# 1. Data Loading and First Impressions

In [1]:
```
options(warn=-1) #Suppresses warnings
library(knitr)
library(boot)
library(Metrics)
library(ggplot2)
library(plyr)
library(dplyr)
library(corrplot)
library(caret)
library(gridExtra)
library(scales)
library(Rmisc)
library(ggrepel)
library(randomForest)
library(xgboost)
library(psych)
library(glmnet)
library(ranger)
library(tidyverse)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:plyr':

    arrange, count, desc, failwith, id, mutate, rename, summarise,
    summarize

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

corrplot 0.84 loaded
Loading required package: lattice

Attaching package: 'lattice'

The following object is masked from 'package:boot':

    melanoma


Attaching package: 'caret'

The following objects are masked from 'package:Metrics':

    precision, recall


Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine

randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.

Attaching package: 'randomForest'

The following object is masked from 'package:gridExtra':

    combine

The following object is masked from 'package:dplyr':

    combine

The following object is masked from 'package:ggplot2':

    margin


Attaching package: 'xgboost'
```

```
The following object is masked from 'package:dplyr':

    slice


Attaching package: 'psych'

The following object is masked from 'package:randomForest':

    outlier

The following objects are masked from 'package:scales':

    alpha, rescale

The following objects are masked from 'package:ggplot2':

    %+%, alpha

The following object is masked from 'package:boot':

    logit

Loading required package: Matrix
Loading required package: foreach
Loaded glmnet 2.0-16


Attaching package: 'glmnet'

The following object is masked from 'package:Metrics':

    auc


Attaching package: 'ranger'

The following object is masked from 'package:randomForest':

    importance

-- Attaching packages --------------------------------------- tidyverse 1.2.1
--
v tibble  2.1.1     v purrr   0.3.2
v tidyr   0.8.3     v stringr 1.4.0
v readr   1.3.1     v forcats 0.4.0
-- Conflicts ------------------------------------------ tidyverse_conflicts()
--
x psych::%+%()           masks ggplot2::%+%()
x purrr::accumulate()    masks foreach::accumulate()
x psych::alpha()         masks scales::alpha(), ggplot2::alpha()
x dplyr::arrange()       masks plyr::arrange()
x readr::col_factor()    masks scales::col_factor()
x randomForest::combine() masks gridExtra::combine(), dplyr::combine()
x purrr::compact()       masks plyr::compact()
x dplyr::count()         masks plyr::count()
```

```
x purrr::discard()         masks scales::discard()
x tidyr::expand()          masks Matrix::expand()
x dplyr::failwith()        masks plyr::failwith()
x dplyr::filter()          masks stats::filter()
x dplyr::id()              masks plyr::id()
x dplyr::lag()             masks stats::lag()
x purrr::lift()            masks caret::lift()
x randomForest::margin()   masks ggplot2::margin()
x dplyr::mutate()          masks plyr::mutate()
x dplyr::rename()          masks plyr::rename()
x xgboost::slice()         masks dplyr::slice()
x dplyr::summarise()       masks plyr::summarise()
x dplyr::summarize()       masks plyr::summarize()
x purrr::when()            masks foreach::when()
```

In [2]:

```
setwd('C:\\Users\\loren\\Desktop\\Data Analytics\\DM Project\\Project submissi
on\\BUDT758T-Team-2-ProjectCode&Data')
```

In [3]:

```
#setwd('C:\\Users\\Rohan\\OneDrive\\Desktop\\Rohan Workspace\\Semester 2\\01 D
ata Mining\\08 Project\\02 Data\\01 Raw Data')
getwd()

#this.dir <- dirname(parent.frame(2)$ofile)
#setwd(this.dir)
```

'C:/Users/loren/Desktop/Data Analytics/DM Project/Project submission/BUDT758T-Team-2-ProjectCode&Data'

In [4]:
```
train <- read.csv('train.csv', stringsAsFactors=F)
test <- read.csv('test.csv', stringsAsFactors = F)
head(train,3)
head(test,3)
```

A data.frame: 3 × 81

| Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | Alley | LotShape | LandContour | Uti |
|---|---|---|---|---|---|---|---|---|---|
| <int> | <int> | <chr> | <int> | <int> | <chr> | <chr> | <chr> | <chr> | < |
| 1 | 60 | RL | 65 | 8450 | Pave | NA | Reg | Lvl | A |
| 2 | 20 | RL | 80 | 9600 | Pave | NA | Reg | Lvl | A |
| 3 | 60 | RL | 68 | 11250 | Pave | NA | IR1 | Lvl | A |

A data.frame: 3 × 80

| Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | Alley | LotShape | LandContour | Uti |
|---|---|---|---|---|---|---|---|---|---|
| <int> | <int> | <chr> | <int> | <int> | <chr> | <chr> | <chr> | <chr> | < |
| 1461 | 20 | RH | 80 | 11622 | Pave | NA | Reg | Lvl | A |
| 1462 | 20 | RL | 81 | 14267 | Pave | NA | IR1 | Lvl | A |
| 1463 | 60 | RL | 74 | 13830 | Pave | NA | IR1 | Lvl | A |

# Data Size / Structure

In [5]:
```
dim(train)
dim(test)
```

1460 81

1459 80

In [6]:
```
str(train)
```

```
'data.frame':   1460 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" "Wd Shng" ...
 $ 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" ...
 $ X1stFlrSF    : int  856 1262 920 961 1145 796 1694 1107 1022 1077 ...
 $ X2ndFlrSF    : 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 ...
 $ X3SsnPorch   : 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  NA NA NA NA ...
 $ 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    : int  208500 181500 223500 140000 250000 143000 307000 20000
0 129900 118000 ...
```

Our DV is SalePrice. The dataset is composed of character and integers variables. Using the command stringasFactor = False, we end up with only characters. We will most likely have to do cleaning, to ensure that the right factors are encoded where we want them, as the default R parser tends to make mistakes.

We will not be needing the ID column.

In [7]:
```
testID <- test$Id

train$Id <- NULL
test$Id <- NULL
```

In [8]:
```
dim(test)
dim(train)
```

1459 79

1460 80

For the purpose of data cleaning, We will merge the train and test datasets together.

In [9]:
```
test$SalePrice <- NA
df <- rbind(train,test)
dim(df)
```

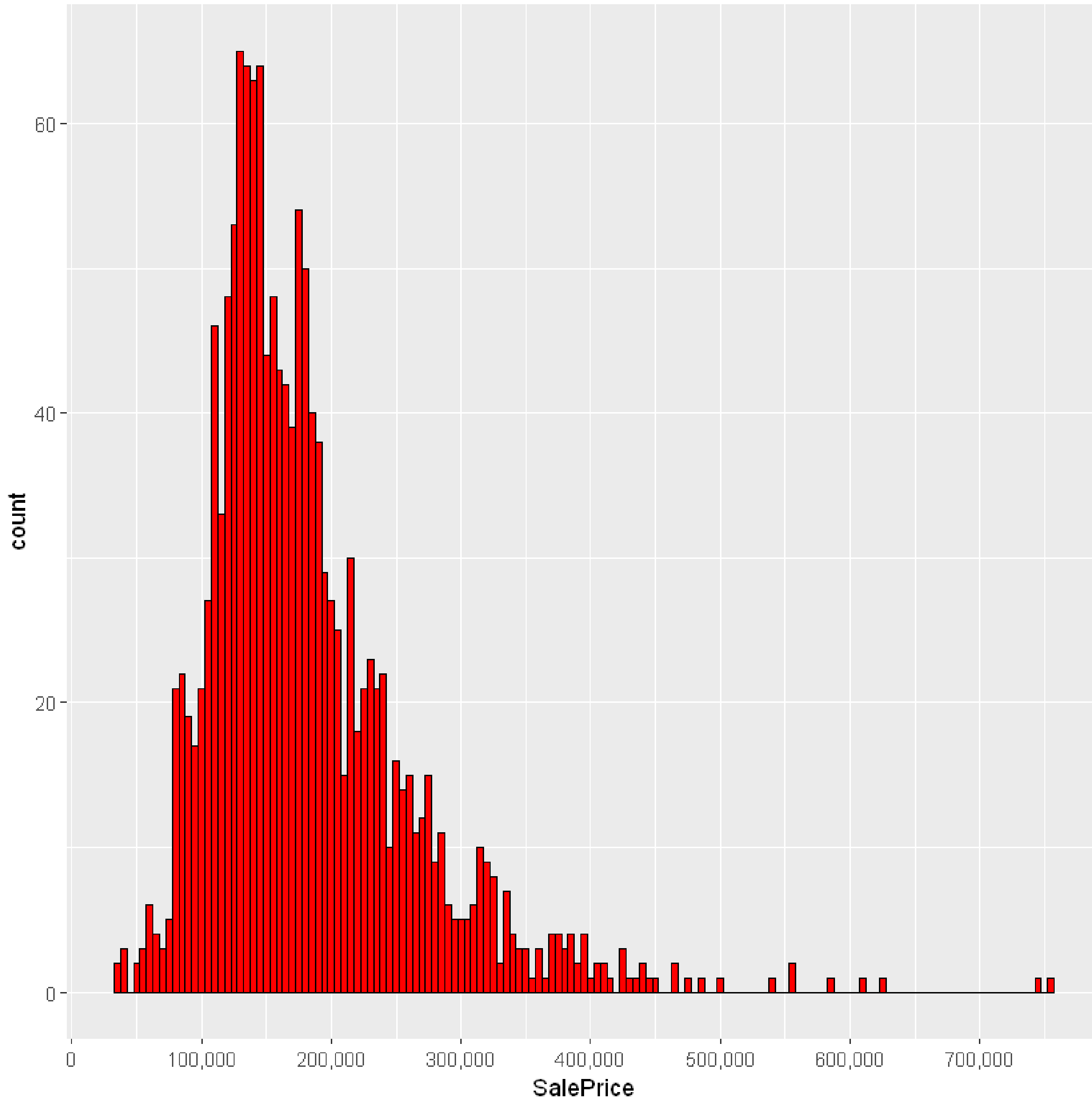
2919 80

We then have our Response Variable, SalePrice, and 79 predictors for 2919 rows of data.

# 2. Variables Exploration

## Sale Price

In [10]:
```
ggplot(data = df[!is.na(df$SalePrice),], aes(x=SalePrice)) +
    geom_histogram(colour='black',fill='red',binwidth=5000)+
    scale_x_continuous(breaks = seq(0,800000,by=100000),labels=comma)
```




The sales prices are rightly skewed, which makes sense since most people cannot afford expensive houses.

In [11]:
```
summary(df$SalePrice)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
  34900  129975  163000  180921  214000  755000    1459 
```

Looking at the summary, we see that the median and the mean are roughly 17k away from each other, which coincides with our visualization since there are outliers skewing the data and bumping the mean compared to the median.

## Numeric Predictors

In [12]:
```
num <- which(sapply(df,is.numeric))
numname <- names(num)
length(num) #37 numeric variables in the dataset
```

37

First, let's explore some of the correlations in the dataset to have a feel for the data, while trying to detect potentially dommageable multicollinearity within the data

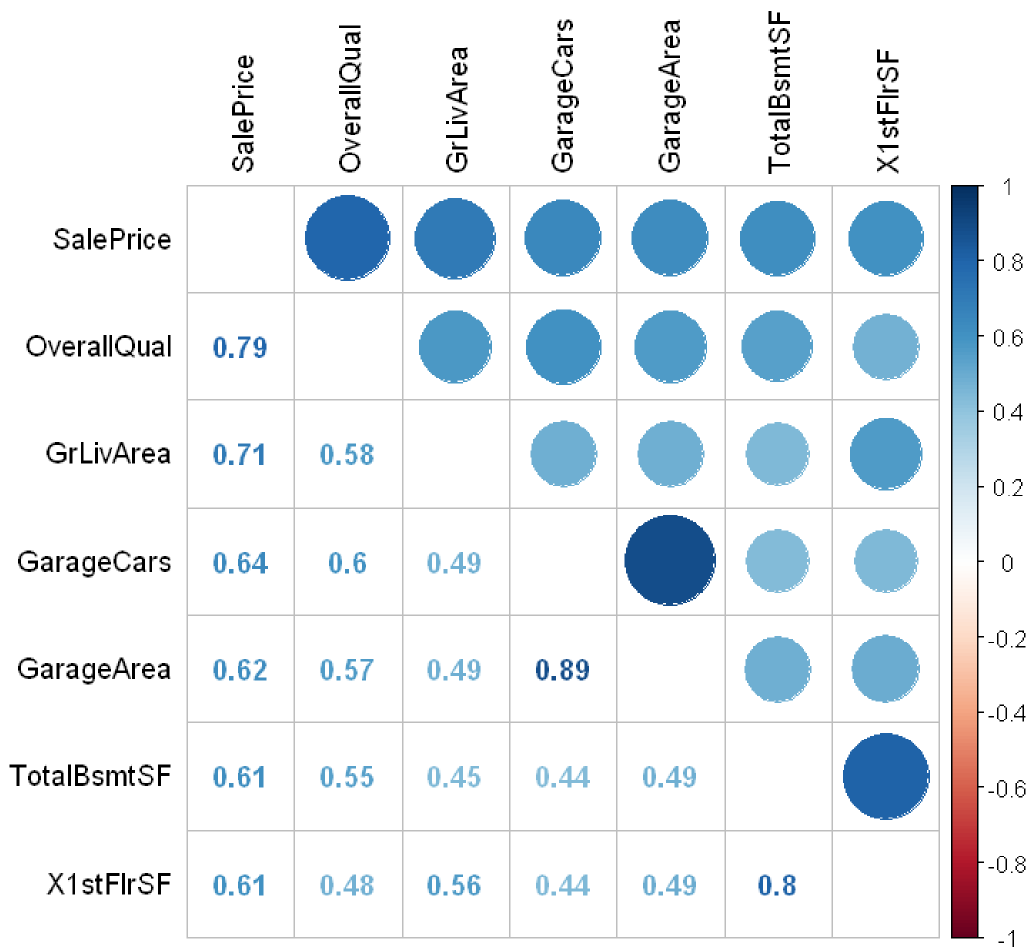
Correlation with SalePrice

In [13]:
```
df_num <- df[,num] #Creates the dataframe with numeric variables only
cor_dfnum <- cor(df_num, use = 'pairwise.complete.obs')

# Sort by decreasing correlations with SalePrice
cor_sort <- as.matrix(sort(cor_dfnum[,'SalePrice'],decreasing = TRUE))

# Only get ones where correlation is above 0.6
cor_strong <- names(which(apply(cor_sort,1,function(x) abs (x) > 0.6)))
cor_dfnum <- cor_dfnum[cor_strong,cor_strong]

corrplot.mixed(cor_dfnum, tl.col = 'black',tl.pos='lt')
```



SalePrice is highly correlated with the Overall Quality and the 'GrLivArea' which corresponds to the area above ground, respectively 0.79 and 0.71.

There seem to be a strong multicollinearity between Garage Area and Garage Cars (which is the size of the garage in sq meters VS size of the garage in terms of how many cars it can hold) with 0.89. Furthermore, there is a strong correlation between X1stFlSF (square footage on the 1st floor) and the Total Basement Square Footage, which makes total sense since basements usually have a very similar square footage to the 1st floor.

