Management and Computer Science
a.a. 2019/2020

Data Analysis For Business - Project

Annalisa Feliziani - Nicolò Pagliari - Michael Saruggia
Group ID: 8

Project: Can You Predict The Price?

Introduction

We are asked to build a machine learning model in order to predict the appropriate price of a house given a set of features.

These features are provided within the "*train.csv*" file: a dataset containing a set of observations along with the response variable (which is *SalePrice*: the price of a house).

We will use this .csv file in order to fit our prediction model; then we will use the fitted model with the "*test.csv*" file to predict the price of the houses.

"*test.csv*" is a dataset containing a set of observations, but without the response variable.

First Steps

In order to predict our response variables we will use **RStudio**.

We have already said that we are gonna use the "*train.csv*" dataset in order to fit the model; but before to do so, we need to **split this dataset into two parts**.

Why?

Because one part will be used to fit the model, and the other part will be used to test his accuracy.

We cannot use the "*test.csv*" dataset to test our model since the actual response variables are missing.

"*train.csv*" is composed of **1,100 observations**, so we have decided to randomly split them using the `sample()` function.

Now, let's start coding.

Launch RStudio and **open the two datasets** using the "`read.csv()`" function:

```
> data_train <- read.csv("/Project data analysis for business/train.csv")
> data_test <- read.csv("/Project data analysis for business/test.csv")
```

For now we will use only the "train.csv" dataset, which is now stored inside "`data_train`".

Our next step is to split data_train using `sample()`:

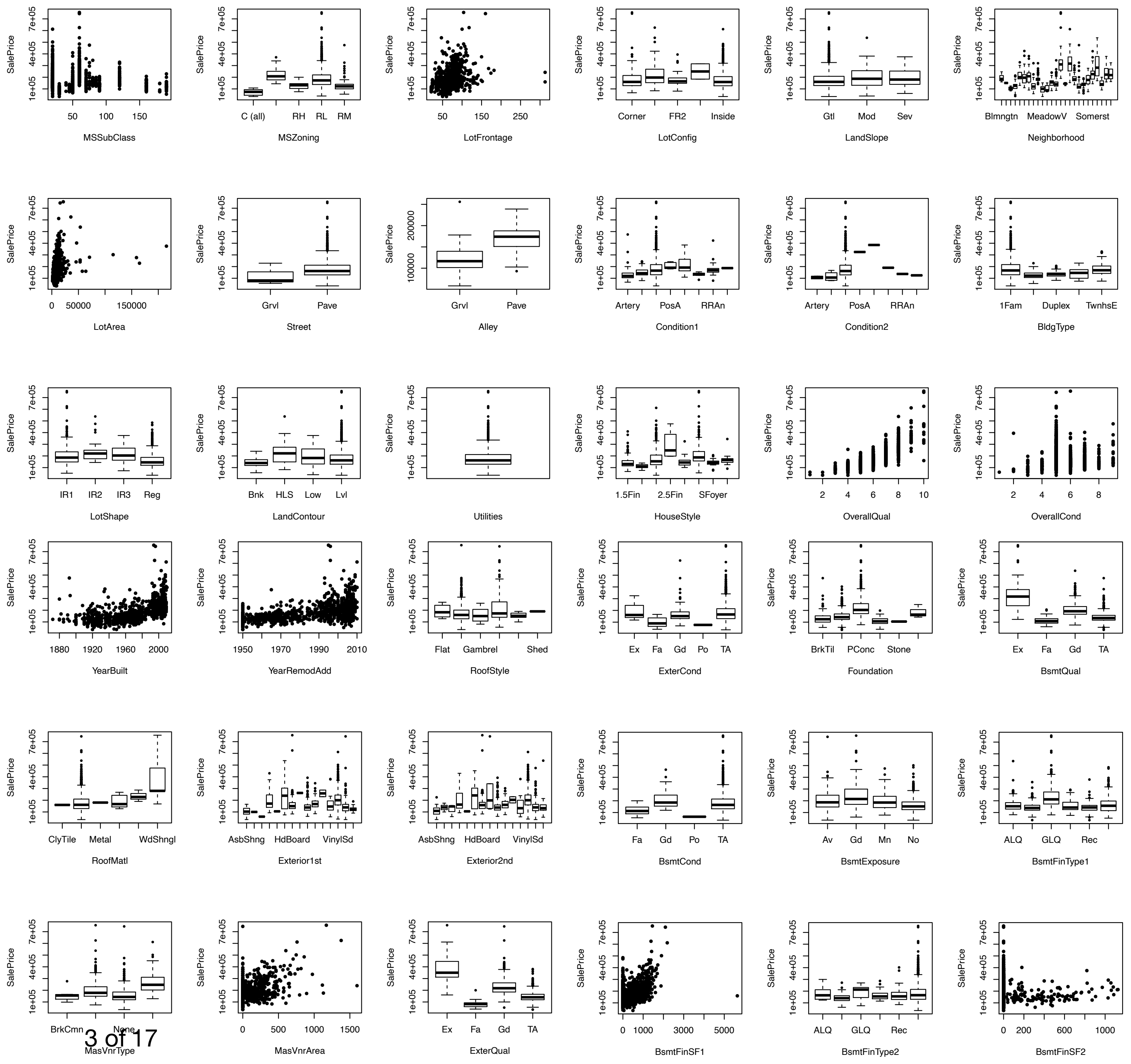
```
> train <- sample(1:nrow(data_train), nrow(data_train)/2)
> test <- (-train)
```

