Kaggle_House_Price

Fan Gong

2017/9/1

Housing Price Analysis

Feature Engineering

```
library(dplyr)
library(e1071)
library(ggplot2)
library(corrplot)
library(reshape2)
library(glmnet)
library(caret)
library(xgboost)
```

Problem Overview

First let me look at each variable and do a philosophical analysis about their meaning and importance.

```
df = read.csv("/Users/gongfan/Desktop/kaggle/Housing Price/train.csv", stringsAsFactors = F)
df_test = read.csv("/Users/gongfan/Desktop/kaggle/Housing Price/test.csv", stringsAsFactors = F)
#Column Names
colnames(df)
```

```
##  [1] "Id"            "MSSubClass"    "MSZoning"      "LotFrontage"
##  [5] "LotArea"       "Street"        "Alley"         "LotShape"
##  [9] "LandContour"   "Utilities"     "LotConfig"     "LandSlope"
## [13] "Neighborhood"  "Condition1"    "Condition2"    "BldgType"
## [17] "HouseStyle"    "OverallQual"   "OverallCond"   "YearBuilt"
## [21] "YearRemodAdd"  "RoofStyle"     "RoofMatl"      "Exterior1st"
## [25] "Exterior2nd"   "MasVnrType"    "MasVnrArea"    "ExterQual"
## [29] "ExterCond"     "Foundation"    "BsmtQual"      "BsmtCond"
## [33] "BsmtExposure"  "BsmtFinType1"  "BsmtFinSF1"    "BsmtFinType2"
## [37] "BsmtFinSF2"    "BsmtUnfSF"     "TotalBsmtSF"   "Heating"
## [41] "HeatingQC"     "CentralAir"    "Electrical"    "X1stFlrSF"
## [45] "X2ndFlrSF"     "LowQualFinSF"  "GrLivArea"     "BsmtFullBath"
## [49] "BsmtHalfBath"  "FullBath"      "HalfBath"      "BedroomAbvGr"
## [53] "KitchenAbvGr"  "KitchenQual"   "TotRmsAbvGrd"  "Functional"
## [57] "Fireplaces"    "FireplaceQu"   "GarageType"    "GarageYrBlt"
## [61] "GarageFinish"  "GarageCars"    "GarageArea"    "GarageQual"
## [65] "GarageCond"    "PavedDrive"    "WoodDeckSF"    "OpenPorchSF"
## [69] "EnclosedPorch" "X3SsnPorch"    "ScreenPorch"   "PoolArea"
## [73] "PoolQC"        "Fence"         "MiscFeature"   "MiscVal"
## [77] "MoSold"        "YrSold"        "SaleType"      "SaleCondition"
## [81] "SalePrice"
```

In order to have some discipline in our analysis, I divide all the variables into three groups: **building**, **space** or **location**. When I say 'bulding', I mean a variable that relates to the physical characteristics of the

building. When I say 'space', I mean a variable that reports space properties of the house. Finally, when I say 'location', I mean a variable that gives inforamtion about the place where the house is located.

```
feature_segmentation = read.csv("Feature_Segmentation.csv")
head(feature_segmentation[,4:6])
```

```
##      Building       Space     Location
## 1  MSSubClass     LotArea     MSZoning
## 2      Street  MasVnrArea  LotFrontage
## 3       Alley  BsmtFinSF1 Neighborhood
## 4    LotShape  BsmtFinSF2   Condition1
## 5 LandContour   BsmtUnfSF   Condition2
## 6   Utilities TotalBsmtSF   GarageType
```

In order to make this table, I can have a general idea about all the variables. Besides, I am able to know some relationship between some variables based on their discription. For example, the variable `GarageCars` and `GarageArea` both illustrate the size of garage.

Data Cleaning

After having an overview about the variables, I would like to then clean the data. Because the prerequisite for performing analysis is that there is no missing data in the datasets.

```
#Combine the training and testing data
df_test$SalePrice = 0
df_combine = rbind(df, df_test)
#NA Check
na_check = apply(is.na(df_combine), 2, sum)
na_check[na_check != 0]
```

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

```
length(na_check[na_check != 0])
```

```
## [1] 34
```

We could see that for `Alley`, `BsmtQual`, `BsmtCond`, `BsmtExposure`, `BsmtFinType1`, `BsmtFinType2`, `FireplaceQu`, `GarageType`, `GarageYrBlt`, `GarageFinish`, `GarageQual`, `GarageCond`, `poolQC`, `Fence`, `MiscFeature`, they have lots of missing data. But here `NA` represents one specific class, not represent for missing data. So we would better replace them to `None`.

However, for `MSZoing`, `LotFrontage`, `Utilities`, `Exterior1st`, `Exterior2nd`, `KitchenQual`, `Functional`, `MasVnrType`, `MasVnrArea`, `Electrical`, `SaleType`, they do have some missing values. Other variable

like BsmtFinSF1, BsmtFinSF2??? BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath, GarageCars, GarageAreaneed to be determined even though they don't have missing value.

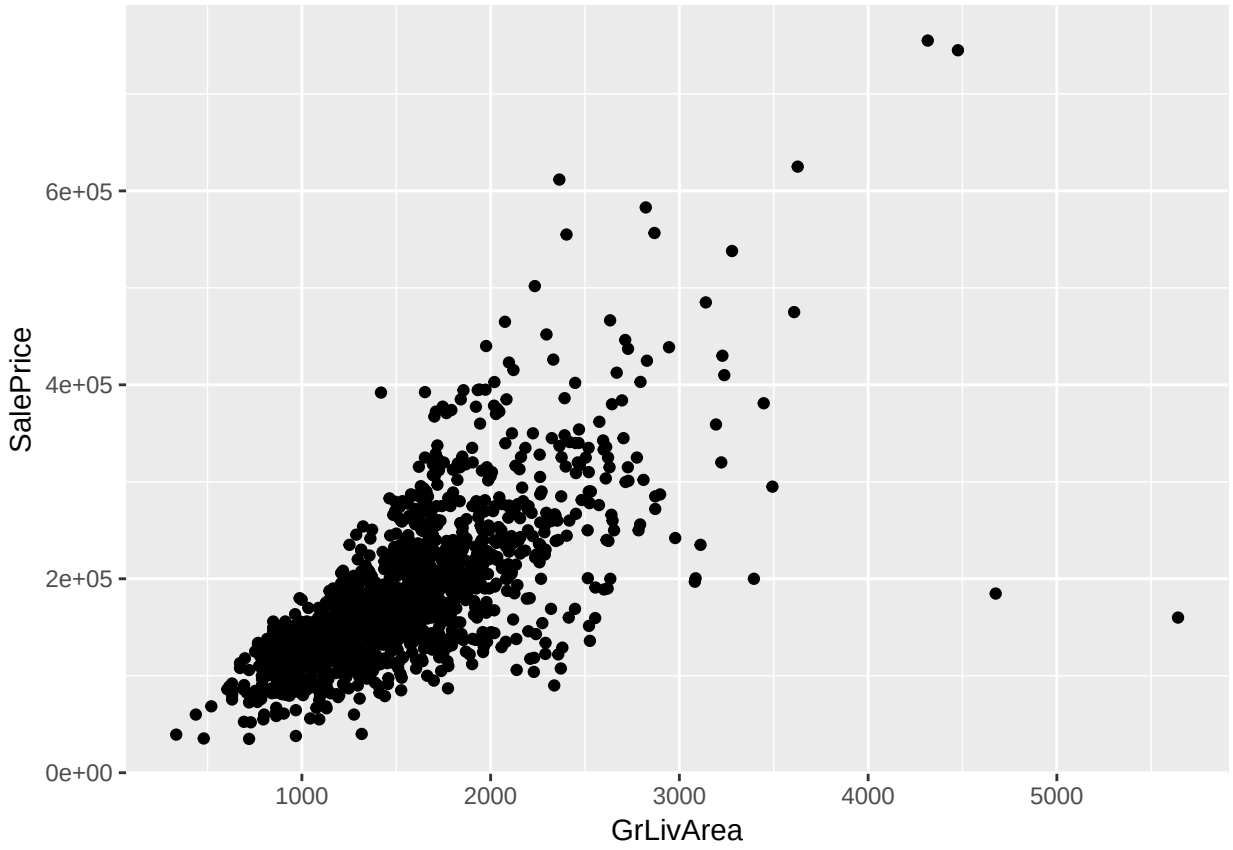
1. For LotFrontage, it means linear feet of street connected to property. Since we have 486 missing values, it is better to exclude this variable.
2. For BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath, GarageCars, GarageArea, the NAs here means there is no basement and no garage. So we could assign all these value to 0.
3. For MSZoning, Utilities, Exterior1st, Exterior2nd, Electrical, KitchenQual, Functional, SaleType, MasVnrType and MasVnrArea, I will use the most frequent value in each column.

```
#Deal with missing data variables
##LotFrontage
df_combine = df_combine %>% select(-LotFrontage)

##Basement and Garage related variables
bsm_na_id = df_combine[is.na(df_combine$BsmtFinSF1),]$Id
new_row = df_combine %>% filter(Id %in% bsm_na_id) %>% mutate(BsmtFinSF1=0, BsmtFinSF2=0,BsmtUnfSF=0,To
df_combine[df_combine$Id %in% bsm_na_id,] = new_row

ga_na_id = df_combine[is.na(df_combine$GarageCars),]$Id
new_row = df_combine %>% filter(Id %in% ga_na_id) %>% mutate(GarageCars=0, GarageArea=0)
df_combine[df_combine$Id %in% ga_na_id,] = new_row

bath_na_id = df_combine[is.na(df_combine$BsmtFullBath),]$Id
new_row = df_combine %>% filter(Id %in% bath_na_id) %>% mutate(BsmtFullBath=0, BsmtHalfBath=0)
df_combine[df_combine$Id %in% bath_na_id,] = new_row

##Impute most frequent value
variables = c('MSZoning', 'Utilities', 'Exterior1st', 'Exterior2nd', 'Electrical', 'KitchenQual', 'Funct

Mode = function(x) {
  ux = unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

for(var in variables){
  if(is.numeric(df_combine[,var])){
    M = as.numeric(Mode(df_combine[,var]))
    df_combine[is.na(df_combine[,var]), var] = M
  }else{
    M = Mode(df_combine[,var])
    df_combine[is.na(df_combine[,var]), var] = M
  }
}

#Replace all the other NA to "None"
df_combine[is.na(df_combine)] = "None"

#Seperate the training and testing data
df = df_combine[1:1460,]
df_test = df_combine[-(1:1460),-ncol(df_combine)]
```

Outlier Removal

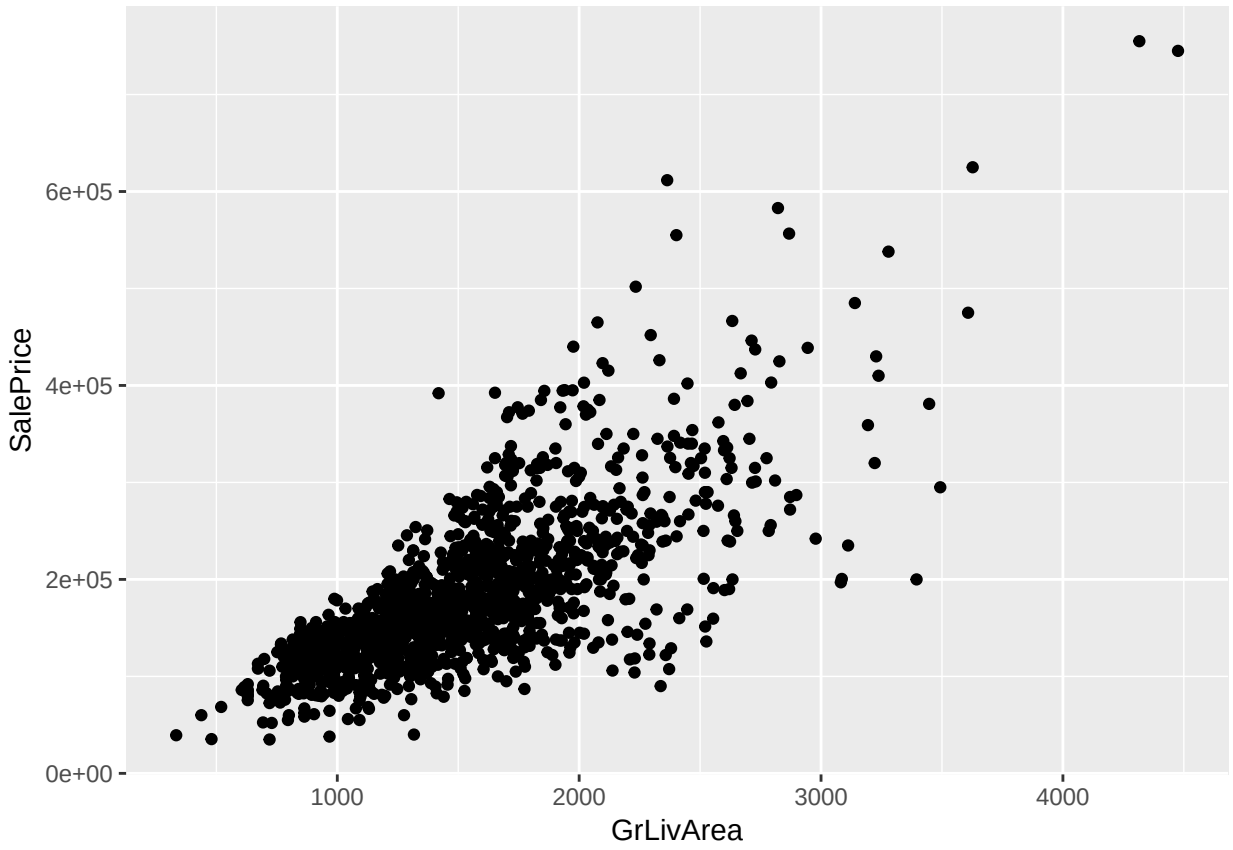
Documentation for the Ames Housing Data indicates that there are outliers present in the training data

```
ggplot(data = df, aes(x = GrLivArea, y = SalePrice)) + geom_point()
```



We can see at the bottom right two with extremely large GrLivArea that are of a low price. These values are huge outliers. Therefore, we can safely delete them.

```
df = df %>% filter(!(GrLivArea > 4000 & SalePrice < 300000))
ggplot(data = df, aes(x = GrLivArea, y = SalePrice)) + geom_point()
```



Variable Transformation

I want our dataset to be strictly numeric values while remaining as much information as possible, I will have to transform any categoric into a binary feature using one-hot encoding.