# 3. Missing Data and Factors/Ordinal

## Missing Values

In [14]:
```
table(is.na(df))
```

```
 FALSE   TRUE 
218096  15424 
```

There are 15424 NA's in the data.

In [15]:
```
#NAS <- which(colSums(is.na(df))>0)
sort(colSums(sapply(df[which(colSums(is.na(df))>0)],is.na)),decreasing = TRUE)
```

| | |
|---|---|
| **PoolQC** | 2909 |
| **MiscFeature** | 2814 |
| **Alley** | 2721 |
| **Fence** | 2348 |
| **SalePrice** | 1459 |
| **FireplaceQu** | 1420 |
| **LotFrontage** | 486 |
| **GarageYrBlt** | 159 |
| **GarageFinish** | 159 |
| **GarageQual** | 159 |
| **GarageCond** | 159 |
| **GarageType** | 157 |
| **BsmtCond** | 82 |
| **BsmtExposure** | 82 |
| **BsmtQual** | 81 |
| **BsmtFinType2** | 80 |
| **BsmtFinType1** | 79 |
| **MasVnrType** | 24 |
| **MasVnrArea** | 23 |
| **MSZoning** | 4 |
| **Utilities** | 2 |
| **BsmtFullBath** | 2 |
| **BsmtHalfBath** | 2 |
| **Functional** | 2 |
| **Exterior1st** | 1 |
| **Exterior2nd** | 1 |
| **BsmtFinSF1** | 1 |
| **BsmtFinSF2** | 1 |
| **BsmtUnfSF** | 1 |
| **TotalBsmtSF** | 1 |
| **Electrical** | 1 |
| **KitchenQual** | 1 |
| **GarageCars** | 1 |
| **GarageArea** | 1 |
| **SaleType** | 1 |

There are 35 columns with missing data

# What to do with the many missing values

We will deal with missing values starting from the columns with the most missing values. Some values can be easily imputable, whereas some others may need some deeper processing. Furthermore, depending on the variable, there may be some character variables that can be encoded as ordinal - for example, the pool quality, going from No Pool = 0 to Excellent = 5.

In [16]:
```
# Function to get number of missing values from any column

na_check <- function(var){
    y = which(colnames(df)==var)
    x = sum(is.na(df[y]))
    z = colnames(df[y])
    cat('There are',x,'missing values in column',z)
}
```

## Pool Quality

In [17]:
```
na_check('PoolQC')
```

```
There are 2909 missing values in column PoolQC
```

In [18]:
```
unique(df$PoolQC)
```

NA 'Ex' 'Fa' 'Gd'

According to the document, these are the different pool qualities available

Ex Excellent

Gd Good

TA Average/Typical

Fa Fair

NA No Pool

And from what we have seen above, the NA's actually mean that there is no pool, so we will simply input a value 'No Pool' in place of the NA. we will also encode an ordinal vector representing the quality of the pool.

In [19]:
```
df$PoolQC[is.na(df$PoolQC)] <- 'No Pool'
```

In [20]:
```
PoolQualities  <- c('No Pool'=0, 'Fa'=1,'Gd'=2,'Ex'=3)
df$PoolQC <- as.integer(revalue(df$PoolQC,PoolQualities))
table(df$PoolQC)
```

```
   0    1    2    3
2909    2    4    4
```

Let's check to make sure there are no houses left with pools that have not been inputted correctly.

In [21]:
```
df[df$PoolArea>0 & df$PoolQC==0,c('PoolArea','PoolQC')]
```

A data.frame: 3 × 2

| | PoolArea | PoolQC |
|---|---|---|
| | <int> | <int> |
| **2421** | 368 | 0 |
| **2504** | 444 | 0 |
| **2600** | 561 | 0 |

Because we cannot know wheter there was a mistake, or perhaps the pool is damaged, we will assume that there is no pool and assign 0 to the pool area.

In [22]:
```
df$PoolArea[df$PoolArea>0 & df$PoolQC==0] <- 0
```

In [23]:
```
# Check
c = c('PoolArea','PoolQC')
df[2421,c]
df[2504,c]
df[2600,c]
```

A data.frame: 1 × 2

| | PoolArea | PoolQC |
|---|---|---|
| | <dbl> | <int> |
| **2421** | 0 | 0 |

A data.frame: 1 × 2

| | PoolArea | PoolQC |
|---|---|---|
| | <dbl> | <int> |
| **2504** | 0 | 0 |

A data.frame: 1 × 2

| | PoolArea | PoolQC |
|---|---|---|
| | <dbl> | <int> |
| **2600** | 0 | 0 |

## Miscellaneous Features

In [24]:
```
na_check('MiscFeature')
unique(df$MiscFeature)
```

```
There are 2814 missing values in column MiscFeature
```

NA 'Shed' 'Gar2' 'Othr' 'TenC'

There are 2814 missing values, and according to the document, the different categories of miscellaneous features are as follows.

```
Elev Elevator
Gar2 2nd Garage (if not described in garage section)
Othr Other
Shed Shed (over 100 SF)
TenC Tennis Court
NA   None
```

Thus I will replace the NA with 'No Features'. Additionally, since there is no order, I can turn this variable into a factor.

In [25]:
```
df$MiscFeature[is.na(df$MiscFeature)] <- 'No Features'
df$MiscFeature <- as.factor(df$MiscFeature)
```

In [26]:
```
table(df$MiscFeature)
```

```
       Gar2 No Features        Othr        Shed        TenC 
          5        2814           4          95           1 
```

## Type of Alley

In [27]:
```
na_check('Alley')
unique(df$Alley)
```

```
There are 2721 missing values in column Alley
```

NA 'Grvl' 'Pave'

There are 2721 missing values, and according to the document, the different categories of alleys are:

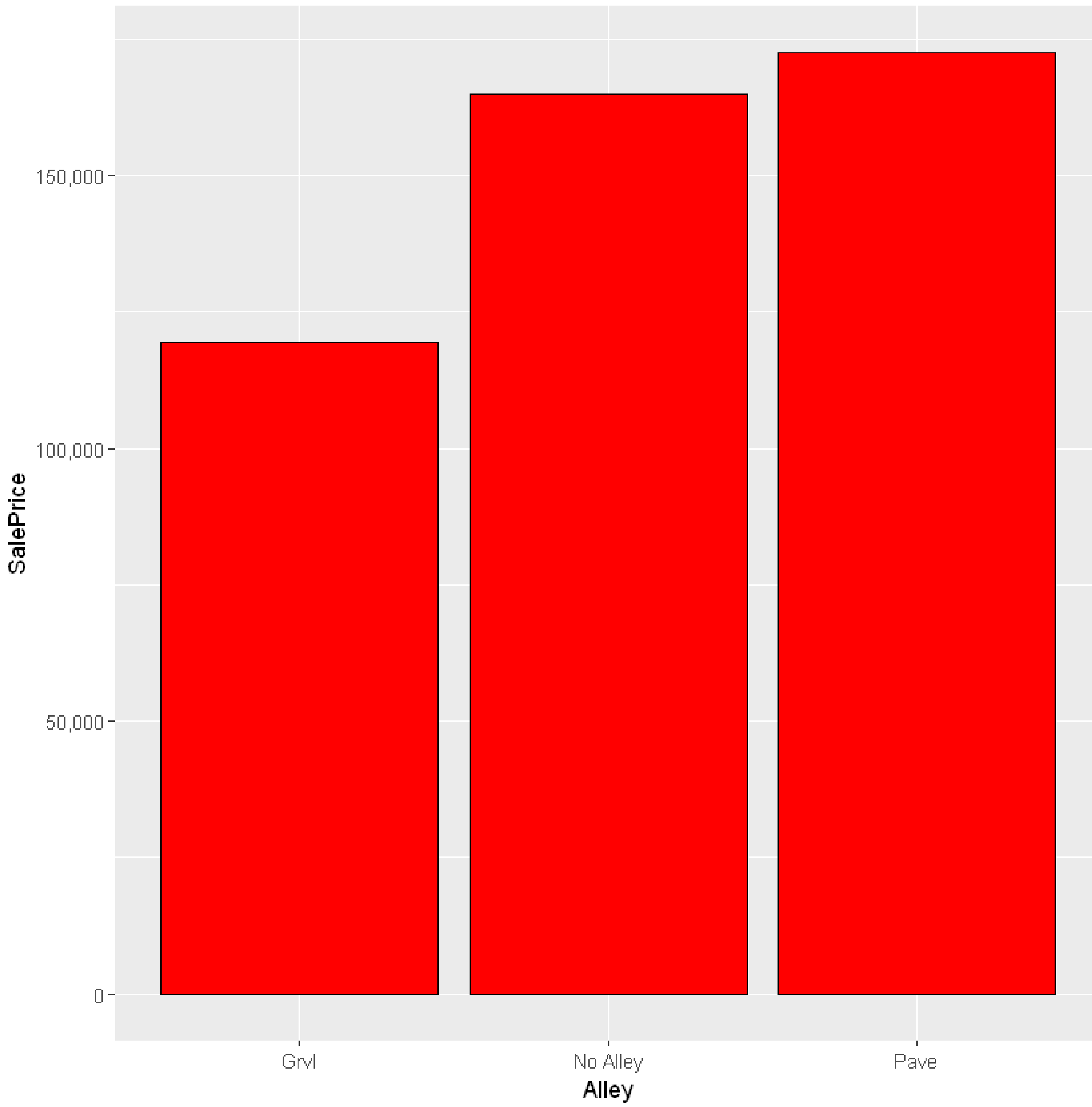
```
Grvl Gravel
Pave Paved
NA   No alley access
```

Thus we will replace the NA with 'No alley'. However, We are not sure whether the variable is ordinal or not.

In [28]:
```
df$Alley[is.na(df$Alley)] <- 'No Alley'
```

In [29]:
```
# Checking if there is any kind of order in the alley variables

ggplot(df[!is.na(df$SalePrice),], aes(x=Alley, y=SalePrice)) +
        geom_bar(stat='summary', fun.y = "median", fill='red',colour='black')+
        scale_y_continuous(breaks= seq(0, 200000, by=50000), labels = comma)
```




In [30]:
```
# There is no ordinality. Thus,
df$Alley <- as.factor(df$Alley)
table(df$Alley)
```

```
    Grvl No Alley     Pave 
     120     2721       78 
```

## Type of Fences

In [31]:
```
na_check('Fence')
unique(df$Fence)
```

```
There are 2348 missing values in column Fence
```

NA 'MnPrv' 'GdWo' 'GdPrv' 'MnWw'

There are 2348 missing values, and according to the document, the fence categories are as follows:

```
GdPrv    Good Privacy
MnPrv    Minimum Privacy
GdWo Good Wood
MnWw Minimum Wood/Wire
NA   No Fence
```

We will replace NA with 'No Fence'. The values do not seem ordinal, We will turn them into a factor.

In [32]:
```
df$Fence[is.na(df$Fence)] <- 'No Fence'
df$Fence <- as.factor(df$Fence)
```

In [33]:
```
table(df$Fence)
```

```
   GdPrv     GdWo    MnPrv     MnWw No Fence
     118      112      329       12     2348
```

## Fireplace Quality and Quantity

In [34]:
```
na_check('FireplaceQu')
unique(df$FireplaceQu)
```

```
There are 1420 missing values in column FireplaceQu
```

NA 'TA' 'Gd' 'Fa' 'Ex' 'Po'

There are 1420 missing values in the Fireplace Quality variable. According to the document, the following are the fireplace qualities:

```
Ex    Excellent - Exceptional Masonry Fireplace
Gd    Good - Masonry Fireplace in main level
TA    Average - Prefabricated Fireplace in main living area or
Fa    Fair - Prefabricated Fireplace in basement
Po    Poor - Ben Franklin Stove
NA    No Fireplace
```

Since the number of houses with no fireplaces matches the number of NAs in fireplace quality, We can replace the NA with 'No Fireplace'. The quality is clearly ordinal.

In [35]:
```
df$FireplaceQu[is.na(df$FireplaceQu)] <- 'No Fireplace'
table(df$FireplaceQu)
```

```
          Ex           Fa           Gd No Fireplace           Po           TA 
          43           74          744         1420           46          592 
```

In [36]:
```
FirePlaceQuality <- c('No Fireplace'=0,'Po'=1,'Fa'=2,'TA'=3,'Gd'=4,'Ex'=5)
df$FireplaceQu <- as.integer(revalue(df$FireplaceQu,FirePlaceQuality))
table(df$FireplaceQu)
```

```
   0    1    2    3    4    5 
1420   46   74  592  744   43 
```

In [37]:
```
na_check('FireplaceQu')
```

```
There are 0 missing values in column FireplaceQu
```

In [38]:
```
na_check('Fireplaces')
table(df$Fireplaces)

 # No missing values in the Fireplaces columns
```

```
There are 0 missing values in column Fireplaces

   0    1    2    3    4 
1420 1268  219   11    1 
```

## Lot Variables

Lot Frontage is the linear feets of street connected to the property.

In [39]:
```
na_check('LotFrontage')
typeof(df$LotFrontage)
```

```
There are 486 missing values in column LotFrontage
```

'integer'

In this case, getting rid of the NAs would significantly reduce our sample size. Thus, it is best to impute the median per neighborhood.

In [40]:
```
for (i in 1:nrow(df)){
    if(is.na(df$LotFrontage[i])){
        df$LotFrontage[i] <- as.integer(median(df$LotFrontage[df$Neighborhood=
=df$Neighborhood[i]], na.rm=TRUE))
    }
}
```

In [41]:
```
# Check if it worked
na_check('LotFrontage')
```

```
There are 0 missing values in column LotFrontage
```

In [42]:
```
summary(df$LotFrontage)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  21.00   60.00   70.00   69.54   80.00  313.00 
```

Lot Shape

In [43]:
```
na_check('LotShape')
```

```
There are 0 missing values in column LotShape
```

General shape of the property, with no missing values. According to the doc, this is what we have:

```
Reg   Regular
IR1   Slightly irregular
IR2   Moderately Irregular
IR3   Irregular
```

There is a clear order, with regular being the best and irregular being the worst. Thus, We will encode this one as an ordinal variable.

