothesis is invalid if the year built is less than 1946 and values of 'MSSubClass' are 20, 60, and 160. The case when the year built is 1946 and newer, and values of 'MSSubClass' are 30, 70, and 120 also shows that the hypothesis is invalid.

```r
dataset$Id[dataset$YearBuilt < 1946 & dataset$MSSubClass %in% c(20, 60, 160)]
```

```
## [1] 1333 1783 2127 2487 2491
```

```r
dataset$Id[dataset$YearBuilt >= 1946 & dataset$MSSubClass %in% c(30, 70, 120)]
```

```
##   [1]   24   35   46   63   82  124  127  190  204  220  230  252  283  285
##  [15]  327  334  344  351  352  358  373  386  401  444  466  471  475  484
##  [29]  512  544  551  560  594  598  632  640  641  681  683  690  691  700
##  [43]  708  713  721  722  731  745  765  776  791  812  820  837  851  852
##  [57]  886  924  973  978 1005 1018 1020 1024 1043 1057 1061 1079 1090 1098
##  [71] 1127 1158 1182 1194 1229 1252 1265 1289 1307 1318 1369 1395 1406 1416
##  [85] 1422 1423 1432 1442 1465 1471 1475 1482 1484 1496 1585 1599 1600 1604
##  [99] 1614 1630 1631 1632 1655 1659 1675 1676 1677 1680 1681 1683 1686 1727
```

```
## [113] 1740 1741 1853 1915 1939 1940 1943 1947 1966 1979 1980 1981 1982 1983
## [127] 1985 1986 1991 2026 2030 2031 2032 2033 2039 2130 2179 2221 2222 2227
## [141] 2240 2242 2261 2262 2266 2281 2282 2285 2305 2306 2307 2308 2309 2310
## [155] 2311 2321 2365 2366 2367 2368 2369 2375 2499 2500 2572 2583 2590 2601
## [169] 2627 2633 2642 2648 2650 2664 2666 2672 2673 2674 2675 2699 2700 2713
## [183] 2854 2895 2896
```

Even if we take the year where the house has been remodeled, we still have many houses with mismatches. Thus, the sixth hypothesis is invalid.

We need to get all data where the GarageType is NA and check if the seventh hypothesis conditions are correct.

```
garage.none <- dataset[is.na(dataset$GarageType), ]
garage.none$Id[!is.na(garage.none$GarageYrBlt)]
```

```
## integer(0)
```

```
garage.none$Id[!is.na(garage.none$GarageFinish)]
```

```
## integer(0)
```

```
garage.none$Id[garage.none$GarageCars != 0]
```

```
## integer(0)
```

```
garage.none$Id[garage.none$GarageArea != 0]
```

```
## integer(0)
```

```
garage.none$Id[!is.na(garage.none$GarageQual)]
```

```
## integer(0)
```

```
garage.none$Id[!is.na(garage.none$GarageCond)]
```

```
## integer(0)
```

Since the conditions are all respected, then the seventh hypothesis is valid.

We need to get all data where the TotalBsmtSF is 0 ft^2 and check if the eighth hypothesis conditions are correct.

```
basement.none <- dataset[dataset$TotalBsmtSF == 0, ]
basement.none$Id[basement.none$BsmtUnfSF != 0]
```

```
## integer(0)
```

```
basement.none$Id[basement.none$BsmtFinSF2 != 0]
```

```
## integer(0)
```

```
basement.none$Id[basement.none$BsmtHalfBath != 0 & !is.na(basement.none$BsmtHalfBath)]
```

```
## integer(0)
```

```
basement.none$Id[basement.none$BsmtFullBath != 0 & !is.na(basement.none$BsmtFullBath)]
```

```
## integer(0)
```

```
basement.none$Id[basement.none$BsmtFinSF1 != 0]
```

```
## integer(0)
```

```
basement.none$Id[!is.na(basement.none$BsmtQual)]
```

```
## integer(0)
```

```
basement.none$Id[!is.na(basement.none$BsmtCond)]
```

```
## integer(0)
```

```
basement.none$Id[!is.na(basement.none$BsmtExposure)]
```

```
## integer(0)
```

```
basement.none$Id[!is.na(basement.none$BsmtFinType1)]
```

```
## integer(0)
```

```
basement.none$Id[!is.na(basement.none$BsmtFinType2)]
```

```
## integer(0)
```

Since the conditions are all respected, then the eighth hypothesis is valid.

We need to get all data where the Fireplaces is 0 and check if the Fireplace Quality is NA.

```
dataset$Id[dataset$Fireplaces != 0 & is.na(dataset$FireplaceQu)]
```

```
## integer(0)
```

Since the conditions are all respected, then the nineth hypothesis is valid.

We need to get all data where the PoolArea is 0 ft^2 and check if the Pool Quality is NA.

```
dataset$Id[dataset$PoolArea != 0 & is.na(dataset$PoolQC)]
```

```
## [1] 2421 2504 2600
```

Since there are 3 pools, the conditions are not respected, then the tenth hypothesis is invalid.

Missing Values

Per the code book of this dataset, we know that the NA values mean 'No' or 'None' and they are used only for categorical features. The other NA values that are not in the code book will be interpreted as if we do not have information for the house's feature. this goes also for the empty strings that will be replaced by NA. Thus, we will replace all of them by zero without losing any information.

Also, we expect for numeric features that the value 0 means the same thing as a NA value. For example, a garage area of 0 means that there is no garage with this house. However, if the value 0 is used for an amount of money, then it is a real 0.