I start by creating a new data frame of just the numeric features.

```
#Exclude ID
df_combine = rbind(df[,-ncol(df)], df_test)
df_combine= df_combine[,-1]
#Extract numeric data
logic_numeric = sapply(df_combine, is.numeric)
df_numeric = df_combine[,logic_numeric]

#Extract categorical data
cat_features = names(which(sapply(df_combine, is.character)))
```

Ordinal Categorical Variable Transformation

We can transform any of the ordinal variables into numeric values. We can do this by determining which order the categories follow and assigning the values an order from 1,2,. . . ,n. We will group each feature by its possible values and return the median SalePrice and mean OverallQual for each unique value, then map the higher priced/better quality homes larger numeric values and the lower priced/lower quality homes samller numeric values.

```
group.df = df_combine[1:nrow(df),]
group.df$SalePrice = df$SalePrice
```

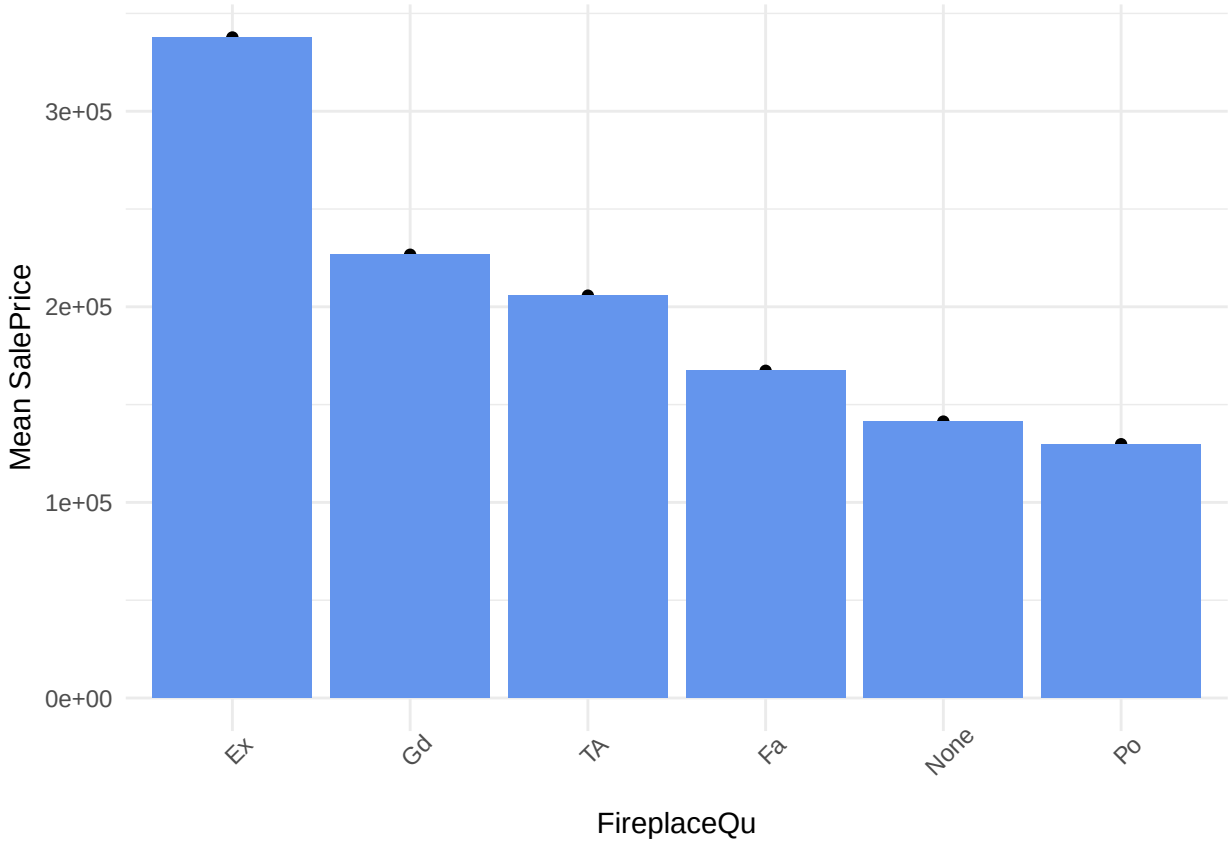
```
#Function that groups a column by its features and returns the median saleprice for each unique feature
group.prices = function(col) {
  group.table = group.df[,c(col, 'SalePrice', 'OverallQual')] %>% group_by_(col) %>%
    summarise(mean.Quality = round(mean(OverallQual),2),
      mean.Price = mean(SalePrice), n = n()) %>%
    arrange(mean.Quality)

  print(qplot(x=reorder(group.table[[col]], -group.table[['mean.Price']]), y=group.table[['mean.Price']]
    geom_bar(stat='identity', fill='cornflowerblue') +
    theme_minimal() +
    labs(x=col, y='Mean SalePrice') +
    theme(axis.text.x = element_text(angle = 45)))

  return(data.frame(group.table))
}

#Function to compute the mean overall quality for each quality
quality.mean = function(col) {
  group.table = df.combined[,c(col, 'OverallQual')] %>%
    group_by_(col) %>%
    summarise(mean.qual = mean(OverallQual)) %>%
    arrange(mean.qual)

  return(data.frame(group.table))
}

#Function that maps a categoric value to its corresponding numeric value and returns that column to the
map.fcn = function(cols, map.list, df){
  for (col in cols){
    df[col] = as.numeric(map.list[df_combine[,col]])
  }
  return(df)
}
```

Any of the columns with the suffix 'Qual' or 'Cond' denote the quality or condition of that specific feature. Each of these columns have the potential values: TA, Fa, Gd, None, Ex, Po. We'll compute the mean house prices for these unique values to get a better sense of what their abbreviations mean.

```
#Categorical Variables (44)
logic_categorical = !(sapply(df, is.numeric))
names(logic_categorical)[logic_categorical]
```

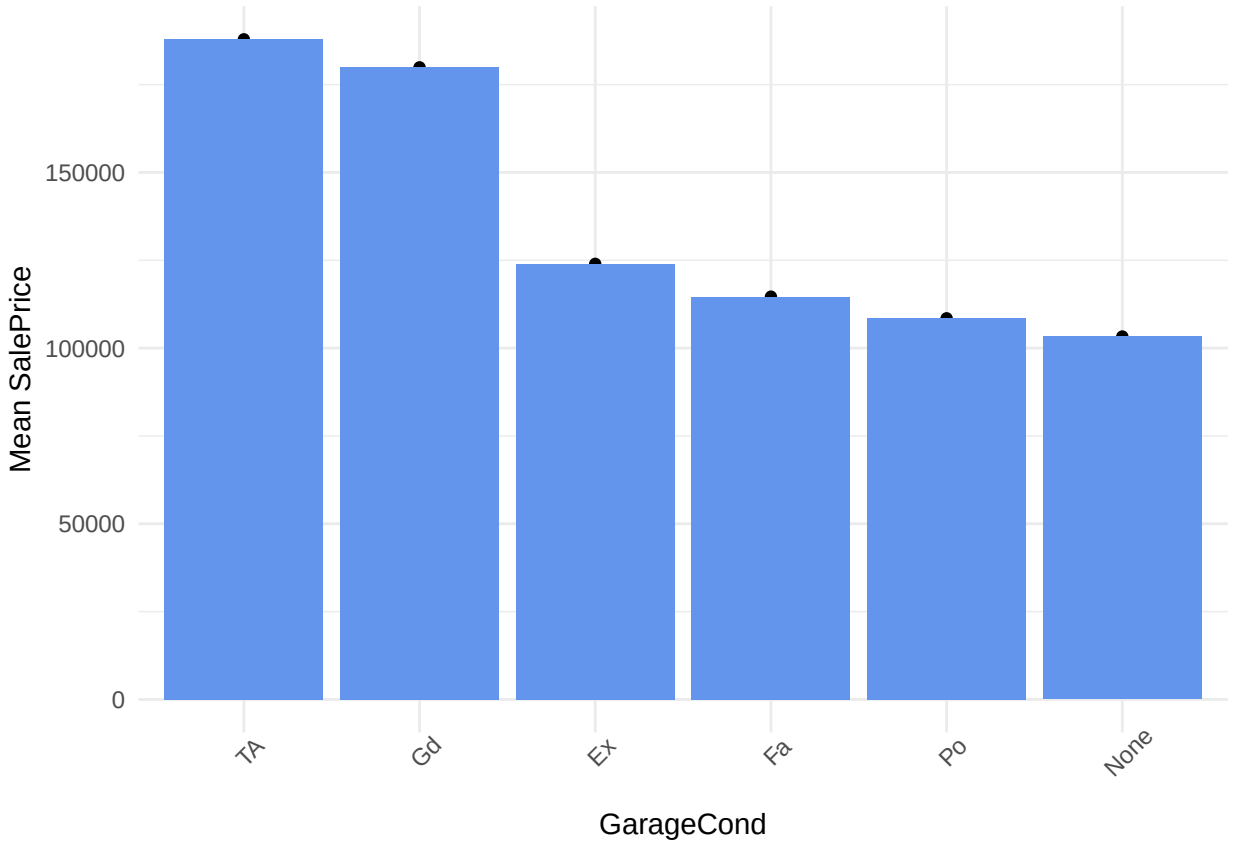
```
##  [1] "MSZoning"      "Street"        "Alley"         "LotShape"
##  [5] "LandContour"   "Utilities"     "LotConfig"     "LandSlope"
##  [9] "Neighborhood"  "Condition1"    "Condition2"    "BldgType"
## [13] "HouseStyle"    "RoofStyle"     "RoofMatl"      "Exterior1st"
## [17] "Exterior2nd"   "MasVnrType"    "ExterQual"     "ExterCond"
## [21] "Foundation"    "BsmtQual"      "BsmtCond"      "BsmtExposure"
## [25] "BsmtFinType1"  "BsmtFinType2"  "Heating"       "HeatingQC"
## [29] "CentralAir"    "Electrical"    "KitchenQual"   "Functional"
## [33] "FireplaceQu"   "GarageType"    "GarageYrBlt"   "GarageFinish"
## [37] "GarageQual"    "GarageCond"    "PavedDrive"    "PoolQC"
## [41] "Fence"         "MiscFeature"   "SaleType"      "SaleCondition"
```

```
#Find the quality columns (8)
qual.cols = c('ExterQual', 'ExterCond', 'GarageQual', 'GarageCond', 'FireplaceQu', 'KitchenQual', 'Heati

group.prices('FireplaceQu')
```




```
##   FireplaceQu mean.Quality mean.Price   n
## 1          Po         4.95   129764.1  20
## 2        None         5.46   141331.5 690
## 3          Fa         5.76   167298.5  33
## 4          TA         6.48   205723.5 313
## 5          Gd         6.88   226637.0 378
## 6          Ex         8.38   337712.5  24
```

```
group.prices('GarageCond')
```




```
##   GarageCond mean.Quality mean.Price    n
## 1       None         4.60   103317.3   81
## 2         Fa         4.83   114654.0   35
## 3         Po         5.14   108500.0    7
## 4         Ex         5.50   124000.0    2
## 5         Gd         5.89   179930.0    9
## 6         TA         6.23   187909.2 1324
```

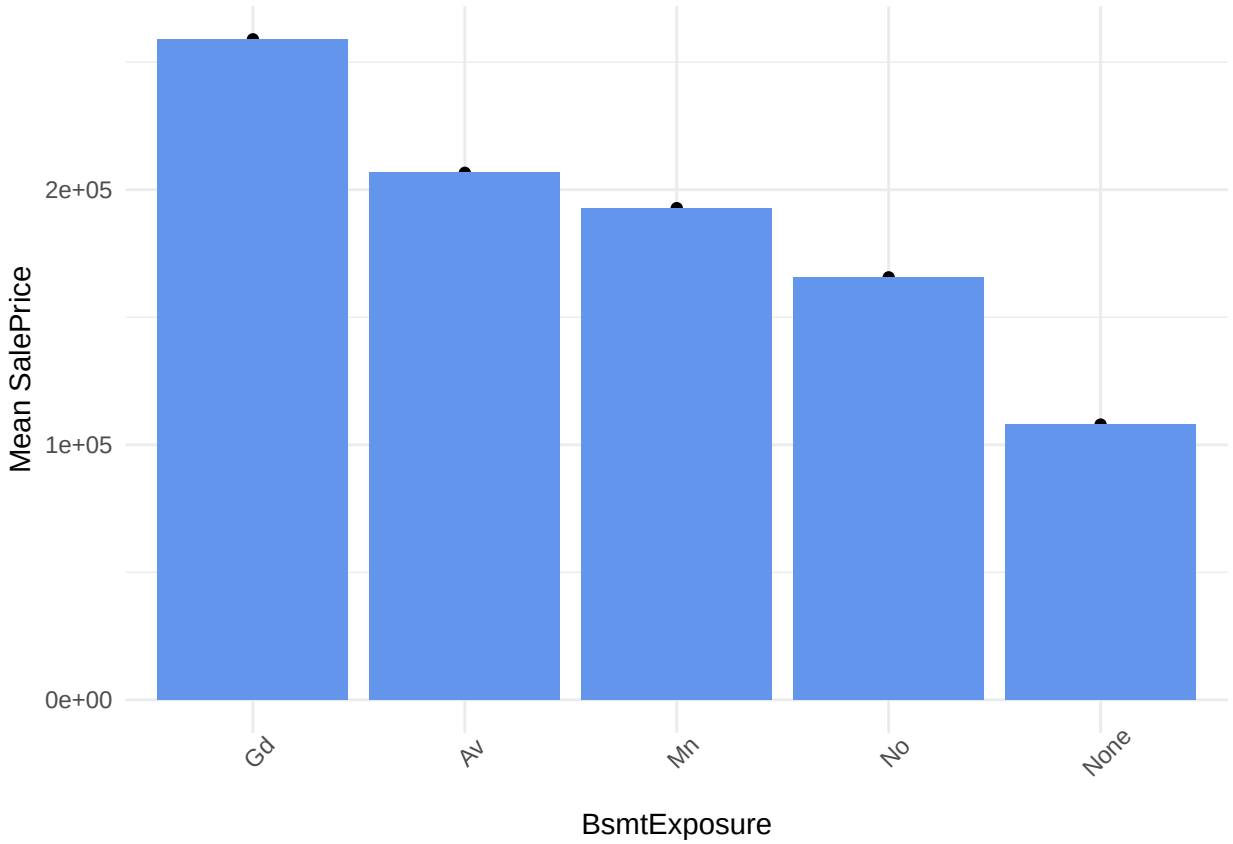
From seeing the mean saleprices from a few of the quality and condition features we can infer that the abbreviations mean poor, fair, typical/average, good and excelent. We'll map numeric values from 0-5 to their corresponding categoric values (including 0 for None) and combine that to our dataframe.

```
qual.list = c('None' = 0, 'Po' = 1, 'Fa' = 2, 'TA' = 3, 'Gd' = 4, 'Ex' = 5)
df_numeric = map.fcn(qual.cols, qual.list, df_numeric)
```

Then we do the same thing to transform other categorical variables into numeric.