In [44]:
```
LotShapeQuality  <- c('IR3'=0, 'IR2'=1, 'IR1'=2, 'Reg'=3)
df$LotShape <- as.integer(revalue(df$LotShape,LotShapeQuality))
table(df$LotShape)
```

```
   0    1    2    3 
  16   76  968 1859 
```

Lot Config

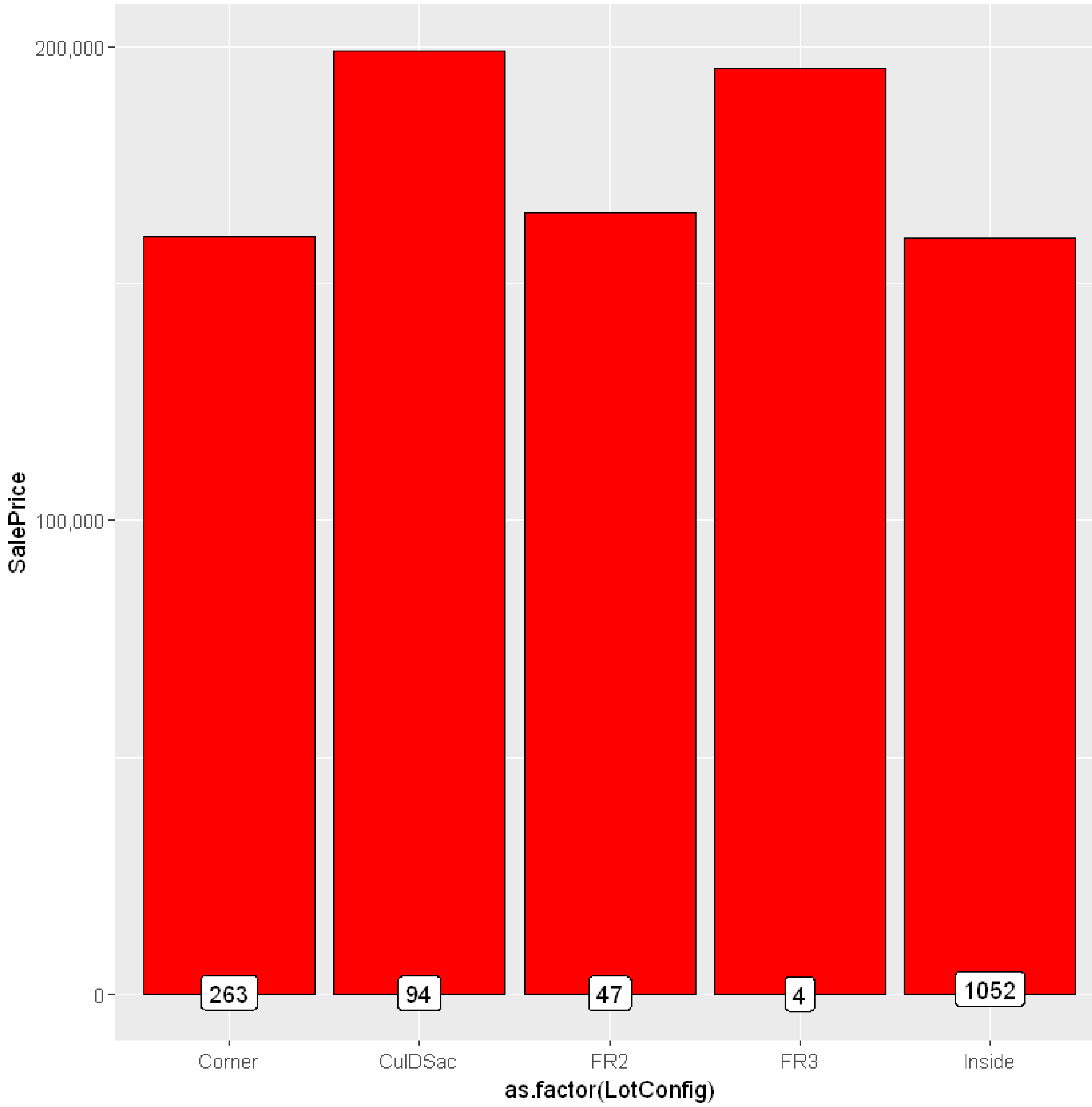
Now onto dealing with Lot Configuration - That it, the overall shape of the property. We are not sure whether the values are ordinal or simply categorical - We'll use a simple visualization to represent it.

In [45]:
```
na_check('LotConfig')
typeof(df$LotConfig)
```

```
There are 0 missing values in column LotConfig
```

'character'

In [46]:
```
ggplot(df[!is.na(df$SalePrice),], aes(x=as.factor(LotConfig), y=SalePrice)) +
        geom_bar(stat='summary', fun.y = "median", fill='red',colour='black')
+
        scale_y_continuous(breaks= seq(0, 800000, by=100000), labels = comma)
+
        geom_label(stat = "count", aes(label = ..count.., y = ..count..))
```




There is no clear order - Thus this variable is a factor.

In [47]:
```
df$LotConfig <- as.factor(df$LotConfig)
table(df$LotConfig)
levels(df$LotConfig) #Quick check
```

```
 Corner CulDSac     FR2     FR3  Inside 
    511     176      85      14    2133 
```

'Corner' 'CulDSac' 'FR2' 'FR3' 'Inside'

## Garage Variables

There are 7 variables related to features of the garage.

```
GarageYrBlt, 159 NA
GarageFinish, 159 NA
   Fin  Finished
   RFn  Rough Finished
   Unf  Unfinished
   NA   No Garage

GarageQual,  159 NA
   Ex   Excellent
   Gd   Good
   TA   Typical/Average
   Fa   Fair
   Po   Poor
   NA   No Garage

GarageCond,  159 NA
   Ex   Excellent
   Gd   Good
   TA   Typical/Average
   Fa   Fair
   Po   Poor
   NA   No Garage

GarageType,  157 NA
   2Types   More than one type of garage
   Attchd   Attached to home
   Basment  Basement Garage
   BuiltIn  Built-In (Garage part of house - typically has room above garage)
   CarPort  Car Port
   Detchd   Detached from home
   NA   No Garage

GarageCars,  1 NA
GarageArea,  1 NA
```

GarageYrBlt can be infered from the year of construction of the house - some garages may have been built after, but it is reasonable to say those constructions require permits, so it would most likely recorded if the house and garage had different construction years.

Garage Finish/quality/condition/type are easy to deal with, just replace NA by 'No Garage'. Some seem ordinal, like condition, some are not, like the type. We will deal with those as we go.

In [48]:
```
Gar <- c('GarageYrBlt','GarageCars','GarageArea','GarageType','GarageCond','GarageQual','GarageFinish')
```

In [49]:
```
df$GarageYrBlt[is.na(df$GarageYrBlt)] <- df$YearBuilt[is.na(df$GarageYrBlt)]
```

We need to find the 2 rows of discrepancy between 159 NA and 157 NA to have everything cleaned up

In [50]:
```
kable(df[!is.na(df$GarageType) & is.na(df$GarageFinish), Gar])
```

```
|     | GarageYrBlt| GarageCars| GarageArea|GarageType |GarageCond |GarageQua
l |GarageFinish |
|:----|-----------:|----------:|----------:|:----------|:----------|:--------
--|:------------|
|2127 |        1910|          1|        360|Detchd     |NA         |NA
|NA           |
|2577 |        1923|         NA|         NA|Detchd     |NA         |NA
|NA           |
```

House 2577 does not have a garage, and house 2127 seems to have one. Thus, there are 158 houses without a garage. For house 2127, I will input the mode - naive imputation.

In [51]:
```
# Fixing 2127

df$GarageCond[2127] <- names(sort(-table(df$GarageCond)))[1]
df$GarageQual[2127] <- names(sort(-table(df$GarageQual)))[1]
df$GarageFinish[2127] <- names(sort(-table(df$GarageFinish)))[1]
```

In [52]:
```
df[2127,Gar]
```

A data.frame: 1 × 7

| | GarageYrBlt | GarageCars | GarageArea | GarageType | GarageCond | GarageQual | GarageFinis |
|---|---|---|---|---|---|---|---|
| | <int> | <int> | <int> | <chr> | <chr> | <chr> | <chı |
| **2127** | 1910 | 1 | 360 | Detchd | TA | TA | U |

In [53]:
```
# Fixing 2577 - Does not have a garage

df$GarageCars[2577] <- 0
df$GarageType[2577] <- NA
df$GarageArea[2577] <- 0
```

In [54]:
```
na_check('GarageType')
```

```
There are 158 missing values in column GarageType
```

In this section, We will provide a brief commentary about the variable and how I transform it

In [55]:
```
# Garage Type - Not ordinal, factor, and NA means no garage

df$GarageType[is.na(df$GarageType)] <- 'No Garage'
df$GarageType <- as.factor(df$GarageType)
table(df$GarageType)
```

```
   2Types    Attchd   Basment   BuiltIn   CarPort    Detchd No Garage
       23      1723        36       186        15       778       158
```

In [56]:
```
# Garage Finish - ordinal, replace NA with no garage

# NA = No Garage, Unf = Unfinished, RFn = Rough Finish, Fin = Finished

df$GarageFinish[is.na(df$GarageFinish)] <- 'No Garage'
df$GarageFinish <- as.integer(revalue(df$GarageFinish,c('No Garage'=0,'Unf'=1,
'RFn'=2,'Fin'=3)))
table(df$GarageFinish)
```

```
   0    1    2    3
 158 1231  811  719
```

In [57]:
```
# Garage Quality - ordinal, NA means no garage

# NA = No Garage, Po = Poor, Fa = Fair, TA = Typical, Gd = Good, Ex = Excellen
t

df$GarageQual[is.na(df$GarageQual)] <- 'No Garage'
Qualities  <- c('No Garage'=0,'Po'=1,'Fa'=2,'TA'=3,'Gd'=4,'Ex'=5)
df$GarageQual <- as.integer(revalue(df$GarageQual,Qualities))
table(df$GarageQual)
```

```
   0    1    2    3    4    5
 158    5  124 2605   24    3
```

In [58]:
```
# Garage Condition - ordinal, NA means no garage

df$GarageCond[is.na(df$GarageCond)] <- 'No Garage'
df$GarageCond <- as.integer(revalue(df$GarageCond,Qualities))
table(df$GarageCond)
```

```
   0    1    2    3    4    5
 158   14   74 2655   15    3
```

In [59]:
```
# What do we have remaining?
sort(colSums(sapply(df[which(colSums(is.na(df))>0)],is.na)),decreasing = TRUE)
```

| | |
|---|---|
| **SalePrice** | 1459 |
| **BsmtCond** | 82 |
| **BsmtExposure** | 82 |
| **BsmtQual** | 81 |
| **BsmtFinType2** | 80 |
| **BsmtFinType1** | 79 |
| **MasVnrType** | 24 |
| **MasVnrArea** | 23 |
| **MSZoning** | 4 |
| **Utilities** | 2 |
| **BsmtFullBath** | 2 |
| **BsmtHalfBath** | 2 |
| **Functional** | 2 |
| **Exterior1st** | 1 |
| **Exterior2nd** | 1 |
| **BsmtFinSF1** | 1 |
| **BsmtFinSF2** | 1 |
| **BsmtUnfSF** | 1 |
| **TotalBsmtSF** | 1 |
| **Electrical** | 1 |
| **KitchenQual** | 1 |
| **SaleType** | 1 |

## Basement Variables

11 variables related to the Basement

5 of those variables have between 79 and 82 NAs

BsmtQual: Evaluates the height of the basement

```
Ex   Excellent (100+ inches)
Gd   Good (90-99 inches)
TA   Typical (80-89 inches)
Fa   Fair (70-79 inches)
Po   Poor (<70 inches
NA   No Basement
```

BsmtCond: Evaluates the general condition of the basement

```
Ex   Excellent
Gd   Good
TA   Typical - slight dampness allowed
Fa   Fair - dampness or some cracking or settling
Po   Poor - Severe cracking, settling, or wetness
NA   No Basement
```

BsmtExposure: Refers to walkout or garden level walls

```
Gd   Good Exposure
Av   Average Exposure (split levels or foyers typically score average or above)
Mn   Mimimum Exposure
No   No Exposure
NA   No Basement
```

BsmtFinType1: Rating of basement finished area

```
GLQ  Good Living Quarters
ALQ  Average Living Quarters
BLQ  Below Average Living Quarters
Rec  Average Rec Room
LwQ  Low Quality
Unf  Unfinshed
NA   No Basement
```

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

```
GLQ   Good Living Quarters
ALQ   Average Living Quarters
BLQ   Below Average Living Quarters
Rec   Average Rec Room
LwQ   Low Quality
Unf   Unfinshed
NA    No Basement
```

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

In [60]:
```
# Are the Nas in one category the same at the ones in the other?

length(which(is.na(df$BsmtQual) & is.na(df$BsmtCond)
             & is.na(df$BsmtExposure) & is.na(df$BsmtFinType1) & is.na(df$Bsmt
FinType2)))
```

79

In [61]:
```
# Where are the additional NAs

df[!is.na(df$BsmtFinType1) & (is.na(df$BsmtCond)|is.na(df$BsmtQual)|is.na(df$B
smtExposure)|is.na(df$BsmtFinType2)),
    c('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2')]
```

A data.frame: 9 × 5

| | BsmtQual | BsmtCond | BsmtExposure | BsmtFinType1 | BsmtFinType2 |
|---|---|---|---|---|---|
| | <chr> | <chr> | <chr> | <chr> | <chr> |
| **333** | Gd | TA | No | GLQ | NA |
| **949** | Gd | TA | NA | Unf | Unf |
| **1488** | Gd | TA | NA | Unf | Unf |
| **2041** | Gd | NA | Mn | GLQ | Rec |
| **2186** | TA | NA | No | BLQ | Unf |
| **2218** | NA | Fa | No | Unf | Unf |
| **2219** | NA | TA | No | Unf | Unf |
| **2349** | Gd | TA | NA | Unf | Unf |
| **2525** | TA | NA | Av | ALQ | Unf |

In [62]:
```
# Imputing modes for each missing variable in houses WITH basements

df$BsmtFinType2[333] <- names(sort(-table(df$BsmtFinType2)))[1]
df$BsmtExposure[c(949, 1488, 2349)] <- names(sort(-table(df$BsmtExposure)))[1]
df$BsmtCond[c(2041, 2186, 2525)] <- names(sort(-table(df$BsmtCond)))[1]
df$BsmtQual[c(2218, 2219)] <- names(sort(-table(df$BsmtQual)))[1]
```

We now have 79 houses with no basement. Now, let's see whether they are ordinal or categorical

In [63]:
```
# Basement Quality - Height of the basement

# Same qualities, clearly ordinal

df$BsmtQual[is.na(df$BsmtQual)] <- 'No Bsmt'
Qualities  <- c('No Bsmt'=0,'Po'=1,'Fa'=2,'TA'=3,'Gd'=4,'Ex'=5)
df$BsmtQual <- as.integer(revalue(df$BsmtQual,Qualities))
table(df$BsmtQual)
```

```
The following `from` values were not present in `x`: Po

   0    2    3    4    5 
  79   88 1285 1209  258 
```

In [64]:
```
# Basement Cond - General condition of the basement

# Same qualities, ordinal as well

df$BsmtCond[is.na(df$BsmtCond)] <- 'No Bsmt'
df$BsmtCond <- as.integer(revalue(df$BsmtCond,Qualities))
table(df$BsmtCond)
```

```
The following `from` values were not present in `x`: Ex

   0    1    2    3    4 
  79    5  104 2609  122 
```

In [65]:
```
# Basement Exps - Walkout/garden level walls

# Ordinal as well, different qualities

df$BsmtExposure[is.na(df$BsmtExposure)] <- 'No Bsmt'
Exposure <- c('No Bsmt'=0, 'No'=1, 'Mn'=2, 'Av'=3, 'Gd'=4)

df$BsmtExposure<-as.integer(revalue(df$BsmtExposure, Exposure))
table(df$BsmtExposure)
```

```
   0    1    2    3    4 
  79 1907  239  418  276 
```

In [66]:
```
# Basement FinType1 - Rating of basement finished area

# Also ordinal, different qualities

df$BsmtFinType1[is.na(df$BsmtFinType1)] <- 'No Bsmt'
FinType <- c('No Bsmt'=0, 'Unf'=1, 'LwQ'=2, 'Rec'=3, 'BLQ'=4, 'ALQ'=5, 'GLQ'=6
)

df$BsmtFinType1<-as.integer(revalue(df$BsmtFinType1, FinType))
table(df$BsmtFinType1)
```

```
  0   1   2   3   4   5   6 
 79 851 154 288 269 429 849 
```

In [67]:
```
# Basement FinType2 - Rating of basement finished area if multiple types
df$BsmtFinType2[is.na(df$BsmtFinType2)] <- 'No Bsmt'

df$BsmtFinType2<-as.integer(revalue(df$BsmtFinType2, FinType))
table(df$BsmtFinType2)
```

```
   0    1    2    3    4    5    6 
  79 2494   87  105   68   52   34 
```

In [68]:
```
# Dealing with the few NAs remaining in the remaining basement variables - her
e we are assuming
# that all missing variables really represent a 0, meaning there is no basemen
t

df$BsmtFullBath[is.na(df$BsmtFullBath)] <- 0
df$BsmtHalfBath[is.na(df$BsmtHalfBath)] <- 0
df$BsmtFinSF1[is.na(df$BsmtFinSF1)] <- 0
df$BsmtFinSF2[is.na(df$BsmtFinSF2)] <- 0
df$BsmtUnfSF[is.na(df$BsmtUnfSF)] <- 0
df$TotalBsmtSF[is.na(df$TotalBsmtSF)] <- 0
```

In [69]:
```
# What do we have remaining?
sort(colSums(sapply(df[which(colSums(is.na(df))>0)],is.na)),decreasing = TRUE)
```

| | |
|---|---|
| **SalePrice** | 1459 |
| **MasVnrType** | 24 |
| **MasVnrArea** | 23 |
| **MSZoning** | 4 |
| **Utilities** | 2 |
| **Functional** | 2 |
| **Exterior1st** | 1 |
| **Exterior2nd** | 1 |
| **Electrical** | 1 |
| **KitchenQual** | 1 |
| **SaleType** | 1 |

## Masonry Variables

2 variables relate to masonry, with Masonry Veneer Type having 24 NAs and Masonry Veneer Area having 23. Logically, if a house has a veneer area, it must have a veneer type, so I must find the one that is missing.