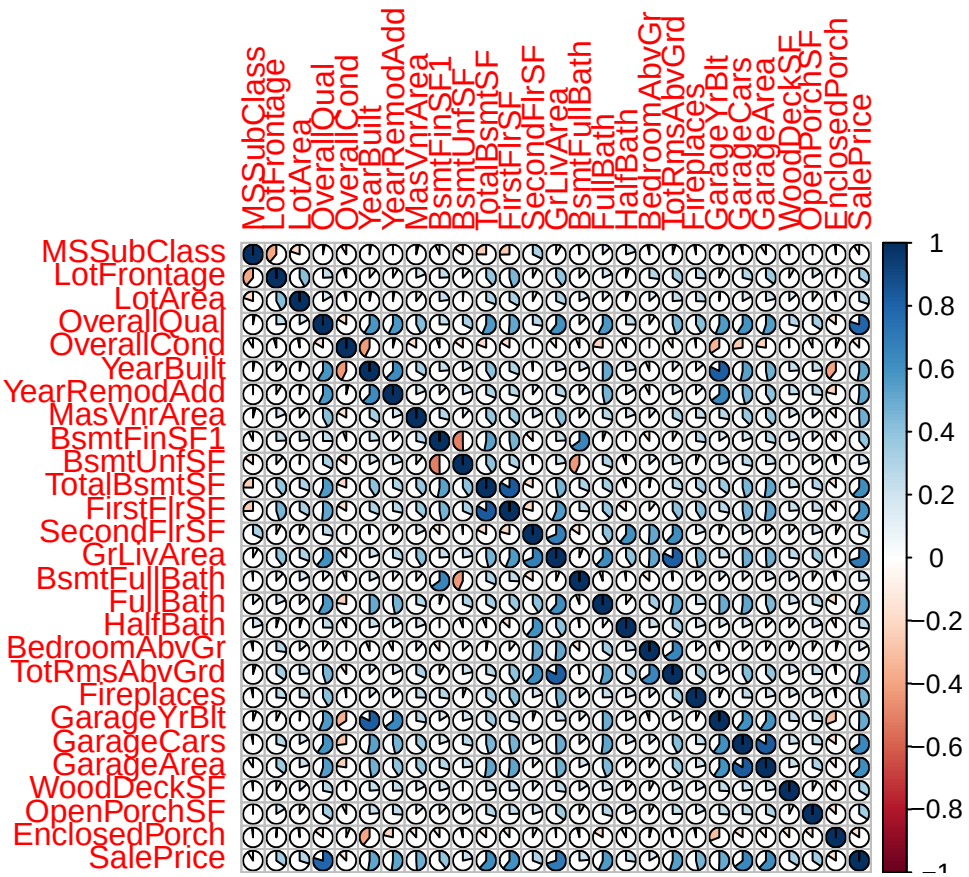
Features

Here is the list of features with their type.

```
## Classes 'data.table' and 'data.frame':   2919 obs. of  81 variables:
##  $ Id           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ MSSubClass   : int  60 20 60 70 60 50 20 60 50 190 ...
##  $ MSZoning     : chr  "RL" "RL" "RL" "RL" ...
##  $ LotFrontage  : int  65 80 68 60 84 85 75 NA 51 50 ...
##  $ LotArea      : int  8450 9600 11250 9550 14260 14115 10084 10382 6120 7420 ...
##  $ Street       : chr  "Pave" "Pave" "Pave" "Pave" ...
##  $ Alley        : chr  NA NA NA NA ...
##  $ LotShape     : chr  "Reg" "Reg" "IR1" "IR1" ...
##  $ LandContour  : chr  "Lvl" "Lvl" "Lvl" "Lvl" ...
##  $ Utilities    : chr  "AllPub" "AllPub" "AllPub" "AllPub" ...
##  $ LotConfig    : chr  "Inside" "FR2" "Inside" "Corner" ...
##  $ LandSlope    : chr  "Gtl" "Gtl" "Gtl" "Gtl" ...
##  $ Neighborhood : chr  "CollgCr" "Veenker" "CollgCr" "Crawfor" ...
##  $ Condition1   : chr  "Norm" "Feedr" "Norm" "Norm" ...
##  $ Condition2   : chr  "Norm" "Norm" "Norm" "Norm" ...
##  $ BldgType     : chr  "1Fam" "1Fam" "1Fam" "1Fam" ...
##  $ HouseStyle   : chr  "2Story" "1Story" "2Story" "2Story" ...
##  $ OverallQual  : int  7 6 7 7 8 5 8 7 7 5 ...
##  $ OverallCond  : int  5 8 5 5 5 5 5 6 5 6 ...
##  $ YearBuilt    : int  2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 ...
##  $ YearRemodAdd : int  2003 1976 2002 1970 2000 1995 2005 1973 1950 1950 ...
##  $ RoofStyle    : chr  "Gable" "Gable" "Gable" "Gable" ...
##  $ RoofMatl     : chr  "CompShg" "CompShg" "CompShg" "CompShg" ...
##  $ Exterior1st  : chr  "VinylSd" "MetalSd" "VinylSd" "Wd Sdng" ...
##  $ Exterior2nd  : chr  "VinylSd" "MetalSd" "VinylSd" "WdShing" ...
##  $ MasVnrType   : chr  "BrkFace" "None" "BrkFace" "None" ...
##  $ MasVnrArea   : int  196 0 162 0 350 0 186 240 0 0 ...
##  $ ExterQual    : chr  "Gd" "TA" "Gd" "TA" ...
##  $ ExterCond    : chr  "TA" "TA" "TA" "TA" ...
```

```
##  $ Foundation   : chr  "PConc" "CBlock" "PConc" "BrkTil" ...
##  $ BsmtQual     : chr  "Gd" "Gd" "Gd" "TA" ...
##  $ BsmtCond     : chr  "TA" "TA" "TA" "Gd" ...
##  $ BsmtExposure : chr  "No" "Gd" "Mn" "No" ...
##  $ BsmtFinType1 : chr  "GLQ" "ALQ" "GLQ" "ALQ" ...
##  $ BsmtFinSF1   : int  706 978 486 216 655 732 1369 859 0 851 ...
##  $ BsmtFinType2 : chr  "Unf" "Unf" "Unf" "Unf" ...
##  $ BsmtFinSF2   : int  0 0 0 0 0 0 0 32 0 0 ...
##  $ BsmtUnfSF    : int  150 284 434 540 490 64 317 216 952 140 ...
##  $ TotalBsmtSF  : int  856 1262 920 756 1145 796 1686 1107 952 991 ...