`BsmtExposure`:

```
#BsmtExposure (8+1=9)
group.prices('BsmtExposure')
```

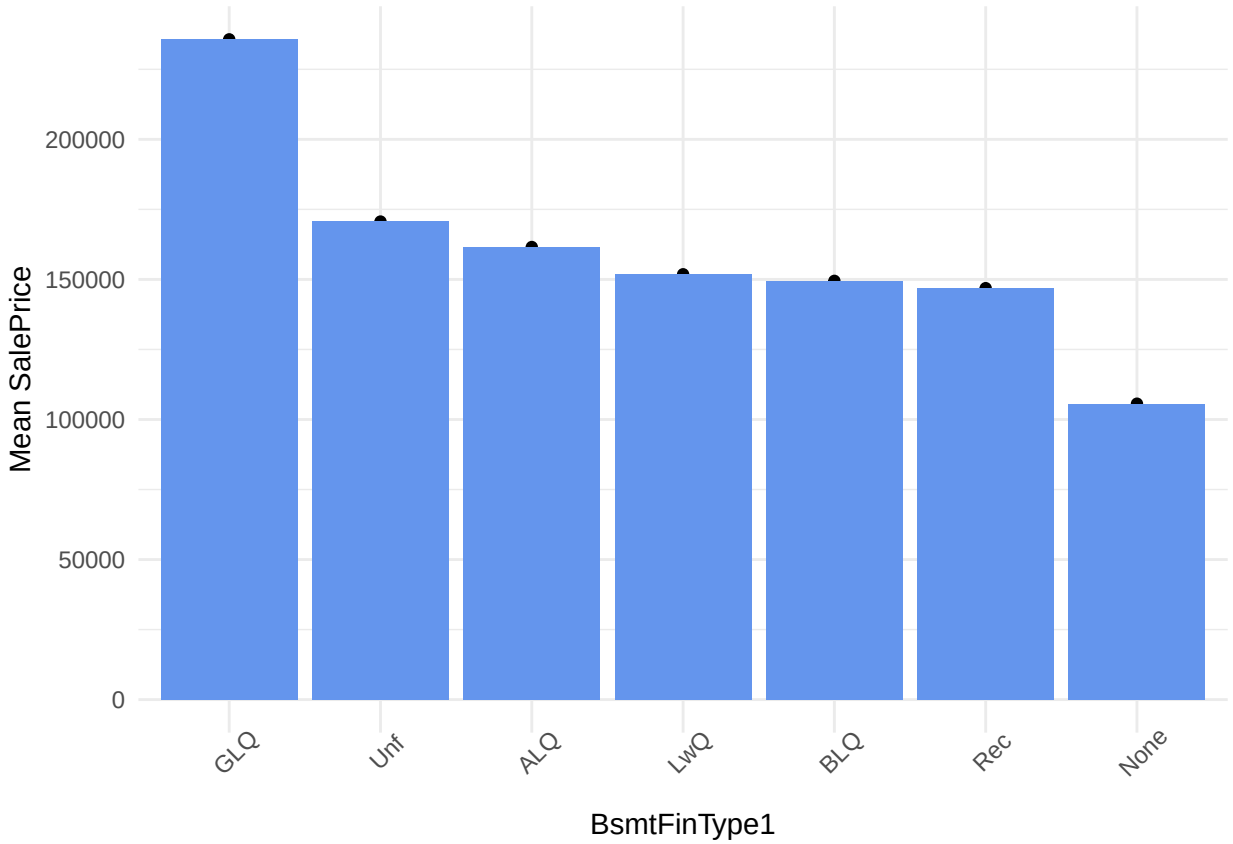



```
##   BsmtExposure mean.Quality mean.Price   n
## 1         None         4.37   107938.3  38
## 2           No         5.92   165652.3 953
## 3           Mn         6.25   192789.7 114
## 4           Av         6.57   206643.4 221
## 5           Gd         6.92   258982.5 132
```

```
bsmt.list = c('None' = 0, 'No' = 1, 'Mn' = 2, 'Av' = 3, 'Gd' = 4)
df_numeric = map.fcn(c('BsmtExposure'), bsmt.list, df_numeric)
```

`BsmtFinType1` and `BsmtFineType2`: `BsmtFinType1` and `BsmtFineType2` represent the quality of the 1st and 2nd finished areas. There should be some order between the different types of qualities, we'll see if we can find a similarity between the 2 features.

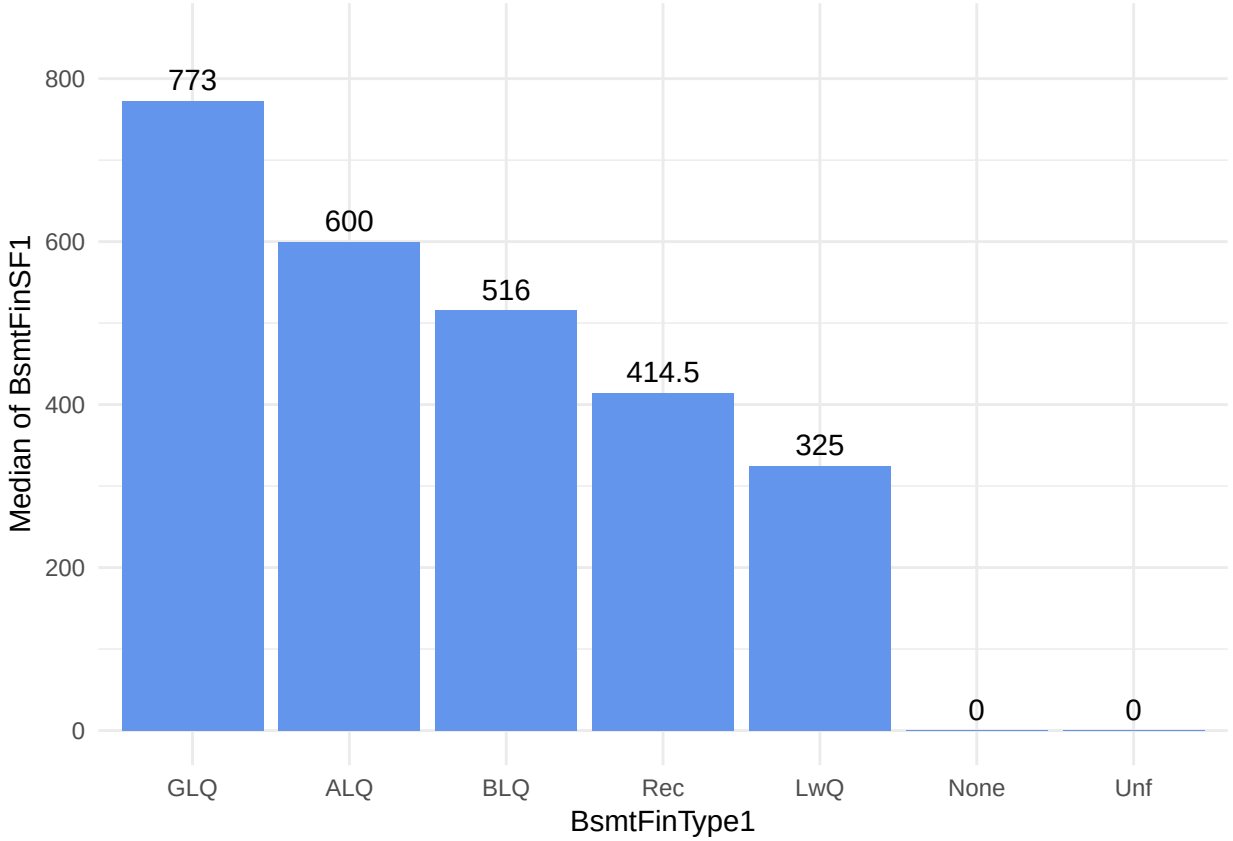
```
group.prices('BsmtFinType1')
```




```
##   BsmtFinType1 mean.Quality mean.Price   n
## 1         None         4.30   105652.9  37
## 2          BLQ         5.35   149493.7 148
## 3          Rec         5.35   146889.2 133
## 4          LwQ         5.54   151852.7  74
## 5          ALQ         5.55   161573.1 220
## 6          Unf         6.20   170670.6 430
## 7          GLQ         7.03   235716.8 416
```

Here returning the mean sale prices might not be as helpful as computing the median basement areas for both columns to determine which quality is better than the other.

```
df_combine[,c('BsmtFinType1', 'BsmtFinSF1')] %>%
  group_by(BsmtFinType1) %>%
  summarise(medianArea = median(BsmtFinSF1), counts = n()) %>%
  arrange(medianArea) %>%
  ggplot(aes(x=reorder(BsmtFinType1,-medianArea), y=medianArea)) +
  geom_bar(stat = 'identity', fill='cornflowerblue') +
  labs(x='BsmtFinType1', y='Median of BsmtFinSF1') +
  geom_text(aes(label = sort(medianArea)), vjust = -0.5) +
  scale_y_continuous(limits = c(0,850)) +
  theme_minimal()
```

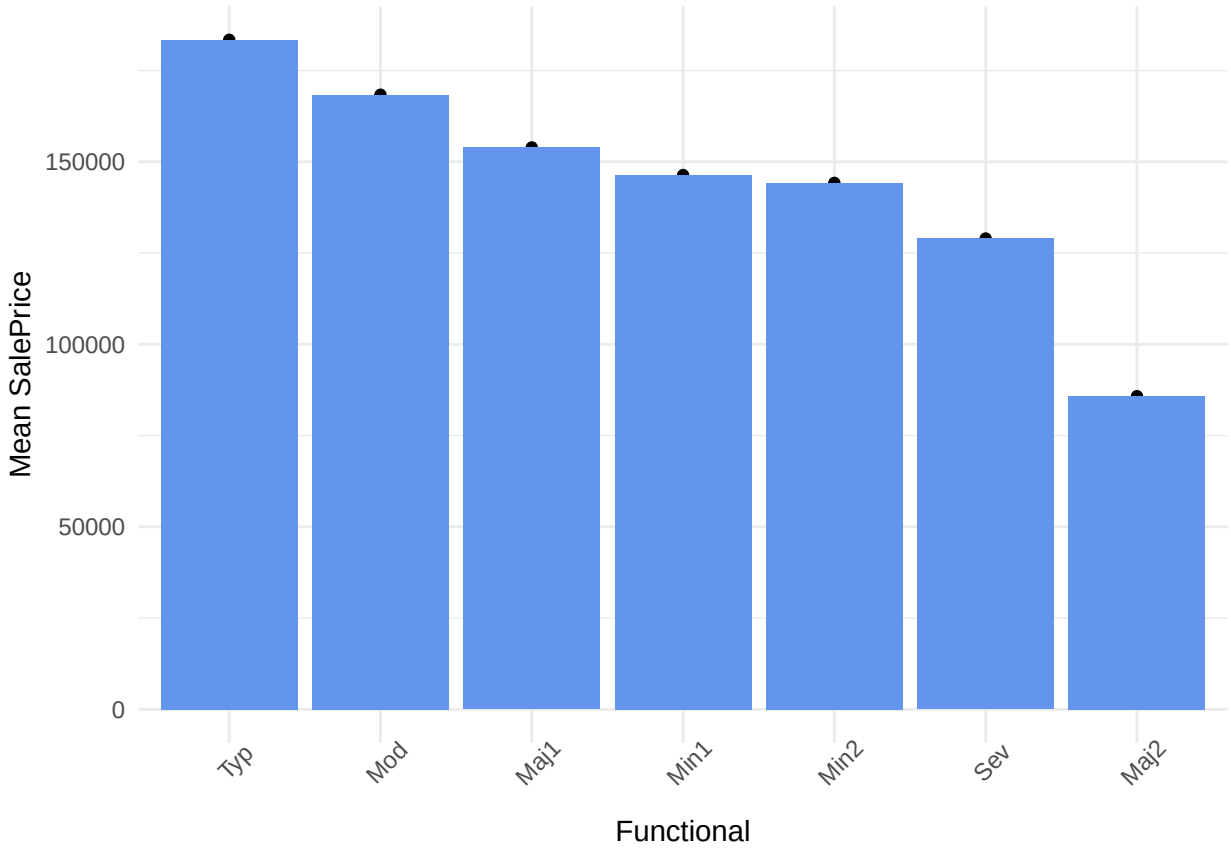


Through investigating the relationships between the basement quality and areas we an see the true order of qualities of each basement to be None < Unf < LwQ < BLQ < Rec < ALQ < GLQ.

```
#BsmtFinType (9+2=11)
bsmt.fin.list = c('None' = 0, 'Unf' = 1, 'LwQ' = 2,'Rec'= 3, 'BLQ' = 4, 'ALQ' = 5, 'GLQ' = 6)
df_numeric = map.fcn(c('BsmtFinType1','BsmtFinType2'), bsmt.fin.list, df_numeric)
```

`Functional`: This feature doesn???t really tell us much and functionality is very vague to tie which other features have a correlation with it. We can compute the mean sale prices for each functional category and assign numeric values accordingly.

```
#Functional (11+1=12)
group.prices('Functional')
```

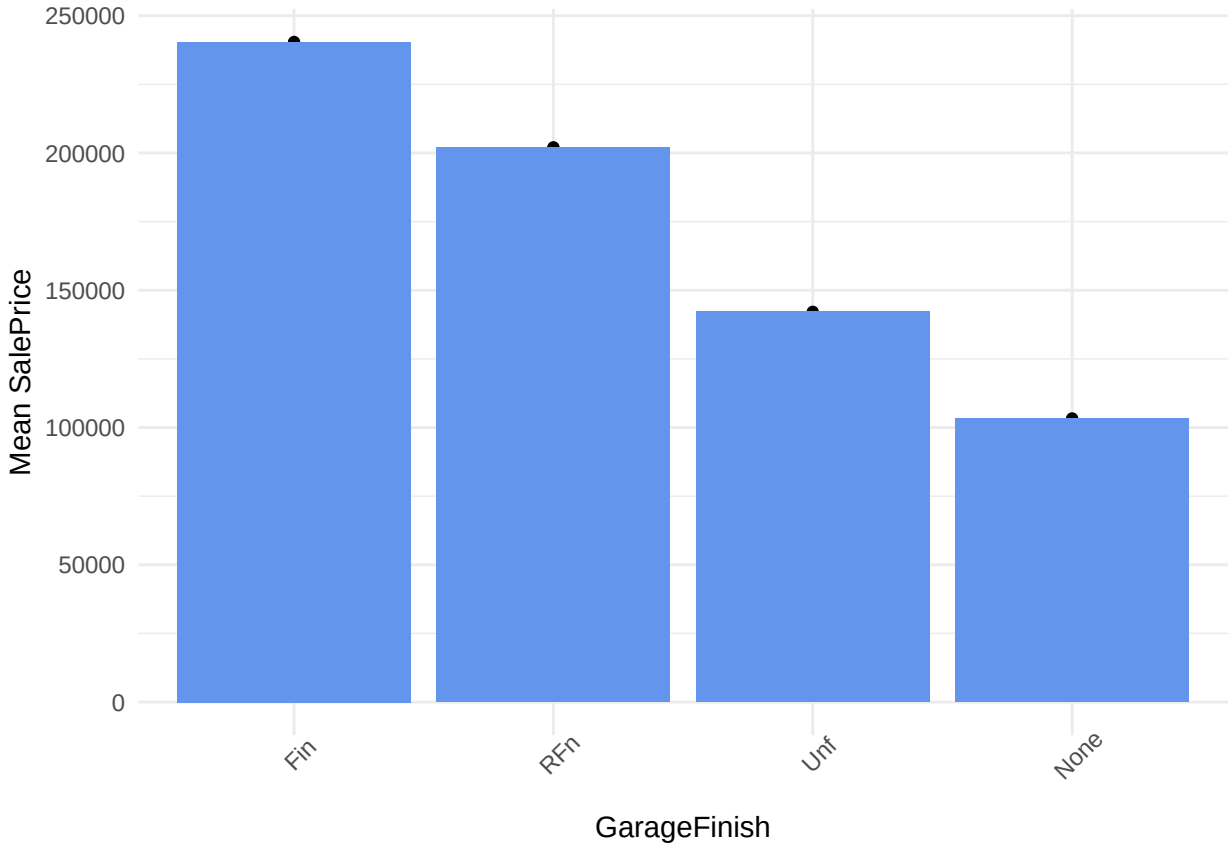



```
##   Functional mean.Quality mean.Price    n
## 1       Min2         4.97   144240.6   34
## 2       Maj2         5.00    85800.0    5
## 3       Min1         5.26   146385.5   31
## 4        Mod         5.40   168393.3   15
## 5       Maj1         5.50   153948.1   14
## 6        Sev         6.00   129000.0    1
## 7        Typ         6.16   183445.4 1358
```

```
functional.list = c('None' = 0, 'Sal' = 1, 'Sev' = 2, 'Maj2' = 3, 'Maj1' = 4, 'Mod' = 5, 'Min2' = 6, 'M

df_numeric['Functional'] = as.numeric(functional.list[df_combine$Functional])
```