In [70]:
```
df[is.na(df$MasVnrType) & !is.na(df$MasVnrArea),c('MasVnrType','MasVnrArea')]
```

A data.frame: 1 × 2

| | MasVnrType | MasVnrArea |
|---|---|---|
| | <chr> | <int> |
| **2611** | NA | 198 |

In [71]:
```
df$MasVnrType[2611] <- names(sort(-table(df$MasVnrType)))[2]
```

In [72]:
```
df[2611,c('MasVnrType','MasVnrArea')]
```

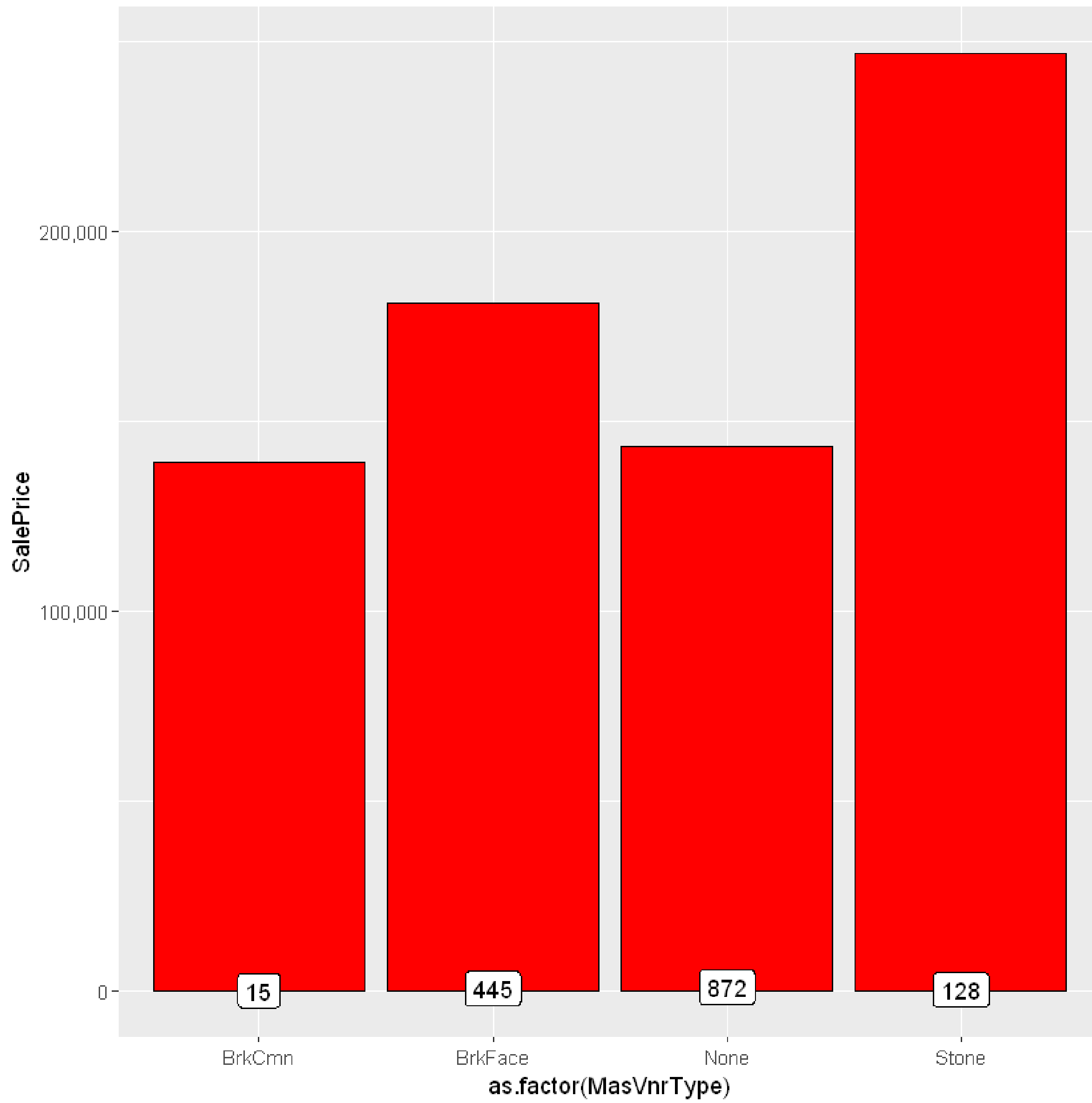
A data.frame: 1 × 2

| | MasVnrType | MasVnrArea |
|---|---|---|
| | <chr> | <int> |
| **2611** | BrkFace | 198 |

Is there ordinality with the type of masonry?

In [73]:
```
df$MasVnrType[is.na(df$MasVnrType)] <- 'None'
```

In [74]:
```
ggplot(df[!is.na(df$SalePrice),], aes(x=as.factor(MasVnrType), y=SalePrice)) +
        geom_bar(stat='summary', fun.y = "median", fill='red',colour='black')
+
        scale_y_continuous(breaks= seq(0, 800000, by=100000), labels = comma)
+
        geom_label(stat = "count", aes(label = ..count.., y = ..count..))
```




Houses made of stone are more expensive. Thus the category is ordinal.

In [75]:
```
MasonryType <- c('None'=0, 'BrkCmn'=0, 'BrkFace'=1, 'Stone'=2)
df$MasVnrType <- as.integer(revalue(df$MasVnrType,MasonryType))
table(df$MasVnrType)
```

```
   0    1    2
1790  880  249
```

In [76]:
```
df$MasVnrArea[is.na(df$MasVnrArea)] <- 0 # We saw previously that those houses
do not have masonry
```

## Dealing with the remaining variables

In [77]:
```
# What do we have remaining?
sort(colSums(sapply(df[which(colSums(is.na(df))>0)],is.na)),decreasing = TRUE)
```

| | |
|---|---|
| **SalePrice** | 1459 |
| **MSZoning** | 4 |
| **Utilities** | 2 |
| **Functional** | 2 |
| **Exterior1st** | 1 |
| **Exterior2nd** | 1 |
| **Electrical** | 1 |
| **KitchenQual** | 1 |
| **SaleType** | 1 |

In [78]:
```
names(sort(-table(df$MSZoning)))[1]
```

'RL'

In [79]:
```
# MS Zoning - Categorical values, 4 NAs

df$MSZoning[is.na(df$MSZoning)] <- names(sort(-table(df$MSZoning)))[1]
df$MSZoning <- as.factor(df$MSZoning)

# Utilities - Kitchen Quality, ordinal , 1 NA

df$KitchenQual[is.na(df$KitchenQual)] <- names(sort(-table(df$KitchenQual)))
[1] #Replace by mode
df$KitchenQual <- as.integer(revalue(df$KitchenQual,Qualities))

# Functional - ordinal, 2 NAs

df$Functional[is.na(df$Functional)] <- names(sort(-table(df$Functional)))[1]
df$Functional <- as.integer(revalue(df$Functional,
                                    c('Sal'=0, 'Sev'=1, 'Maj2'=2, 'Maj1'=3, 'M
od'=4, 'Min2'=5, 'Min1'=6, 'Typ'=7)))

# Exterior 1st
df$Exterior1st[is.na(df$Exterior1st)] <- names(sort(-table(df$Exterior1st)))
[1]
df$Exterior1st <- as.factor(df$Exterior1st)

# Exterior 2nd
df$Exterior2nd[is.na(df$Exterior2nd)] <- names(sort(-table(df$Exterior2nd)))
[1]
df$Exterior2nd <- as.factor(df$Exterior2nd)

# ExterQual
df$ExterQual <- as.integer(revalue(df$ExterQual,Qualities))

# ExterCond
df$ExterCond <- as.integer(revalue(df$ExterCond,Qualities))

# Electrical
df$Electrical[is.na(df$Electrical)] <- names(sort(-table(df$Electrical)))[1]
df$Electrical <- as.factor(df$Electrical)

# Sale Type
df$SaleType[is.na(df$SaleType)] <- names(sort(-table(df$SaleType)))[1]
df$SaleType <- as.factor(df$SaleType)

# Sale Condition
df$SaleCondition <- as.factor(df$SaleCondition)
```

```
The following `from` values were not present in `x`: No Bsmt, Po
The following `from` values were not present in `x`: Sal
The following `from` values were not present in `x`: No Bsmt, Po
The following `from` values were not present in `x`: No Bsmt
```

In [80]:
```
# Utilities - ordinal

table(df$Utilities)
```

```
AllPub NoSeWa
  2916      1
```

In [81]:
```
# All houses have full utilities, besides ones - useless predictor

df$Utilities <- NULL
```

In [82]:
```
# What do we have remaining?
sort(colSums(sapply(df[which(colSums(is.na(df))>0)],is.na)),decreasing = TRUE)

# No more NAs - The NAs for the sale price correspond to the test data
```

**SalePrice:** 1459

# 4. Character Variables

We still have to take care of the character variables that did not have any missing values.

In [83]:
```
char  <- names(df[,sapply(df,is.character)])
char
cat('There are',length(char),'character columns')
```

'Street' 'LandContour' 'LandSlope' 'Neighborhood' 'Condition1' 'Condition2' 'BldgType' 'HouseStyle' 'RoofStyle' 'RoofMatl' 'Foundation' 'Heating' 'HeatingQC' 'CentralAir' 'PavedDrive'

```
There are 15 character columns
```

In [84]:
```
# Foundation - factor
df$Foundation <- as.factor(df$Foundation)

# LandContour - factor
df$LandContour <- as.factor(df$LandContour)

# LandSlope - ordinal
df$LandSlope <- as.integer(revalue(df$LandSlope, c('Sev'=0, 'Mod'=1, 'Gtl'=2
)))

# Neighborhood - factor
df$Neighborhood <- as.factor(df$Neighborhood)

# Condition1 - factor
df$Condition1 <- as.factor(df$Condition1)

# Condition2 - factor
df$Condition2 <- as.factor(df$Condition2)

# BldgType - factor
df$BldgType <- as.factor(df$BldgType)

# HouseStyle - factor
df$HouseStyle <- as.factor(df$HouseStyle)

# RoofStyle - factor
df$RoofStyle <- as.factor(df$RoofStyle)

# RoofMatl - factor
df$RoofMatl <- as.factor(df$RoofMatl)

# Heating - factor
df$Heating <- as.factor(df$Heating)

# Heating Quality - ordinal
df$HeatingQC <- as.integer(revalue(df$HeatingQC,Qualities))

# CentralAir - Yes/No, factor
df$CentralAir <- as.factor(df$CentralAir)

# PavedDrive - ordinal
df$PavedDrive <- as.integer(revalue(df$PavedDrive,c('N'=0, 'P'=1, 'Y'=2)))

# Street - ordinal
df$Street <- as.integer(revalue(df$Street,c('Grvl'=0, 'Pave'=1)))
```

```
The following `from` values were not present in `x`: No Bsmt
```

# 5. Numerical Variables to Factors

In [85]:
```
num
```

| | |
|---|---|
| **MSSubClass** | 1 |
| **LotFrontage** | 3 |
| **LotArea** | 4 |
| **OverallQual** | 17 |
| **OverallCond** | 18 |
| **YearBuilt** | 19 |
| **YearRemodAdd** | 20 |
| **MasVnrArea** | 26 |
| **BsmtFinSF1** | 34 |
| **BsmtFinSF2** | 36 |
| **BsmtUnfSF** | 37 |
| **TotalBsmtSF** | 38 |
| **X1stFlrSF** | 43 |
| **X2ndFlrSF** | 44 |
| **LowQualFinSF** | 45 |
| **GrLivArea** | 46 |
| **BsmtFullBath** | 47 |
| **BsmtHalfBath** | 48 |
| **FullBath** | 49 |
| **HalfBath** | 50 |
| **BedroomAbvGr** | 51 |
| **KitchenAbvGr** | 52 |
| **TotRmsAbvGrd** | 54 |
| **Fireplaces** | 56 |
| **GarageYrBlt** | 59 |
| **GarageCars** | 61 |
| **GarageArea** | 62 |
| **WoodDeckSF** | 66 |
| **OpenPorchSF** | 67 |
| **EnclosedPorch** | 68 |
| **X3SsnPorch** | 69 |
| **ScreenPorch** | 70 |
| **PoolArea** | 71 |
| **MiscVal** | 75 |
| **MoSold** | 76 |
| **YrSold** | 77 |
| **SalePrice** | 80 |

Some of these variables do not make sense as a numerical variable

```
Year Sold
Month Sold
MSSubClass - Code for the type of dwelling involved in the sale
    For example: 20  1-STORY 1946 & NEWER ALL STYLES
                 70  2-STORY 1945 & OLDER
```

In [86]:
```
df$YrSold <- as.factor(df$YrSold)
df$MoSold <- as.factor(df$MoSold)
df$MSSubClass <- as.factor(df$MSSubClass)
```

In [87]:
```
numericVars <- which(sapply(df, is.numeric)) #index vector numeric variables
factorVars <- which(sapply(df, is.factor)) #index vector factor variables
cat('There are', length(which(sapply(df, is.numeric))),
    'numeric variables, and', length(which(sapply(df, is.factor))), 'categoric
variables')
```

```
There are 54 numeric variables, and 25 categoric variables
```