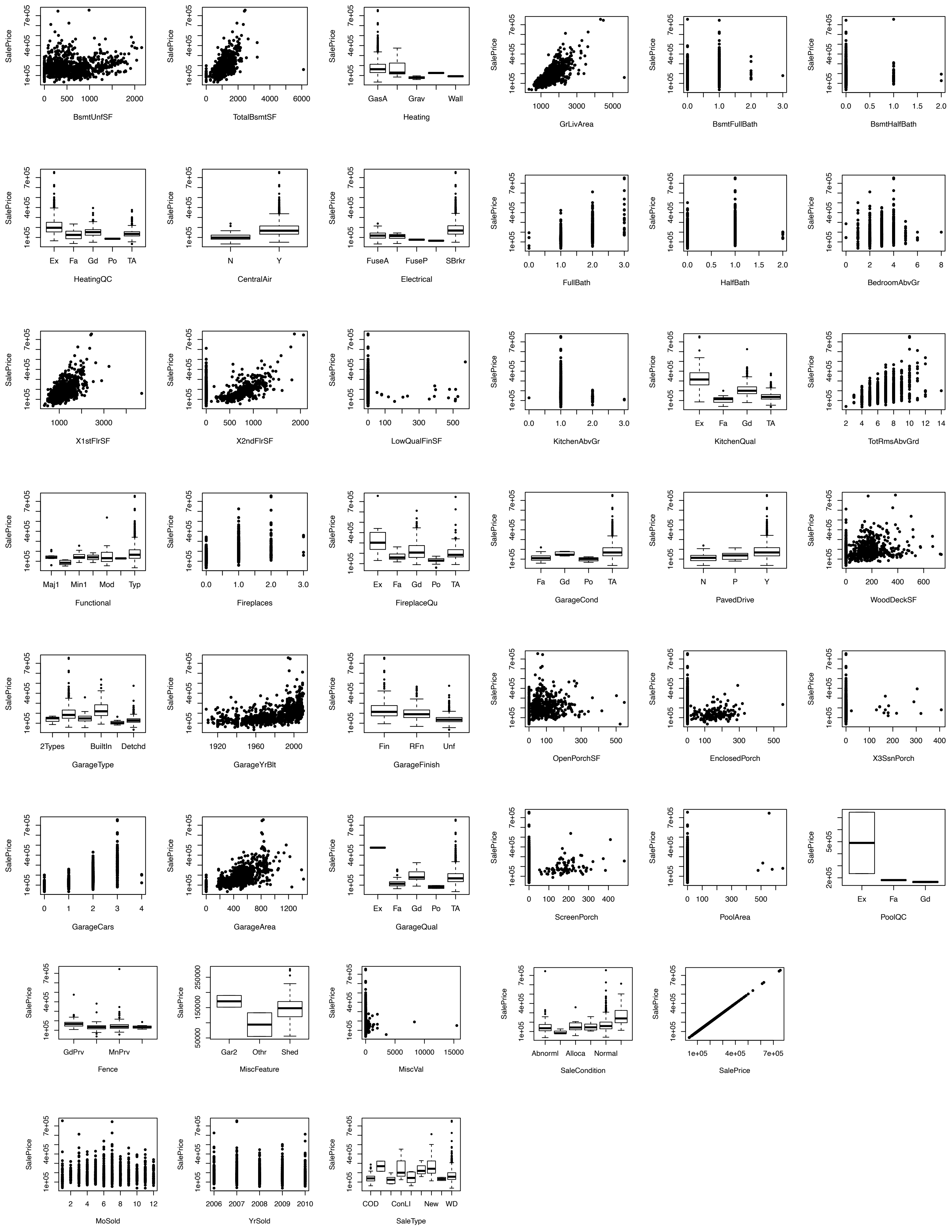
Now, we are ready to start.

Verifying correlations: Exploratory Data Analysis

Now that our dataset is loaded, we can start with some *Exploratory Data Analysis* (EDA). We are interested in verifying the correlations between the response SalePrice and every other predictor. To do this, we draw and export some plots using these commands:

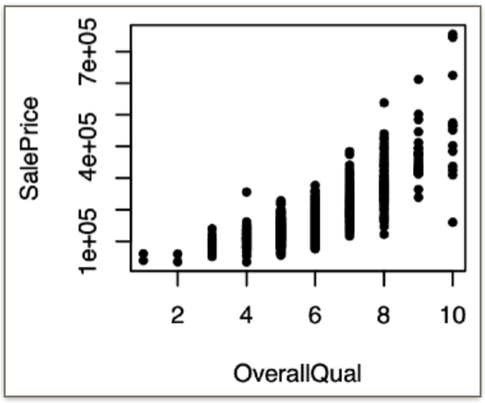
```
> pdf("my_plot.pdf")
> par(mfrow = c(4,3))
> for(i in 1:ncol(data_train)){
>   plot(data_train[,i], data_train$SalePrice, xlab=names(data_train[i]),
ylab="SalePrice", pch=20)
   }
> dev.off()
```




SalePrice
BsmtUnfSF
TotalBsmtSF
Heating
GrLivArea
BsmtFullBath
BsmtHalfBath
HeatingQC
CentralAir
Electrical
FullBath
HalfBath
BedroomAbvGr
X1stFlrSF
X2ndFlrSF
LowQualFinSF
KitchenAbvGr
KitchenQual
TotRmsAbvGrd
Functional
Fireplaces
FireplaceQu
GarageCond
PavedDrive
WoodDeckSF
GarageType
GarageYrBlt
GarageFinish
OpenPorchSF
EnclosedPorch
X3SsnPorch
GarageCars
GarageArea
GarageQual
ScreenPorch
PoolArea
PoolQC
Fence
MiscFeature
MiscVal
SaleCondition
SalePrice
MoSold
YrSold
SaleType

We are looking for *correlation* among predictors and response; that is: as a predictor increases, the response either decreases or increases (in the case of quantitative variables).

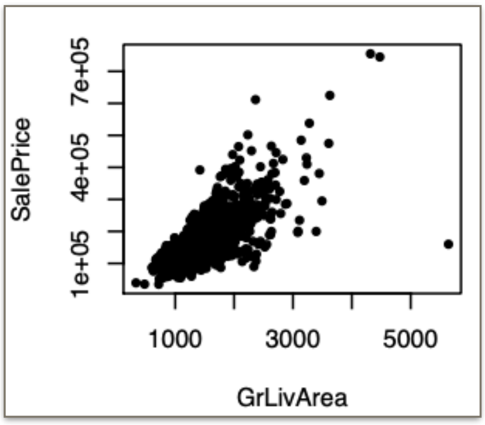
For example, *OverallQual* appears to be highly correlated with *SalePrice*: as the value of *OverallQual* increases, we expect an higher *SalePrice*.



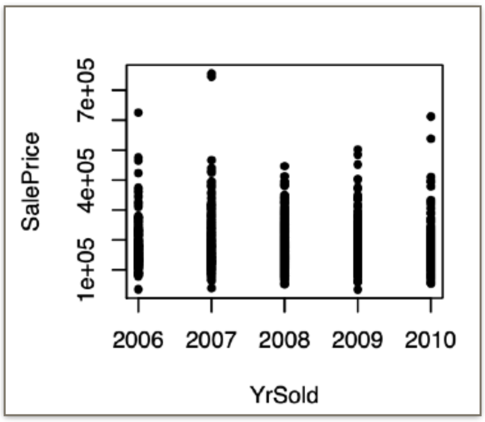
We can say the same thing about *GrLivArea*, which is the ground living area in square feet.

This is not a surprise, since it is common that a larger house will likely have an higher price.