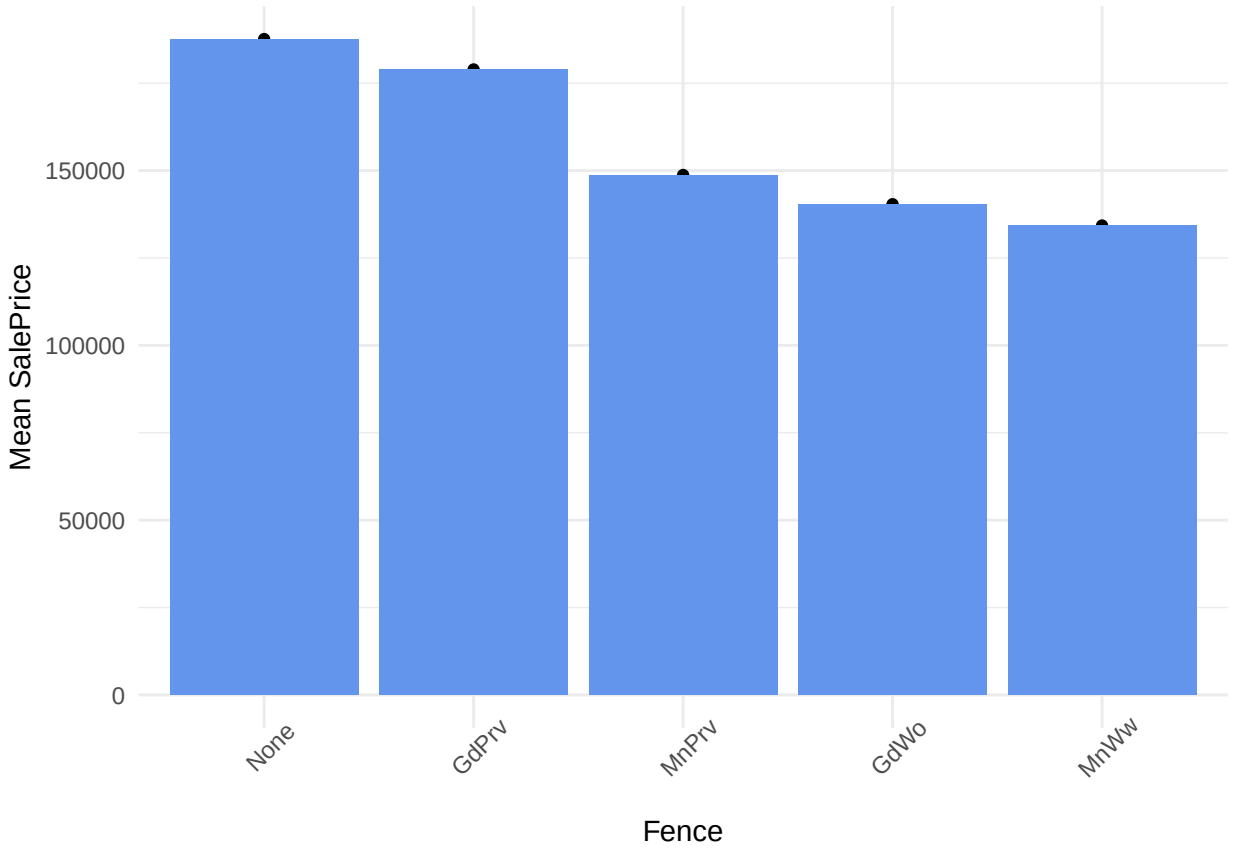
`GarageFinish` and `Fence` We???ll continue to do the same for GarageFinish and Fence:

```
#GarageFinish and Fence (12+2=14)
group.prices('GarageFinish')
```




```
##   GarageFinish mean.Quality mean.Price   n
## 1         None         4.60   103317.3  81
## 2          Unf         5.40   142156.4 605
## 3          RFn         6.57   202068.9 422
## 4          Fin         7.05   240439.4 350
```

```
garage.fin.list = c('None' = 0,'Unf' = 1, 'RFn' = 1, 'Fin' = 2)
df_numeric['GarageFinish'] = as.numeric(garage.fin.list[df_combine$GarageFinish])

group.prices('Fence')
```

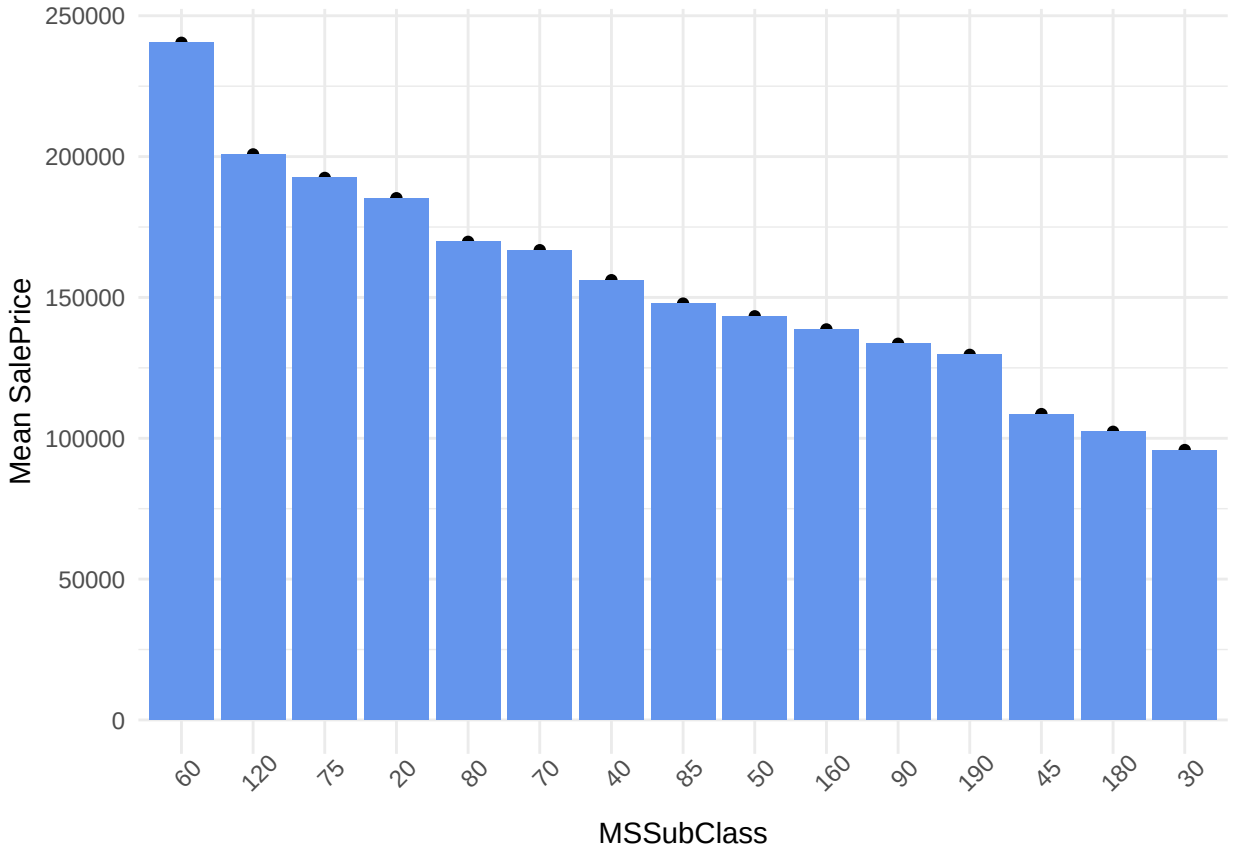



```
##   Fence mean.Quality mean.Price    n
## 1  GdWo         5.17   140379.3   54
## 2  MnWw         5.18   134286.4   11
## 3 MnPrv         5.45   148751.1  157
## 4  None         6.22   187622.7 1177
## 5 GdPrv         6.31   178927.5   59
```

```
fence.list = c('None' = 0, 'MnWw' = 1, 'GdWo' = 1, 'MnPrv' = 2, 'GdPrv' = 4)
df_numeric['Fence'] = as.numeric(fence.list[df_combine$Fence])
```

MSSubClass:

```
#MSSubClass (14+1=15)
group.prices('MSSubClass')
```




```
##    MSSubClass mean.Quality mean.Price   n
## 1          30         4.51   95829.72  69
## 2         180         4.60  102300.00  10
## 3          90         4.96  133541.08  52
## 4         190         4.97  129613.33  30
## 5          50         5.40  143302.97 144
## 6          85         5.45  147810.00  20
## 7          45         5.50  108591.67  12
## 8          40         5.75  156125.00   4
## 9          80         5.97  169736.55  58
## 10         20         6.03  185224.81 536
## 11         70         6.12  166772.42  60
## 12        160         6.14  138647.38  63
## 13         75         6.62  192437.50  16
## 14         60         7.04  240403.54 297
## 15        120         7.06  200779.08  87
```

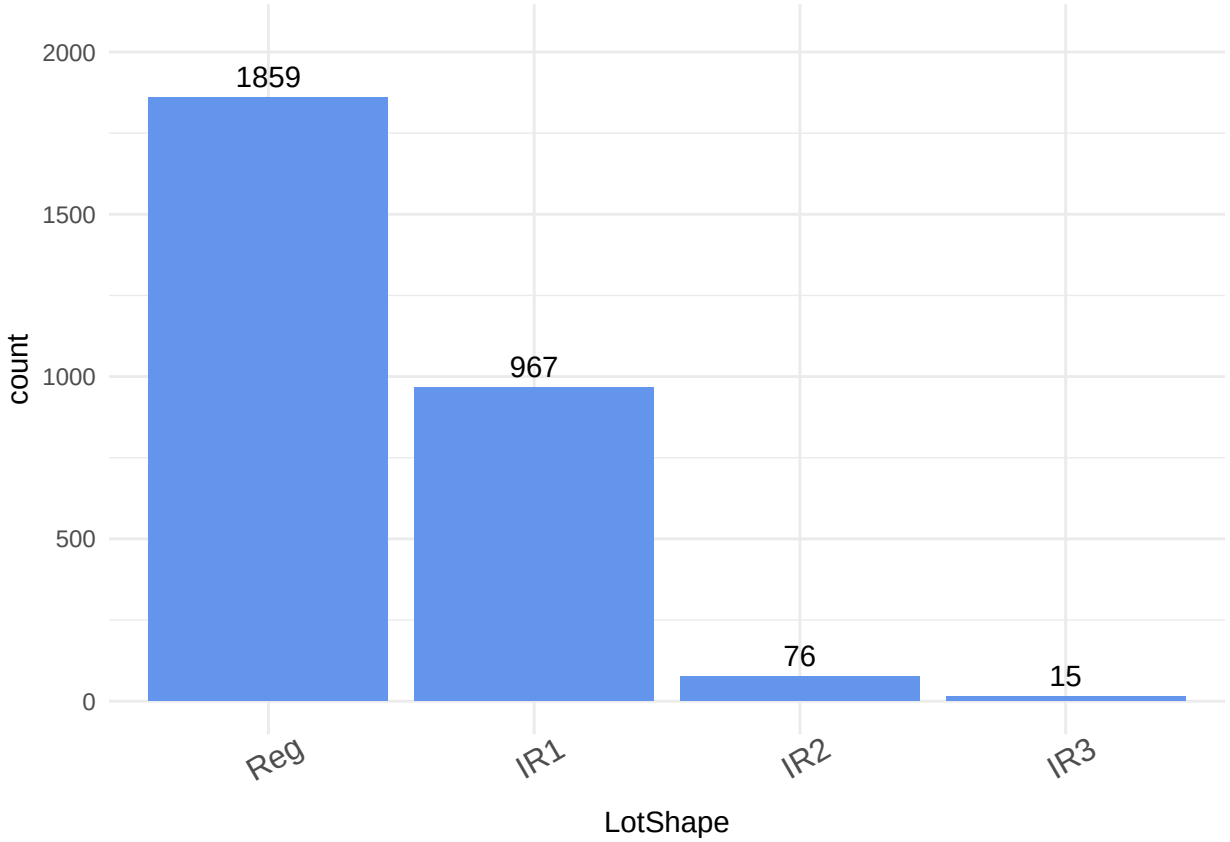
```
MSdwelling.list = c('20' = 1, '30'= 0, '40' = 0, '45' = 0,'50' = 0, '60' = 1, '70' = 0, '75' = 0, '80' =

df_numeric['NewerDwelling'] = as.numeric(MSdwelling.list[as.character(df_combine$MSSubClass)])
```

Nominal Categorical Variable Transformation

Now for some of the nominal variables we can take one of the categories that is distinct from the others and create a binary feature that returns 1 if the house has that specific value and 0 if it does not.

`LotShape`:

```r
#Helper function for plotting categoric data for easier data visualization.
plot.categoric = function(cols, df){
  for (col in cols) {
    order.cols = names(sort(table(df_combine[,col]), decreasing = TRUE))
    num.plot = qplot(df[,col]) +
      geom_bar(fill = 'cornflowerblue') +
      geom_text(aes(label = ..count..), stat='count', vjust=-0.5) +
      theme_minimal() +
      scale_y_continuous(limits = c(0,max(table(df[,col]))*1.1)) +
      scale_x_discrete(limits = order.cols) +
      xlab(col) +
      theme(axis.text.x = element_text(angle = 30, size=12))

    print(num.plot)
  }
}

plot.categoric('LotShape', df_combine)
```



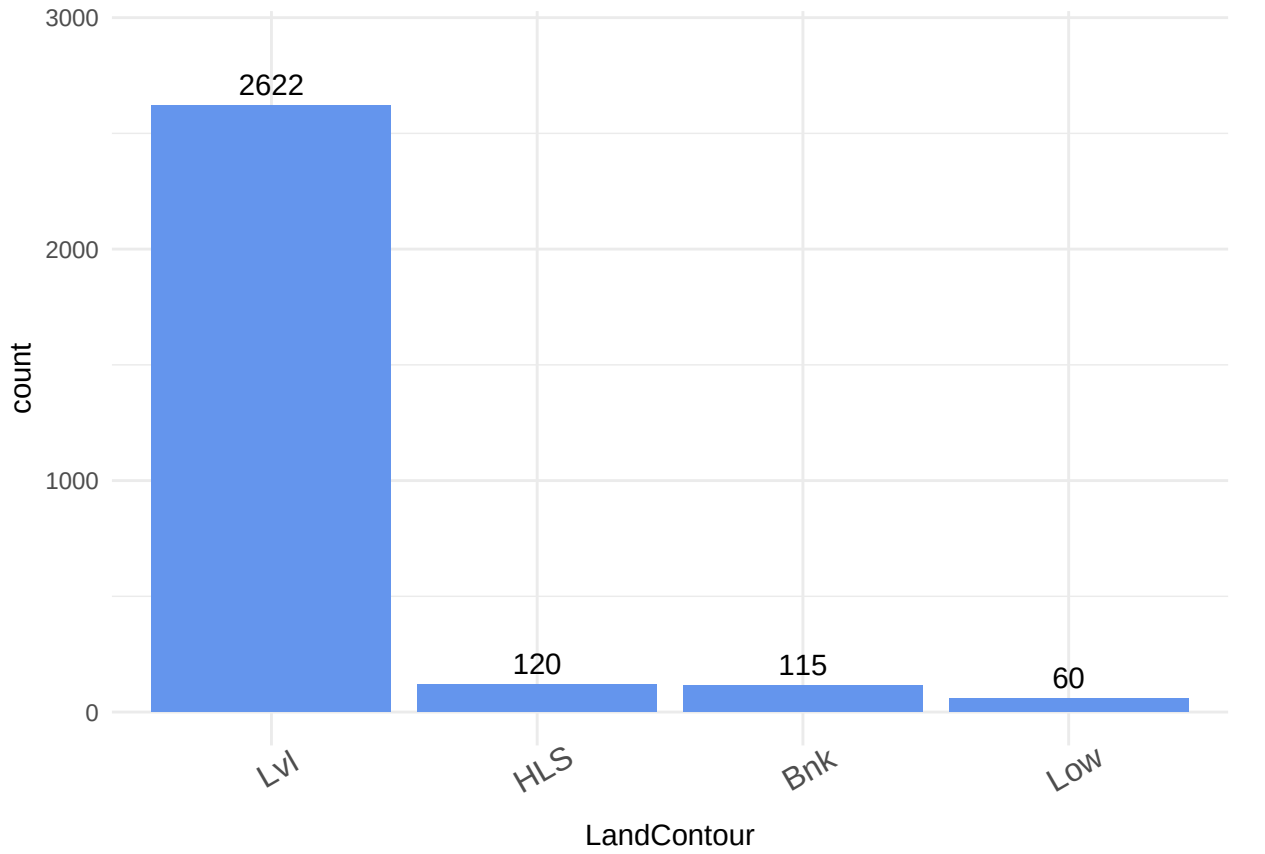
We could see that LotShape has 3 values for having an irregular shape and only 1 for regular. We can create a binary column that returns 1 for houses with a regular lot shape and 0 for houses with any of the 3 irregular lot shapes. Using this method of turning a categoric feature into a binary column will ultimately help our data train better through boosted models without using numeric placeholders on nominal data.

```r
#LotShape (15+1=16)
df_numeric['RegularLotShape'] = (df_combine$LotShape == 'Reg') * 1
```