# 6. Important Variables - Visualization

Numeric Variables

In [88]:
```
numericVars
```

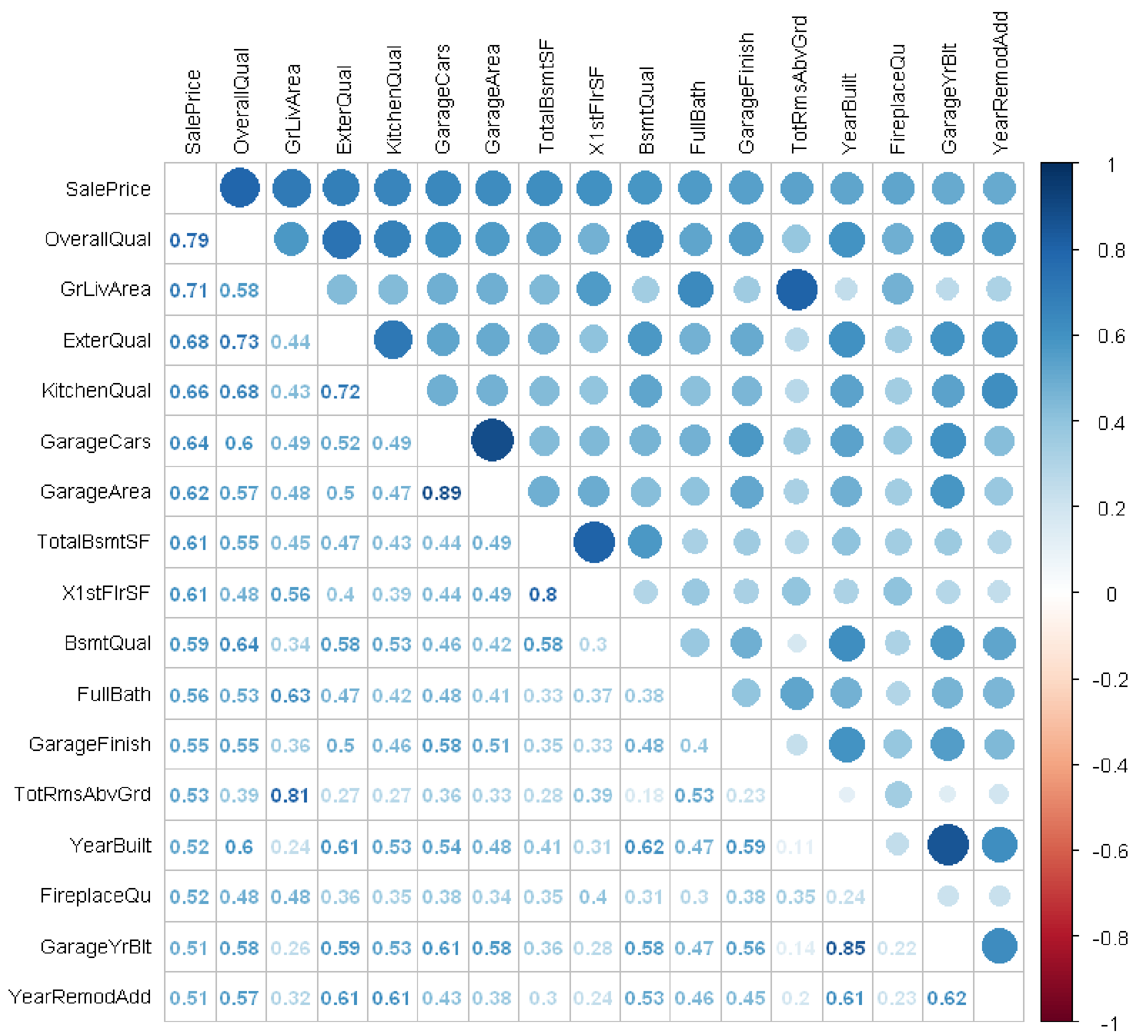
| | |
|---|---|
| **LotFrontage** | 3 |
| **LotArea** | 4 |
| **Street** | 5 |
| **LotShape** | 7 |
| **LandSlope** | 10 |
| **OverallQual** | 16 |
| **OverallCond** | 17 |
| **YearBuilt** | 18 |
| **YearRemodAdd** | 19 |
| **MasVnrType** | 24 |
| **MasVnrArea** | 25 |
| **ExterQual** | 26 |
| **ExterCond** | 27 |
| **BsmtQual** | 29 |
| **BsmtCond** | 30 |
| **BsmtExposure** | 31 |
| **BsmtFinType1** | 32 |
| **BsmtFinSF1** | 33 |
| **BsmtFinType2** | 34 |
| **BsmtFinSF2** | 35 |
| **BsmtUnfSF** | 36 |
| **TotalBsmtSF** | 37 |
| **HeatingQC** | 39 |
| **X1stFlrSF** | 42 |
| **X2ndFlrSF** | 43 |
| **LowQualFinSF** | 44 |
| **GrLivArea** | 45 |
| **BsmtFullBath** | 46 |
| **BsmtHalfBath** | 47 |
| **FullBath** | 48 |
| **HalfBath** | 49 |
| **BedroomAbvGr** | 50 |
| **KitchenAbvGr** | 51 |
| **KitchenQual** | 52 |
| **TotRmsAbvGrd** | 53 |
| **Functional** | 54 |
| **Fireplaces** | 55 |
| **FireplaceQu** | 56 |
| **GarageYrBlt** | 58 |
| **GarageFinish** | 59 |
| **GarageCars** | 60 |
| **GarageArea** | 61 |
| **GarageQual** | 62 |
| **GarageCond** | 63 |
| **PavedDrive** | 64 |
| **WoodDeckSF** | 65 |
| **OpenPorchSF** | 66 |
| **EnclosedPorch** | 67 |

| | |
|---|---|
| **X3SsnPorch** | 68 |
| **ScreenPorch** | 69 |
| **PoolArea** | 70 |
| **PoolQC** | 71 |
| **MiscVal** | 74 |
| **SalePrice** | 79 |

In [89]:
```
df_numVar <- df[, numericVars]
cor_numVar <- cor(df_numVar, use="pairwise.complete.obs") #correlations of all numeric variables

#sort by decreasing correlations with SalePrice
cor_sorted <- as.matrix(sort(cor_numVar[,'SalePrice'], decreasing = TRUE))
 #select only high correlations
CorHigh <- names(which(apply(cor_sorted, 1, function(x) abs(x)>0.5)))
cor_numVar <- cor_numVar[CorHigh, CorHigh]

corrplot.mixed(cor_numVar, tl.col="black", tl.pos = "lt", tl.cex = 0.7,cl.cex = .7, number.cex=.7)
```




Compared to our first visualization, we get 2 more variables with a high correlation, for a total of 8 numeric variables with a correlation > 0.6

How important are the different variables?

The goal of this short section is to get a sense of which variables are most impotant. For this, I will run a Random Forest with both categorical and numerical predictors.

In [90]:
```
set.seed(123)
RF_important_variables <- randomForest(x=df[1:1460,-79],y=df$SalePrice[1:1460]
,ntre=500,importance=TRUE)
```

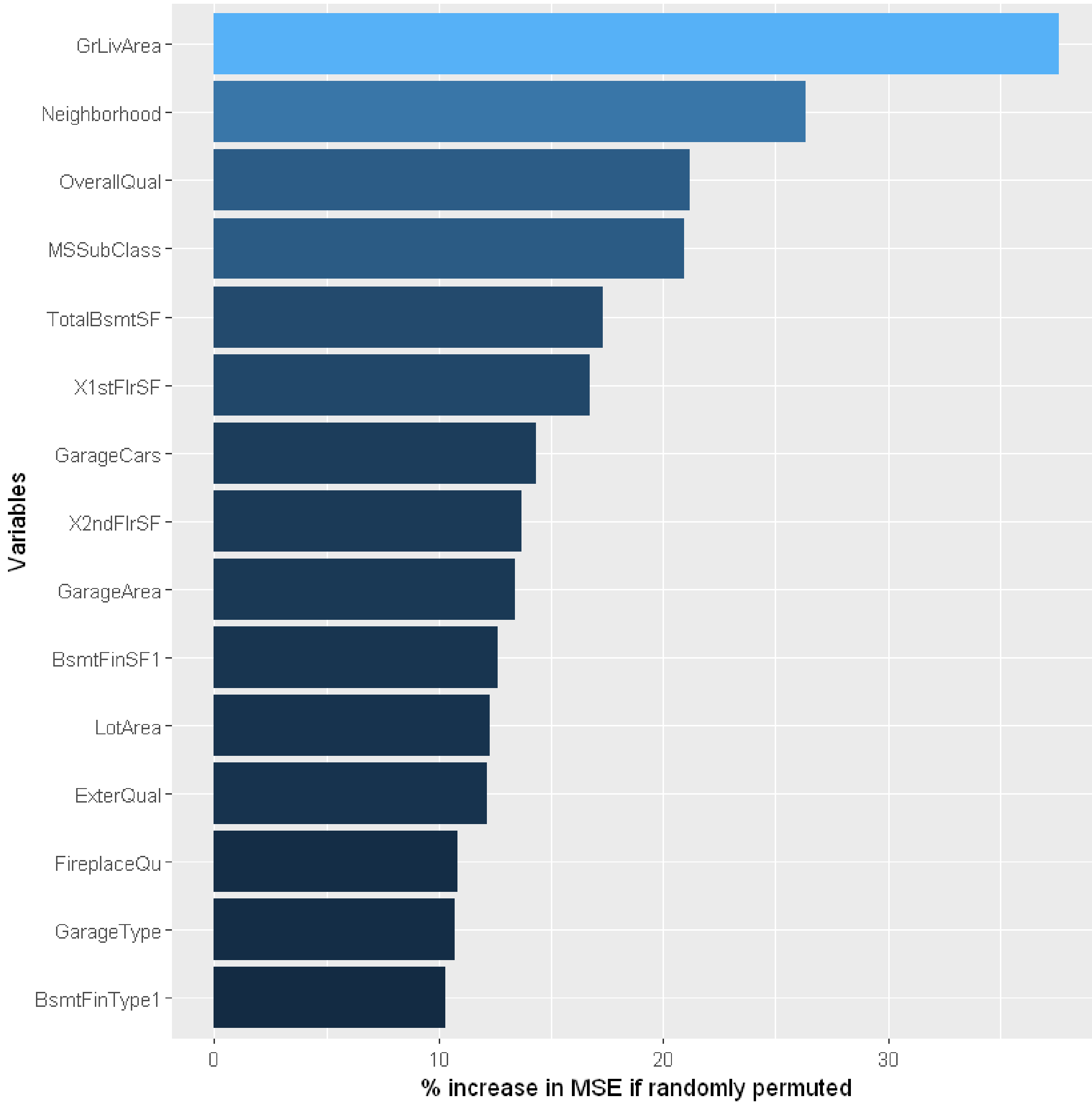
In [91]:
```
# Necessity to specify package from which 'importance' fct is extracted from a
s it collides
# with another fct of the same name in an another package

imp = randomForest::importance(RF_important_variables)
imp = data.frame(Variables = row.names(imp), MSE = imp[,1])
imp = imp[order(imp$MSE, decreasing = TRUE),]
```

- Chart Logic

We only want the first 15 variables, showing by how much the MSE would increase if we randomly moved those variables - that is, how important they are in explaining the model. In reality, this is a rough Random Forest just to get a quick feeling of how the variables interact with eachother. We then flipped the axis to make it easier to read.

In [92]:
```
ggplot(imp[1:15,],aes(x=reorder(Variables,MSE), y = MSE, fill=MSE)) +
    geom_bar(stat='identity') +
    labs(x='Variables', y = '% increase in MSE if randomly permuted') +
    coord_flip() + theme(legend.position='none')
```




Out of the most important 15 variables, only 3 are categorical - the Neighborhood, the MSSubclass and the Garage Type, in this order. According to the Random Forest, it seems that the numeric variables are the most important ones in determining the sale price of a house.

# 7. Feature Engineering

As we have seen during the data cleaning part, some variables are broken down into several sub-variables - for example, the proch variables (4 sub-porch variables), which seems like a bit of an overkill for our purposes.

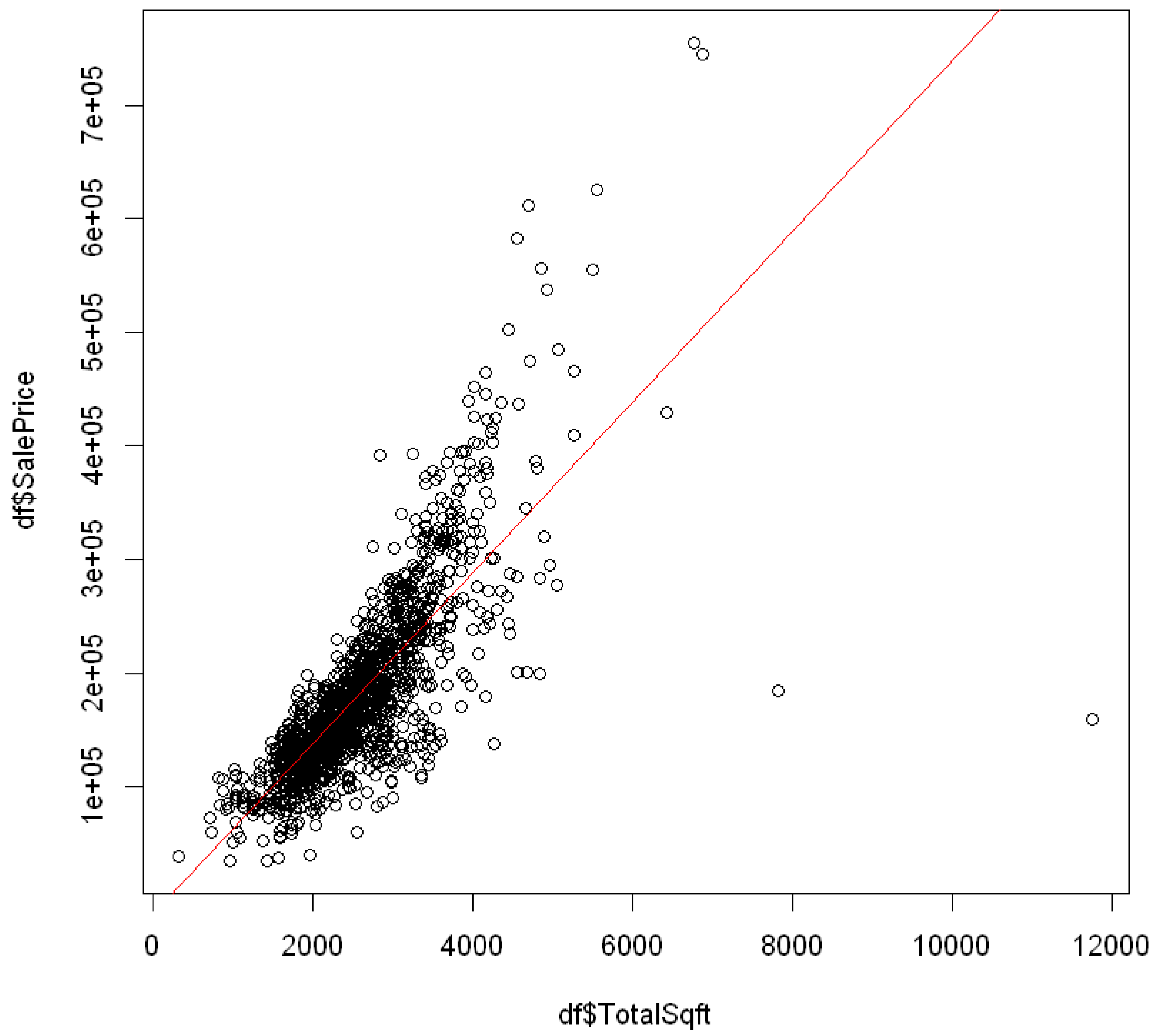
In this section, I will try to come up with new variables that could hopefully deliver more insights running the analysis.

