Kaggle House Pricing Prediction

Rahul Singh

Contents

Introduction

Feature engineering is a very important part of the process of developing prediction models. It is considered, by many authors, an art, and it involves human-driven design and intuition.

The experimental dataset we are going to use is the House Prices Dataset. It includes 79 explanatory variables of residential homes. For more details on the dataset and the competition see https://www.kaggle.com/c/house-prices-advanced-regression-techniques.

What is my goal?

- I want to predict predict the final price of each home (Therefore, this is a regression task)
- I have to use the feature engineering techniques explained in class to transform the dataset.

Data Reading and preparation

The dataset is offered in two separated fields, one for the training and another one for the test set.

```
training_data = read.csv(file = file.path("~/Desktop/IE MBD Term2/Machine Learning 2/CODES/Practice 1/H
test_data = read.csv(file = file.path("~/Desktop/IE MBD Term2/Machine Learning 2/CODES/Practice 1/House
```

As some of you asked me in class, you do not have to apply the Feature Engineering process two times (once for training and once for test). You just have to join both datasets (using the `rbind` function), apply your FE and then split the datasets again.

However, if we try to do join the two dataframes as they are, we will get an error because they do not have the same columns: `test_data` does not have a column `SalePrice`. Therefore, we first create this column in the test set and then we join the data

```
test_data$SalePrice <- 0
dataset <- rbind(training_data, test_data)
```

Feature Engineering

Hunting NAs

Our dataset is filled with missing values, therefore, before we can build any predictive model we'll clean our data by filling in all NA's with more appropriate values.

Counting columns with null values.

```
na.cols <- which(colSums(is.na(dataset)) > 0)
sort(colSums(sapply(dataset[na.cols], is.na)), decreasing = TRUE)
```

```
##       PoolQC  MiscFeature        Alley        Fence  FireplaceQu
##         2909         2814         2721         2348         1420
##  LotFrontage  GarageYrBlt GarageFinish   GarageQual   GarageCond
##          486          159          159          159          159
##   GarageType     BsmtCond BsmtExposure     BsmtQual BsmtFinType2
##          157           82           82           81           80
## BsmtFinType1   MasVnrType   MasVnrArea     MSZoning    Utilities
##           79           24           23            4            2
## BsmtFullBath BsmtHalfBath   Functional  Exterior1st  Exterior2nd
##            2            2            2            1            1
##   BsmtFinSF1   BsmtFinSF2    BsmtUnfSF  TotalBsmtSF   Electrical
##            1            1            1            1            1
##  KitchenQual   GarageCars   GarageArea     SaleType
##            1            1            1            1
```

```
paste('There are', length(na.cols), 'columns with missing values')
```

```
## [1] "There are 34 columns with missing values"
```

First, we deal with numerical values. According to the documentation we can safely assume that `NAs` in these variables means 0.

```
# LotFrontage : NA most likely means no lot frontage
dataset$LotFrontage[is.na(dataset$LotFrontage)] <- 0
dataset$MasVnrArea[is.na(dataset$MasVnrArea)] <- 0
dataset$BsmtFinSF1[is.na(dataset$BsmtFinSF1)] <- 0
dataset$BsmtFinSF2[is.na(dataset$BsmtFinSF2)] <- 0
dataset$BsmtUnfSF[is.na(dataset$BsmtUnfSF)] <- 0
dataset$TotalBsmtSF[is.na(dataset$TotalBsmtSF)] <- 0
dataset$BsmtFullBath[is.na(dataset$BsmtFullBath)] <- 0
```

```
dataset$BsmtHalfBath[is.na(dataset$BsmtHalfBath)] <- 0
dataset$GarageCars[is.na(dataset$GarageCars)] <- 0
dataset$GarageArea[is.na(dataset$GarageArea)] <- 0
```

One special case is the variable "GarageYrBlt". We can assume that the year that the garage was built is the same than when the house itself was built.

```
dataset$GarageYrBlt[is.na(dataset$GarageYrBlt)] <- dataset$YearBuilt[is.na(dataset$GarageYrBlt)]
```

We even have a garage from the future (built in 2207).

```
summary(dataset$GarageYrBlt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1872    1958    1978    1976    2001    2207
```

Besides the curiosity (which is obviously a type), you have to always inspect your dataset. Such kind of mistakes are more usual than one would like. Let's fix it

```
dataset$GarageYrBlt[dataset$GarageYrBlt==2207] <- 2007
```

Now we deal with `NAs` in categorical values.

`NAs` in this dataset might be due to: 1) Missing data or 2) Empty values for this feature (for instance, a house does not have Garage).

Firstly, we'll address "real" NAs, that is, values which are actually missing. To that end, we will impute them with the most common value for this feature (i.e., the mode of the value)

```
# For those values without meaning for NA values, we replace them with the mode value
dataset$KitchenQual[is.na(dataset$KitchenQual)] <- names(sort(-table(dataset$KitchenQual)))[1]
dataset$MSZoning[is.na(dataset$MSZoning)] <- names(sort(-table(dataset$MSZoning)))[1]
dataset$SaleType[is.na(dataset$SaleType)] <- names(sort(-table(dataset$SaleType)))[1]
dataset$Exterior1st[is.na(dataset$Exterior1st)] <- names(sort(-table(dataset$Exterior1st)))[1]
dataset$Exterior2nd[is.na(dataset$Exterior2nd)] <- names(sort(-table(dataset$Exterior2nd)))[1]
dataset$Functional[is.na(dataset$Functional)] <- names(sort(-table(dataset$Functional)))[1]
```

For empty values, we just change the `NA` value to a new value (No, None, etc...) meaning that the house does not have such feature, thus avoiding later errors.

```
# For the rest we change NAs to their actual meaning
dataset$Alley = factor(dataset$Alley, levels=c(levels(dataset$Alley), "None"))
dataset$Alley[is.na(dataset$Alley)] = "None"

# Bsmt : NA for basement features is "no basement"
dataset$BsmtQual = factor(dataset$BsmtQual, levels=c(levels(dataset$BsmtQual), "No"))
dataset$BsmtQual[is.na(dataset$BsmtQual)] = "No"

dataset$BsmtCond = factor(dataset$BsmtCond, levels=c(levels(dataset$BsmtCond), "No"))
dataset$BsmtCond[is.na(dataset$BsmtCond)] = "No"

dataset$BsmtExposure[is.na(dataset$BsmtExposure)] = "No"

dataset$BsmtFinType1 = factor(dataset$BsmtFinType1, levels=c(levels(dataset$BsmtFinType1), "No"))
dataset$BsmtFinType1[is.na(dataset$BsmtFinType1)] = "No"

dataset$BsmtFinType2 = factor(dataset$BsmtFinType2, levels=c(levels(dataset$BsmtFinType2), "No"))
dataset$BsmtFinType2[is.na(dataset$BsmtFinType2)] = "No"
```

```
# Fence : NA means "no fence"
dataset$Fence = factor(dataset$Fence, levels=c(levels(dataset$Fence), "No"))
dataset$Fence[is.na(dataset$Fence)] = "No"

# FireplaceQu : NA means "no fireplace"
dataset$FireplaceQu = factor(dataset$FireplaceQu, levels=c(levels(dataset$FireplaceQu), "No"))
dataset$FireplaceQu[is.na(dataset$FireplaceQu)] = "No"

# Garage : NA for garage features is "no garage"
dataset$GarageType = factor(dataset$GarageType, levels=c(levels(dataset$GarageType), "No"))
dataset$GarageType[is.na(dataset$GarageType)] = "No"

dataset$GarageFinish = factor(dataset$GarageFinish, levels=c(levels(dataset$GarageFinish), "No"))
dataset$GarageFinish[is.na(dataset$GarageFinish)] = "No"

dataset$GarageQual = factor(dataset$GarageQual, levels=c(levels(dataset$GarageQual), "No"))
dataset$GarageQual[is.na(dataset$GarageQual)] = "No"

dataset$GarageCond = factor(dataset$GarageCond, levels=c(levels(dataset$GarageCond), "No"))
dataset$GarageCond[is.na(dataset$GarageCond)] = "No"

# MasVnrType : NA most likely means no veneer
dataset$MasVnrType[is.na(dataset$MasVnrType)] = "None"

# MiscFeature : NA = "no misc feature"
dataset$MiscFeature = factor(dataset$MiscFeature, levels=c(levels(dataset$MiscFeature), "No"))
dataset$MiscFeature[is.na(dataset$MiscFeature)] = "No"

# PoolQC : data description says NA means "no pool"
dataset$PoolQC = factor(dataset$PoolQC, levels=c(levels(dataset$PoolQC), "No"))
dataset$PoolQC[is.na(dataset$PoolQC)] = "No"

# Electrical : NA means "UNK"
dataset$Electrical = factor(dataset$Electrical, levels=c(levels(dataset$Electrical), "UNK"))
dataset$Electrical[is.na(dataset$Electrical)] = "UNK"

# GarageYrBlt: It seems reasonable that most houses would build a garage when the house itself was buil
idx <- which(is.na(dataset$GarageYrBlt))
dataset[idx, 'GarageYrBlt'] <- dataset[idx, 'YearBuilt']
```