##  $ Heating      : chr  "GasA" "GasA" "GasA" "GasA" ...
##  $ HeatingQC    : chr  "Ex" "Ex" "Ex" "Gd" ...
##  $ CentralAir   : chr  "Y" "Y" "Y" "Y" ...
##  $ Electrical   : chr  "SBrkr" "SBrkr" "SBrkr" "SBrkr" ...
##  $ FirstFlrSF   : int  856 1262 920 961 1145 796 1694 1107 1022 1077 ...
##  $ SecondFlrSF  : int  854 0 866 756 1053 566 0 983 752 0 ...
##  $ LowQualFinSF : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ GrLivArea    : int  1710 1262 1786 1717 2198 1362 1694 2090 1774 1077 ...
##  $ BsmtFullBath : int  1 0 1 1 1 1 1 1 0 1 ...
##  $ BsmtHalfBath : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ FullBath     : int  2 2 2 1 2 1 2 2 2 1 ...
##  $ HalfBath     : int  1 0 1 0 1 1 0 1 0 0 ...
##  $ BedroomAbvGr : int  3 3 3 3 4 1 3 3 2 2 ...
##  $ KitchenAbvGr : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ KitchenQual  : chr  "Gd" "TA" "Gd" "Gd" ...
##  $ TotRmsAbvGrd : int  8 6 6 7 9 5 7 7 8 5 ...
##  $ Functional   : chr  "Typ" "Typ" "Typ" "Typ" ...
##  $ Fireplaces   : int  0 1 1 1 1 0 1 2 2 2 ...
##  $ FireplaceQu  : chr  NA "TA" "TA" "Gd" ...
##  $ GarageType   : chr  "Attchd" "Attchd" "Attchd" "Detchd" ...
##  $ GarageYrBlt  : int  2003 1976 2001 1998 2000 1993 2004 1973 1931 1939 ...
##  $ GarageFinish : chr  "RFn" "RFn" "RFn" "Unf" ...
##  $ GarageCars   : int  2 2 2 3 3 2 2 2 2 1 ...
##  $ GarageArea   : int  548 460 608 642 836 480 636 484 468 205 ...
##  $ GarageQual   : chr  "TA" "TA" "TA" "TA" ...
##  $ GarageCond   : chr  "TA" "TA" "TA" "TA" ...
##  $ PavedDrive   : chr  "Y" "Y" "Y" "Y" ...
##  $ WoodDeckSF   : int  0 298 0 0 192 40 255 235 90 0 ...
##  $ OpenPorchSF  : int  61 0 42 35 84 30 57 204 0 4 ...
##  $ EnclosedPorch: int  0 0 0 272 0 0 0 228 205 0 ...
##  $ ThreeSsnPorch: int  0 0 0 0 0 320 0 0 0 0 ...
##  $ ScreenPorch  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolArea     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolQC       : chr  "" "" "" "" ...
##  $ Fence        : chr  NA NA NA NA ...
##  $ MiscFeature  : chr  NA NA NA NA ...
##  $ MiscVal      : int  0 0 0 0 0 700 0 350 0 0 ...
##  $ MoSold       : int  2 5 9 2 12 10 8 11 4 1 ...
##  $ YrSold       : int  2008 2007 2008 2006 2008 2009 2007 2009 2008 2008 ...
##  $ SaleType     : chr  "WD" "WD" "WD" "WD" ...
##  $ SaleCondition: chr  "Normal" "Normal" "Normal" "Abnorml" ...
##  $ SalePrice    : num  208500 181500 223500 140000 250000 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

We see now a plot of the correlation between numeric features of the train set.



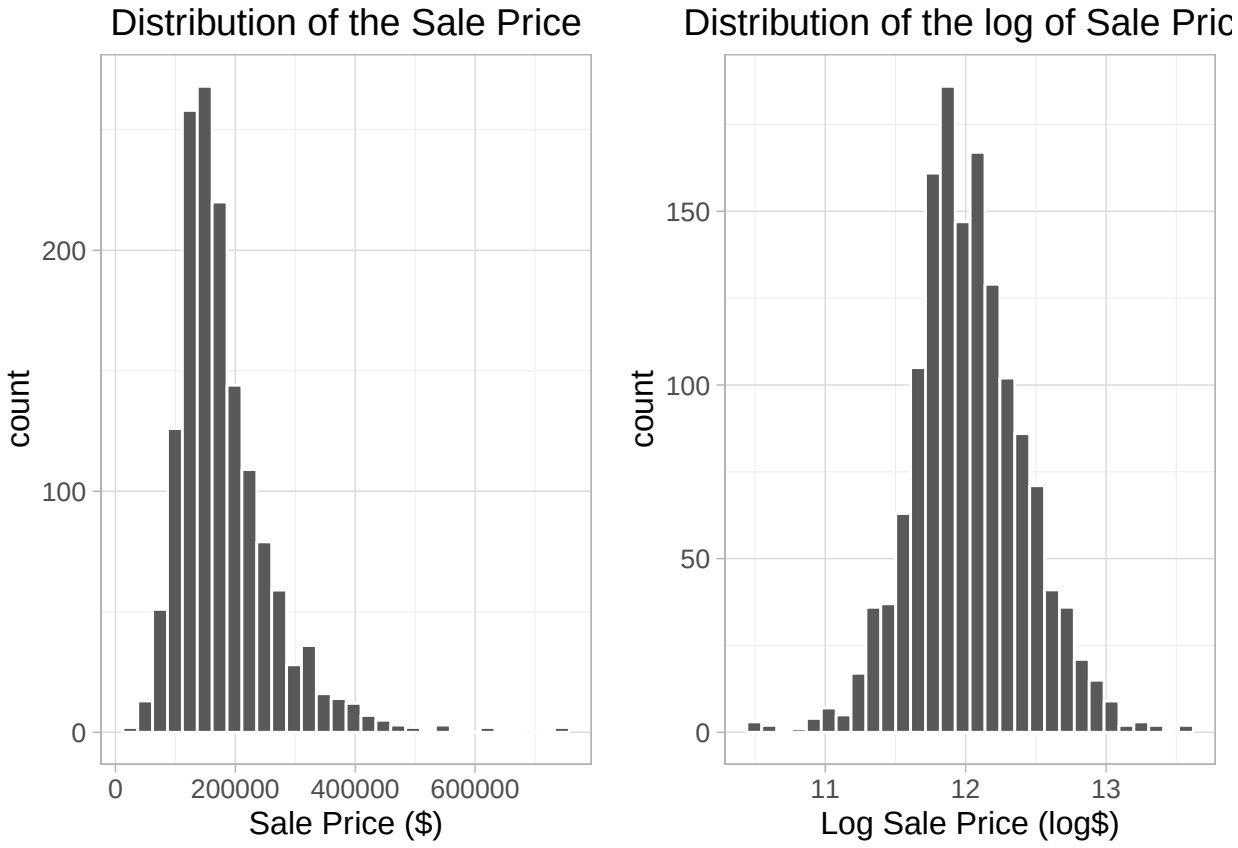