However we can clearly see something strange on the right: the house with the highest GrLivArea has a relatively small price…



YrSold, instead, does not seem to have any relationship with our response.

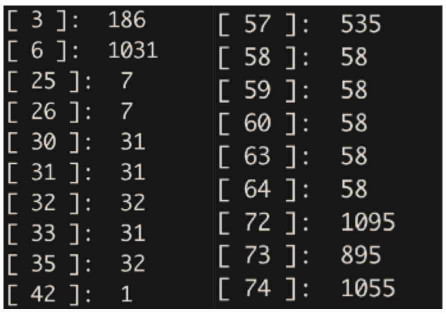


By looking at this plots, we can start making an idea about which predictors could be useful for the model that we are going to fit.

For example, this useful-predictors could be: LotFrontage, Neighborhood, Condition2, OverallQual, HouseStyle, YearBuilt, TotalBsmtSF, X1stFlrSF, X2ndFlrSF, GrLivArea, FullBath, TotRmsAbvGrd, GarageYrBlt, GarageCars and others…

We have noticed that this dataset contains a lot of NA values.
Using this function we can print how many NAs there are inside each predictor column:

```
> for (i in 1:ncol(data_train)){
    n <- sum(is.na(data_train[,i]))
    if (n > 100){
        cat("[", i, "]: ", n, "\n")
    }
}
```



```
[ 3 ]:  186      [ 57 ]:  535
[ 6 ]:  1031     [ 58 ]:  58
[ 25 ]:  7       [ 59 ]:  58
[ 26 ]:  7       [ 60 ]:  58
[ 30 ]:  31      [ 63 ]:  58
[ 31 ]:  31      [ 64 ]:  58
[ 32 ]:  32      [ 72 ]:  1095
[ 33 ]:  31      [ 73 ]:  895
[ 35 ]:  32      [ 74 ]:  1055
[ 42 ]:  1
```

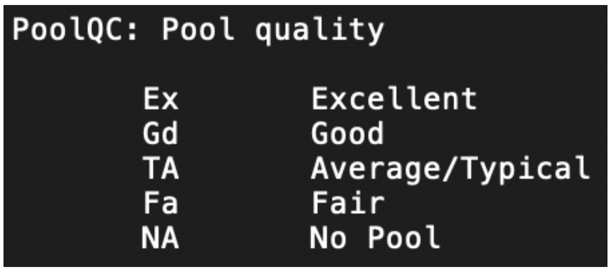
We have decided to remove the predictors which showed a number of NAs > 500.
This can be done with the following commands:

```
> indexes_NA <- c() #boolean vector: TRUE means that
predictor i has NA values
for (i in 1:ncol(data_train)){
    n <- sum(is.na(data_train[,i]))
    if (n > 500){
        indexes_NA <- c(indexes_NA, TRUE)
    }
    else{ indexes_NA <- c(indexes_NA, FALSE) }
}
#remove:
data_train <- data_train[,!indexes_NA]
```

We need to remove the predictor "Utilities", which contains only 1 level of factor.

```
> data_train$Utilities <- NULL
```

How we can deal with the remaining NAs values?
We reasoned as follows: reading the file data_description.txt, we discovered that some NA values are not associated to a missing observation of the predictor, but it represents that the feature is not present in the house.



```
PoolQC: Pool quality

        Ex      Excellent
        Gd      Good
        TA      Average/Typical
        Fa      Fair
        NA      No Pool
```

Since NAs can raise problems during the fitting of the model, we can decide either to remove the whole predictors, or to substitute them with some value.
Based on our observation of data_description.txt, we wanted to try substituting some of these "false" NAs with a new factor that has value "No".
This can be done in this way:

```
> data_train$GarageFinish <- as.character(data_train$GarageFinish)
> data_train$GarageFinish[is.na(data_train$GarageFinish)] <- "No"
> data_train$GarageFinish <- as.factor(data_train$GarageFinish)

> data_train$GarageQual <- as.character(data_train$GarageQual)
> data_train$GarageQual[is.na(data_train$GarageQual)] <- "No"
> data_train$GarageQual <- as.factor(data_train$GarageQual)
```

And so on for all the remaining predictors...
For the NAs coming from quantitative predictors, we decided to substitute them with either the mean of the other values, or a 0 value.

```
> data_train$GarageYrBlt[is.na(data_train$GarageYrBlt)] <- 0
> data_train$MasVnrArea[is.na(data_train$MasVnrArea)] <-
mean(data_train$MasVnrArea, na.rm = TRUE) #substitute with mean
> data_train$LotFrontage[is.na(data_train$LotFrontage)] <-
mean(data_train$LotFrontage, na.rm = TRUE)
```

In the NA observations of GarageYrBlt we have not assigned the mean of the other values because this predictor corresponds to the year in which the garage was built. If the garage is not present in the house we assign a 0 value to GarageYrBlt.
For the predictor "Electrical" we assigned the value "SBrkr" since this is the value that appears in almost every observation.

```
> data_train$Electrical[is.na(data_train$Electrical)] <- "SBrkr"
```

We can verify if we have any remaining NA in the dataset using the following command:

```
> sum(is.na(data_train))
```

Model 1 - Fitting a Linear Regression

We start by making an assumption about the shape of our model, and we decided to try with a **linear** one. Linear regression is a very simple approach for supervised learning.
Since we have 79 total predictors, our regression will be a **multiple linear regression**.
That's how we're going to proceed: we're going to perform a linear regression using all the 79 predictors, and then we're going to perform variable selection analyzing the p-values. In particular, the way of reasoning would be to look for predictors with a small associated p-value in order to detect which are most significant ones.