## Total Square Footage

Surprisingly enough, the total square footage is nowhere to be found in the dataset. Thus, I will create a new variable adding the livable space above and below ground.

In [93]:
```
df$TotalSqft  <- df$GrLivArea + df$TotalBsmtS
```

In [94]:
```
plot(df$TotalSqft,df$SalePrice)
abline(lm(df$SalePrice~df$TotalSqft,data=df),col='red')
```




In [95]:
```
cor(df$SalePrice,df$TotalSqft,use='pairwise.complete.obs')
```

0.778958828994226

In [96]:
```
# Isolate the 2 clear outliers and remove them to run the correlation again

(df[df$TotalSqft > 10000,c('TotalSqft','SalePrice')])
(df[df$TotalSqft >7000 & df$TotalSqft < 8000,c('TotalSqft','SalePrice')])
```

A data.frame: 2 × 2

| | TotalSqft | SalePrice |
|---|---|---|
| | <dbl> | <int> |
| **1299** | 11752 | 160000 |
| **2550** | 10190 | NA |

A data.frame: 1 × 2

| | TotalSqft | SalePrice |
|---|---|---|
| | <dbl> | <int> |
| **524** | 7814 | 184750 |

In [97]:
```
# Running the correlation again removing the data points at index 524 and 129
9, the correlation is much stronger,
# indicating that the new feature added (total square feet) is statistically s
ignificant

cor(df$SalePrice[-c(524,1299)],df$TotalSqft[-c(524,1299)],use='pairwise.comple
te.obs')
```

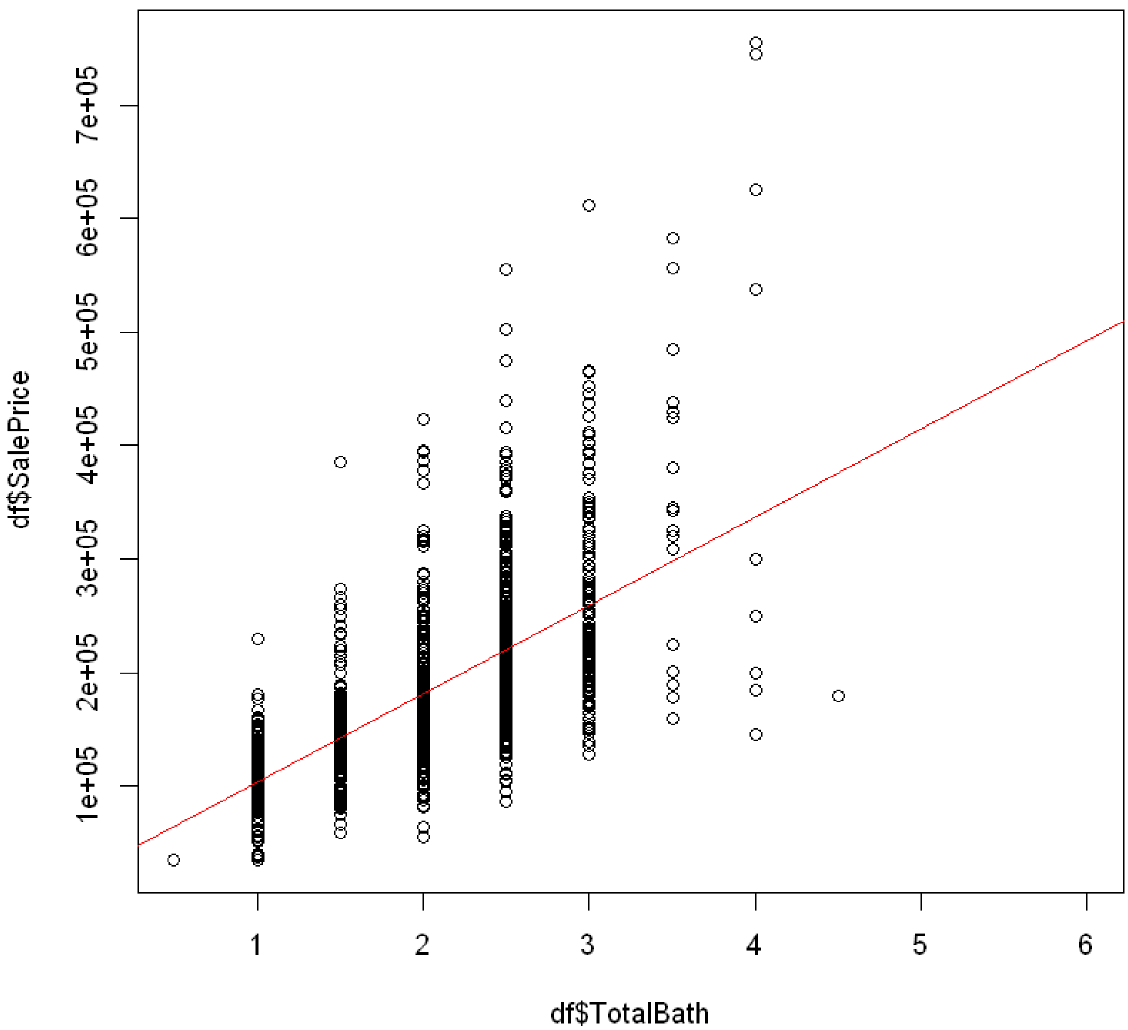
0.829041978106551

## Bathroom Variables

From the dataset, there are 4 bathroom variables: full bathroom and half bathroom for living areas above and below ground. We will add them all up adding weights based on where the bathroom is located. In this section, We actually tried weighing halfbathroom the same as full ones, and then again with a weight of 0.5, and the correlation is stronger when half bathrooms are weighted down. We also played around with the weight for the full bathroom in the above and below ground living area, and it turns out that the strongest correlation appears when the Full bathroom in the basement is weighted down to 0.5 as well.

In [98]:
```
df$TotalBath <- df$FullBath + (df$BsmtFullBath*0.5) + (df$HalfBath*0.5) + (df$
BsmtHalfBath*0.5)
```

In [99]:
```
plot(df$TotalBath,df$SalePrice)
abline(lm(df$SalePrice~df$TotalBath,data=df),col='red')
```




In [100]:
```
cor(df$TotalBath,df$SalePrice,use='pairwise.complete.obs')
```

0.654312210590486

# 8. Preparing Data for modeling

### Removing outliers

In [101]:
```
df  <- df[-c(524,1299),]
```

### Dropping Highly Correlated Variables

From the past sections, we have seen several highly correlated variables (see correlation matrix in section 6). Out of the highly correlated pair of variables, I will drop the one with the least correlation with our DV, Sale Price

In [102]:
```
df[,c('GarageYrBlt', 'GarageArea', 'GarageCond', 'TotalBsmtSF', 'TotalRmsAbvGrd', 'BsmtFinSF1')] <- NULL
```

### Predictor Variables

In [103]:
```
numericVars <- which(sapply(df, is.numeric)) #index vector numeric variables
```

In [104]:
```
length(numericVars)
```

51

In [105]:
```
dfnum <- df[,numericVars]
```

Many of these numeric variables are actually ordinal, which we will append to the factor dataframe, without transforming them into factors.

In [106]:
```
ord = dfnum[,c('OverallQual','OverallCond','LandSlope','PavedDrive','Street', 'HeatingQC','KitchenQual',
               'Functional','ExterQual','ExterCond','MasVnrType','PoolQC','FireplaceQu','LotShape','GarageFinish',
               'GarageQual','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2')]
```

In [107]:
```
ord1 <- c('OverallQual','OverallCond','LandSlope','PavedDrive','Street','HeatingQC','KitchenQual',
               'Functional','ExterQual','ExterCond','MasVnrType','PoolQC','FireplaceQu','LotShape','GarageFinish',
               'GarageQual','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2')
```

In [108]:
```
dfnum <- dfnum %>% select (-ord1)
```

In [109]:
```
length(dfnum)
```

30

In [110]:
```
factorVars <- which(sapply(df, is.factor)) #index vector factor variables
```

In [111]:
```
length(factorVars)
```

25

In [112]:
```
dffactor <- df[,factorVars]
```

In [113]:
```
dffactor <- cbind(dffactor,ord)
```

In [114]:
```
cat('There are', length(dfnum), 'numeric variables, and', length(dffactor), 'factor variables')
```

```
There are 30 numeric variables, and 46 factor variables
```

### Standardization

In [115]:
```
head(dfnum,2)
```

A data.frame: 2 × 30

| LotFrontage | LotArea | YearBuilt | YearRemodAdd | MasVnrArea | BsmtFinSF2 | BsmtUnfSF | X1stFlrS |
|---|---|---|---|---|---|---|---|
| <int> | <int> | <int> | <int> | <dbl> | <dbl> | <dbl> | <int |
| 65 | 8450 | 2003 | 2003 | 196 | 0 | 150 | 85 |
| 80 | 9600 | 1976 | 1976 | 0 | 0 | 284 | 126 |

In [116]:
```
dfnorm  <- dfnum %>% mutate_at(scale, .vars = vars(-SalePrice))
```

In [117]:
```
head(dfnorm,2)
```

A data.frame: 2 × 30

| LotFrontage | LotArea | YearBuilt | YearRemodAdd | MasVnrArea | BsmtFinSF2 | BsmtUnfSF | X1: |
|---|---|---|---|---|---|---|---|
| <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <( |
| -0.2089718 | -0.21639955 | 1.0470513 | 0.8975478 | 0.5339956 | -0.2930841 | -0.9336018 | -0.7 |
| 0.4984058 | -0.06909653 | 0.1555794 | -0.3947970 | -0.5669270 | -0.2930841 | -0.6288477 | 0.2 |

Encoding categorical variables

In [118]:
```
dfdummies <- as.data.frame(model.matrix(~.-1,dffactor))
```

In [119]:
```
dim(dfdummies)
```

2917 198

In [120]:
```
head(dfdummies,2)
```

A data.frame: 2 × 198

| MSSubClass20 | MSSubClass30 | MSSubClass40 | MSSubClass45 | MSSubClass50 | MSSubClass60 |
|---|---|---|---|---|---|
| <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |

Removing levels with few or no observations in train or test as this will cause us issues during the modeling phase

In [121]:
```
PredictorsWithNoObs <- which(colSums(dfdummies[(nrow(df[!is.na(df$SalePrice),]
)+1):nrow(df),])==0)
colnames(dfdummies[PredictorsWithNoObs])
```

'Condition2RRAe' 'Condition2RRAn' 'Condition2RRNn' 'HouseStyle2.5Fin' 'RoofMatlMembran' 'RoofMatlMetal' 'RoofMatlRoll' 'Exterior1stImStucc' 'Exterior1stStone' 'Exterior2ndOther' 'HeatingOthW' 'ElectricalMix' 'MiscFeatureTenC'

Removing these columns

In [122]:
```
dfdummies <- dfdummies[,-PredictorsWithNoObs]
```

Check if values are absent in the training set

In [123]:
```
PredictorsWithNoObsTrain <- which(colSums(dfdummies[1:nrow(df[!is.na(df$SalePr
ice),]),])==0)
colnames(dfdummies[PredictorsWithNoObsTrain])
```

'MSSubClass150'

Removing the column

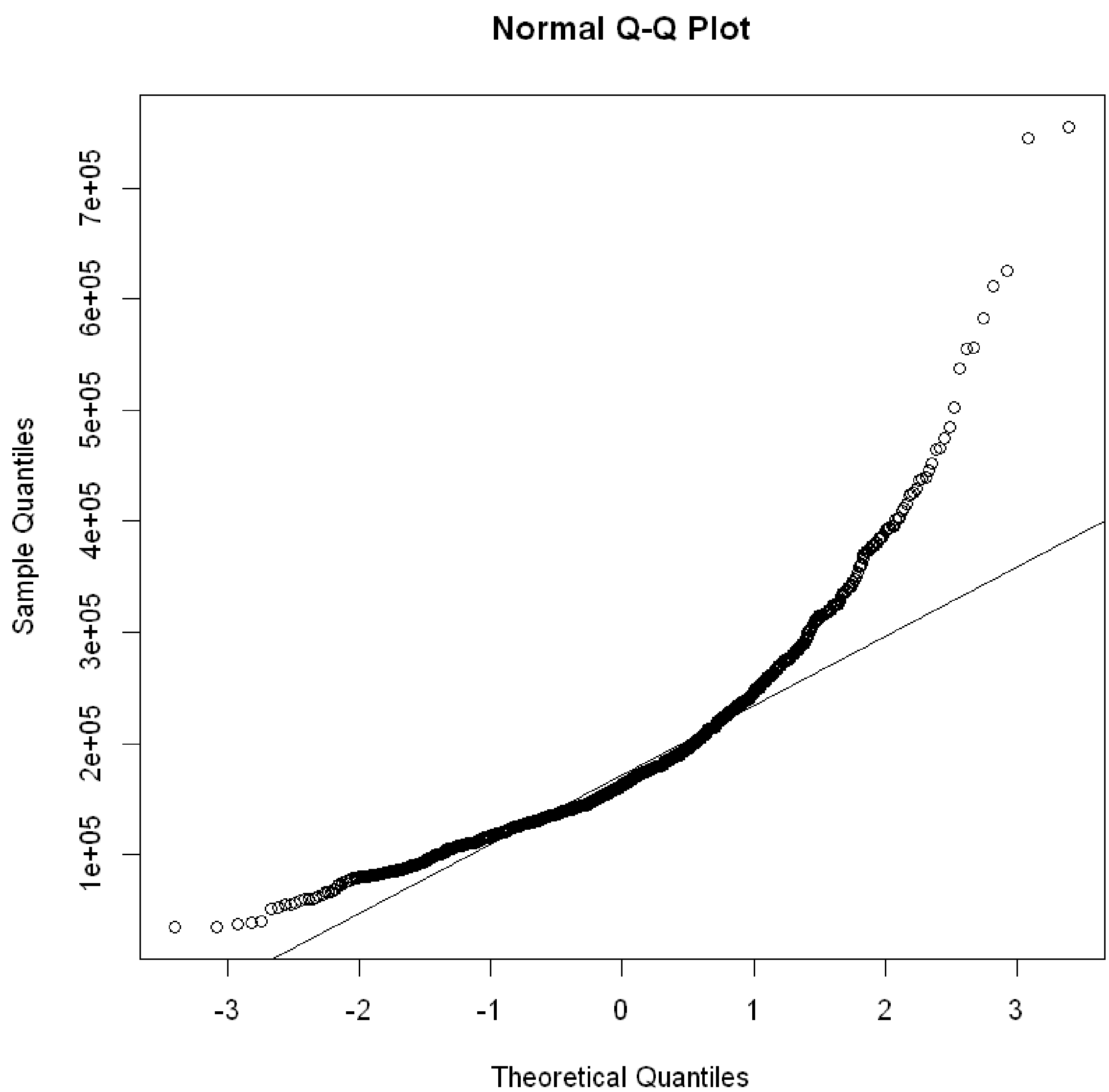
In [124]:
```
dfdummies <- dfdummies[,-PredictorsWithNoObsTrain]
```