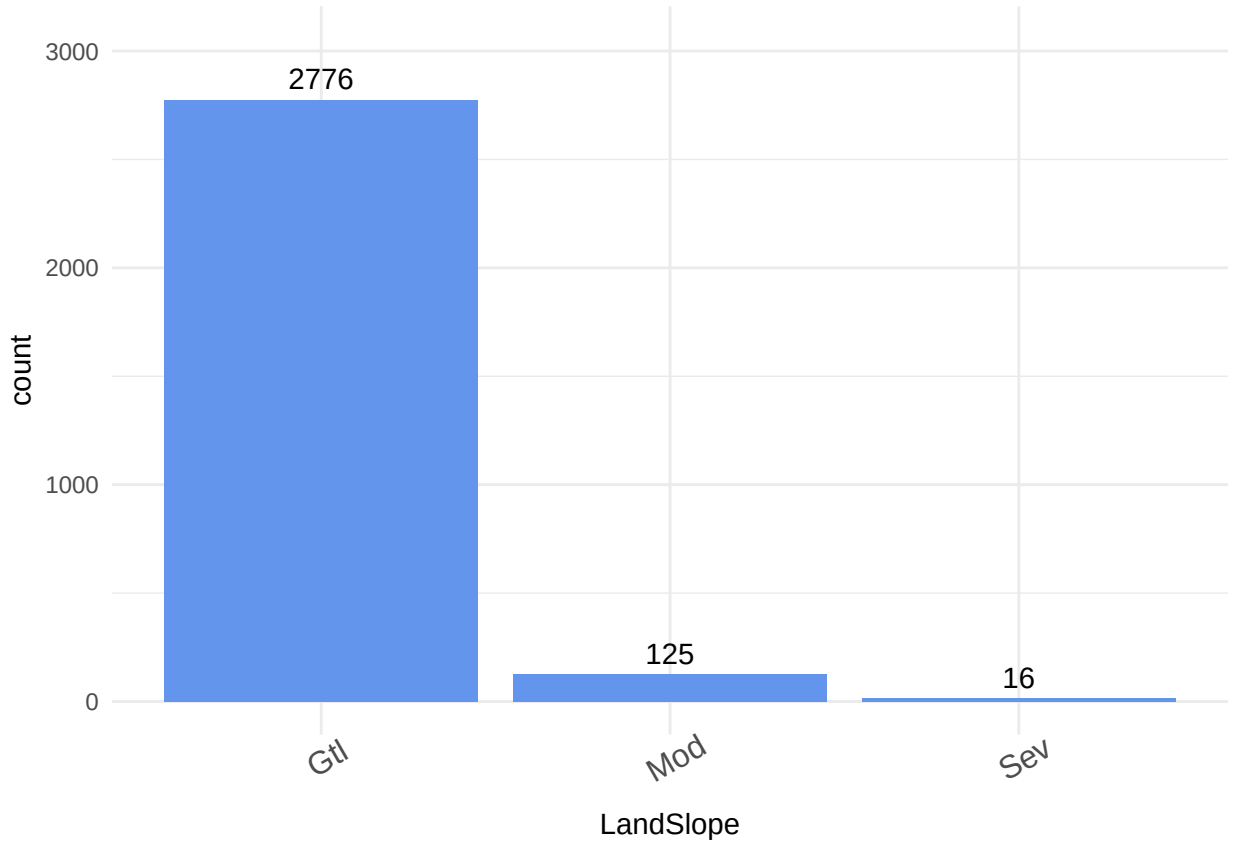
We will use this exact same method for: `LandContour`, `LandSlope`, `Electrical`, `GarageType`, `PavedDrive`,

MiscFeature

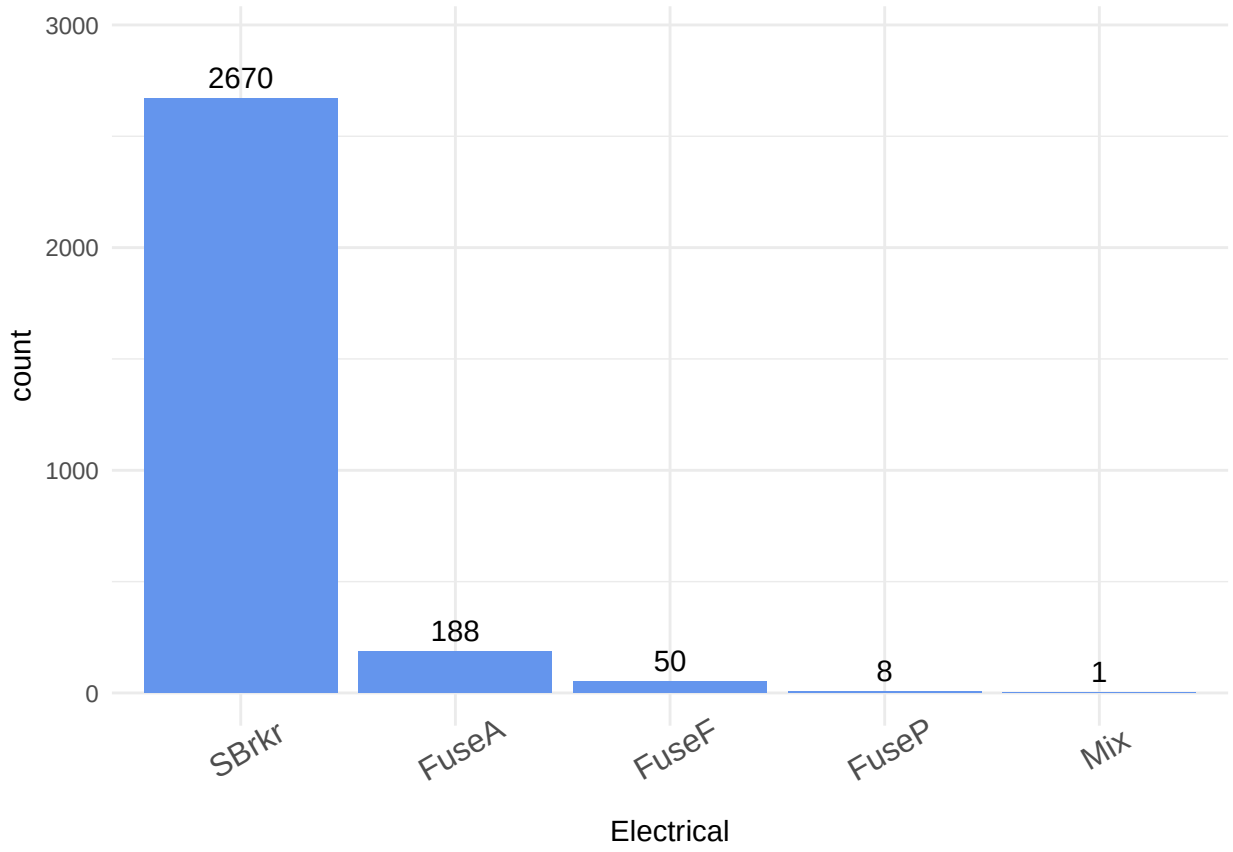
```
#LandContour, LandSlope, Electrical, GarageType, PavedDrive, MiscFeature (16+6=22)
plot.categoric('LandContour', df_combine)
```



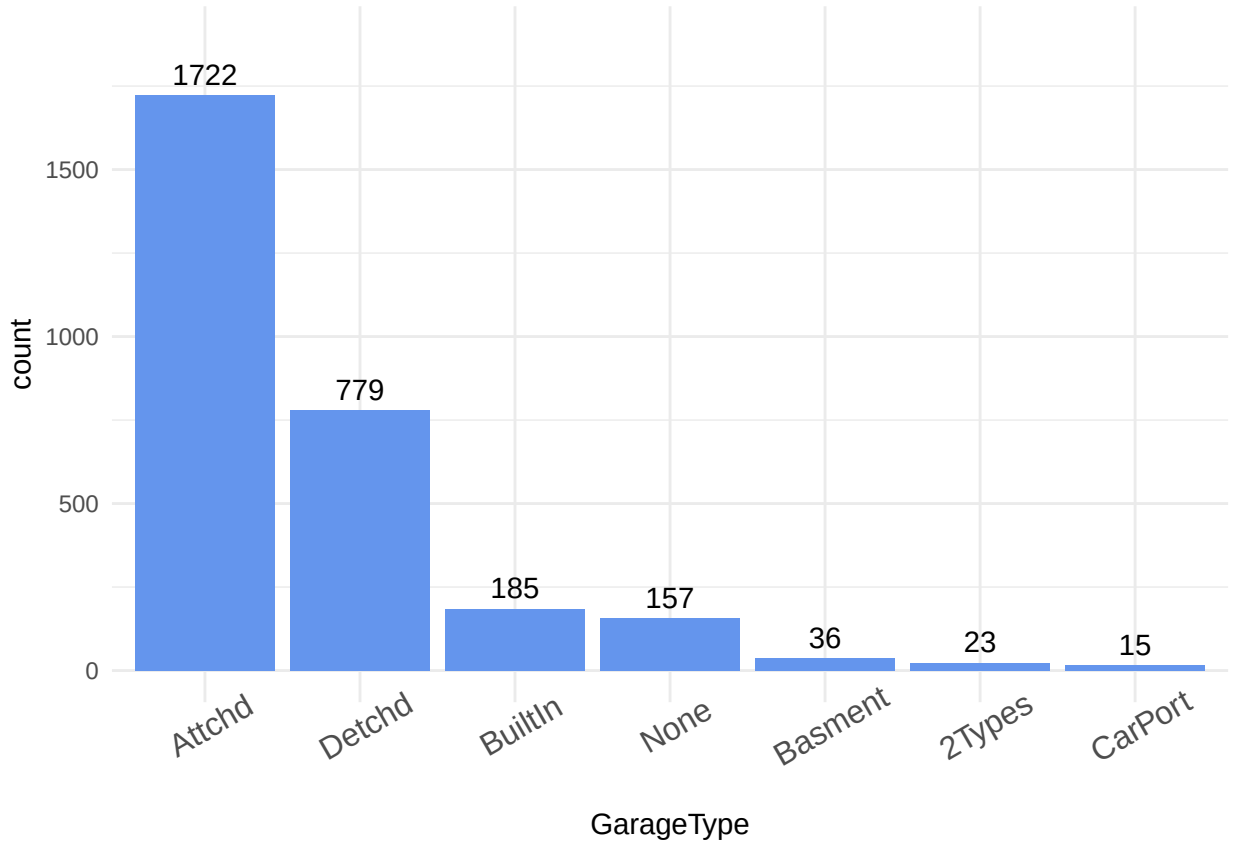

```
df_numeric['LandLeveled'] = (df_combine$LandContour == 'Lvl') * 1

plot.categoric('LandSlope', df_combine)
```



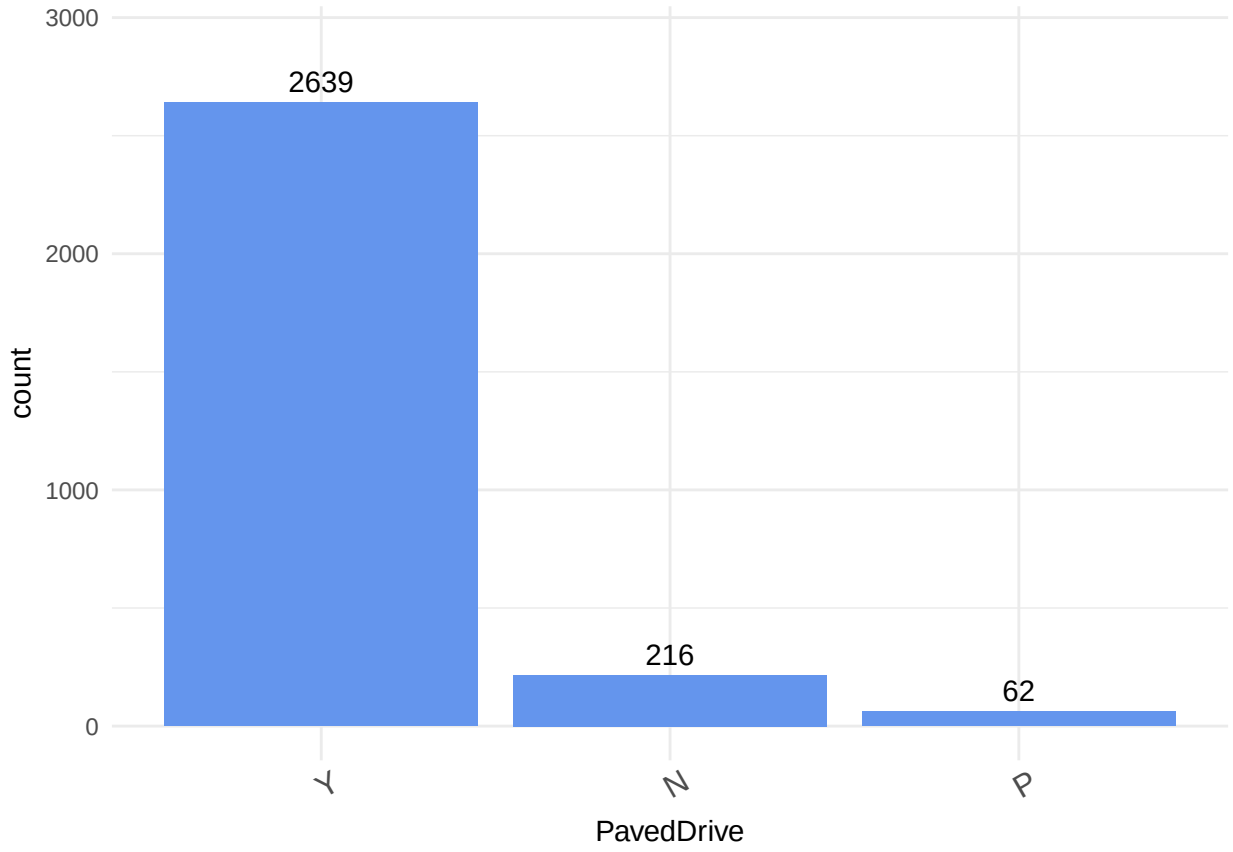

```
df_numeric['LandSlopeGentle'] = (df_combine$LandSlope == 'Gtl') * 1

plot.categoric('Electrical', df_combine)
```