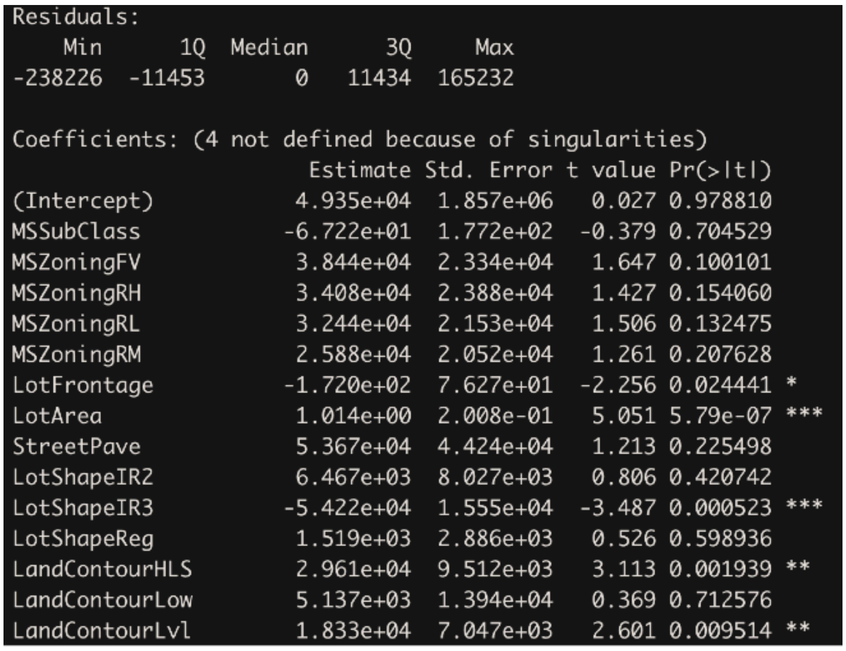
Before proceeding, we have to point out something: performing variable selection with p-values is not so correct. We realized this only after the model was completely fitted. However, we decided to keep this type of analysis both for completeness and for comparison with other models that are shown later in this paper.
Why is performing variable selection using p-values not a good idea?
Because the test statistic, and thus the p-values, runs together both the estimate of the actual size of the coefficient, and how well we can measure that particular coefficient.
So, p-values do not help us to answer to the question "*is this variable relevant to the response*?", but rather help us with the question "*can we reliably detect that this coefficient isn't exactly zero*?".
To fit the linear model we use the `lm()` function.

```
> lm.fit <- lm(SalePrice~., data=data_train[train,])
```

R automatically creates *dummy variables* for qualitative predictors.
Now we have to perform **variable selection**: which of the predictors are useful in predicting the response?
Using the **`summary()`** function we can analyze the p-values of all the predictors.

```
> summary(lm.fit)
```

Here it is a snippet of the output of the previous command -->
The p-values are the numbers of the column "`PR(>|t|)`". It's easy to see that the predictor "*LotArea*" has a low p-value relatively to the other values; this means that we are going to keep *LotArea* in our model.



```
Residuals:
    Min      1Q  Median      3Q     Max
-238226  -11453       0   11434  165232

Coefficients: (4 not defined because of singularities)
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          4.935e+04  1.857e+06   0.027 0.978810
MSSubClass          -6.722e+01  1.772e+02  -0.379 0.704529
MSZoningFV           3.844e+04  2.334e+04   1.647 0.100101
MSZoningRH           3.408e+04  2.388e+04   1.427 0.154060
MSZoningRL           3.244e+04  2.153e+04   1.506 0.132475
MSZoningRM           2.588e+04  2.052e+04   1.261 0.207628
LotFrontage         -1.720e+02  7.627e+01  -2.256 0.024441 *
LotArea              1.014e+00  2.008e-01   5.051 5.79e-07 ***
StreetPave           5.367e+04  4.424e+04   1.213 0.225498
LotShapeIR2          6.467e+03  8.027e+03   0.806 0.420742
LotShapeIR3         -5.422e+04  1.555e+04  -3.487 0.000523 ***
LotShapeReg          1.519e+03  2.886e+03   0.526 0.598936
LandContourHLS       2.961e+04  9.512e+03   3.113 0.001939 **
LandContourLow       5.137e+03  1.394e+04   0.369 0.712576
LandContourLvl       1.833e+04  7.047e+03   2.601 0.009514 **
```

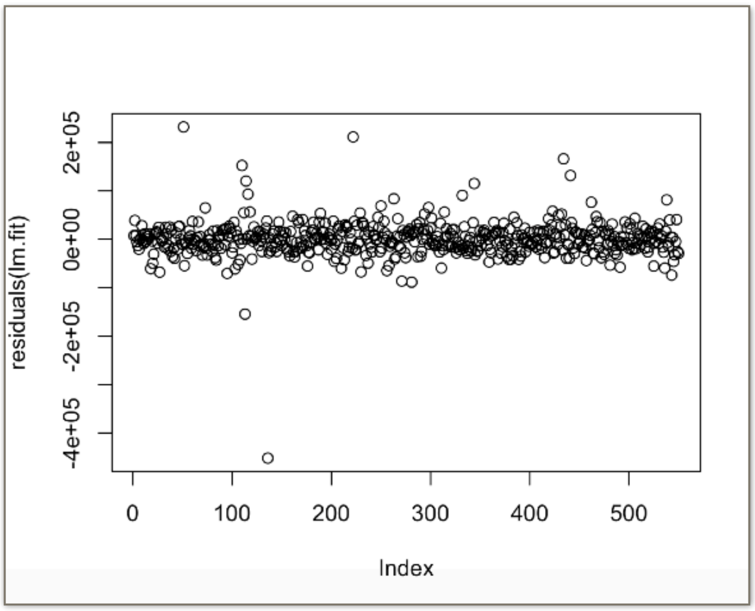
All the variables with a low p-value are:
KitchenQual, X2ndFlrSF, X1stFlrSF, BsmtQual, MasVnrArea, OverallQual, OverallCond, LotArea.

So we have fitted a model with these predictors along with some of the predictors found thanks to the EDA.

```
> lm.fit <- lm(SalePrice ~ KitchenQual + X2ndFlrSF + X1stFlrSF +
BsmtQual + MasVnrArea + OverallQual + OverallCond + LotArea +
HouseStyle + LotFrontage + YearBuilt +TotalBsmtSF, data =
data_train[train,])
```

Our linear model is now fitted.
In the first step we made the assumption that the shape of our model is linear.
One question we could raise is: "*is the linear model a good model for our prediction?*"
The answer can be found by looking at the residuals plot.
This plot can be visualized by using the function `residuals()`.

```
> plot(residuals(lm.fit))
```



If the residual plot seems to exhibit a U-shape, then we have some evidence of non-linearity of data.
In our case, we observe that the residual plot exhibits roughly a straight line; this confirms the fact that **the linear model is good for our prediction**.
There is something else we can notice in this graph. Some residuals are quite far from the others: they could be ***outliers***.
An outlier is a point for which the actual value is far from the value predicted by the model.
Even a single outlier could substantially increment the RSE.
To understand if we are dealing with an outlier or not we can compute the *studentized residuals*; if it's absolute value is greater than 3, then we are dealing with a possible outlier.
We can compute studentized residuals using the "`rstudent()`" function.

```
out <- abs(rstudent(lm.fit))
for(i in 1:length(out)){
    if(out[i] > 3){ print(out[i]) }
}
```