**Merging the standardized numerical values dataframe and the encoded one**

In [125]:
```
dfmodel <- cbind(dfnorm,dfdummies)
```

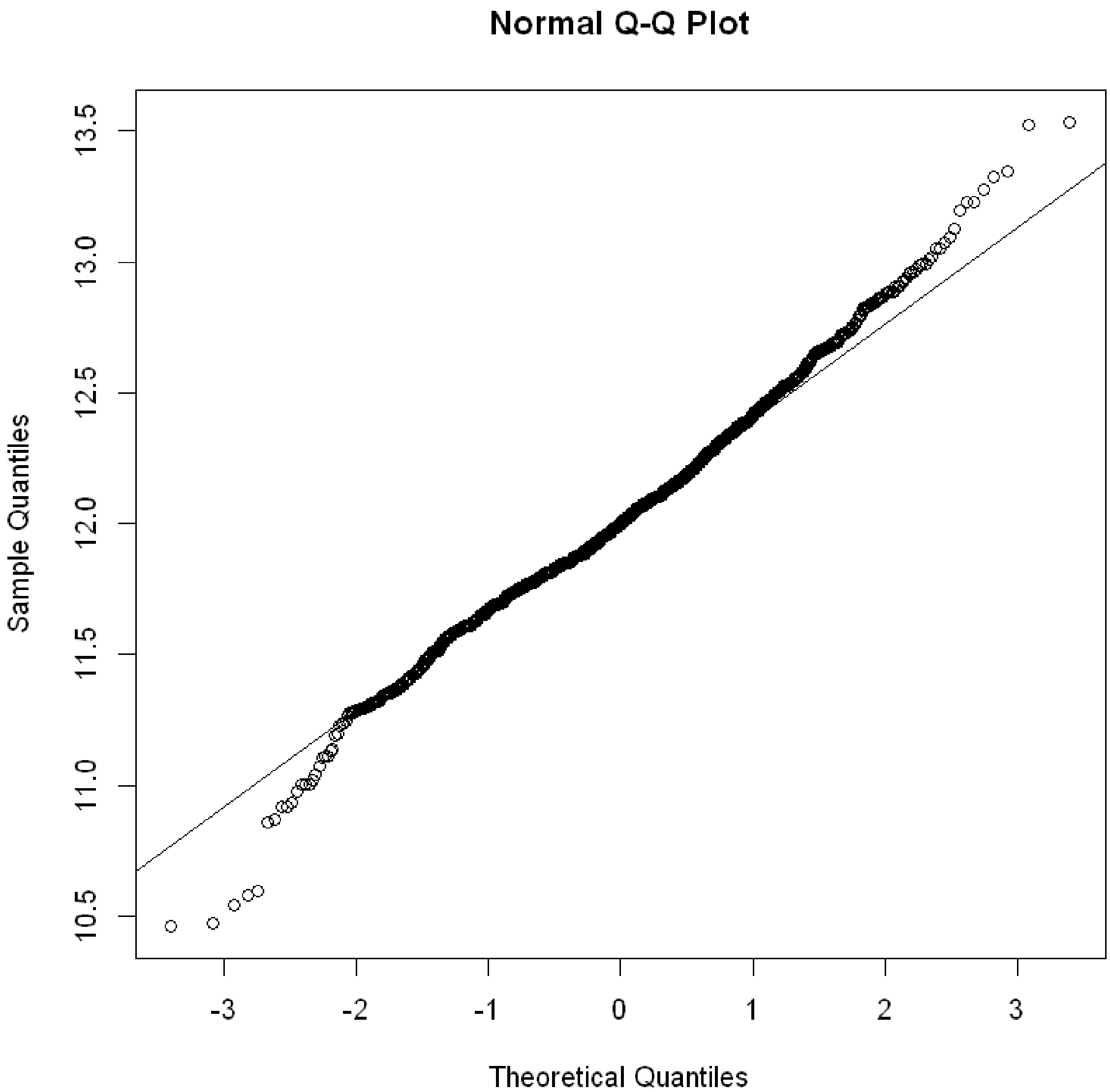
### Skewness of the DV

In [126]:
```
qqnorm(dfmodel$SalePrice)
qqline(dfmodel$SalePrice)
```




This is not satisfactory - we want the line to be as straight as possible. Applying log, we get a much more acceptable qqplot.

In [127]:
```
dfmodel$SalePrice <- log(dfmodel$SalePrice)
qqnorm(dfmodel$SalePrice)
qqline(dfmodel$SalePrice)
```




- Much better

Renaming Columns with out spaces

In [128]:
```
names(dfmodel)[names(dfmodel) == 'AlleyNo Alley'] <- 'AlleyNo_Alley'
names(dfmodel)[names(dfmodel) == 'RoofMatlTar&Grv'] <- 'RoofMatlTar_Grv'
names(dfmodel)[names(dfmodel) == 'Exterior1stWd Sdng'] <- 'Exterior1stWd_Sdng'
names(dfmodel)[names(dfmodel) == 'Exterior2ndBrk Cmn'] <- 'Exterior2ndBrkCmn'
names(dfmodel)[names(dfmodel) == 'Exterior2ndWd Sdng'] <- 'Exterior2ndWdSdng'
names(dfmodel)[names(dfmodel) == 'Exterior2ndWd Shng'] <- 'Exterior2ndWdShng'
names(dfmodel)[names(dfmodel) == 'GarageTypeNo Garage'] <- 'GarageTypeNoGarag
e'
names(dfmodel)[names(dfmodel) == 'FenceNo Fence'] <- 'FenceNoFence'
names(dfmodel)[names(dfmodel) == 'MiscFeatureNo Features'] <- 'MiscFeatureNoFe
atures'
```

Train and Test Sets ready for modeling

In [129]:
```
train1 <- dfmodel[!is.na(dfmodel$SalePrice),]
test1 <- dfmodel[is.na(dfmodel$SalePrice),]
```

In [130]:
```
head(train1,2)
head(test1,2)
```

A data.frame: 2 × 214

| LotFrontage | LotArea | YearBuilt | YearRemodAdd | MasVnrArea | BsmtFinSF2 | BsmtUnfSF | X1: |
|---|---|---|---|---|---|---|---|
| <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <( |
| -0.2089718 | -0.21639955 | 1.0470513 | 0.8975478 | 0.5339956 | -0.2930841 | -0.9336018 | -0.7 |
| 0.4984058 | -0.06909653 | 0.1555794 | -0.3947970 | -0.5669270 | -0.2930841 | -0.6288477 | 0.2 |

A data.frame: 2 × 214

| | LotFrontage | LotArea | YearBuilt | YearRemodAdd | MasVnrArea | BsmtFinSF2 | BsmtUnfSF |
|---|---|---|---|---|---|---|---|
| | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> | <dbl[,1]> |
| **1461** | 0.4984058 | 0.1899006 | -0.3396827 | -1.112766 | -0.56692703 | 0.5578182 | -0.6606877 |
| **1462** | 0.5455643 | 0.5286975 | -0.4387351 | -1.256360 | 0.03970378 | -0.2930841 | -0.3513851 |

# 9. Modeling

Now, having all features processed and engineered, we can start the model training then testing with all existing features as model predictors

# 9.1 Linear Regression with cross validation

We are going to start off with a Linear Regression model with cross validation to set our baseline

This will be used to compare the other models that we are going to build to see how well they perform. Ideally our RMSE values for the following models should be better than our baseline model.

Going to run a linear regression model considering SalePrice as our dependent variable and all other vairables as the predictor or independent variables

In [131]:
```
set.seed(123)
lm_mod <- train(SalePrice~.,
                  data = train1,
                  method = "lm",
                  trControl=trainControl(
                    method = "cv",
                    number=5,
                    savePredictions = TRUE,
                    verboseIter = TRUE)
)
```

```
+ Fold1: intercept=TRUE
- Fold1: intercept=TRUE
+ Fold2: intercept=TRUE
- Fold2: intercept=TRUE
+ Fold3: intercept=TRUE
- Fold3: intercept=TRUE
+ Fold4: intercept=TRUE
- Fold4: intercept=TRUE
+ Fold5: intercept=TRUE
- Fold5: intercept=TRUE
Aggregating results
Fitting final model on full training set
```

In [132]:
```
lm_mod$results$RMSE
```

0.123383759964574

The RMSE is 0.1233.. which is a what we will be trying to beat with our subsequent models that we will build

In [133]:
```
#Predicting on test dataset

lm.predict.test <- predict(lm_mod, test1)
predictions_lm <- exp(lm.predict.test) #need to reverse the log to the real va
lues
head(predictions_lm)

##code to ADD ID column
```

| | |
|---|---|
| **1461** | 118905.679599649 |
| **1462** | 148824.472067011 |
| **1463** | 177338.305675302 |
| **1464** | 202477.917726432 |
| **1465** | 201390.424659108 |
| **1466** | 171068.417269435 |

In [134]:
```
lm.sol <- data.frame(Id = testID, SalePrice = predictions_lm)
```

In [135]:
```
write.csv(lm.sol, "TestPredlinear.csv",row.names=FALSE)
```

## 9.2 Lasso Regression

Lasso shrinks coefficients all the way to zero, effectively dropping unecessary variables.

In [136]:
```
SP <- c('SalePrice')
```

For Lasso, We remove the variable SalePrice from the samples and run it.

In [137]:
```
train2 <- train1 %>% select (-SP)
test2 <- test1 %>% select(-SP)
```

In [138]:
```
set.seed(123)
my_control <-trainControl(method="cv", number=5)
lassoGrid <- expand.grid(alpha = 1, lambda = seq(0.001,0.1,by = 0.0005))

lasso_mod <- train(x=train2, y=dfmodel$SalePrice[!is.na(dfmodel$SalePrice)], m
ethod='glmnet', trControl= my_control, tuneGrid=lassoGrid)
lasso_mod$bestTune
```

A data.frame: 1 × 2

| | alpha | lambda |
|---|---|---|
| | <dbl> | <dbl> |
| **4** | 1 | 0.0025 |

In [139]:
```
min(lasso_mod$results$RMSE)
min(lasso_mod$results$Rsquared)
min(lasso_mod$results$MAE)
```

0.113738283115503

0.830879200867707

0.0796113087326443

### Feature selection from Lasso

In [140]:
```
lassoVarImp <- varImp(lasso_mod,scale=F)
lassoImportance <- lassoVarImp$importance

varsSelected <- length(which(lassoImportance$Overall!=0))
varsNotSelected <- length(which(lassoImportance$Overall==0))
```

### Variables selected

In [141]:
```
varsSelected
```

109

### Variables dropped by Lasso

In [142]:
```
varsNotSelected
```

104

### Predicting Sales price on test dataset

In [143]:
```
LassoPred <- predict(lasso_mod, test1)
predictions_lasso <- exp(LassoPred) #need to reverse the log to the real value
s
head(predictions_lasso)

##code to ADD ID column
```

| | |
|---|---|
| **1461** | 113144.752908856 |
| **1462** | 158486.952315333 |
| **1463** | 177001.144404305 |
| **1464** | 200043.181318015 |
| **1465** | 200164.202673657 |
| **1466** | 170571.281858478 |

In [144]:
```
lass.sol <- data.frame(Id = testID, SalePrice = predictions_lasso)
```

In [145]:
```
write.csv(lass.sol, "TestPredlasso.csv",row.names=FALSE)
```

# 9.3 Ridge Regression Model

Ridge regression performs shrinkage without exclusion of predictors. We will first create a matrix of all predictors and a vector of the response before we pass it into the glmner function.

In [146]:
```
train.ridgeMatrix <- as.matrix(train1[,names(train1) != c("SalePrice")])
test.ridgeMatrix <- as.matrix(test1[,names(test1) != c("SalePrice")])
```

In [147]:
```
train.ridge.y <- train1$SalePrice
```

In [148]:
```
set.seed(123)
```

Creating a grid of lambda values which is basically the tuning grid

In [149]:
```
grid = 10^seq(10,-2, length = 100)
```

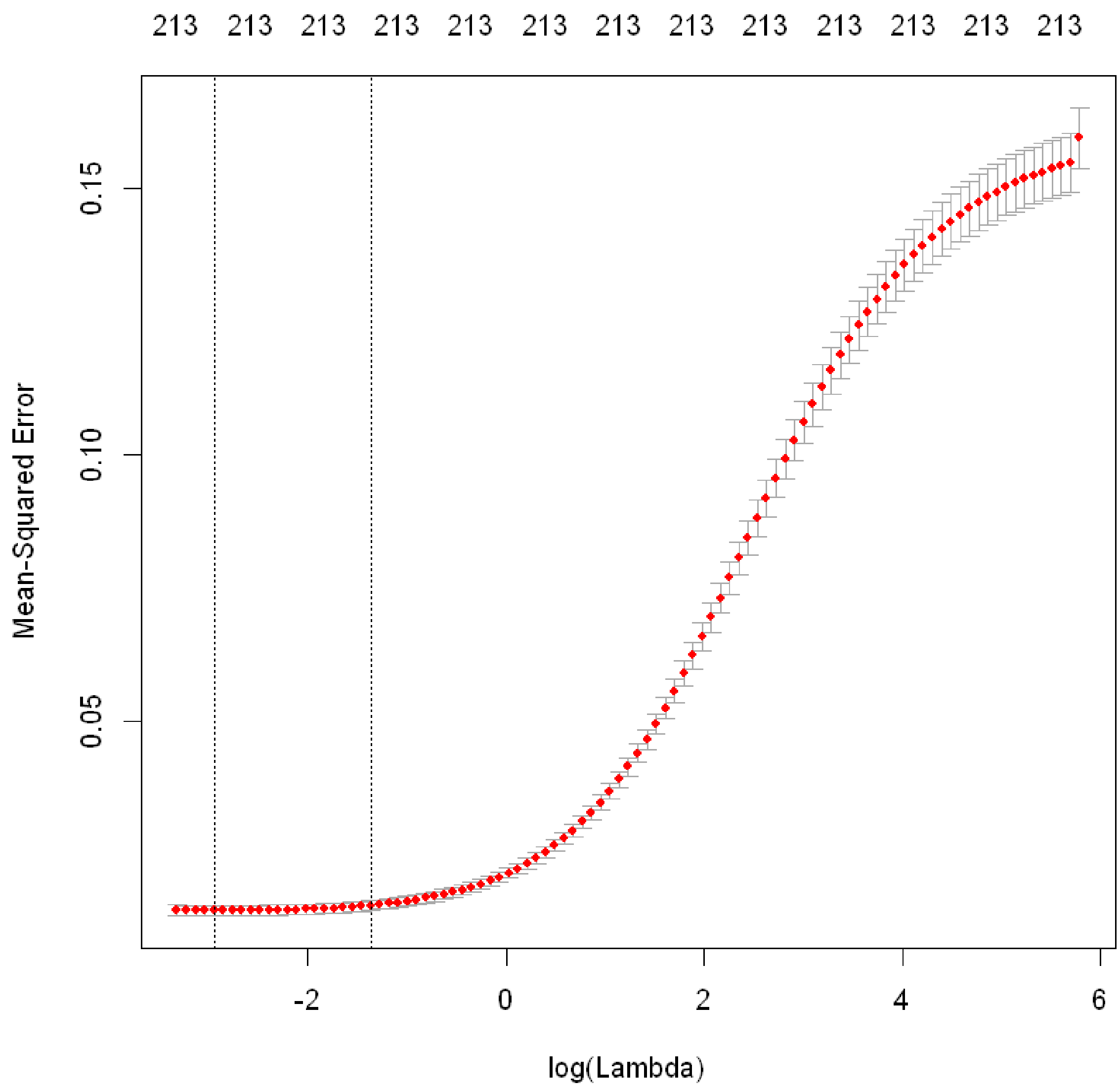
In [150]:
```
# train the Ridge Regression model using the grid of selected lambdas with alp
ha=0
ridge.mod <- glmnet(train.ridgeMatrix,train.ridge.y,alpha = 0, lambda = grid)
dim(coef(ridge.mod))
```