```
##                SalePriceCorrelation
## SalePrice                 1.0000000
## OverallQual               0.7978807
## GrLivArea                 0.7051536
## GarageCars                0.6470336
## GarageArea                0.6193296
## TotalBsmtSF               0.6156122
## FirstFlrSF                0.6079691
## FullBath                  0.5666274
## TotRmsAbvGrd              0.5470674
## YearBuilt                 0.5253936
## YearRemodAdd              0.5212533
## GarageYrBlt               0.5047530
## MasVnrArea                0.4886582
## Fireplaces                0.4618727
## BsmtFinSF1                0.3903005
## LotFrontage               0.3442698
## OpenPorchSF               0.3433538
## WoodDeckSF                0.3368551
## SecondFlrSF               0.3068790
## LotArea                   0.2999622
## HalfBath                  0.2685603
## BsmtFullBath              0.2367374
## BsmtUnfSF                 0.2131287
```

```
## BedroomAbvGr            0.1668139
## MSSubClass             -0.0880317
## OverallCond            -0.1243912
## EnclosedPorch          -0.1548432
```

Regarding the sale price, we note that some features are more than 60% correlated with the sale price. We will produce plots for each of them to get insights.

Sale Price

The sale price should follow the normal distribution. However, we need to normalize the sale price by taking its logarithm.



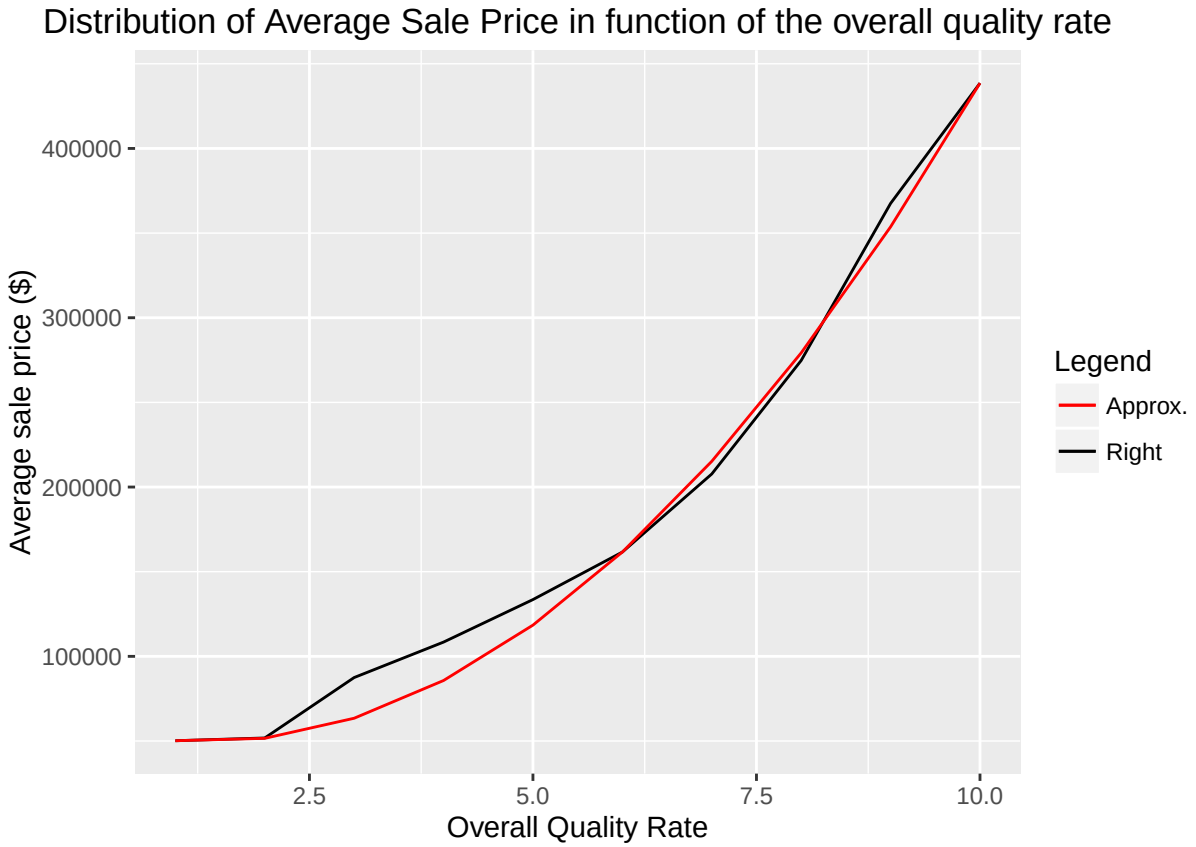

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  130000  163000  180900  214000  755000
```

Overall Quality Rate

The overall quality rate is the most correlated feature to the sale price as seen previously. We look at the average sale price for each overall quality rate and try to figure out an equation that will best approximate our data.

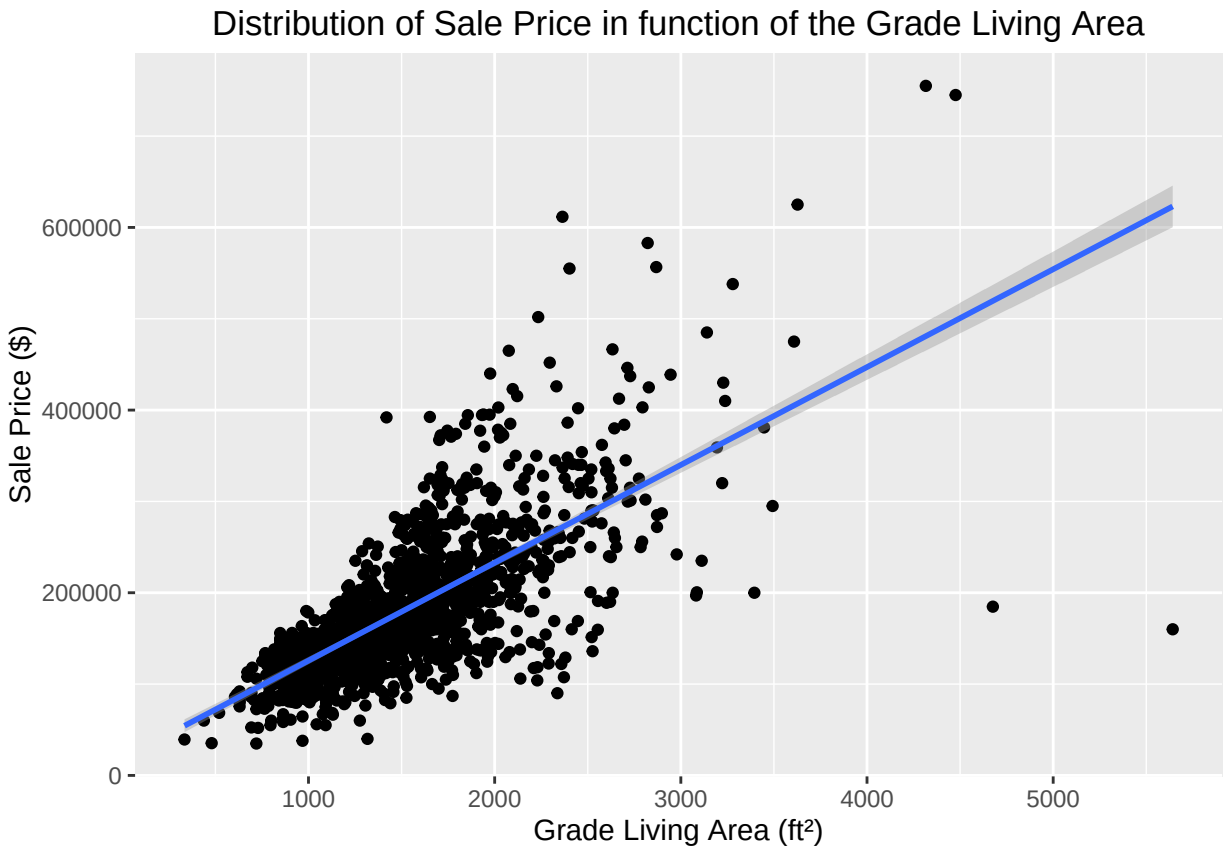
```
## Source: local data table [10 x 2]
##
##    OverallQual MeanSalePrice
```

```
##          (int)         (dbl)
## 1            1      50150.00
## 2            2      51770.33
## 3            3      87473.75
## 4            4     108420.66
## 5            5     133523.35
## 6            6     161603.03
## 7            7     207716.42
## 8            8     274735.54
## 9            9     367513.02
## 10          10     438588.39
```



Note that the equation used to approximate is a parabola where the equation has been built from 3 points (OverallQual, MeanSalePrice) where the overall quality rates chosen are 1, 6 and 10 with their corresponding average sale price. The equation used to approximate is $M(Q) = \frac{939113}{180}Q^2 - \frac{2561483}{180}Q + \frac{354979}{6}$ where Q is the overall quality rate and $M(Q)$ is the mean sale price in function of Q.