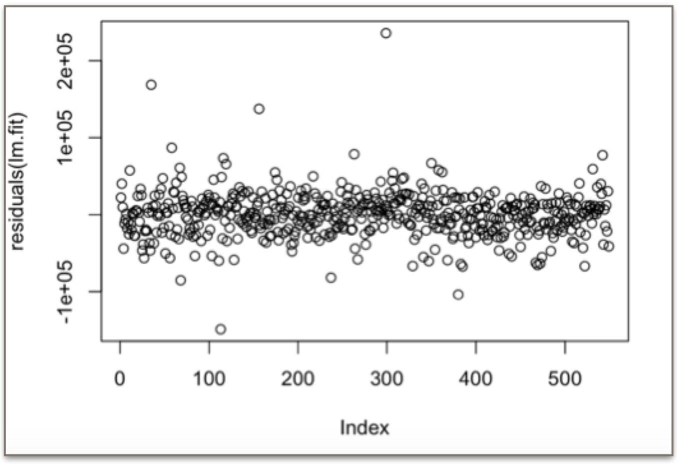
The indexes of the potential outliers are: 1003, 613, 120, 929, 558, 122, 630, 264, 399.

A solution for outliers is to simply remove the related observation.
We decided to remove only the first three observations with the highest residuals, which are the observations [1003], [558] and [122].

```
data_train <- data_train[-1003,]
data_train <- data_train[-558,]
data_train <- data_train[-122,]
```

In this way we have removed the observations of interest from the dataset. It is important to delete the observation in decreasing order of indexes; otherwise, the indexes order will be altered every time and the deletion will not delete the outlier.
Now, we have removed a total of 3 observations, so our dataset now contains 1,097 observations.
Now we fit the linear model again.

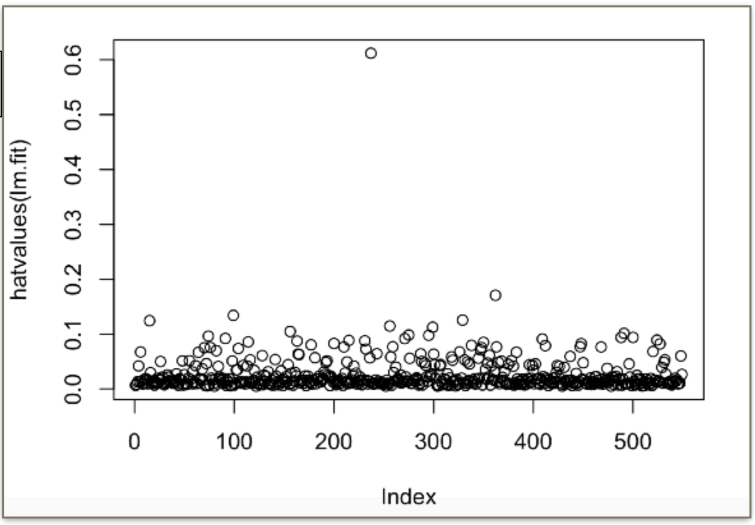
```
lm.fit <- lm(SalePrice ~ KitchenQual + X2ndFlrSF + X1stFlrSF + BsmtQual
+ MasVnrArea + OverallQual + OverallCond + LotArea + HouseStyle +
LotFrontage + YearBuilt +TotalBsmtSF, data = data_train[train,])
```



This is the new residual plot. It's clear to see that the highest outliers have been removed.
By analyzing this image, we can also see that there is not a "*funnel shape*" in the residual plot; this means that error terms have a constant variance.

The next step is to use the "**`hatvalues()`**" function to compute *leverage statistics*.
Leverage is a measure of how far away the values of an observation are from those of other observation.

```
> plot(hatvalues(lm.fit))
```



We can clearly see that one point is very far from the others.

Using this simple algorithm we can find which is the observation related to this point:

```
> hat <- hatvalues(lm.fit)
for(i in 1:length(hat)){
    if(hat[i] > 0.20) {print(hat[i])}
}
```

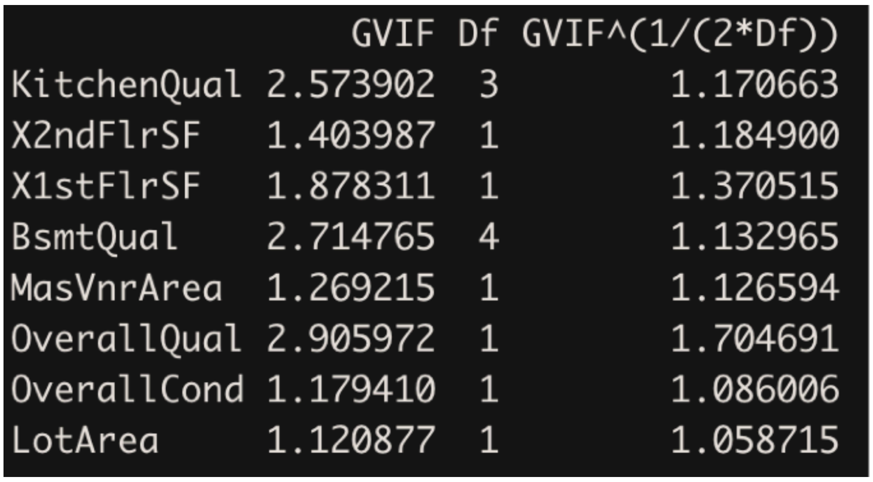
The index of the observation we are looking for is [116], and has a corresponding leverage of 0.612.
So, we have decided to delete it.

```
> data_train <- data_train[-116,]
```

The next step is to look for *collinearity*.
Collinearity refers to the situation in which two or more predictor variables are closely related to each other.
To detect collinearity we can compute the *Variance Inflaction Factor* (VIF).
A VIF value that exceed 5 indicates a problematic amount of collinearity.
The VIF can be easily calculated using the `vif()` function of the `car` package:

```
> library(car)
> vif(lm.fit)
```

The output clearly shows that collinearity is not a problem of our model:



```
                GVIF Df GVIF^(1/(2*Df))
KitchenQual 2.573902  3        1.170663
X2ndFlrSF   1.403987  1        1.184900
X1stFlrSF   1.878311  1        1.370515
BsmtQual    2.714765  4        1.132965
MasVnrArea  1.269215  1        1.126594
OverallQual 2.905972  1        1.704691
OverallCond 1.179410  1        1.086006
LotArea     1.120877  1        1.058715
```

Our model is now complete. We just need to use it to predict the values.
We will first use the model to predict the responses of the test part of `data_train`.
Once this responses have been predicted we will compare them to the actual responses in order to verify the accuracy of our model.