We now check again if we have null values

```
na.cols <- which(colSums(is.na(dataset)) > 0)
paste('There are now', length(na.cols), 'columns with missing values')
```

```
## [1] "There are now 1 columns with missing values"
```

Perfect! We have fixed all `NAs`

Data Cleaning

We remove meaningless features and incomplete cases

```
dataset <- dataset[,-which(names(dataset) == "Utilities")]
dataset <- dataset[,-which(names(dataset) == "Id")]
```

Outliers

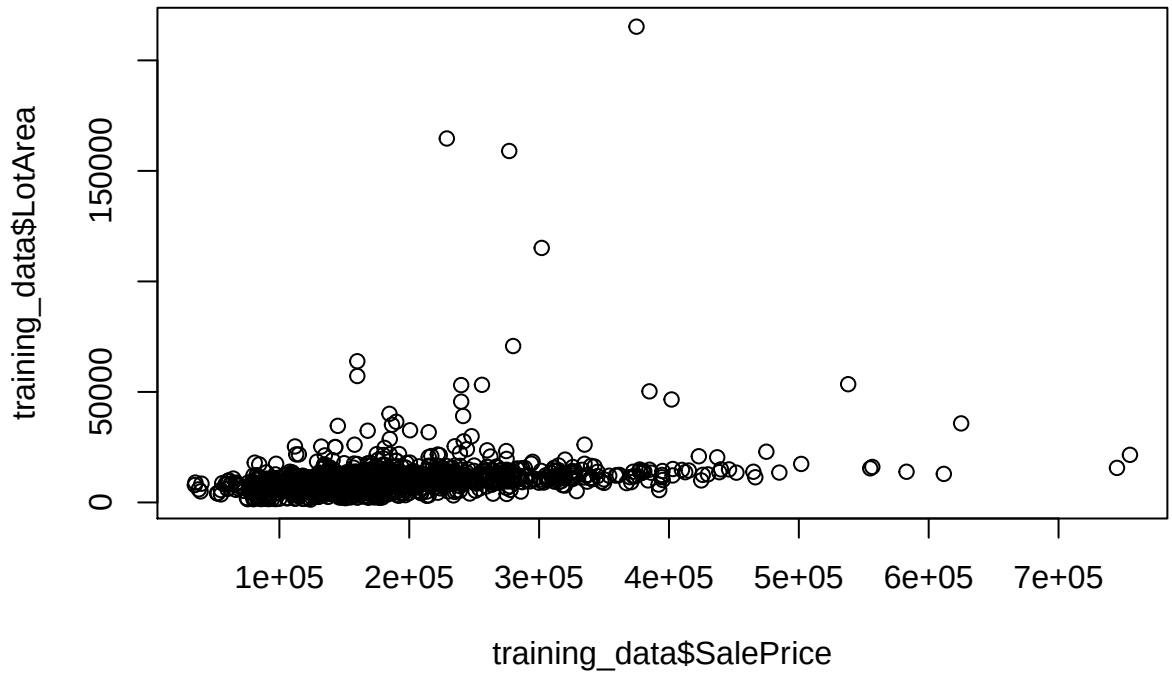
We will now focus on numerical values. If `NAs` where our enemy in categorical values, the main problem with numerical values are outliers (values which largely differ from the rest). Outliers can mislead the training of our models resulting in less accurate models and ultimately poorer results.

In this section we will seek to identify outliers to then properly deal with them. If we summarize the dataset, we can see variables which "Max." is much larger than the rest of values. These are features susceptible of containing outliers.

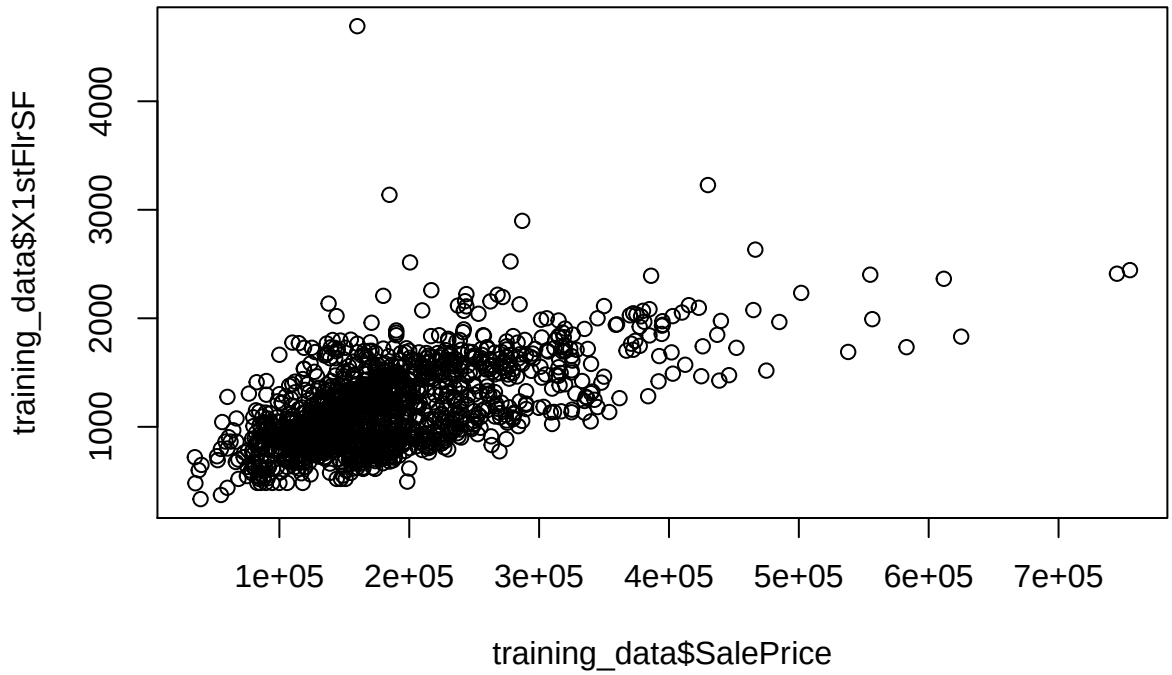
Let's plot these features against the target variable

```
plot(training_data$SalePrice, training_data$GrLivArea)
```