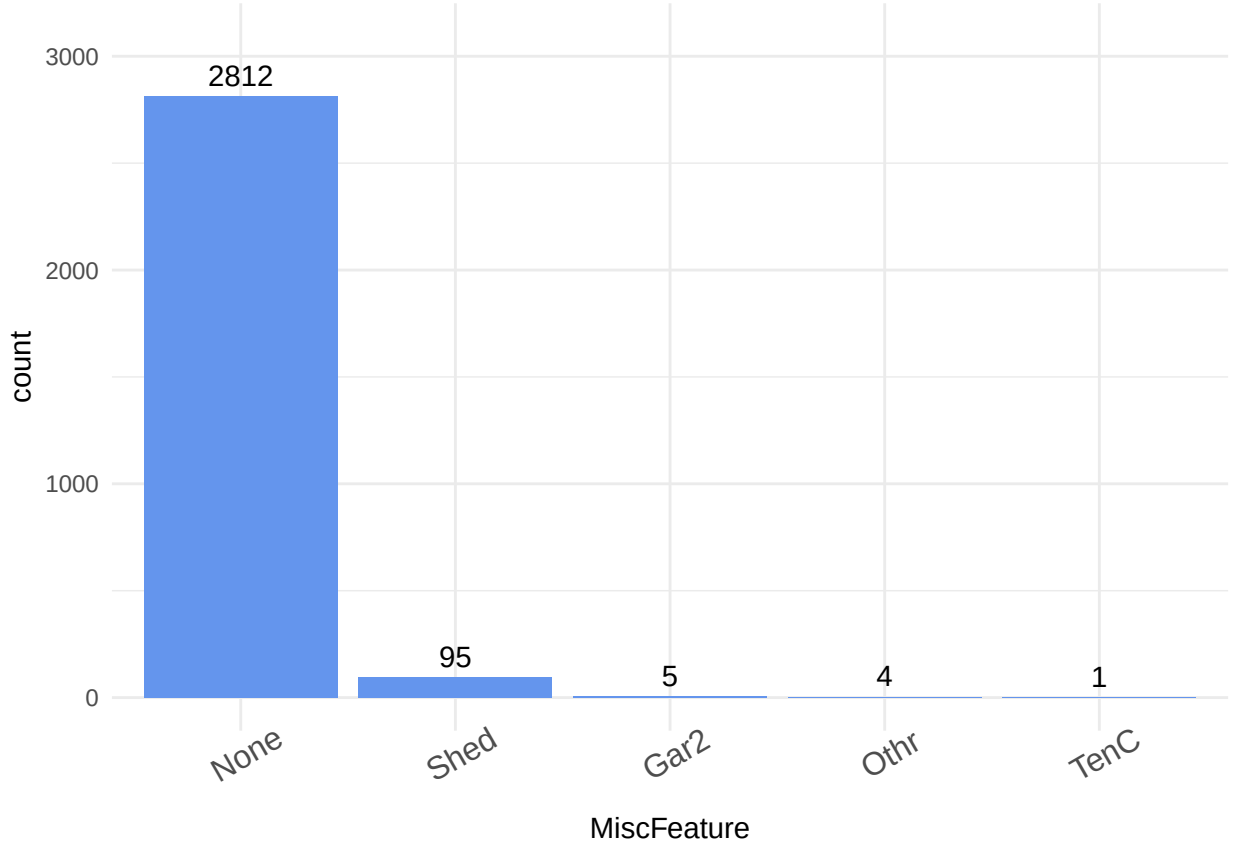

```
df_numeric['ElectricalSB'] = (df_combine$Electrical == 'SBrkr') *1

plot.categoric('GarageType', df_combine)
```




```
df_numeric['GarageDetchd'] = (df_combine$GarageType == 'Detchd') * 1

plot.categoric('PavedDrive', df_combine)
```




```
df_numeric['HasPavedDrive'] = (df_combine$PavedDrive == 'Y') * 1

plot.categoric('MiscFeature', df_combine)
```




```
df_numeric['HasShed'] = (df_combine$MiscFeature == 'Shed') * 1
```

`YearBuilt` and `YearRemodAdd`: Many of the houses recorded the same year for `YearBuilt` and `YearRemodAdd`. We can create a new column that records that a house was remodelled if the year it was built is different than the remodel year.

```
#YearBuilt and YearRemodAdd (22+2=24)
df_numeric['Remodeled'] = (df_combine$YearBuilt != df_combine$YearRemodAdd) * 1
```

We can also create a column that seperates which houses have been recently remodelled vs those who are not. Houses that have been remodelled after the year they were sold will fall into this category.

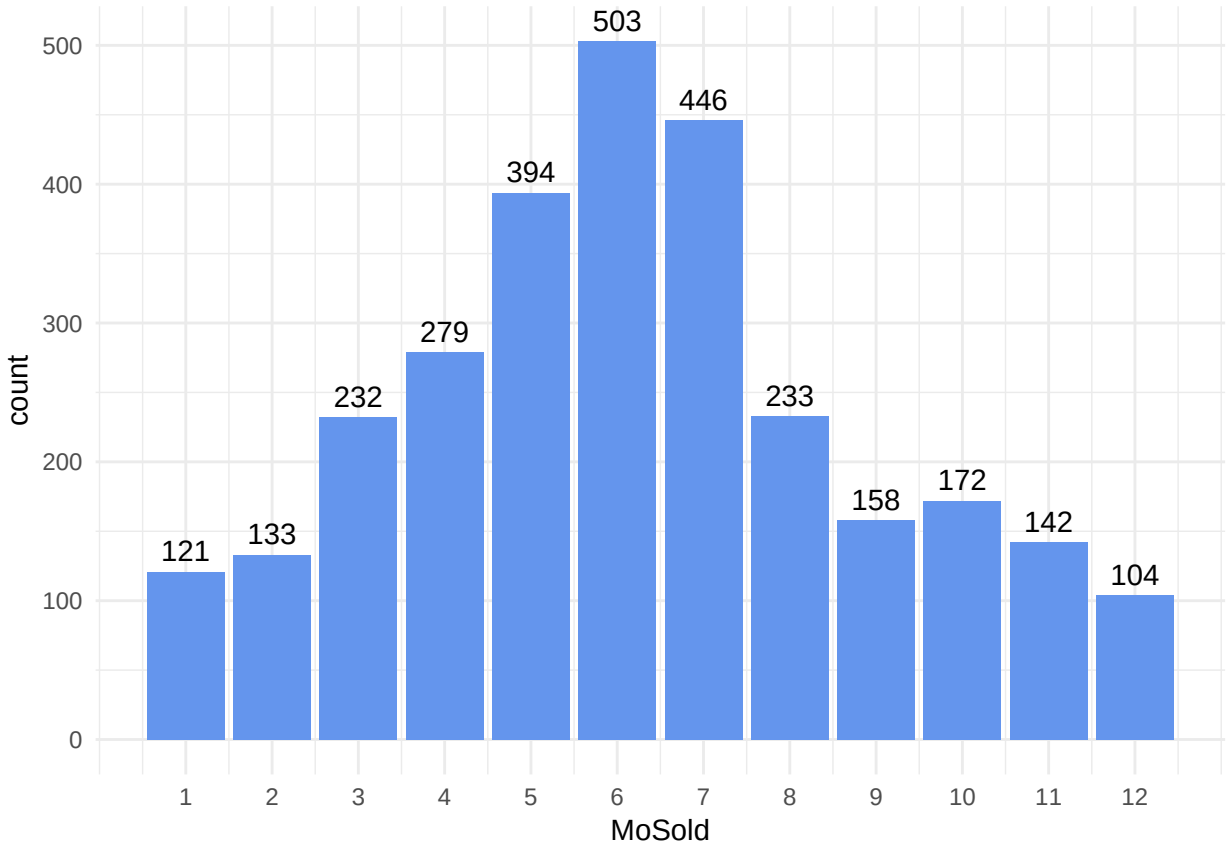
```
df_numeric['RecentRemodel'] = (df_combine$YearRemodAdd >= df_combine$YrSold) * 1
```

There can be potential value to homes who were sold the same year they were built as this could be an indicator that these houses were hot in the market.

```
df_numeric['NewHouse'] = (df_combine$YearBuilt == df_combine$YrSold) * 1
```

We know how important the year a house was built and sold but what about the specific month it was sold? How do houses sold during summer compare to the other seasons?

```
ggplot(df_combine, aes(x=MoSold)) + geom_bar(fill = "cornflowerblue") + geom_text(aes(label = ..count..
```

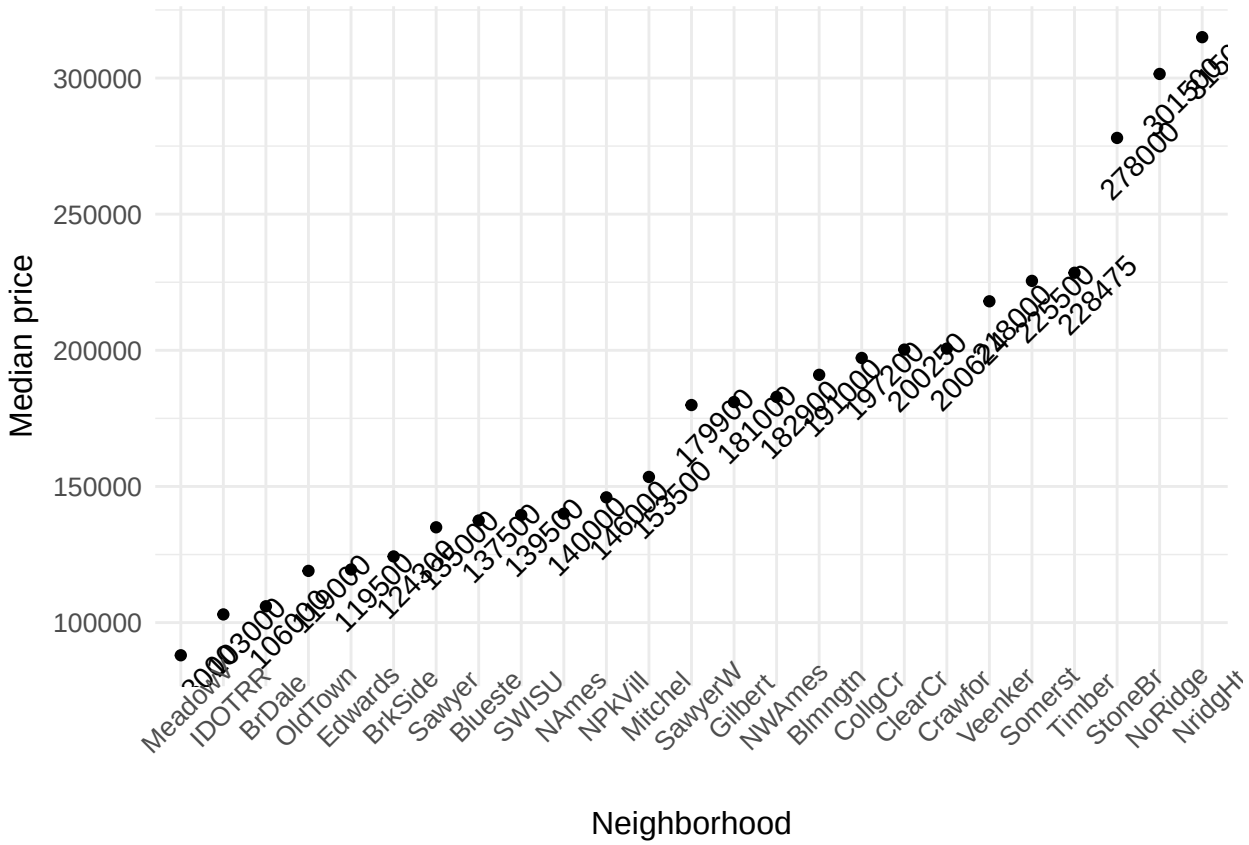


The largest proportion of houses sold is during the summer months: May, June, july. Lets add a column that seperates the summer houses from the rest.

```
df_numeric['HighSeason'] = (df_combine$MoSold %in% c(5,6,7)) * 1
```

`Neighborhoods`: What about which neighborhoods are more expensive than others?

```
#Neighborhood (24+1=25)
df[,c('Neighborhood', 'SalePrice')] %>% group_by(Neighborhood) %>% summarise(median.price = median(Sale
  geom_point() +
  geom_text(aes(label = median.price, angle = 45), vjust = 2) + theme_minimal() + labs(x='Neighborhood'
```

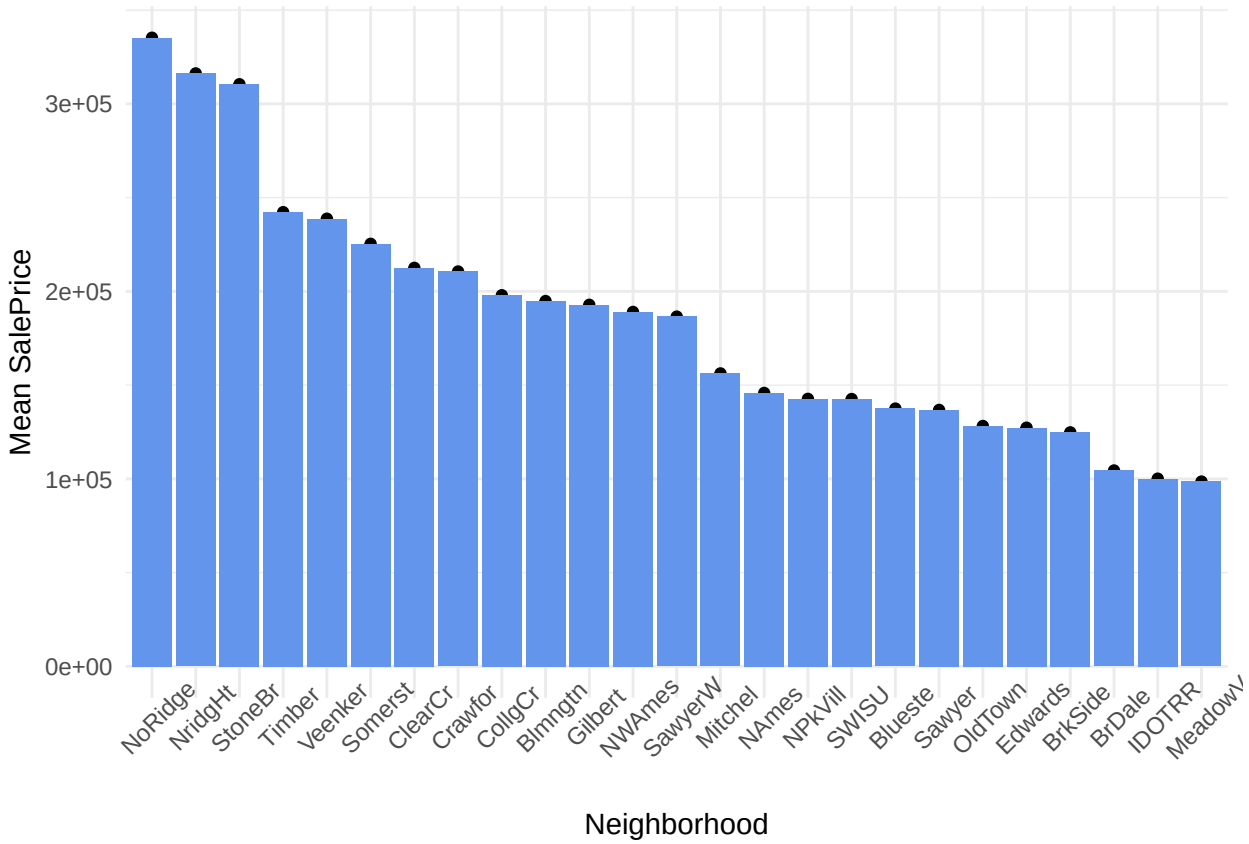


Lets one-hot encode the more expensive neighborhoods and add that to our dataframe

```
nbrh.rich = c('Crawfor', 'Somerst', 'Timber', 'StoneBr', 'NoRidge', 'NridgeHt')
df_numeric['NbrhRich'] = (df_combine$Neighborhood %in% nbrh.rich) * 1
```