Assessing the Accuracy of Our Model - Linear

It is now time to predict the responses of `data_train[test,]` using the `predict()` function.
The first parameter of the function `predict()` is the model that we have fitted;
the second parameter is the data set in which we want to predict the responses.
We assign these predicted responses to a vector named "`predicted`".
We want to compare these predicted responses to the actual responses of `data_train[test,]`. In order to do this, it is necessary to create another vector that contains the actual responses.
The quality of a linear regression fit is typically assessed using the *MSE* (Mean Squared Error), the *RSE* (Residual Standard Error) and the R^2 Statistic.
We are going to compute the MSE using the k-fold cross validation method.

MSE ~ K-Fold Cross-Validation

The MSE measures the average of the squares of the errors, that is the average squared difference between the estimated values and the actual values.
The K-Fold approach involves randomly dividing the set of observations into *k* groups, or *folds*, of approximately equal size.
The first fold is treated as a *validation set*, and the method is fit on the remaining *k – 1* folds.
The MSE is computed on the observations in the held-out fold. This procedure is repeated *k* times; each time, a different group of observations is treated as a validation set.
This process results in k estimates of the test error, MSE_1, MSE_2, ..., MSE_k.
The k-fold CV estimate is computed by averaging these values.
To do this, we have developed the following code:

```
> mse.lm <- rep(0,10)
> for(i in 1:10){
    set.seed(i)
    train <- sample(1:nrow(data_train), nrow(data_train)/2)
    test <- (-train)
    y <- data_train[test,]$SalePrice
    lm.fit <- lm(SalePrice ~ KitchenQual + X2ndFlrSF + X1stFlrSF + 
BsmtQual + MasVnrArea + OverallQual + OverallCond + LotArea + 
HouseStyle + LotFrontage + YearBuilt +TotalBsmtSF, data = 
data_train[train,])
    pred <- predict(lm.fit, data_train[test,])
    mse.lm[i] <- mean((y - pred)^2)
}
> print(sqrt(mean(mse.lm)))
```

The resulting average RMSE is **29826.37**.

Model 2 - Fitting the Lasso

The Lasso is a *shrinkage* method that can shrink coefficient estimates to exactly zero; this means that if the coefficient is zero, then the corresponding variable drops out of the model. Thus, the Lasso method also performs by itself a kind of variable selection.
For this regression we are going to use the same dataset of the previous created model.
To fit the model using the Lasso method, we use the function `glmnet()` of the `gmlnet` package.
First of all, let's install the `glmnet` package:

```
> install.packages("glmnet")
> library(glmnet)
```

To fit our model, we decided to directly use the k-fold method using the following code that we have developed:

```
mse.lasso <- rep(0,10)
for (i in 1:10){
    x <- model.matrix(SalePrice~., data_train)[,-1] #Predictors
    y <- data_train$SalePrice #Response
    set.seed(i)
    train <- sample(1:nrow(x), nrow(x)/2) #split train and test
    test <- (-train)
    y.test = y[test]
    lasso.mod <- glmnet(x[train,], y[train], alpha=1) #fitting a lasso model
    cv.out <- cv.glmnet(x[train,], y[train], alpha=1) #find best lambda
    bestlam <- cv.out$lambda.min
    lasso.pred <- predict(lasso.mod, s=bestlam, newx=x[test,])
    mse.lasso[i] <- mean((y.test - lasso.pred)^2)
}
print(sqrt(mean(mse.lasso)))
```

The average MSE of this model is **26409.51**, a great step forward with respect to the previous linear model.

Average R^2

R^2 takes of the form a proportion and so it always takes on a value between 0 and 1.
It measures the proportion of variability in the response that can be explained using the regressors.
An R^2 Statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression model.
So, the closer it is to 1, the better.
To compute the R^2 in R we have developed the following function:

```
rsq <- function(predicted, actual){
    mean <- mean(actual)
    tss <- 0
    rss <- 0
    for(i in 1:length(predicted)){
        tss <- tss + (actual[i] - mean)^2
        rss <- rss + (actual[i] - predicted[i])^2
    }
    rs <- 1 - rss/tss
}
```

We are going to use this function to compute the average R^2 among 10 different dataset splits.

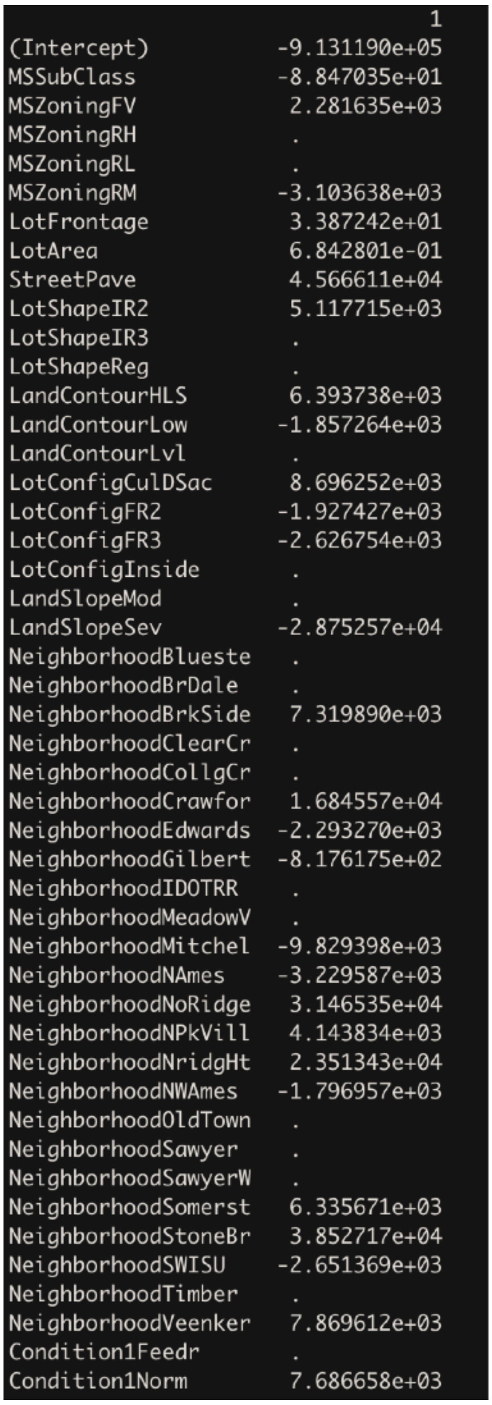
```
rs.lasso <- rep(0,10)
for (i in 1:10){
    x <- model.matrix(SalePrice~., data_train)[,-1] #Predictors
    y <- data_train$SalePrice #Response
    set.seed(i)
    train <- sample(1:nrow(x), nrow(x)/2) #split train and test
    test <- (-train)
    y.test = y[test]
    lasso.mod <- glmnet(x[train,], y[train], alpha=1) #fitting a lasso model
    cv.out <- cv.glmnet(x[train,], y[train], alpha=1) #find best lambda
    bestlam <- cv.out$lambda.min
    lasso.pred <- predict(lasso.mod, s=bestlam, newx=x[test,])
    rs.lasso[i] <-rsq(lasso.pred, y.test)
}
print(sqrt(mean(rs.lasso)))
```