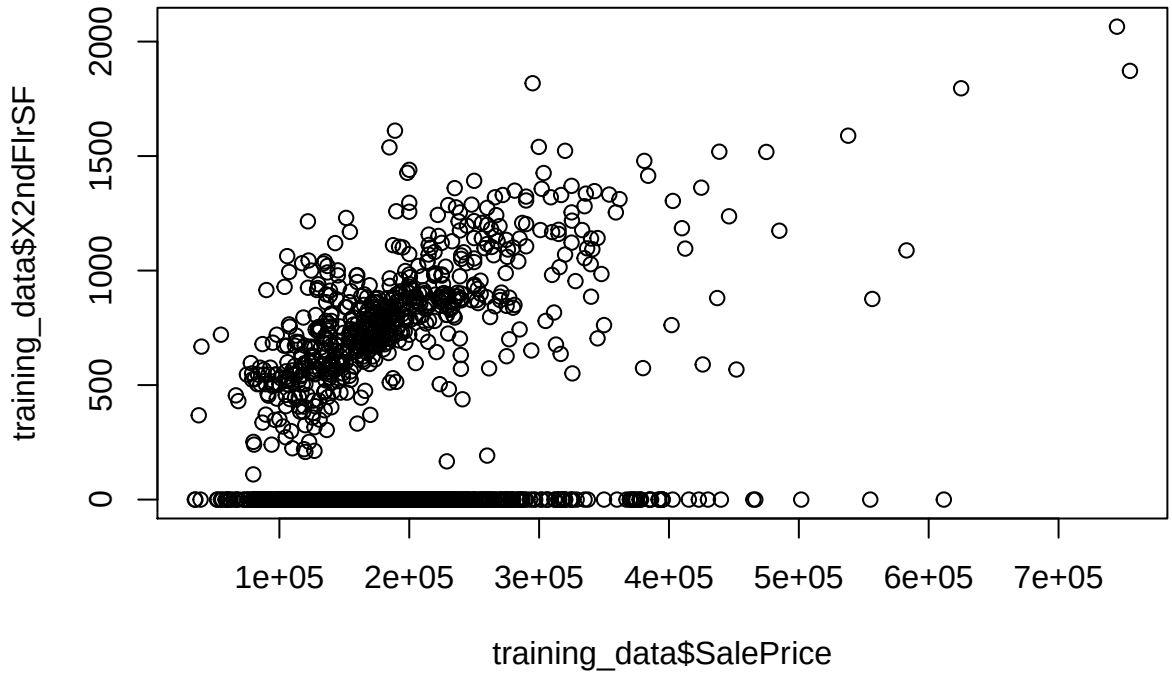

```
plot(training_data$SalePrice, training_data$LotArea)
```



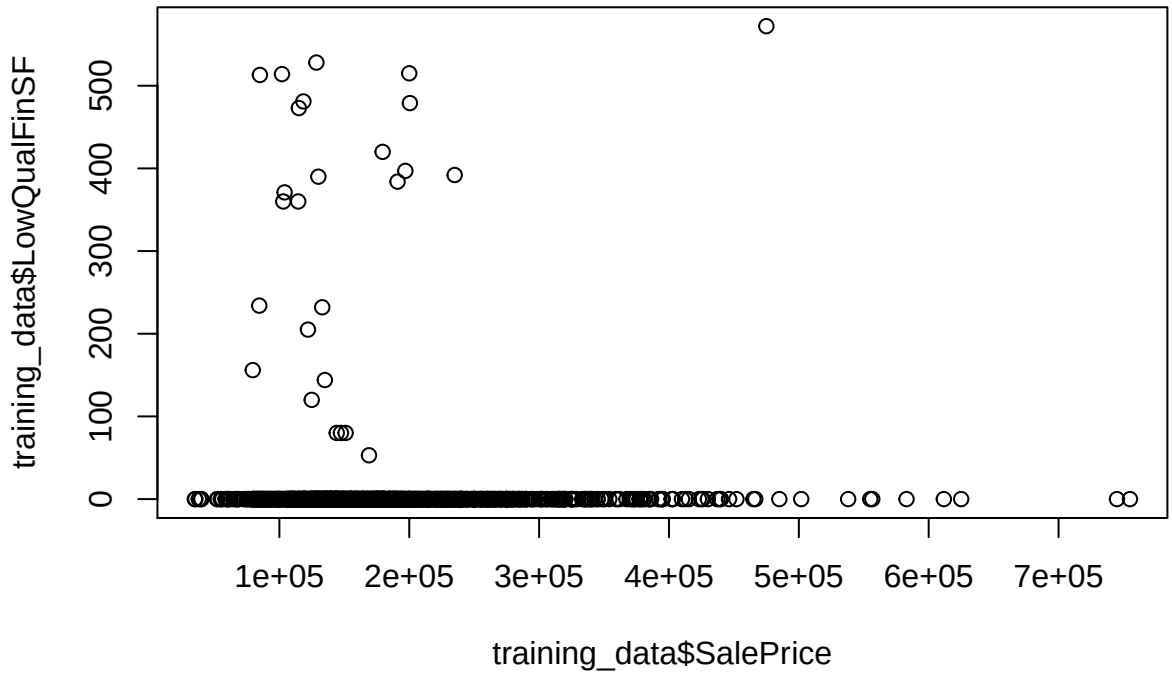

```
plot(training_data$SalePrice, training_data$X1stFlrSF)
```



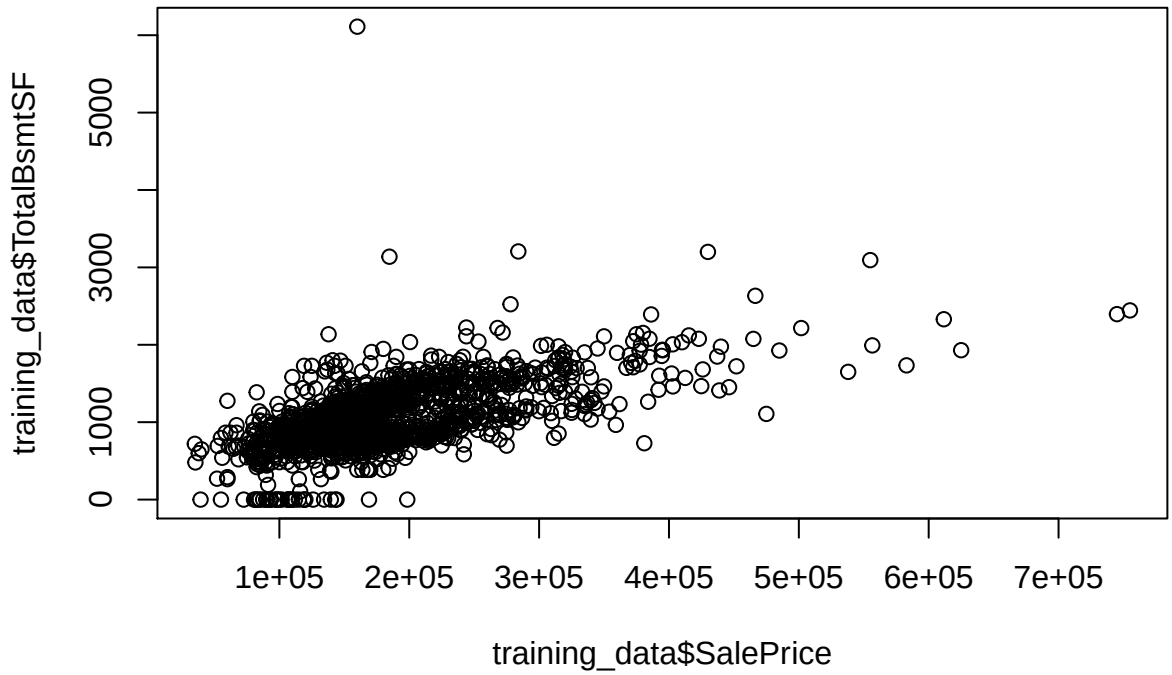

```
plot(training_data$SalePrice, training_data$X2ndFlrSF)
```