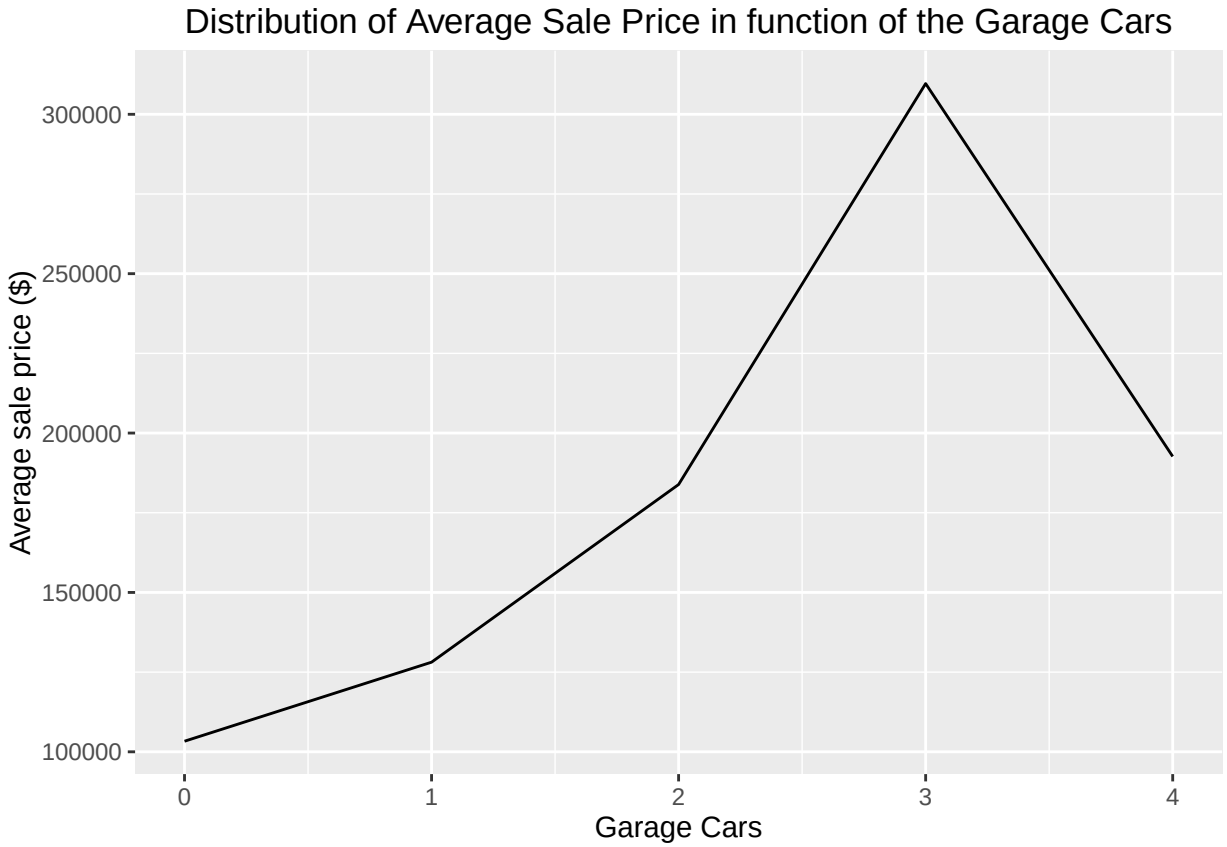
Above grade (ground) living area



Garage Cars

We verify that if the Garage Cars is 0, then the Garage Area is also 0. The converse is true since a Garage area of 0 means that there is no garage.

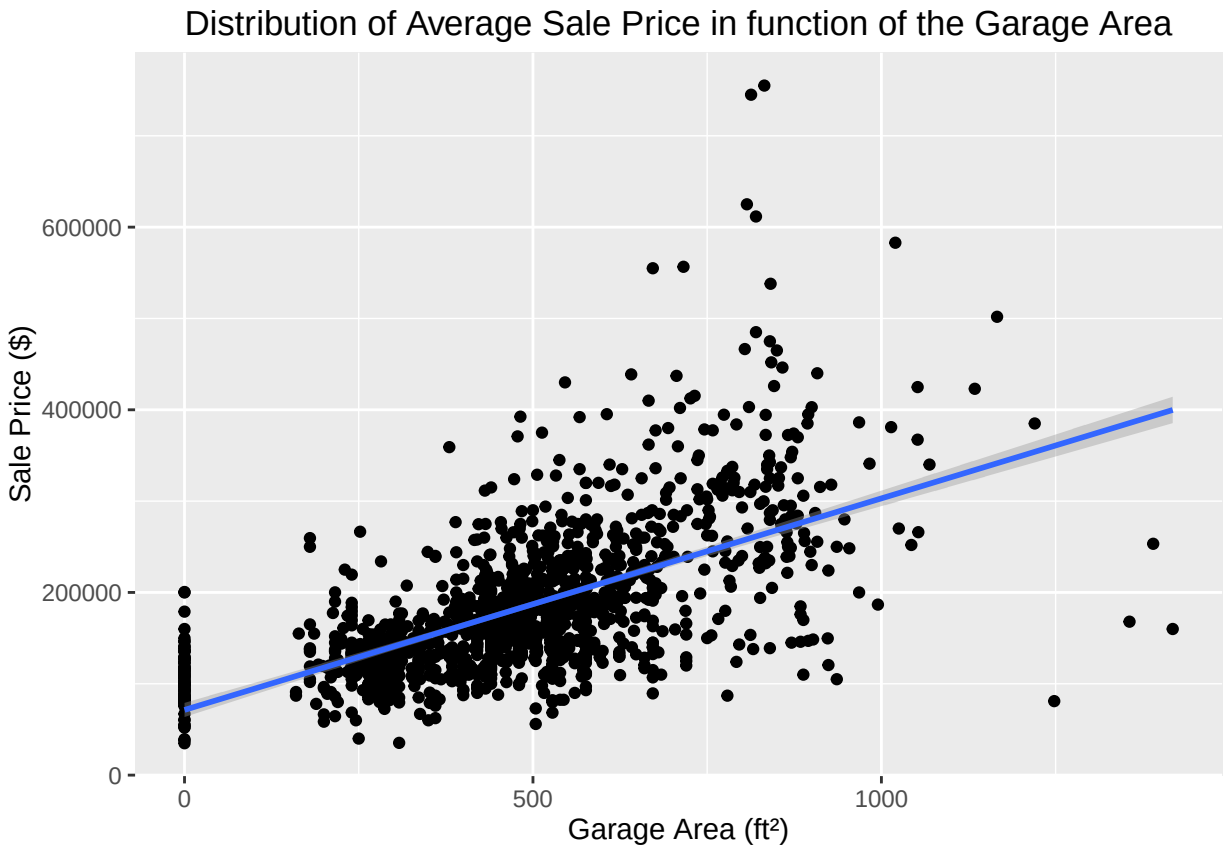
```
## Source: local data table [5 x 2]
##
##   GarageCars MeanSalePrice
##        (int)         (dbl)
## 1          0      103317.3
## 2          1      128116.7
## 3          2      183851.7
## 4          3      309636.1
## 5          4      192655.8
```