$R^2 \approx 0.94$

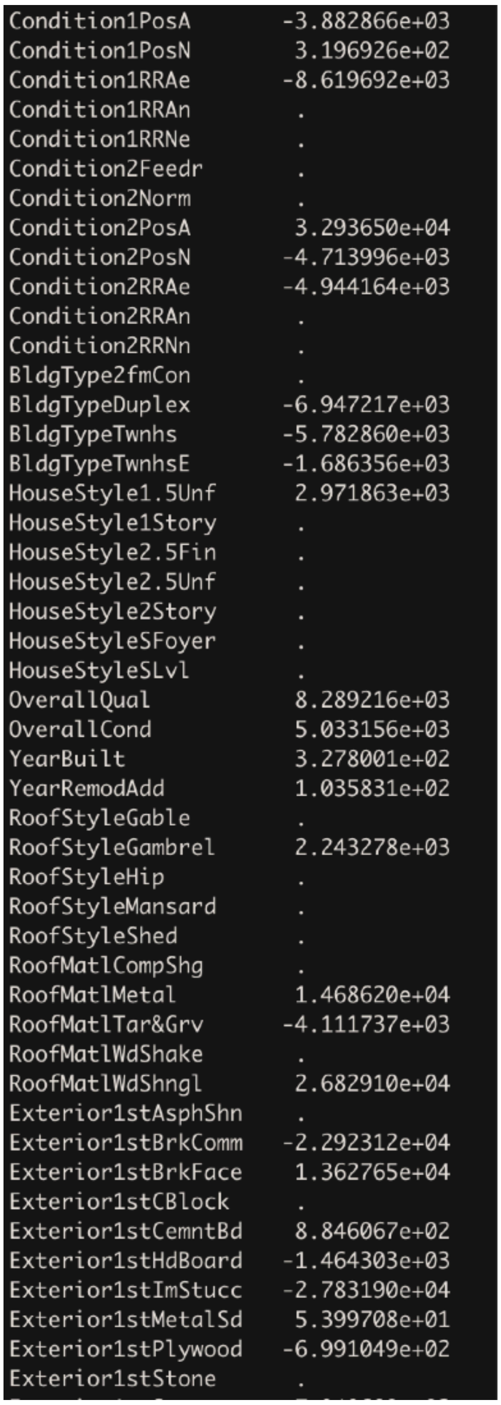
Our model seems to have a pretty good level of accuracy.

For the sake of completeness, let's print the coefficient estimates for the lasso:

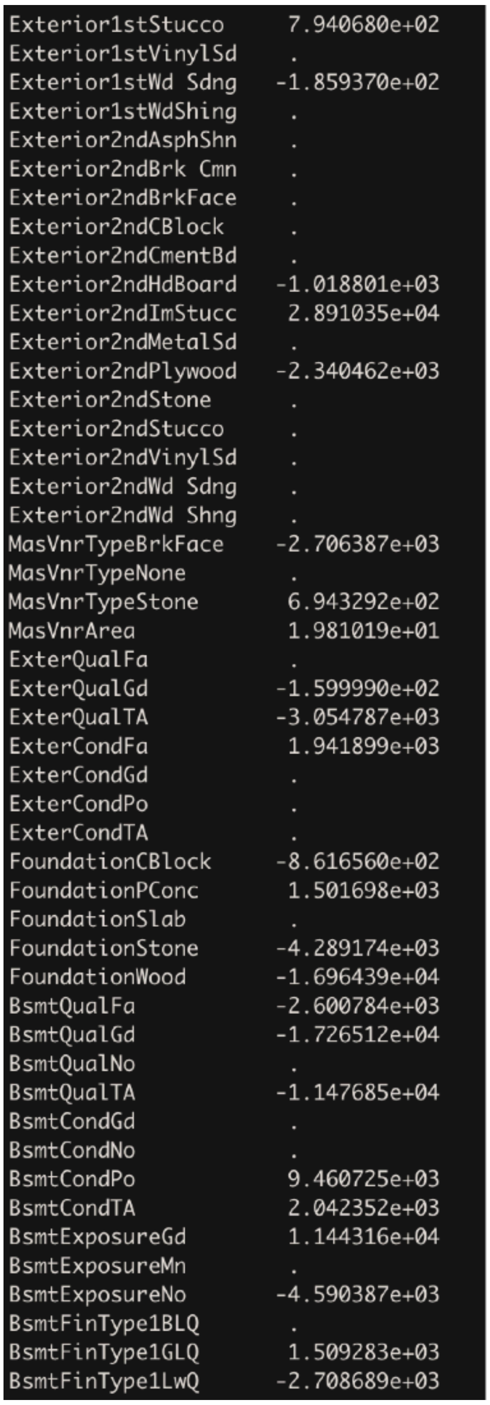
```
> out <- glmnet(x,y, alpha=1)
> lasso.coef <- predict(out, type="coefficient", s=bestlam)
> print(lasso.coef)
```