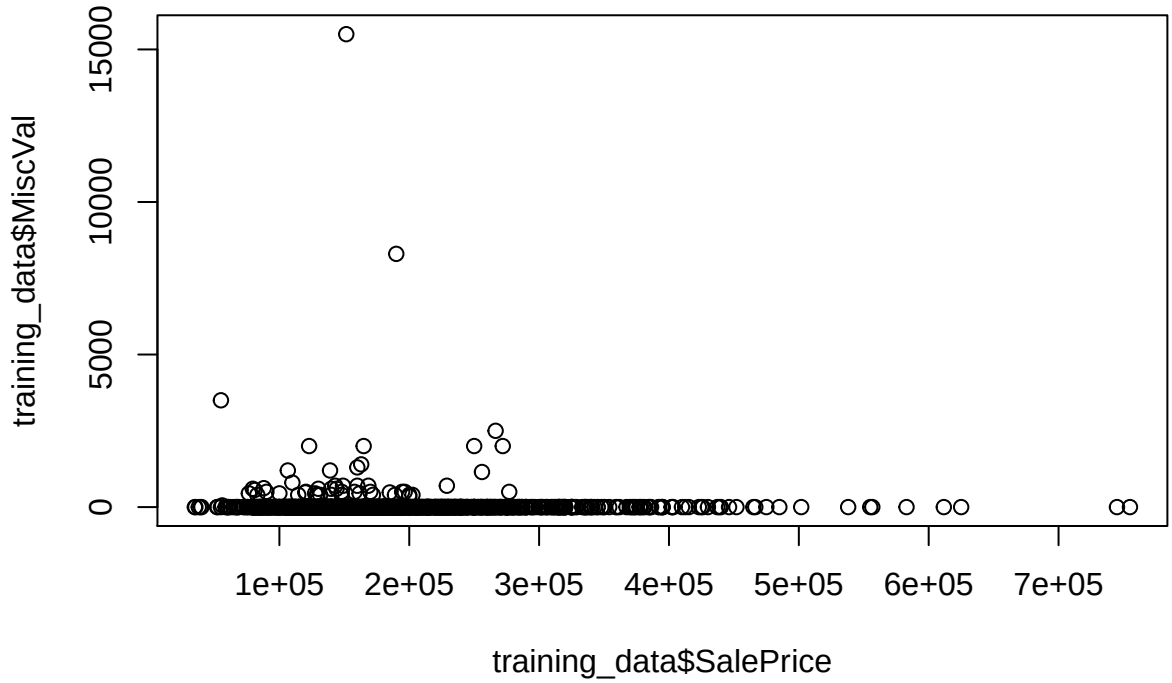

```
plot(training_data$SalePrice, training_data$LowQualFinSF)
```




```
plot(training_data$SalePrice, training_data$TotalBsmtSF)
```




```
plot(training_data$SalePrice, training_data$MiscVal)
```



By reviewing the plots we see that `X2ndFlrSF` does not have significant outliers. `MiscVal` and `LowQualFinSF` does not present outliers as such. They are 0 for most of the houses, but they do have values for some of them. Therefore, they might be valuable to predict the price of these houses.

We transform the rest of the outliers by assigning them the mean of each variable. More complex imputations can be also applied. For instance, fit a regression model or a decision tree to predict the value of the outliers based on the rest of the features. Nevertheless, for the majority of cases, simple median or mean based imputations usually perform good enough and there is no need for more complex models. Only if your dataset is highly affected by outliers, you should propose a more sophisticated approach.

```
dataset$GrLivArea[dataset$GrLivArea>4000] <- mean(dataset$GrLivArea)%>%as.numeric
dataset$LotArea[dataset$LotArea>35000] <- mean(dataset$LotArea)%>%as.numeric
dataset$X1stFlrSF[dataset$X1stFlrSF>3000] <- mean(dataset$X1stFlrSF)%>%as.numeric
dataset$TotalBsmtSF[dataset$TotalBsmtSF>2900] <- mean(dataset$TotalBsmtSF)%>%as.numeric
```

Character variables into continuous numerical variables

There are some categories which are clearly a ranking (they goes from None to Excellent for instance). If we leave them as categories (factors) this ranking is lost. Excellent will have the same weight as None. To codify this ranking, we are going to recode these features as numerical, assigning larger values to higher-ranking categories.

```
dataset$ExterQual<- recode(dataset$ExterQual,"None"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$ExterCond<- recode(dataset$ExterCond,"None"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$BsmtQual<- recode(dataset$BsmtQual,"No"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$BsmtCond<- recode(dataset$BsmtCond,"No"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$BsmtExposure<- recode(dataset$BsmtExposure,"No"=0,"No"=1,"Mn"=2,"Av"=3,"Gd"=6)
dataset$BsmtFinType1<- recode(dataset$BsmtFinType1,"No"=0,"Unf"=1,"LwQ"=2,"Rec"=3,"BLQ"=4,"ALQ"=5,"GLQ"=
dataset$BsmtFinType2<- recode(dataset$BsmtFinType2,"No"=0,"Unf"=1,"LwQ"=2,"Rec"=3,"BLQ"=4,"ALQ"=5,"GLQ"=
dataset$HeatingQC<- recode(dataset$HeatingQC,"None"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=5)
dataset$KitchenQual<- recode(dataset$KitchenQual,"None"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$Functional<- recode(dataset$Functional,"None"=0,"Sev"=1,"Maj2"=2,"Maj1"=3,"Mod"=4,"Min2"=5,"Mini
dataset$FireplaceQu<- recode(dataset$FireplaceQu,"No"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$GarageFinish<- recode(dataset$GarageFinish,"No"=0,"Unf"=1,"RFn"=2,"Fin"=3)
dataset$GarageQual<- recode(dataset$GarageQual,"No"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$GarageCond<- recode(dataset$GarageCond,"No"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$PoolQC<- recode(dataset$PoolQC,"No"=0,"Po"=1,"Fa"=2,"TA"=3,"Gd"=4,"Ex"=6)
dataset$Fence<- recode(dataset$Fence,"No"=0,"MnWw"=1,"GdWo"=2,"MnPrv"=3,"GdPrv"=6)
```

In addition to capture their ranking meaning, we are going to create a new binary feature for each one of them to reward good and penalize bad qualities and conditions

```
dataset['IsExterQualBad'] <- ifelse(dataset$ExterQual< 3, 1, 0)
dataset['IsExterCondlBad'] <- ifelse(dataset$ExterCond< 3, 1, 0)
dataset['IsBsmtQualBad'] <- ifelse(dataset$BsmtQual< 3, 1, 0)
dataset['IsBsmtCondBad'] <- ifelse(dataset$BsmtCond< 3, 1, 0)
dataset['IsBsmtExposureBad'] <- ifelse(dataset$BsmtExposure< 3, 1, 0)
dataset['IsHeatingQCBad'] <- ifelse(dataset$HeatingQC< 3, 1, 0)
dataset['IsKitchenQualBad'] <- ifelse(dataset$KitchenQual< 3, 1, 0)
dataset['IsFireplaceQuBad'] <- ifelse(dataset$FireplaceQu< 3, 1, 0)
dataset['IsGarageQualBad'] <- ifelse(dataset$GarageQual< 3, 1, 0)
dataset['IsGarageCondBad'] <- ifelse(dataset$GarageCond< 3, 1, 0)
dataset['IsPoolQCBad'] <- ifelse(dataset$PoolQC< 3, 1, 0)

dataset['IsExterQualGood'] <- ifelse(dataset$ExterQual >= 3, 1, 0)
dataset['IsExterCondlGood'] <- ifelse(dataset$ExterCond >= 3, 1, 0)
dataset['IsBsmtQualGood'] <- ifelse(dataset$BsmtQual >= 3, 1, 0)
dataset['IsBsmtCondGood'] <- ifelse(dataset$BsmtCond >= 3, 1, 0)
dataset['IsBsmtExposureGood'] <- ifelse(dataset$BsmtExposure >= 3, 1, 0)
dataset['IsHeatingQCGood'] <- ifelse(dataset$HeatingQC >= 3, 1, 0)
dataset['IsKitchenQualGood'] <- ifelse(dataset$KitchenQual >= 3, 1, 0)
dataset['IsFireplaceQuGood'] <- ifelse(dataset$FireplaceQu >= 3, 1, 0)
dataset['IsGarageQualGood'] <- ifelse(dataset$GarageQual >= 3, 1, 0)
dataset['IsGarageCondGood'] <- ifelse(dataset$GarageCond >= 3, 1, 0)
```

```
dataset['IsPoolQCGood'] <- ifelse(dataset$PoolQC >= 3, 1, 0)
```