How about a numeric mapping to the neighborhoods who have higher quality houses and run for larger sale prices?

```
group.prices('Neighborhood')
```

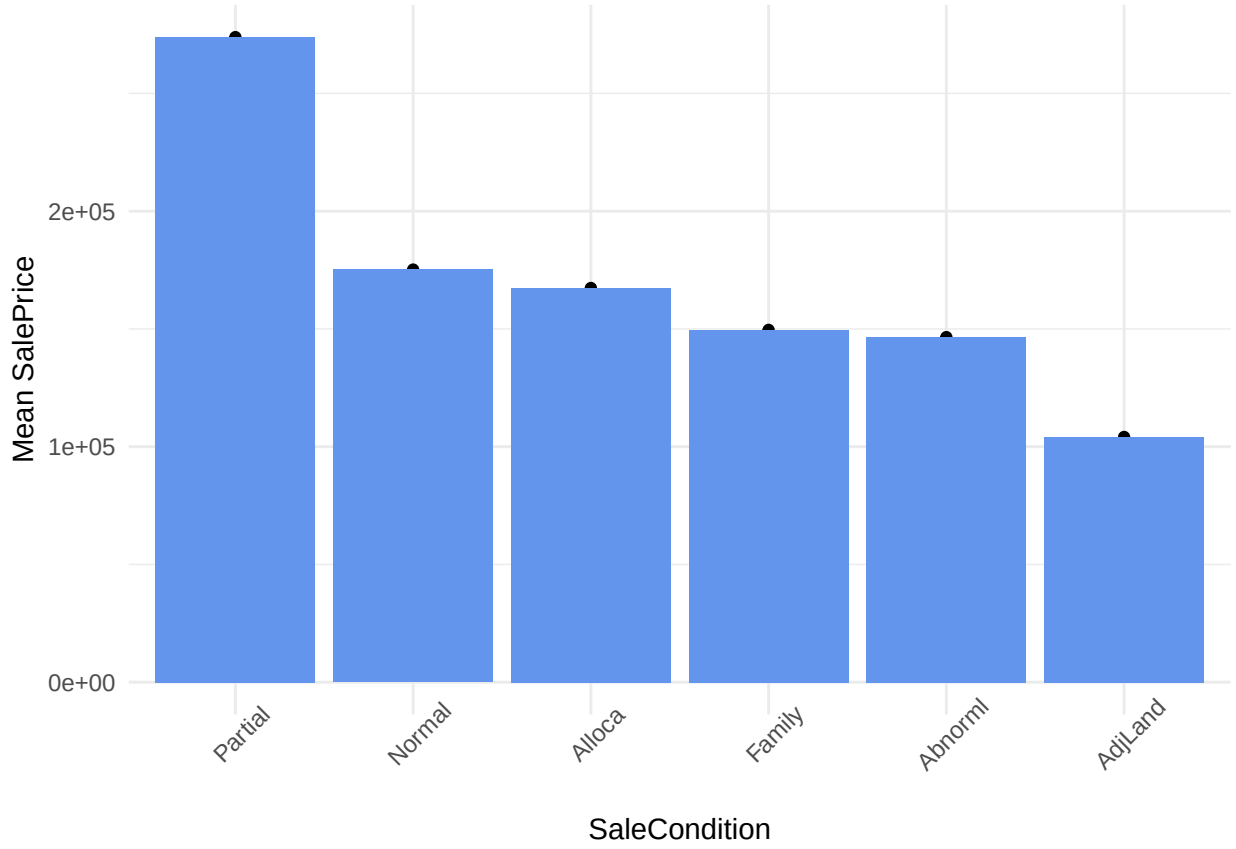



```
##    Neighborhood mean.Quality mean.Price   n
## 1       MeadowV         4.47   98576.47  17
## 2        IDOTRR         4.76  100123.78  37
## 3       Edwards         4.98  127318.57  98
## 4        Sawyer         5.03  136793.14  74
## 5       BrkSide         5.05  124834.05  58
## 6         NAmes         5.36  145847.08 225
## 7       OldTown         5.39  128225.30 113
## 8         SWISU         5.44  142591.36  25
## 9       Mitchel         5.59  156270.12  49
## 10       BrDale         5.69  104493.75  16
## 11      ClearCr         5.89  212565.43  28
## 12      Blueste         6.00  137500.00   2
## 13      NPkVill         6.00  142694.44   9
## 14      Crawfor         6.27  210624.73  51
## 15      SawyerW         6.32  186555.80  59
## 16       NWAmes         6.33  189050.07  73
## 17      Gilbert         6.56  192854.51  79
## 18      CollgCr         6.64  197965.77 150
## 19      Veenker         6.73  238772.73  11
## 20       Timber         7.16  242247.45  38
## 21      Blmngtn         7.18  194870.88  17
## 22      Somerst         7.34  225379.84  86
## 23      NoRidge         7.93  335295.32  41
## 24      StoneBr         8.16  310499.00  25
## 25      NridgHt         8.26  316270.62  77
```

```
nbrh.map = c('MeadowV' = 0, 'IDOTRR' = 1, 'Sawyer' = 1, 'BrDale' = 1, 'OldTown' = 1, 'Edwards' = 1,
             'BrkSide' = 1, 'Blueste' = 1, 'SWISU' = 2, 'NAmes' = 2, 'NPkVill' = 2, 'Mitchel' = 2,
             'SawyerW' = 2, 'Gilbert' = 2, 'NWAmes' = 2, 'Blmngtn' = 2, 'CollgCr' = 2, 'ClearCr' = 3,
             'Crawfor' = 3, 'Veenker' = 3, 'Somerst' = 3, 'Timber' = 3, 'StoneBr' = 4, 'NoRidge' = 4,
             'NridgHt' = 4)

df_numeric['NeighborhoodBin'] = as.numeric(nbrh.map[df_combine$Neighborhood])
```

SaleCondition:

```
#SaleCondition (25+1=26)
group.prices('SaleCondition')
```

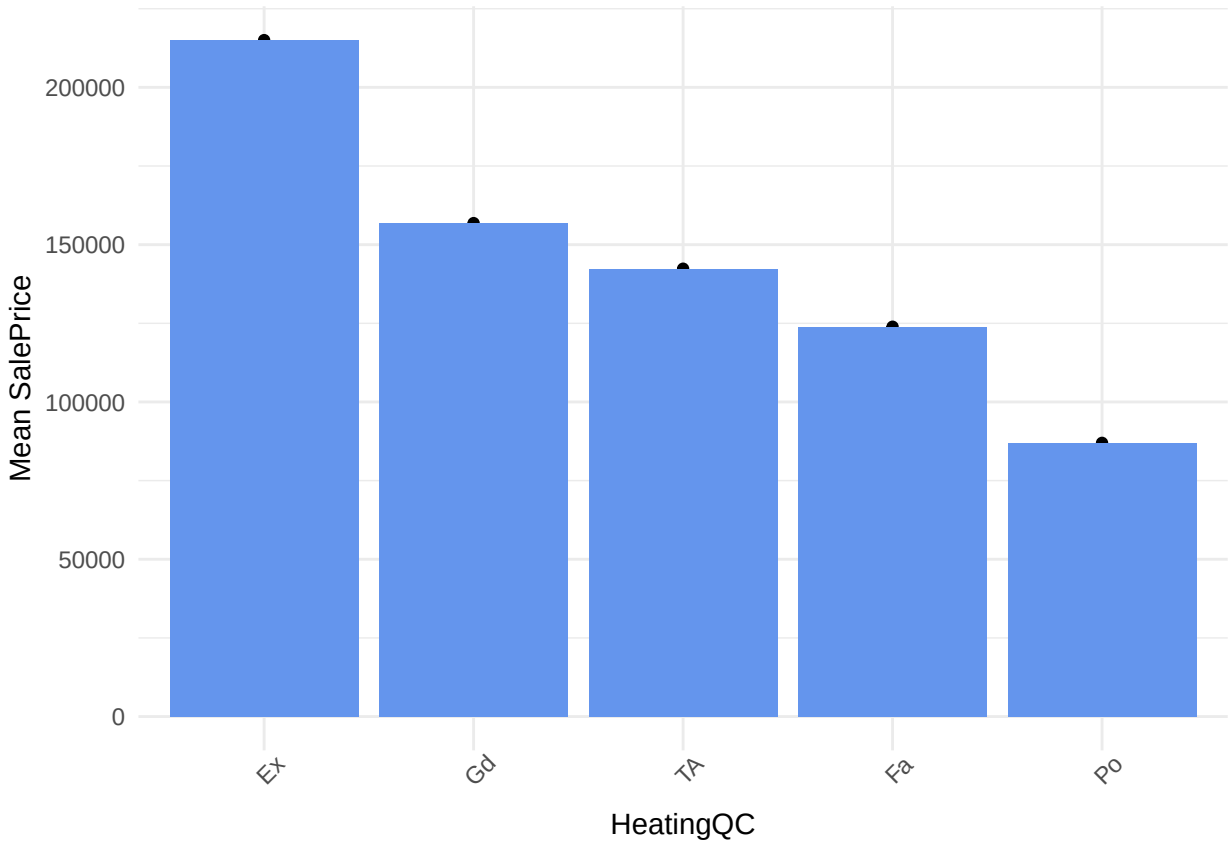



```
##   SaleCondition mean.Quality mean.Price    n
## 1       AdjLand         5.00   104125.0    4
## 2        Alloca         5.42   167377.4   12
## 3       Abnorml         5.57   146526.6  101
## 4        Family         5.80   149600.0   20
## 5        Normal         6.01   175202.2 1198
## 6       Partial         7.52   273916.4  123
```

```
df_numeric['PartialPlan'] = (df_combine$SaleCondition == 'Partial') * 1
```

HeatingQC

```
#HeatingQC (26+1=27)
group.prices('HeatingQC')
```

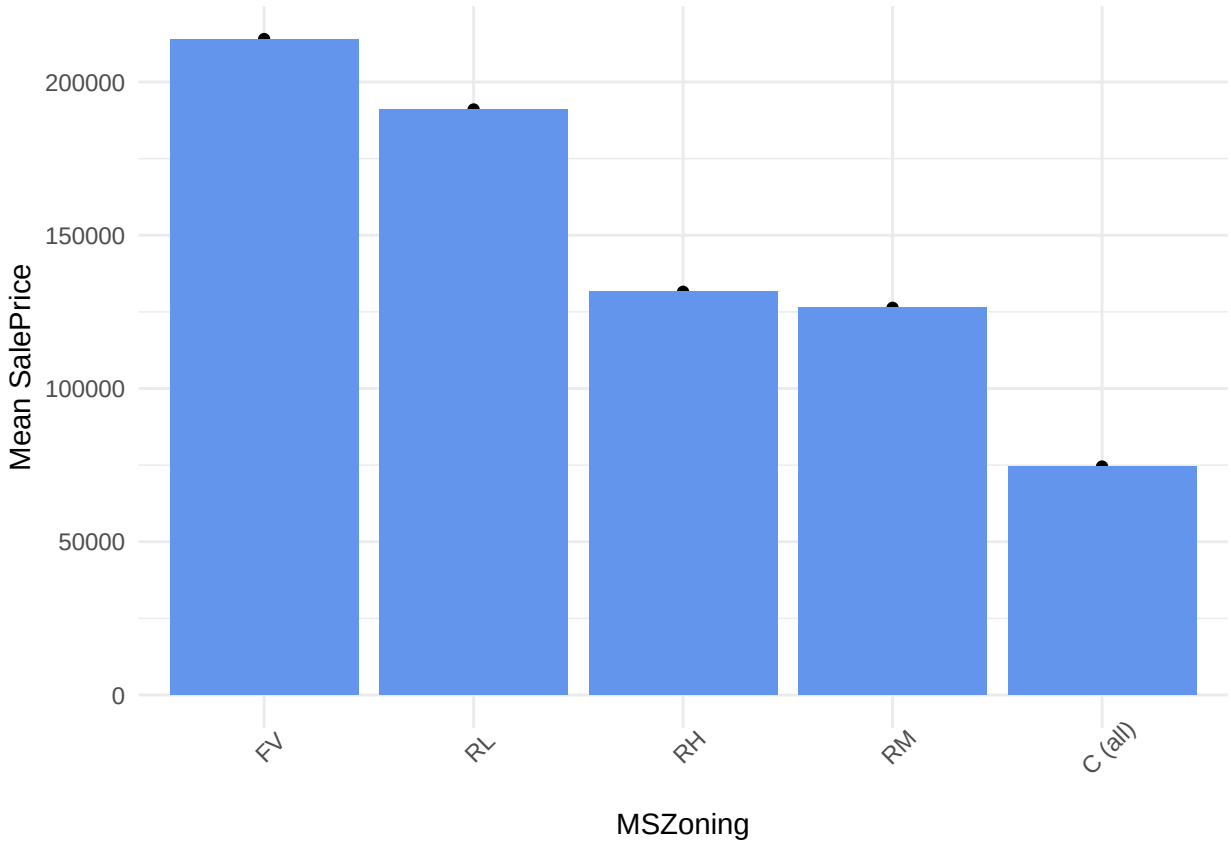



```
##   HeatingQC mean.Quality mean.Price   n
## 1        Fa         5.00   123919.5  49
## 2        Po         5.00    87000.0   1
## 3        TA         5.37   142362.9 428
## 4        Gd         5.70   156858.9 241
## 5        Ex         6.71   215029.6 739
```

```
heating.list = c('Po' = 0, 'Fa' = 1, 'TA' = 2, 'Gd' = 3, 'Ex' = 4)
df_numeric['HeatingScale'] = as.numeric(heating.list[df_combine$HeatingQC])
```

MSZoning

```
group.prices('MSZoning')
```




```
##   MSZoning mean.Quality mean.Price    n
## 1  C (all)         3.90    74528.0   10
## 2       RH         5.25   131558.4   16
## 3       RM         5.45   126316.8  218
## 4       RL         6.18   191037.4 1149
## 5       FV         7.20   214014.1   65
```

Other Categorical Variable Transformation

For the rest of the categoric features we can one-hot encode each value to get as many splits in the data as possible since sparse data performs better for trees and xgboost.

```
#Use dymmyVars
dummy = dummyVars(" ~ .",data=df_combine[,cat_features])
df_categoric = data.frame(predict(dummy,newdata=df_combine[,cat_features]))
```

Numeric Variable Transformation

New Numeric Variable Transformation

We know the area of the house is one of the most important factor that effects its sale price. So first I would like to calculate the `TotalArea` and `AreaInside` based the variable we have.

```
area.cols = c('LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
              'TotalBsmtSF', 'X1stFlrSF', 'X2ndFlrSF', 'GrLivArea', 'GarageArea', 'WoodDeckSF',
              'OpenPorchSF', 'EnclosedPorch', 'X3SsnPorch', 'ScreenPorch', 'LowQualFinSF', 'PoolArea')
```

```
df_numeric['TotalArea'] = as.numeric(rowSums(df_combine[, area.cols]))

df_numeric['AreaInside'] = as.numeric(df_combine$X1stFlrSF + df_combine$X2ndFlrSF)
```

We've seen how strong of an effect the year of a house built has on the house price, therefore, as this dataset collects houses up until 2010 we can determine how old a house is and how long ago the house was sold:

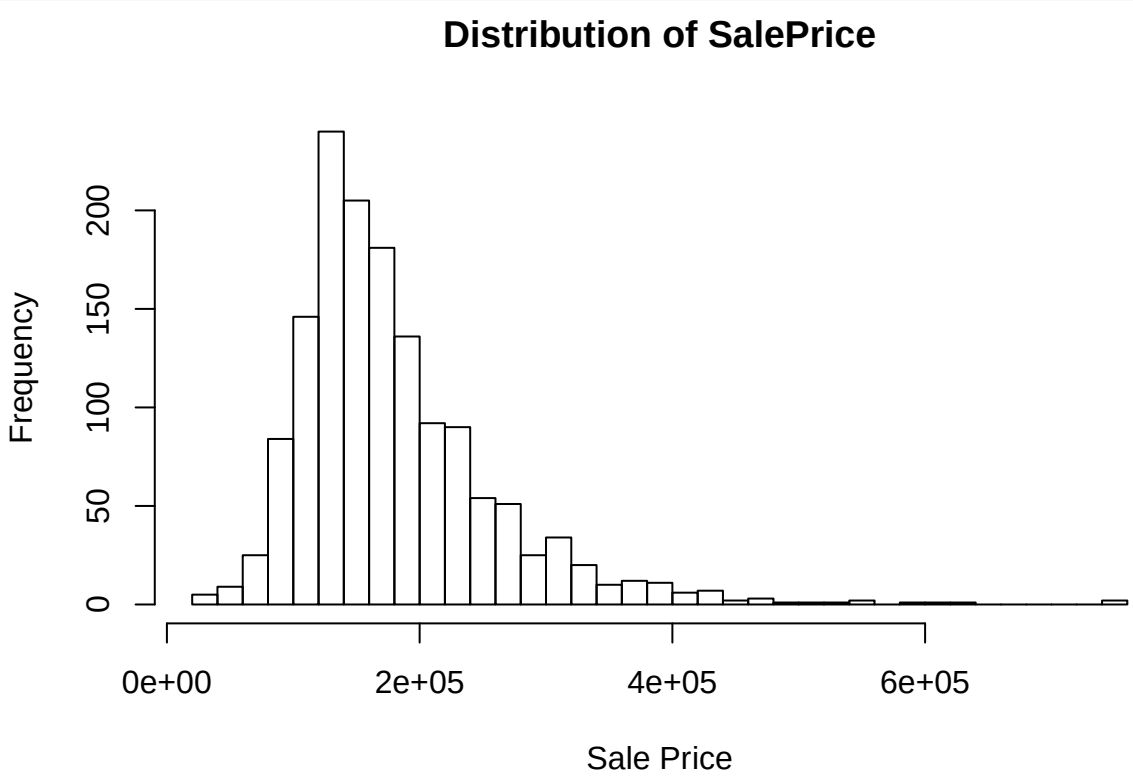
```
df_numeric['Age'] = as.numeric(2010 - df_combine$YearBuilt)

# how many years since the house was remodelled and sold
df_numeric['TimeSinceSold'] = as.numeric(2010 - df_combine$YrSold)
```

Variable Normalization

Linear models assume normality from dependant variables. For skewnesses outside the range of -0.5 to 0.5 and Kurtosises outside of -3.0 to 3.0 do not satisfy the assumption of normality. So we can make a non-linear transforamtion like a log-transformation such that $f(x) = log(x + 1)$ when a column has 0 and $f(x) = log(x)$ otherwise.

I will first focus on the dependent variable ('SalePrice') and try to know a little bit more about it.

```
hist(df$SalePrice, breaks = 50, main = "Distribution of SalePrice", xlab = "Sale Price")
```




```
skewness(df$SalePrice)
```

```
## [1] 1.877427
```

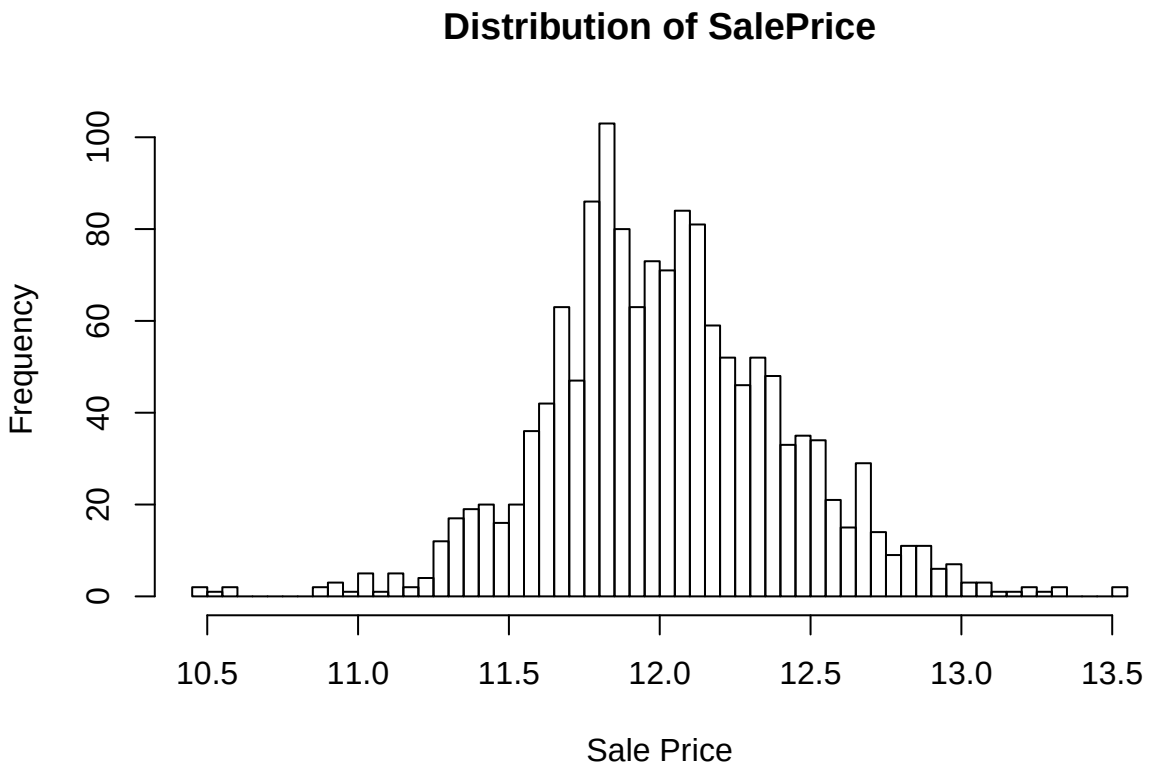
```
kurtosis(df$SalePrice)
```

```
## [1] 6.483584
```

SalePrice is our target variable. The SalePrice feature has a positive skewness (right skewed). Since linear

model usually perform better on normally distributed data, I need to transform this distribution to a normal one.

```
hist(log(df$SalePrice), breaks = 50, main = "Distribution of SalePrice", xlab = "Sale Price")
```




```
skewness(log(df$SalePrice))
```

```
## [1] 0.1213182
```

```
kurtosis(log(df$SalePrice))
```

```
## [1] 0.7926861
```

```
df$SalePrice = log(df$SalePrice)
```

So, after taking the logarithm, `SalePrice` is almost normal distribution.

```
skewed = apply(df_numeric, 2, skewness)
skewed = skewed[(skewed>0.8) | (skewed< -0.8)]

kurtosis = apply(df_numeric, 2, kurtosis)
kurtosis = kurtosis[(kurtosis > 3.0) | (kurtosis < -3.0)]


for(col in names(skewed)){
  if(0 %in% df_numeric[, col]){
    df_numeric[,col] = log(1+df_numeric[,col])
  }else{
    df_numeric[,col] = log(df_numeric[,col])
  }
}
```

```
scaler = preProcess(df_numeric)
df_numeric = predict(scaler, df_numeric)

#Final DataFrame
df_final = cbind(df_numeric, df_categoric)
```

Feature Selection

Some of these features may have become zero-variance predictors, such that a few samples may have an insignificant influence on the model. These near-zero-variance may cause overfitting or will prevent our model from generalizing over the data at a more sufficient rate. The package caret offers a function nearZeroVar, which checks the frequency of the most common value over the second most frequent value, which would be closer to 1 for well-behaved predictors and very large for highly-unbalanced features. It also checks the number of unique values divided by the n number of samples which will approach zero as the level of detail in the feature increases. We can remove all of the near-zero-variance variables from out dataframe.

```
nzv.data = nearZeroVar(df_final, saveMetrics = T)
drop.cols = rownames(nzv.data)[nzv.data$nzv == TRUE]
df_final = df_final[,!names(df_final) %in% drop.cols]

paste('The dataframe now has', dim(df_final)[1], 'rows and', dim(df_final)[2], 'columns')
```

```
## [1] "The dataframe now has 2917 rows and 151 columns"
```

Model Training

XGBoost

XGboost fits shallow regression trees to our data and then additional trees to the residuals, we will repeat this process for 30000 rounds so that our model has learned from the data as much as possible without overfitting. XGboost is a Gradient Boosted Method(GBM), which is an ensemble learning method that uses a very large number of decision trees, which are typically weak learners and combines them into one final prediction. For gradient boosted trees, the new trees added to the model are the weak learnings.

Dataset transformation

```
x_train = df_final[1:1458,]
y_train = df$SalePrice

x_test = df_final[1459:nrow(df_final),]

dtrain = xgb.DMatrix(as.matrix(x_train), label = y_train)
dtest = xgb.DMatrix(as.matrix(x_test))
```

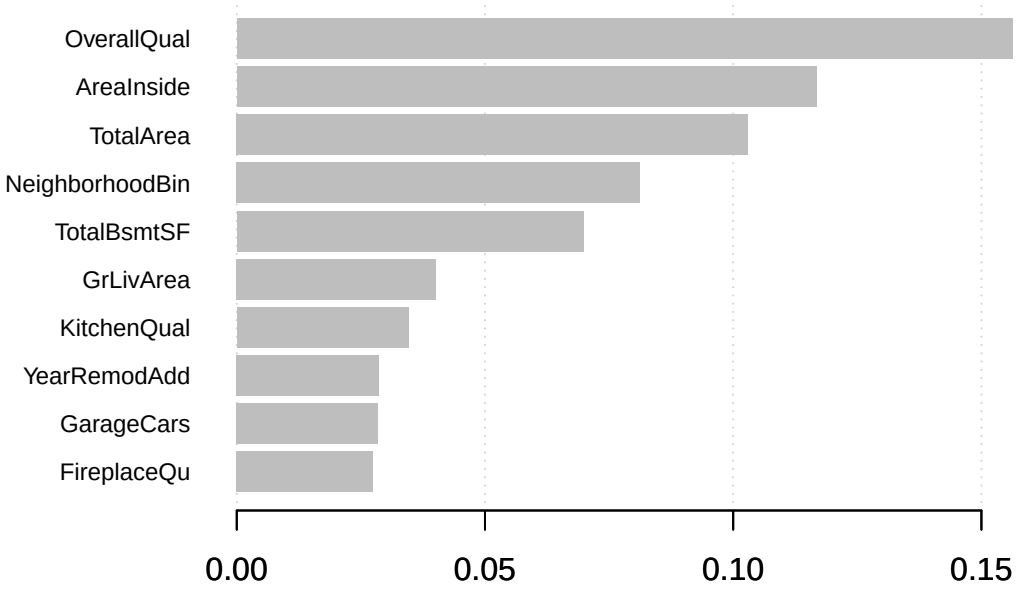
Parameters Tunning

```
cv.ctrl = trainControl(method = 'repeatedcv', repeats = 1, number = 4, allowParallel = T)

#Tunning parameter range
xgb.grid = expand.grid(nrounds = 750, eta = c(0.01,0.005,0.001), max_depth = c(4,6,8),colsample_bytree=
        min_child_weight = 2,
        subsample=c(0,0.2,0.4,0.6),
```

```
        gamma=0.01)
set.seed(40)


#xgb_tune = train(as.matrix(x_train),
             # y_train,
             # method="xgbTree",
             # trControl=cv.ctrl,
             # tuneGrid=xgb.grid,
             # verbose=T,
             # metric="RMSE",
             # nthread =3)


xgb_params = list(
  booster = 'gbtree',
  objective = 'reg:linear',
  colsample_bytree=1,
  eta=0.005,
  max_depth=4,
  min_child_weight=3,
  alpha=0.3,
  lambda=0.4,
  gamma=0.01, # less overfit
  subsample=0.6,
  seed=5,
  silent=TRUE)
```

XGboost Model Training

```
bst = xgb.train(xgb_params, dtrain, nrounds = 10000)


rmse_eval = function(y.true, y.pred) {
  mse_eval = sum((y.true - exp(y.pred)-1)^2) / length(y.true)
  return(sqrt(mse_eval))
}


y_pred.xgb = as.double(predict(bst, dtest))
y_pred.xgb = as.double(exp(y_pred.xgb))


#We could check the variable importance
model.names = dimnames(dtrain)[[2]]
importance_matrix = xgb.importance(model.names, model = bst)
xgb.plot.importance(importance_matrix[1:10])
```



Ridge and Lasso

One limitation to using XGBoost in particular is its inability to extrapolate and because of this we can use linear model to better predict any sale prices outside the range of prices given in our training set.

We know there exists multicollinearity in our dependent variables as area features are a determinant for a house having a certain amount of rooms, we know that the garage variables have heavy dependency and the list goes on. Due to this a simple linear regression model will not be of much help to predict accurate sale prices, which is why we can make use of both ridge and lasso regression.

Here we have the RMSE for elastic-net, lasso and ridge are all very similar yet not as low as the RMSE we got from XGBoost, however, ridge, lasso, and elastic-nets ability to extrapolate saleprices beyond the range we were given in the training set will be crucial for predicting more accurately towards unseen data (future houses). If we combine all 4 of these predictions on our test data by averaging all of our predictions we can get what we hope to be an accurate submission.

```
glm.cv.ridge = cv.glmnet(as.matrix(x_train), y_train, alpha = 0)
glm.cv.lasso = cv.glmnet(as.matrix(x_train), y_train, alpha = 1)
glm.cv.net = cv.glmnet(data.matrix(x_train), y_train, alpha = 0.001)

# use the lamdba that minimizes the error
penalty.ridge = glm.cv.ridge$lambda.min
penalty.lasso = glm.cv.lasso$lambda.min
penalty.net = glm.cv.net$lambda.min

glm.ridge = glmnet(x = as.matrix(x_train), y = y_train, alpha = 0, lambda = penalty.ridge )
glm.lasso = glmnet(x = as.matrix(x_train), y = y_train, alpha = 1, lambda = penalty.lasso)
glm.net = glmnet(x = as.matrix(x_train), y = y_train, alpha = 0.001, lambda = penalty.net)

y_pred.ridge = as.numeric(predict(glm.ridge, as.matrix(x_train)))
y_pred.lasso = as.numeric(predict(glm.lasso, as.matrix(x_train)))
y_pred.net = as.numeric(predict(glm.net, as.matrix(x_train)))
```

Final Boosting Result

```
y_pred.ridge = as.double(predict(glm.ridge, as.matrix(x_test)))
y_pred.lasso = as.double(predict(glm.lasso, as.matrix(x_test)))
y_pred.net = as.double(predict(glm.net, as.matrix(x_test)))

y_pred.ridge = as.double(exp(y_pred.ridge))
y_pred.lasso = as.double(exp(y_pred.lasso))
y_pred.net = as.double(exp(y_pred.net))

y_pred.xgb = as.double(predict(bst, dtest))
y_pred.xgb = as.double(exp(y_pred.xgb))

# take the average of our predictions for our ensemble
y_pred = (y_pred.xgb + y_pred.ridge + y_pred.lasso + y_pred.net)/4.0

#result = data.frame(ID = c(1461:2919), SalePrice = y_pred)
#write.csv(result, 'result.csv', row.names = F)
```