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

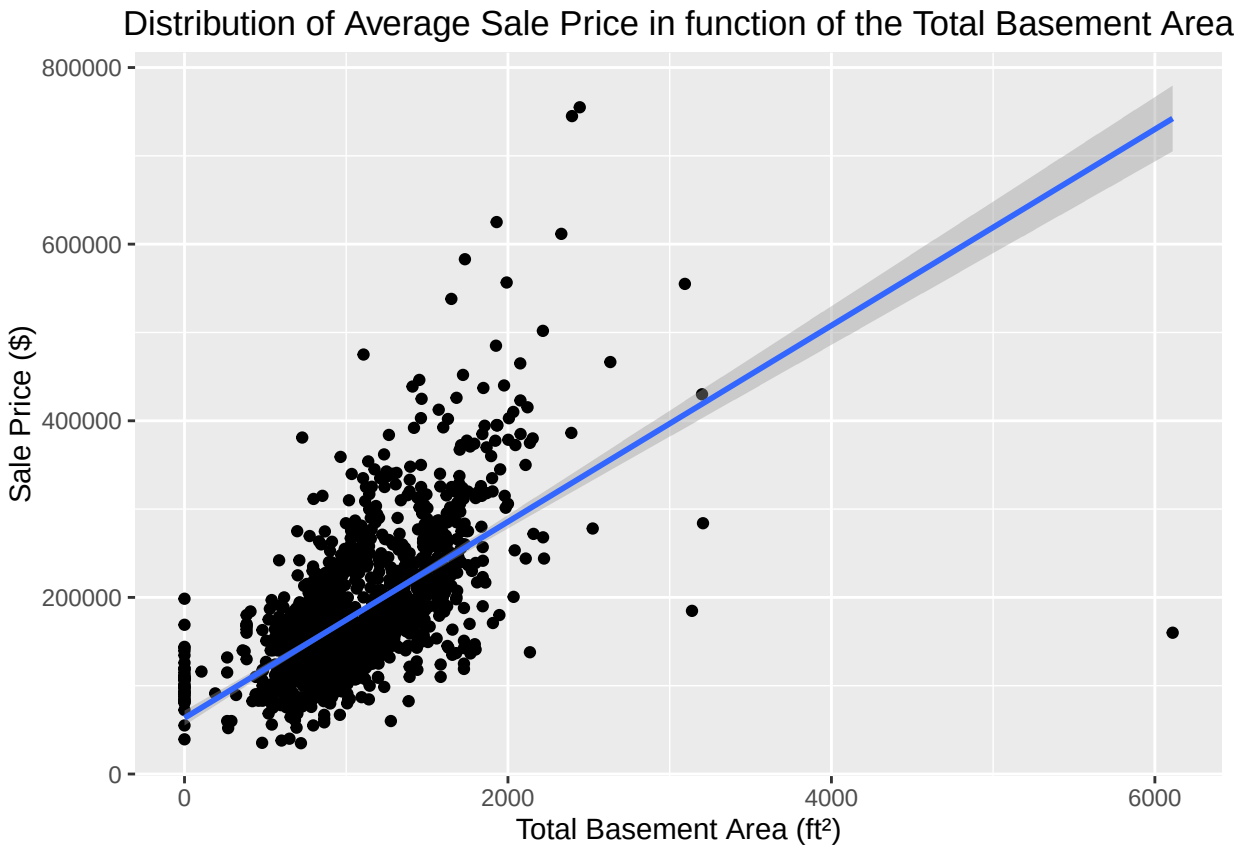
Garage Area

We precise that a garage area of 0 ft^2 means that there is no garage with the house.

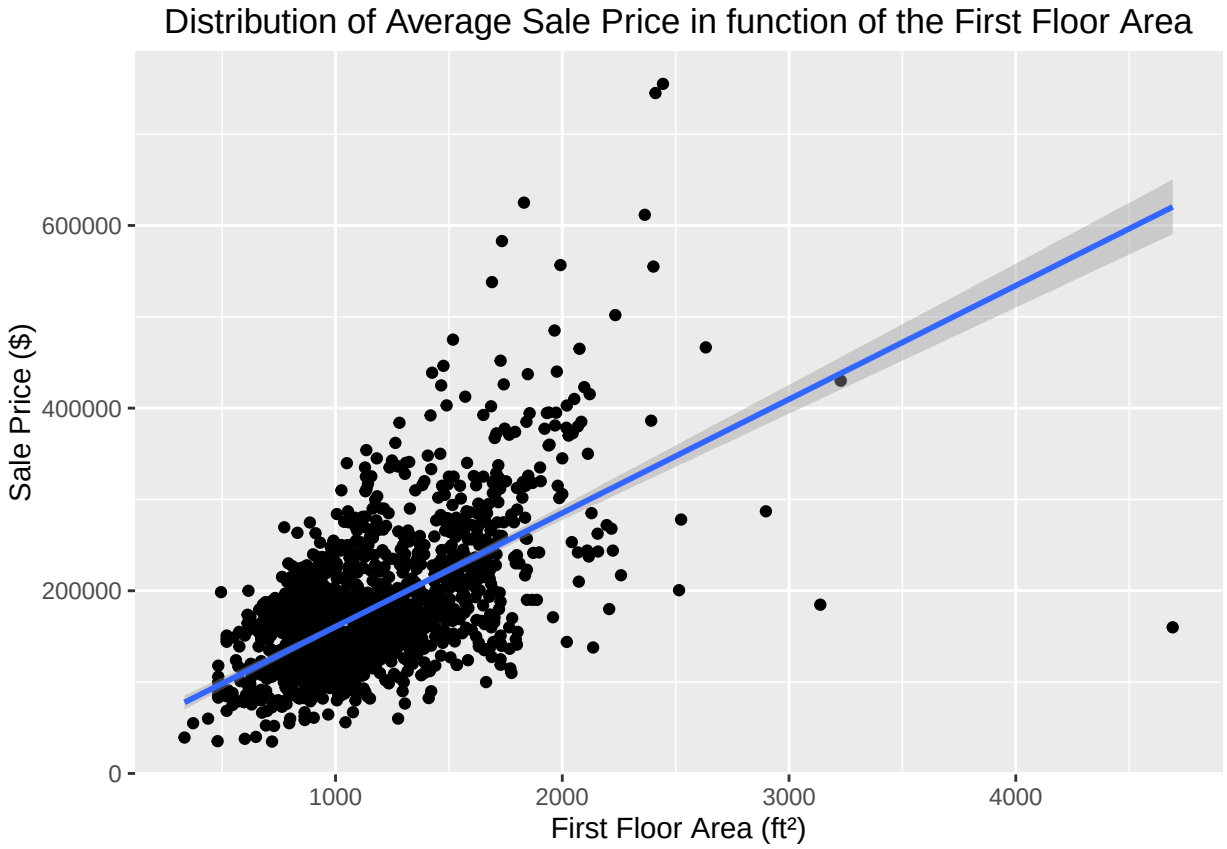


Total Basement Area

We precise that a basement area of 0 ft² means that there is no basement in the house.


Distribution of Average Sale Price in function of the Total Basement Area
800000
600000
400000
200000
0
Sale Price ($)
0
2000
4000
6000
Total Basement Area (ft²)

First Floor Area



Feature Selection

In this section, we will use 2 methods to select features of our dataset. We will check which features are strongly correlated and remove them if the correlation coefficient is above a certain threshold (which will mean strongly correlated). Then, we will rank the features by importance by using the XGBoost algorithm to train the model and estimate the features importance.