Adding new features

We are not experts in the real estate domain; however, we can come up with some sensible features to increase the performance of our model. This does not pretend to be an exhaustive list, given that the possibilities are almost infinite, but just an example of the kind of features you can create. In fact, some of you have proposed some interesting features which are not included here and might boost the performance of your models.

```
# Has been the house remodeled?: If the YearBuilt is different than the remodel year
dataset['HasBeenRemodeled'] <- ifelse(dataset$YearRemodAdd == dataset$YearBuilt, 0, 1)

# Has been the house remodelled after the year it was sold?
dataset['HasBeenRecentlyRemodeled'] <- ifelse(dataset$YearRemodAdd == dataset$YrSold, 0, 1)

# Has been the house sold the year it was built
dataset['IsNewHouse'] <-ifelse(dataset$YearBuilt == dataset$YrSold, 1, 0)

# How old it is
dataset['Age'] <- as.numeric(2010 - dataset$YearBuilt)

# Time since last selling
dataset['TimeSinceLastSelling'] <- as.numeric(2010 - dataset$YrSold)

# Time since remodeled and sold
dataset['TimeSinceRemodeledAndSold'] <- as.numeric(dataset$YrSold - dataset$YearRemodAdd)

areas <- c('LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
               'TotalBsmtSF', 'X1stFlrSF', 'X2ndFlrSF', 'GrLivArea', 'GarageArea', 'WoodDeckSF',
               'OpenPorchSF', 'EnclosedPorch', 'X3SsnPorch', 'ScreenPorch', 'LowQualFinSF', 'PoolArea')

# Total surface of the house, combining the area-related features
dataset['TotalSF'] <- as.numeric(rowSums(dataset[,areas]))

# Total surface of the house, combining the total inside surfacae
dataset['TotalInsideSF'] <- as.numeric(dataset$X1stFlrSF + dataset$X2ndFlrSF)

# There are more number of sales in April, May, June and July, which may indicate some stationality. we
dataset['IsHotMonth'] = recode(dataset$MoSold,"1"=0,"2"=0,"3"=0,"4"=1,"5"=1, "6"=1, "7"=1, "8"=0, "9"=0
```

Polynomic degrees of more correlated features

There are some features which are more related to the target variable. We can select them by computing the correlations of all the features to the target variable and select the most correlate. You can also select them by applying some knowledge domain you may have on the data. We can boost their importance by adding some polynomials (the values to the power of 2, 3, the square values...) on their values. This would include some non-linearity in our data that may be able to better predict the target variable. More on polynomial regression here: https://onlinecourses.science.psu.edu/stat501/node/325

I've go the idea from https://www.kaggle.com/hdnguyenfl/advanced-regression-on-house-prices-in-ames and I took the top-performing ones in the notebook, you are free to test some others.

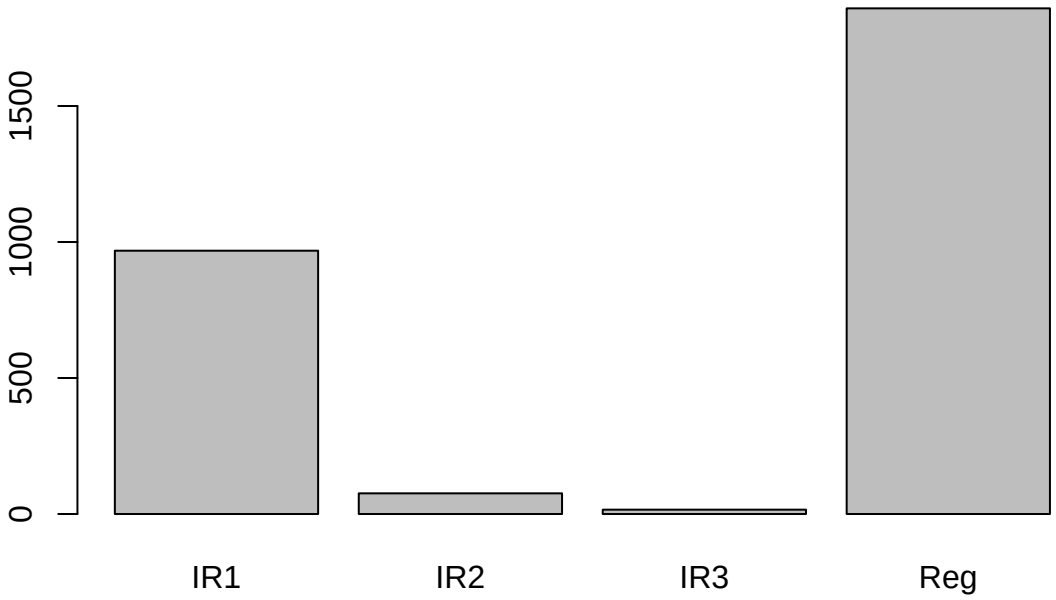
```
dataset["OverallQual-s2"] <- sapply(dataset$OverallQual, function(x){x**2})
dataset["OverallQual-s3"] <- sapply(dataset$OverallQual, function(x){x**3})
dataset["OverallQual-Sq"] <- sqrt(dataset["OverallQual"])
dataset["TotalSF-2"] <- sapply(dataset$TotalSF, function(x){x**2})
dataset["TotalSF-3"] = sapply(dataset$TotalSF, function(x){x**3})
dataset["TotalSF-Sq"] = sqrt(dataset["TotalSF"])
dataset["GrLivArea-2"] = sapply(dataset$GrLivArea, function(x){x**2})
dataset["GrLivArea-3"] = sapply(dataset$GrLivArea, function(x){x**3})
dataset["GrLivArea-Sq"] = sqrt(dataset["GrLivArea"])
dataset["ExterQual-2"] = sapply(dataset$ExterQual, function(x){x**2})
dataset["ExterQual-3"] = sapply(dataset$ExterQual, function(x){x**3})
dataset["ExterQual-Sq"] = sqrt(dataset["ExterQual"])
dataset["GarageCars-2"] = sapply(dataset$GarageCars, function(x){x**2})
dataset["GarageCars-3"] = sapply(dataset$GarageCars, function(x){x**3})
dataset["GarageCars-Sq"] = sqrt(dataset["GarageCars"])
dataset["KitchenQual-2"] = sapply(dataset$KitchenQual, function(x){x**2})
dataset["KitchenQual-3"] = sapply(dataset$KitchenQual, function(x){x**3})
dataset["KitchenQual-Sq"] = sqrt(dataset["KitchenQual"])
```

Binarizing Features

There are some variables that can be encoded as binary because they mostly present a unique value.

For instance, If we take a look to the `LotShape` value distribution, we can see that there are mainly two values: The house has or does not have a regular shape. Therefore, we can binarize these values according to this criteria.