214 100

## 9.3.1 Ridge regression by K-Fold cross validation

In [151]:
```
set.seed(123)
cv.out <- cv.glmnet(train.ridgeMatrix, train.ridge.y,alpha=0)
```

In [152]:
```
plot(cv.out)
```




Finding out the best lambda from the cross validation exercise

In [153]:
```
bestlam <- cv.out$lambda.min
bestlam
```

0.0522611783512238

The best lambda value that results in the smallest cross validation error is 0.0522611783512238. Let's see the RMSE associated with this value of lambda

In [154]:
```
#use it in fitting the training data
ridge.pred <- predict(ridge.mod, s=bestlam, newx=train.ridgeMatrix)
rmse(train.ridge.y,ridge.pred)
```

0.100425934450849

Now that we have finalized our ridge regression model we will predict SalePrice on the test data set

In [155]:
```
ridge.test.pred <- predict(ridge.mod, s=bestlam, newx=test.ridgeMatrix)
predictions_Ridge <- exp(ridge.test.pred)
#Code to Add ID column
```

In [156]:
```
ridge.sol  <- data.frame(Id = testID, SalePrice = predictions_Ridge)
```

In [157]:
```
write.csv(ridge.sol, "TestPredRidge.csv",row.names=FALSE)
```

# 9.4 Random Forests

Random forests are built on the same fundamental principles as decision trees and bagging. Since the algorithm randomly selects a bootstrap sample to train on and predictors to use at each split, tree correlation will be lessened beyond bagged trees.

In [158]:
```
# for reproduciblity
set.seed(123)
```

We will first start off with a basic model before we try and tune the different parameters available to us

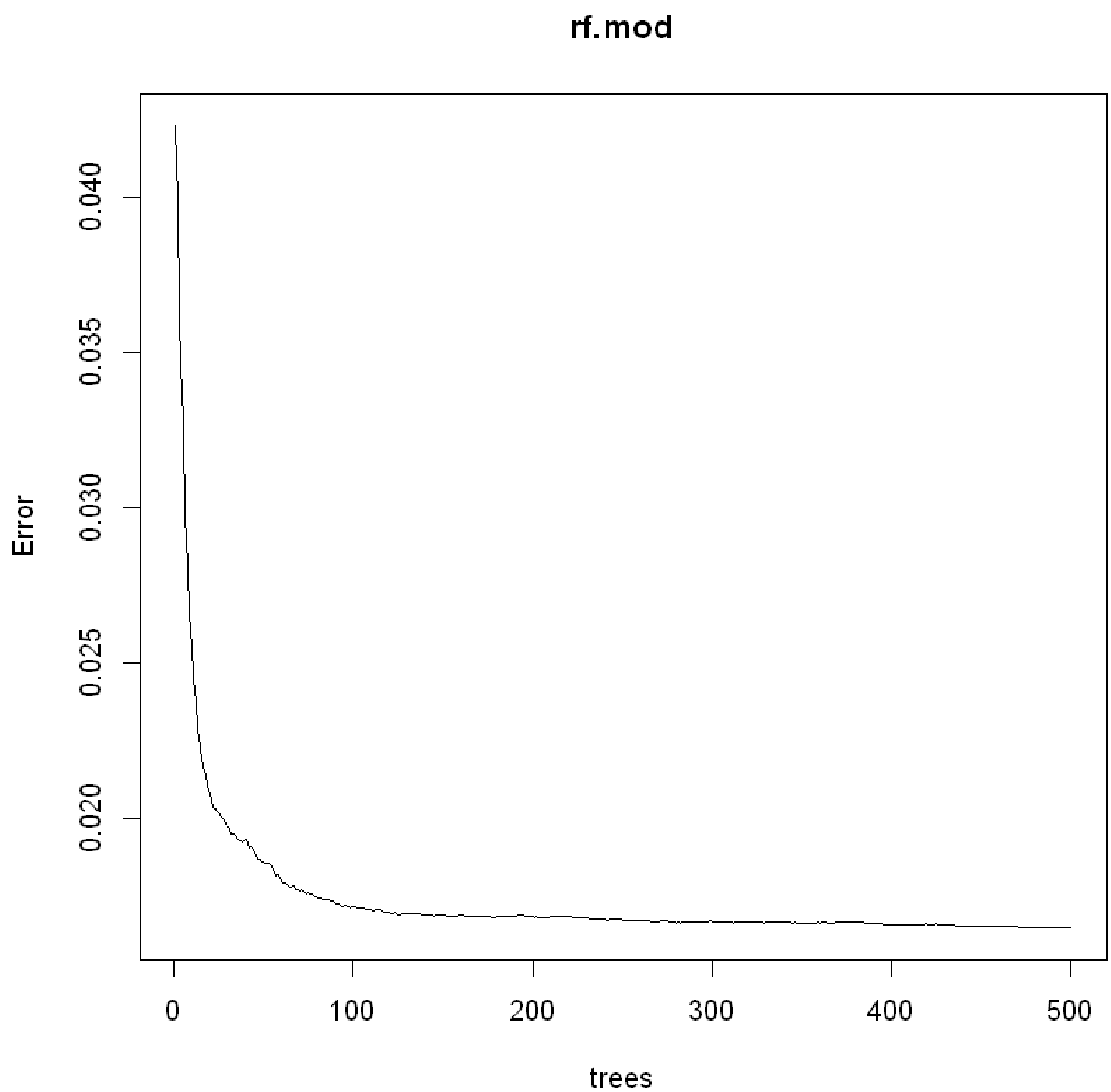
In [159]:
```
rf.mod <- randomForest(
  formula = SalePrice~.,
  data    = train1)

rf.mod
```

```
Call:
 randomForest(formula = SalePrice ~ ., data = train1)
              Type of random forest: regression
                    Number of trees: 500
No. of variables tried at each split: 71

          Mean of squared residuals: 0.01649832
                    % Var explained: 89.67
```

In [160]:
```
plot(rf.mod)
```




Plotting the model will illustrate the error rate as we average across more trees and shows that our error rate stabalizes with around 100 trees but continues to decrease slowly until around 300 or so trees.

Random forests are fairly easy to tune since there are only a handful of tuning parameters. Typically, the primary concern when starting out is tuning the number of candidate variables to select from at each split. However, there are a few additional hyperparameters that we will be tuning with the help of a grid.

In [161]:
```
# hyperparameter grid search
rf.grid <- expand.grid(
  mtry        = seq(20, 200, by = 5),
  node_size   = seq(3, 9, by = 2),
  sampe_size = c(.70, .80),
  OOB_RMSE    = 0
)

# total number of combinations
nrow(rf.grid)
```

296

In [162]:
```
for(i in 1:nrow(rf.grid)) {

  # train model
  rf.mod <- ranger(
    formula         = SalePrice ~ .,
    data            = train1,
    num.trees       = 500,
    mtry            = rf.grid$mtry[i],
    min.node.size   = rf.grid$node_size[i],
    sample.fraction = rf.grid$sampe_size[i],
    seed            = 123
  )

  # add OOB error to grid
  rf.grid$OOB_RMSE[i] <- sqrt(rf.mod$prediction.error)
}

rf.grid %>%
  dplyr::arrange(OOB_RMSE) %>%
  head(10)
```

A data.frame: 10 × 4

| mtry | node_size | sampe_size | OOB_RMSE |
|---|---|---|---|
| <dbl> | <dbl> | <dbl> | <dbl> |
| 70 | 5 | 0.7 | 0.1287370 |
| 50 | 3 | 0.7 | 0.1287771 |
| 70 | 7 | 0.7 | 0.1290654 |
| 70 | 3 | 0.8 | 0.1290725 |
| 55 | 3 | 0.8 | 0.1290956 |
| 70 | 3 | 0.7 | 0.1291581 |
| 85 | 3 | 0.8 | 0.1291800 |
| 70 | 7 | 0.8 | 0.1293032 |
| 85 | 5 | 0.8 | 0.1293726 |
| 95 | 3 | 0.8 | 0.1293965 |

Currently, the best random forest model we have found uses mtry = 70, terminal node size of 5 observations, and a sample size of 70%. We will repeat this to see what we should expect as error rates

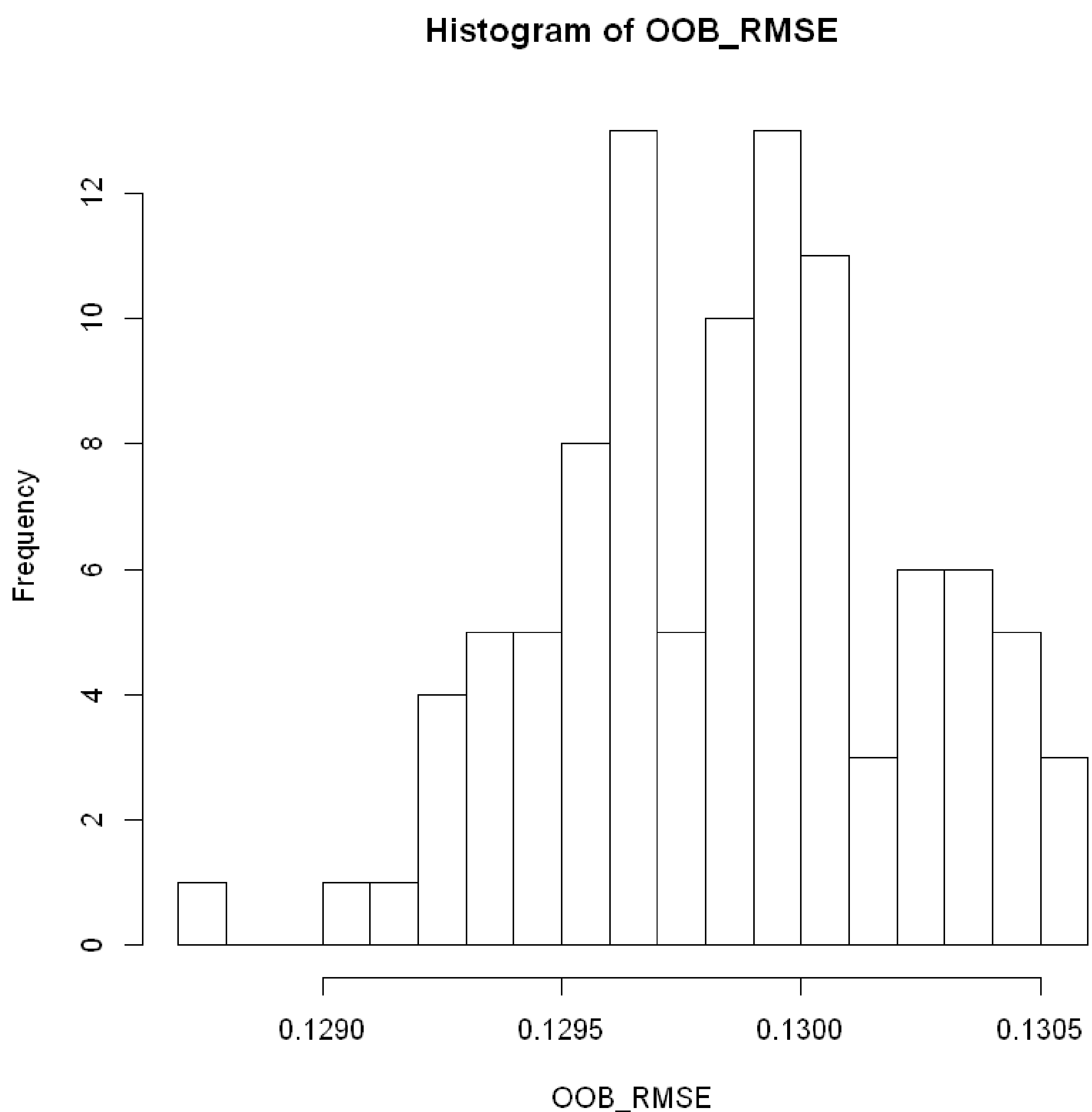
In [163]:
```
OOB_RMSE <- vector(mode = "numeric", length = 100)

for(i in seq_along(OOB_RMSE)) {

  optimal_ranger <- ranger(
    formula         = SalePrice ~ .,
    data            = train1,
    num.trees       = 500,
    mtry            = 70,
    min.node.size   = 5,
    sample.fraction = .7,
    importance      = 'impurity'
  )

  OOB_RMSE[i] <- sqrt(optimal_ranger$prediction.error)
}

hist(OOB_RMSE, breaks = 20)
```



Finalizing the best Random Forest Model

In [164]:
```
set.seed(123)
optimal_ranger <- ranger(
    formula         = SalePrice ~ .,
    data            = train1,
    num.trees       = 500,
    mtry            = 70,
    min.node.size   = 5,
    sample.fraction = .7,
    importance      = 'impurity'
  )
```

In [165]:
```
sqrt(optimal_ranger$prediction.error)
```

0.128881753547348

Now that we have finalised our Random Forest model we can precdict the SalePrice on the test dataset

In [166]:
```
#
pred_ranger <- predict(optimal_ranger, test2)

rf.predictions <- exp(pred_ranger$predictions) #need to reverse the log to the
real values
head(rf.predictions)
```

121128.318457925 156760.832420942 179915.115146642 181642.176343631
193888.077568018 183044.945709552

In [167]:
```
rf.sol  <- data.frame(Id = testID, SalePrice = rf.predictions)
```

In [168]:
```
write.csv(rf.sol,"testPredRF.csv",row.names=FALSE)
```

# 9.5 Ensemble Model with Weights

In [169]:
```
# construct data frame for the ensembled solution
solution <- data.frame(Id = testID,
                       SalePrice =  as.numeric(predictions_Ridge*.2 + predicti
ons_lasso*.7 + predictions_lm*.1))
```

In [170]:
```
write.csv(solution,"ensemble_sol.csv",row.names=FALSE)
```