Correlation

```
# features.remove <- findCorrelation(cor(correlations), cutoff = 0.75, verbose = TRUE)
#
# if(length(features.remove) > 0)
# {
#     cat("Features strongly correlated removed: ", colnames(dataset)[features.remove], sep = "\n")
#     cat("Total features removed:", length(dataset[, features.remove]))
#
#     dataset <- subset(dataset, select = -features.remove)
# }
# str(dataset)
```

Ranking Features by Importance

To rank the features by importance, we need to transform every categorical features to integers. The train set must be a numeric matrix which will be used as input to the algorithm. The NA values will be 0 and the categorical features will be 1-base.

```
features.string <- which(sapply(dataset, is.character))
setDT(dataset)

for(feature in features.string)
{
    set(dataset, i = NULL, j = feature, value = as.numeric(factor(dataset[[feature]])))
}

dataset[is.na(dataset), ] <- 0

train <- dataset[dataset$SalePrice > -1, ]
test <- dataset[dataset$SalePrice == -1, ]

sale.price <- train$SalePrice
train$SalePrice <- NULL
test.id <- test$Id
test$Id <- NULL
train$Id <- NULL
```

Now, we train the model with the random forest algorithm and rank features by importance using the recursive feature elimination method.

```
# # define the control using a random forest selection function
# control <- rfeControl(functions = rfFuncs, method = "cv", number = 5)
# # run the RFE algorithm
# results <- rfe(train, sale.price, sizes = c(1:ncol(train)), rfeControl = control)
# # summarize the results
# print(results)
# # list the chosen features
# features.selected <- predictors(results)
# print(features.selected)
# # plot the results
# plot(results, type = c("g", "o"))
# # keep only the selected features
# dataset <- dataset %>%
#     subset(select = c(features.selected, "SalePrice"))

x <- c("GrLivArea", "OverallQual", "TotalBsmtSF", "SecondFlrSF",
"FirstFlrSF",  "GarageCars", "YearBuilt", "BsmtFinSF1",
"GarageArea",  "LotArea", "MSZoning", "ExterQual",
"Neighborhood", "MSSubClass", "CentralAir", "FireplaceQu",
"Fireplaces",  "GarageYrBlt", "KitchenQual", "GarageType",
"BsmtQual", "FullBath", "GarageFinish", "BldgType",
"BsmtUnfSF", "HalfBath", "BedroomAbvGr", "BsmtFullBath",
"BsmtFinType1", "OpenPorchSF", "HeatingQC", "KitchenAbvGr",
"BsmtCond", "TotRmsAbvGrd", "GarageCond", "WoodDeckSF",
"GarageQual", "Foundation", "SalePrice")
```

```
dataset <- dataset %>%
    subset(select = x)
```

Feature Engineering

In this section, we create new features depending on the selected ones to help prediction.

```
train <- dataset[dataset$SalePrice > -1, ]
test <- dataset[dataset$SalePrice == -1, ]

train$SalePrice <- NULL
test$SalePrice <- NULL
gc()
```

```
##           used (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 1752737 93.7    2637877 140.9  2637877 140.9
## Vcells 2585963 19.8    4701432  35.9  4701418  35.9
```

Models Building

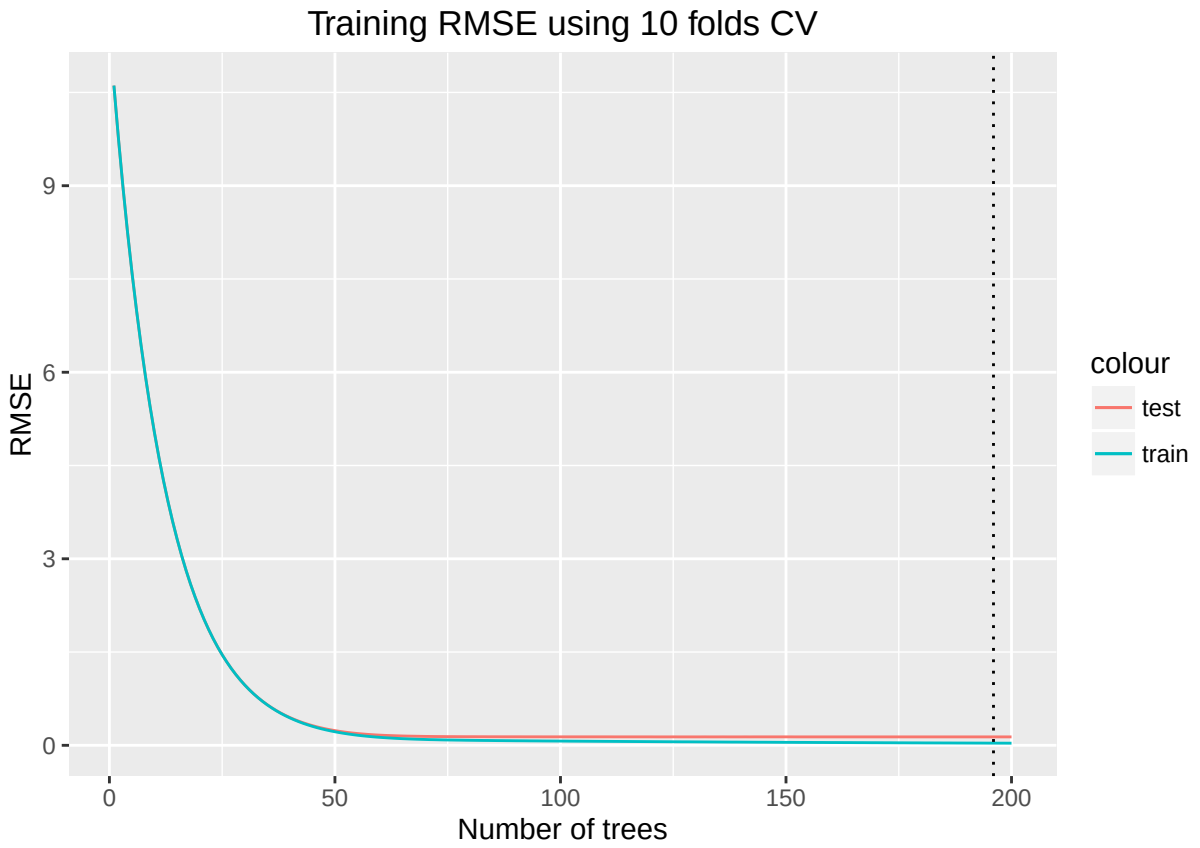
In this section, we train different models and give predictions on the sale price of each house. We will use the random forest and the extreme gradient boosting trees algorithms to build models.

Extreme Gradient Boosted Regression Trees

We proceed to a 5-fold cross-validation to get the optimal number of trees and the RMSE score which is the metric used for the accuracy of our model. We use randomly subsamples representing 80% of the training set. The training set will be split in 5 samples where each sample has 292 observations (activities).

For each tree, we will have the average of 5 error estimates to obtain a more robust estimate of the true prediction error. This is done for all trees and we get the optimal number of trees to use for the test set.

We also display 2 curves indicating the test and train RMSE mean progression. The vertical dotted line is the optimal number of trees. This plot shows if the model overfits or underfits.