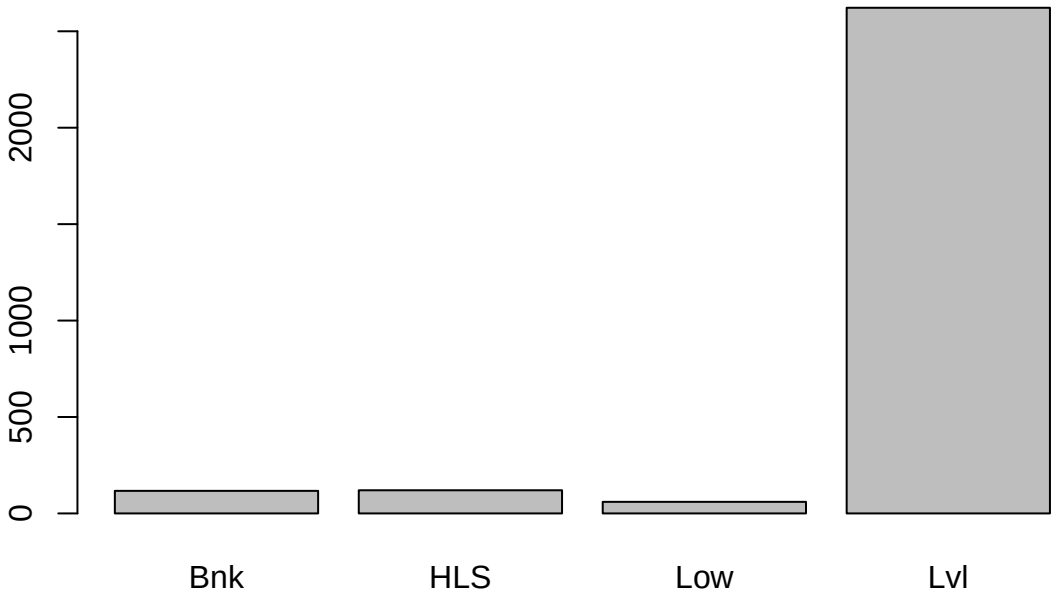
```
plot(dataset$LotShape)
```




```
dataset$IsRegLotShape <- ifelse(dataset$LotShape == 'Reg', 1, 0)
```

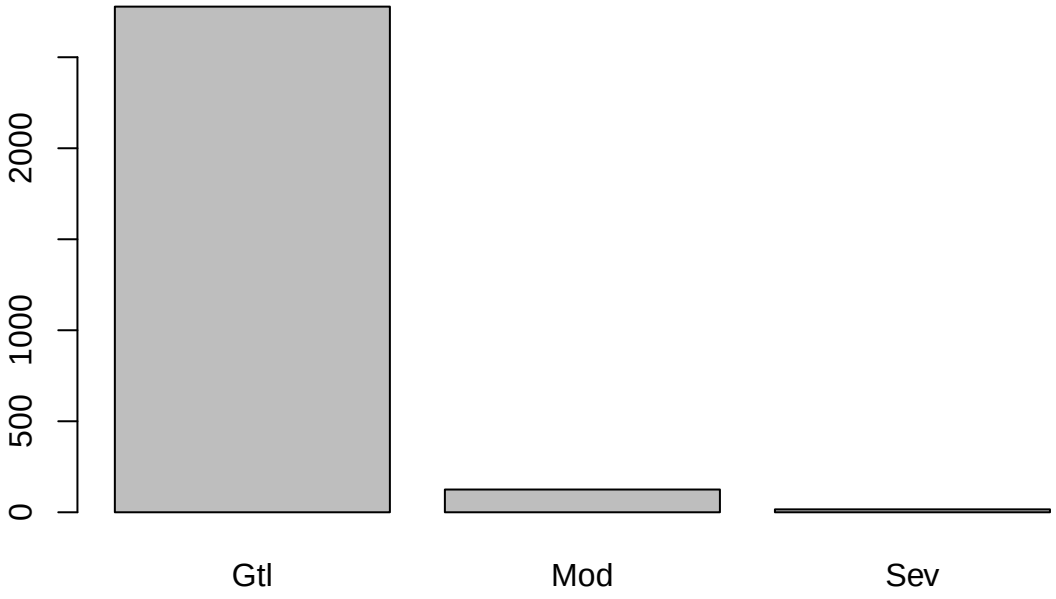
Similarly, we can binarize other values that present the same situation

```
plot(dataset$LandContour)
```



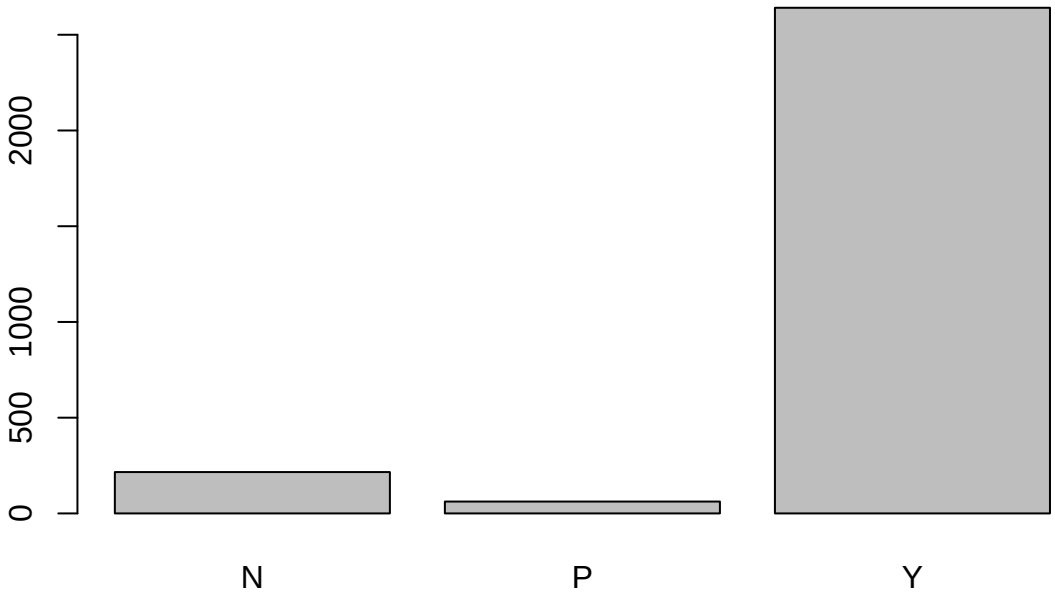

```
dataset['IsLandLvl'] <- ifelse(dataset$LandContour == 'Lvl', 1, 0)
```

```
plot(dataset$LandSlope)
```



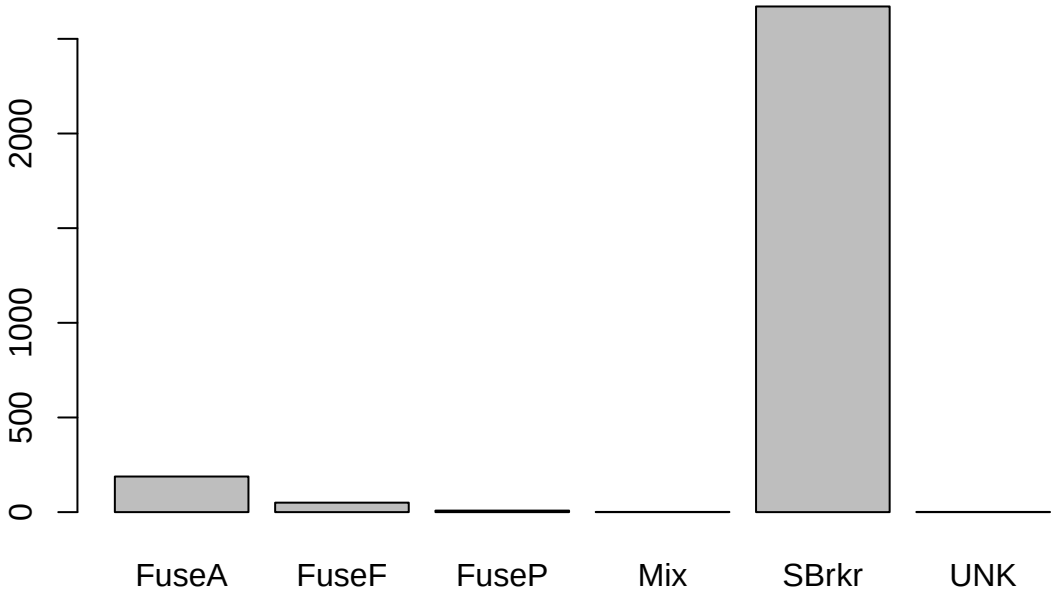

```
dataset['IsLandSlopeGtl'] <-  ifelse(dataset$LandSlope == 'Gtl', 1, 0)
```

```
plot(dataset$PavedDrive)
```




```
dataset['HasPavedDrive'] <-  ifelse(dataset$PavedDrive == 'Y', 1, 0)
```

```
plot(dataset$Electrical)
```




```
dataset['IsElectricalSBrkr'] <- ifelse(dataset$Electrical == 'SBrkr', 1, 0)
```

We can also binarize area-related features (e.g., `X2ndFlrSF` larger than 0 means that the house has a second floor)

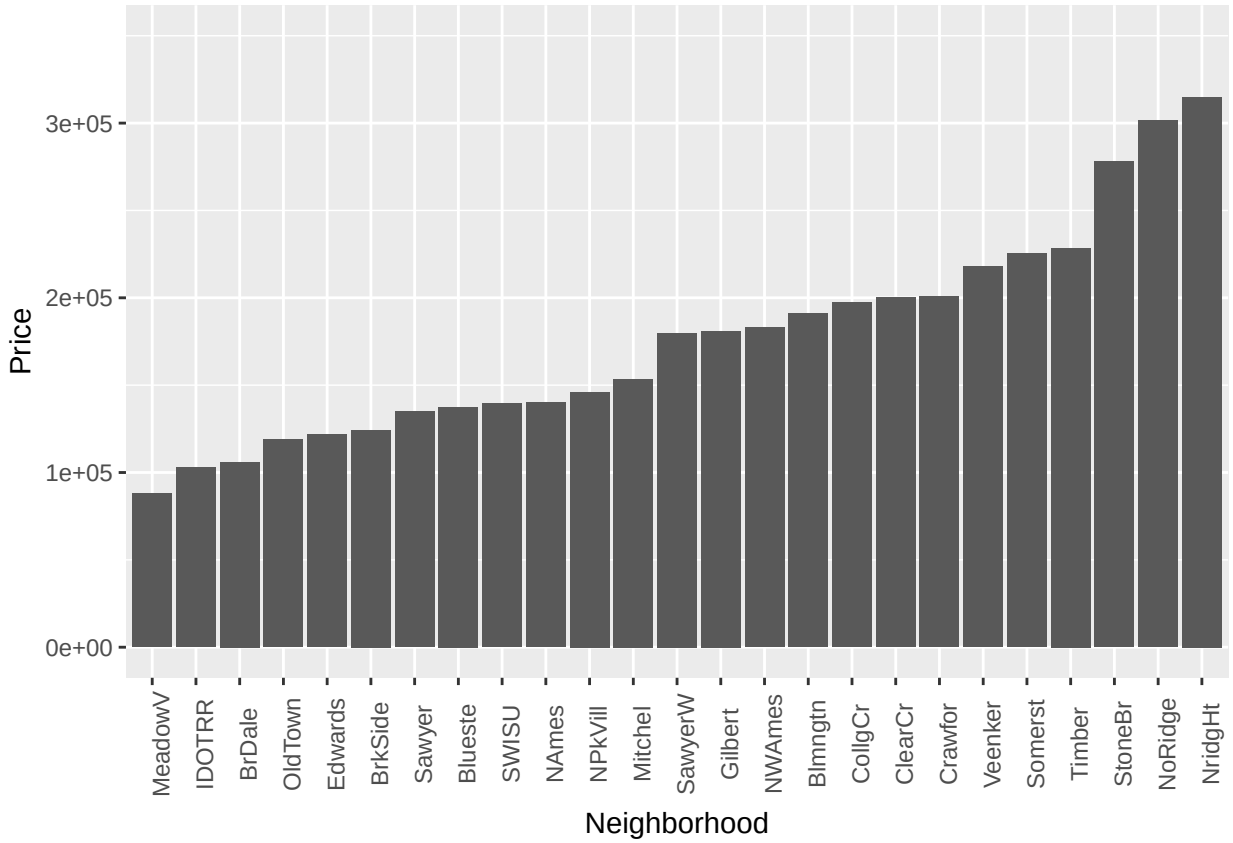
```
##Binary
area_features <- c('X2ndFlrSF', 'MasVnrArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', 'X3SsnPorch

for (area_feature in area_features){
  dataset[str_c('Has',area_feature)] <- ifelse(dataset[,area_feature] != 0, 1, 0)
}
```

Finally, from this kernel https://www.kaggle.com/patjob/real-estate-prediction I've got the idea of the importance of the neighborhood of the house. If we plot the median value of the houses in the neighborhoods,

we can appreciate that there are some "rich" neighborhoods. We expect "rich" neighborhoods to include expensive houses.

Therefore, I've create two new features. A binary feature to indicate if the house is in a rich neighborhood and a numerical feature to codify the ranking of the neighborhoods according to their median house value.

```
training_data[,c('Neighborhood','SalePrice')] %>%
  group_by(Neighborhood) %>%
  summarise(avg = median(SalePrice, na.rm = TRUE)) %>%
  arrange(avg) %>%
  mutate(sorted = factor(Neighborhood, levels=Neighborhood)) %>%
  ggplot(aes(x=sorted, y=avg)) +
  geom_bar(stat = "identity") +
  labs(x='Neighborhood', y='Price') +
  ylim(NA, 350000) +
  theme(axis.text.x = element_text(angle=90))
```




```
richNeighborhood <- c('Crawfor', 'ClearCr', 'Veenker', 'Somerst', 'Timber', 'StoneBr', 'NridgeHt', 'NoR
dataset['IsNeighborhoodRich'] <- (dataset$Neighborhood %in% richNeighborhood) *1
dataset$NeighborhoodScored <- recode(dataset$Neighborhood, 'MeadowV' = 0, 'IDOTRR' = 0, 'Sawyer' = 1, '
```

Factorize features

Some numerical features are actually categories.

```
dataset$MSSubClass <- as.factor(dataset$MSSubClass)
dataset$MoSold <- as.factor(dataset$MoSold)
```

```
dataset$YrSold <- as.factor(dataset$YrSold)
dataset$MoSold <- as.factor(dataset$MoSold)
```

Feature Dummification.

Dummify the categorical variables to let the model learn a better weight for each feature value. (Note: when you use caret you can omit this step, caret performs it by itself)

```
dataset <-dummy.data.frame(dataset,dummy.classes = "factor")
```

Remove some specific features

I've got the idea from https://www.kaggle.com/klyusba/lasso-model-for-regression-problem

```
# We remove them because only one value is nonzero
dataset = dataset[, - which(names(dataset) %in% c('RoofMatlClyTile', 'Condition2PosN'))]

# We remove them becaus they lead to overfitting
dataset = dataset[, - which(names(dataset) %in% c('MSZoningC (all)', 'MSSubClass160'))]
```

Skewness

Transform the target value applying log

```
# Log transform the target for official scoring
dataset$SalePrice <- log1p(dataset$SalePrice)
```

The same "skewness" observed in the target variable before also affects other variables. To facilitate the application of the regression model we are going to also eliminate this skewness.