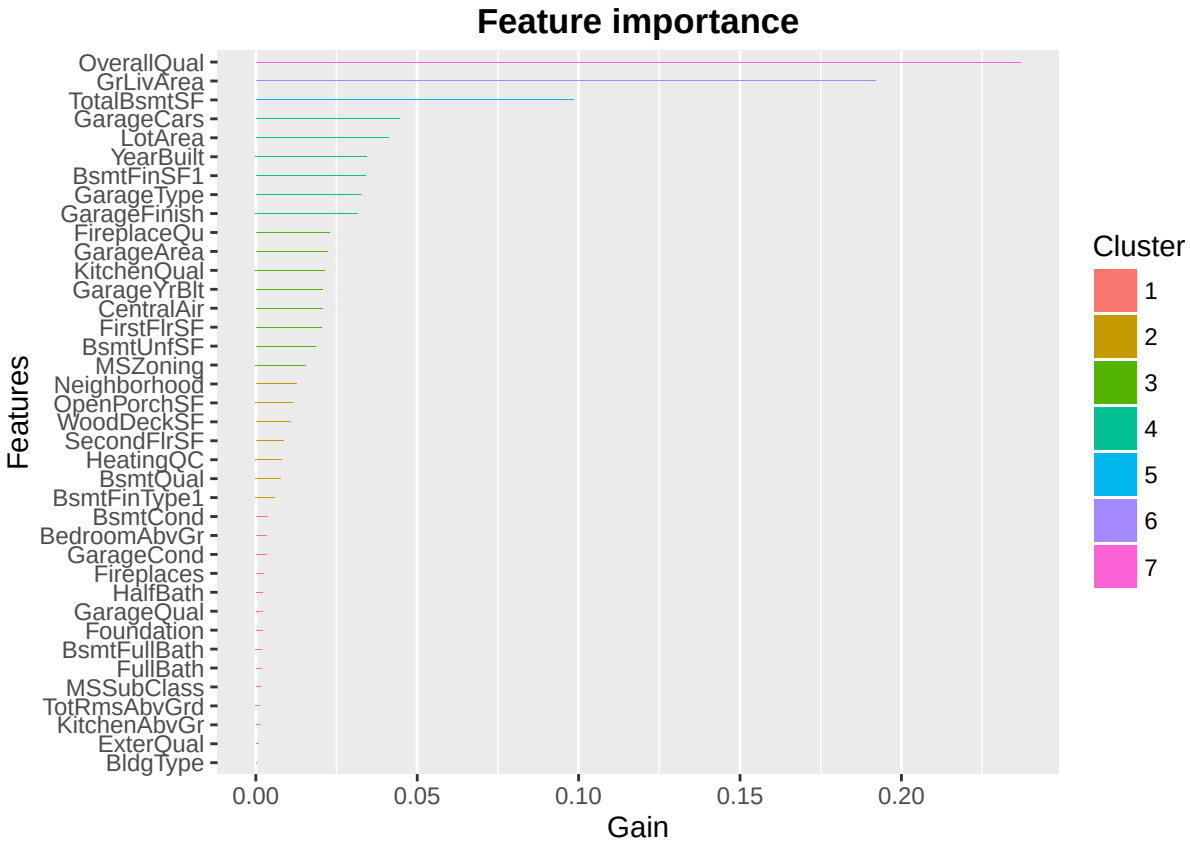



```
##      train.rmse.mean train.rmse.std test.rmse.mean test.rmse.std names
##   1:       10.610395       0.003143      10.610269      0.031763     1
##   2:        9.763560       0.002891       9.763431      0.032045     2
##   3:        8.984507       0.002745       8.984373      0.032202     3
##   4:        8.267750       0.002456       8.267529      0.031530     4
##   5:        7.608304       0.002272       7.608149      0.029814     5
##  ---
## 196:        0.035131       0.001986       0.135270      0.011101   196
## 197:        0.034845       0.001993       0.135320      0.011086   197
## 198:        0.034633       0.001964       0.135319      0.011044   198
## 199:        0.034318       0.001898       0.135299      0.011074   199
## 200:        0.034110       0.001917       0.135294      0.011064   200

##
## Optimal testing set RMSE score: 0.13527

##
## Associated training set RMSE score: 0.035131

##
## Interval testing set RMSE score: [ 0.124169 ,  0.146371 ].

##
## Difference between optimal training and testing sets RMSE: 0.100139
```

```
##
## Optimal number of trees: 196
```

Using the optimal number of trees given by the cross-validation, we can build the model using the test set as input.

```
##          Feature         Gain        Cover   Frequence
##  1:  OverallQual 0.2369048636 0.0564552128 0.032992237
##  2:    GrLivArea 0.1919220789 0.0912498930 0.075688073
##  3:  TotalBsmtSF 0.0983255068 0.0603373543 0.058045166
##  4:   GarageCars 0.0445300959 0.0043736274 0.005469301
##  5:      LotArea 0.0410493946 0.0787611107 0.080098800
##  6:    YearBuilt 0.0343994563 0.0335450724 0.039872971
##  7:   BsmtFinSF1 0.0338856929 0.0561016878 0.068983769
##  8:   GarageType 0.0323496785 0.0127388366 0.009350741
##  9: GarageFinish 0.0316534593 0.0033150425 0.010938603
## 10:  FireplaceQu 0.0228280084 0.0129252165 0.015878617
## 11:   GarageArea 0.0222012221 0.0586884557 0.070395201
## 12:  KitchenQual 0.0214625037 0.0188210574 0.009174312
## 13:  GarageYrBlt 0.0207411076 0.0495631347 0.050282287
## 14:   CentralAir 0.0205623628 0.0051741989 0.003175723
## 15:   FirstFlrSF 0.0203194674 0.0553939747 0.054693013
## 16:    BsmtUnfSF 0.0185178710 0.0817591082 0.094565984
## 17:     MSZoning 0.0155398121 0.0316222417 0.012350035
## 18: Neighborhood 0.0125638915 0.0428833043 0.040755116
## 19:  OpenPorchSF 0.0115313545 0.0417643614 0.056986591
## 20:   WoodDeckSF 0.0106905063 0.0422465614 0.050635145
## 21:  SecondFlrSF 0.0085636445 0.0374331835 0.040402258
## 22:    HeatingQC 0.0080748860 0.0083605663 0.015172900
## 23:     BsmtQual 0.0078266783 0.0040897463 0.005998589
## 24: BsmtFinType1 0.0059616798 0.0165466915 0.013232181
## 25:     BsmtCond 0.0036428801 0.0121750539 0.007233592
## 26: BedroomAbvGr 0.0034030213 0.0132018017 0.014114326
## 27:   GarageCond 0.0032685350 0.0040592357 0.002646436
## 28:   Fireplaces 0.0024745172 0.0043212288 0.005822159
## 29:     HalfBath 0.0021041321 0.0043809234 0.005998589
## 30:   GarageQual 0.0019999047 0.0045202109 0.002293578
## 31:   Foundation 0.0019849877 0.0080110210 0.006704305
## 32: BsmtFullBath 0.0019316070 0.0124390368 0.007939308
## 33:     FullBath 0.0017139982 0.0139360458 0.005292872
## 34:   MSSubClass 0.0015121308 0.0032951442 0.009527170
## 35: TotRmsAbvGrd 0.0012725127 0.0057963497 0.010056457
## 36: KitchenAbvGr 0.0012163107 0.0062420697 0.002293578
## 37:    ExterQual 0.0008839843 0.0026040129 0.003705011
## 38:     BldgType 0.0001862554 0.0008682254 0.001235004
##          Feature         Gain        Cover   Frequence
```



Random Forest

Root Mean Square Error (RMSE)

We can verify how our predictions score under the RMSE score. We take our predictions applied to the train set and we compare to the real outcome values of the train set.

```
## RMSE = 0.03578788
```

Results

We write the 'activity_id' associated to the predicted outcome in the submission file.

```
submission <- data.frame(Id = test.id, SalePrice = prediction.test)
write.csv(submission, "Submission.csv", row.names = FALSE)
```

Conclusion