For numeric feature with excessive skewness, instead of the original log transformation, we apply BoxCox Transformation, which gives us better results.

```
column_types <- sapply(names(dataset),function(x){class(dataset[[x]])})
numeric_columns <-names(column_types[column_types != "character"])

skew <- sapply(numeric_columns,function(x){skewness(dataset[[x]],na.rm = T)})

dkskew <- skew[skew > 0.75]
for (x in names(skew)) {
  bc = BoxCoxTrans(dataset[[x]], lambda = 0.15)
  dataset[[x]] = predict(bc, dataset[[x]])
}
```

Train, test splitting

For facilitating the data cleaning and feature engineering we merged train and test datasets. We now split them again to create our final model.

```
fe_training <- dataset[1:1460,]
fe_test <- dataset[1461:2919,]
```

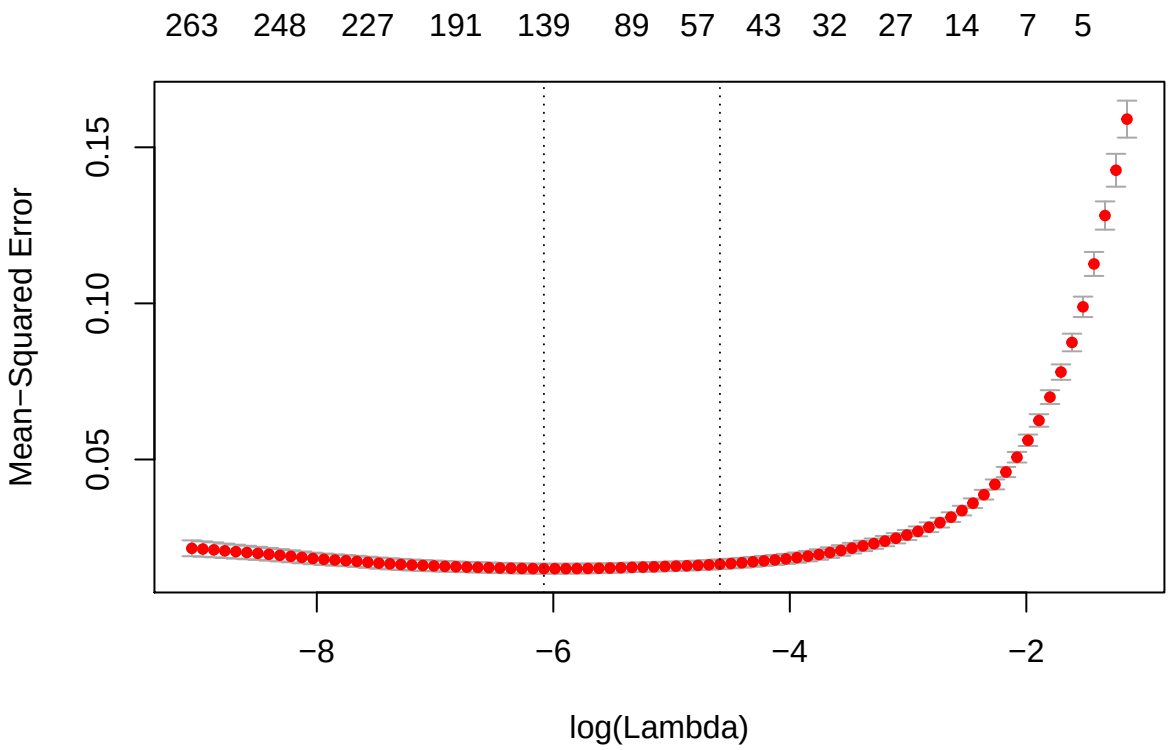
Embedded

Finally, we will apply embedded methods to create the final model. In particular we are going to focus on Lasso Regression.

Lasso Regression

Making use of `glmnet` library we fit a cross-validated lasso regression model to find the best lambda value.

```
set.seed(123)
lasso <- cv.glmnet(x = data.matrix(fe_training[, - which(names(fe_training) %in% c('SalePrice'))]), y =
plot(lasso)
```



As seen in the figure, lambda min is close to 0. In particular it is equal to:

```
lasso$lambda.min
```

```
## [1] 0.002289408
```

Cross-Validated error (in terms of RMSE)

```
sqrt(lasso$cvm[lasso$lambda == lasso$lambda.min])
```

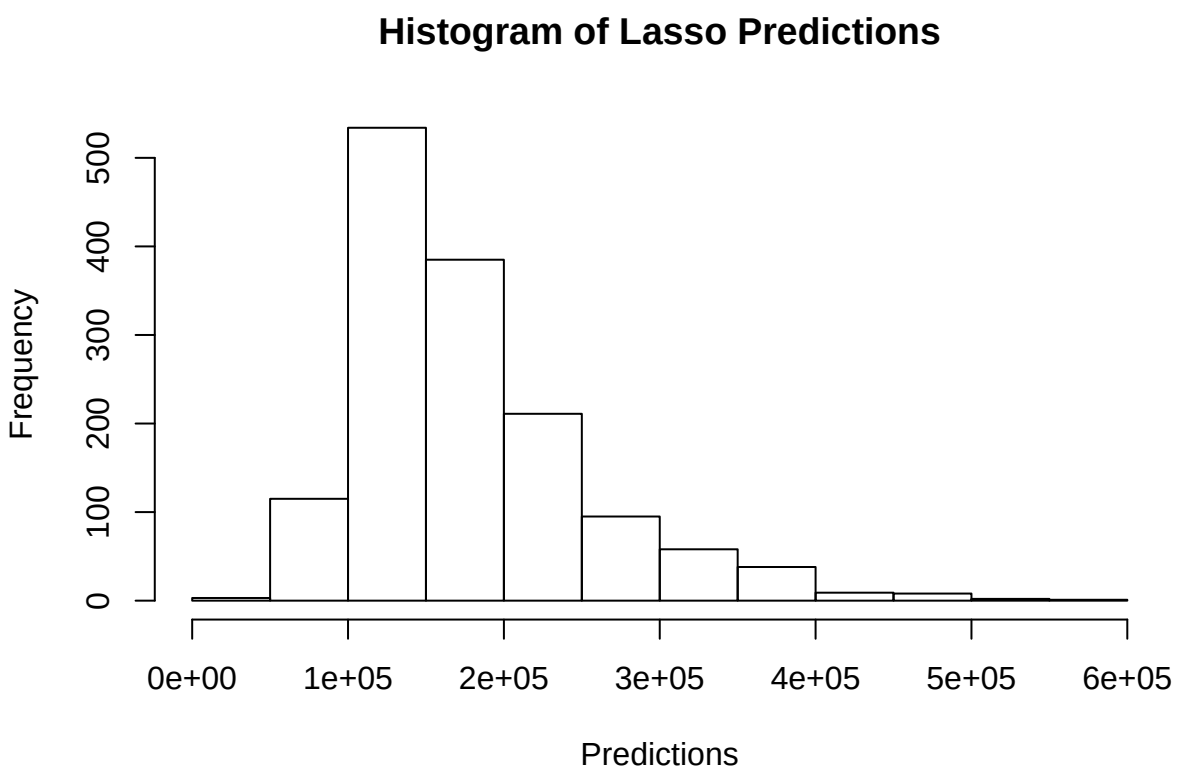
```
## [1] 0.1226024
```

Final Submission

Final submission using lasso.

```
set.seed(46)
lasso <-  cv.glmnet(x = data.matrix(fe_training[, - which(names(fe_training) %in% c('SalePrice'))]), y =
```

```
lasso_pred <- as.numeric(exp(predict(lasso, newx = data.matrix(fe_test[, - which(names(fe_test) %in% c(
hist(lasso_pred, main="Histogram of Lasso Predictions", xlab = "Predictions")
```




```
lasso_submission <- data.frame(Id = test_data$Id, SalePrice= (lasso_pred))
colnames(lasso_submission) <-c("Id", "SalePrice")
write.csv(lasso_submission, file = "lasso_submission.csv", row.names = FALSE)

# Results: 0.11752
```