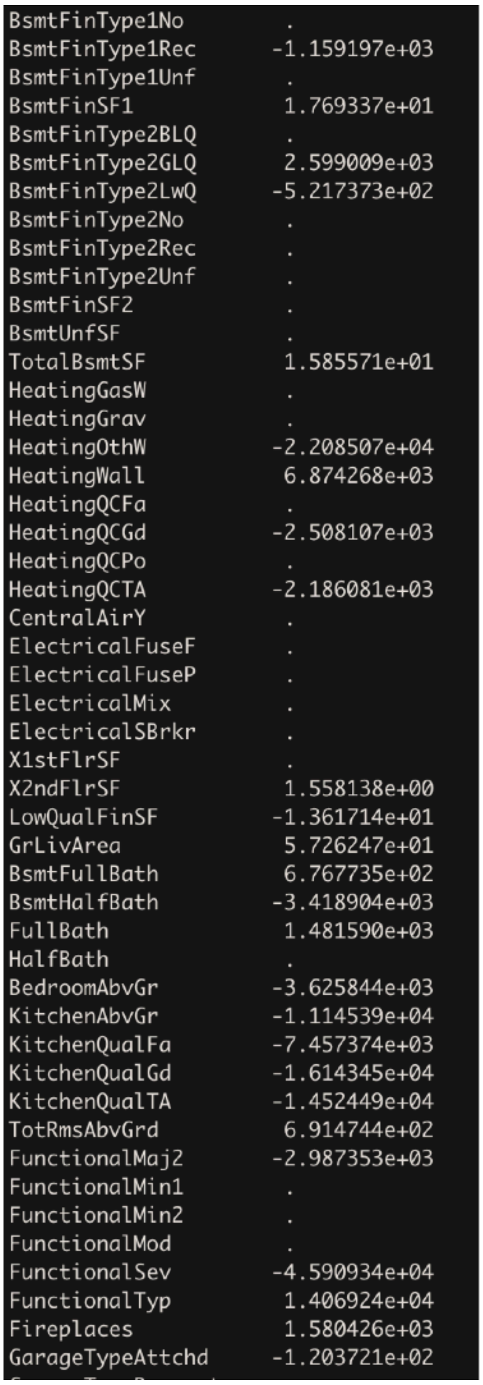
```
                                 1
(Intercept)          -9.131190e+05
MSSubClass           -8.847035e+01
MSZoningFV            2.281635e+03
MSZoningRH            .
MSZoningRL            .
MSZoningRM           -3.103638e+03
LotFrontage           3.387242e+01
LotArea               6.842801e-01
StreetPave            4.566611e+04
LotShapeIR2           5.117715e+03
LotShapeIR3           .
LotShapeReg           .
LandContourHLS        6.393738e+03
LandContourLow       -1.857264e+03
LandContourLvl        .
LotConfigCulDSac      8.696252e+03
LotConfigFR2         -1.927427e+03
LotConfigFR3         -2.626754e+03
LotConfigInside       .
LandSlopeMod          .
LandSlopeSev         -2.875257e+04
NeighborhoodBlueste   .
NeighborhoodBrDale    .
NeighborhoodBrkSide   7.319890e+03
NeighborhoodClearCr   .
NeighborhoodCollgCr   .
NeighborhoodCrawfor   1.684557e+04
NeighborhoodEdwards  -2.293270e+03
NeighborhoodGilbert  -8.176175e+02
NeighborhoodIDOTRR    .
NeighborhoodMeadowV   .
NeighborhoodMitchel  -9.829398e+03
NeighborhoodNAmes    -3.229587e+03
NeighborhoodNoRidge   3.146535e+04
NeighborhoodNPkVill   4.143834e+03
NeighborhoodNridgHt   2.351343e+04
NeighborhoodNWAmes   -1.796957e+03
NeighborhoodOldTown   .
NeighborhoodSawyer    .
NeighborhoodSawyerW   .
NeighborhoodSomerst   6.335671e+03
NeighborhoodStoneBr   3.852717e+04
NeighborhoodSWISU    -2.651369e+03
NeighborhoodTimber    .
NeighborhoodVeenker   7.869612e+03
Condition1Feedr       .
Condition1Norm        7.686658e+03
```



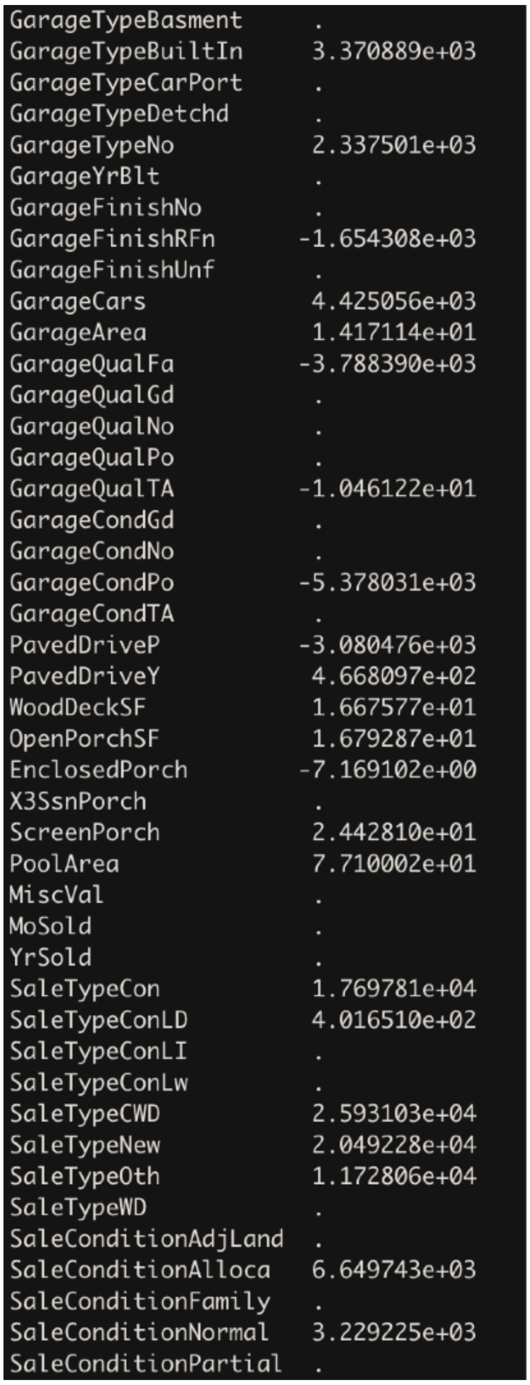
```
Condition1PosA       -3.882866e+03
Condition1PosN        3.196926e+02
Condition1RRAe       -8.619692e+03
Condition1RRAn        .
Condition1RRNe        .
Condition2Feedr       .
Condition2Norm        .
Condition2PosA        3.293650e+04
Condition2PosN       -4.713996e+03
Condition2RRAe       -4.944164e+03
Condition2RRAn        .
Condition2RRNn        .
BldgType2fmCon        .
BldgTypeDuplex       -6.947217e+03
BldgTypeTwnhs        -5.782860e+03
BldgTypeTwnhsE       -1.686356e+03
HouseStyle1.5Unf      2.971863e+03
HouseStyle1Story      .
HouseStyle2.5Fin      .
HouseStyle2.5Unf      .
HouseStyle2Story      .
HouseStyleSFoyer      .
HouseStyleSLvl        .
OverallQual           8.289216e+03
OverallCond           5.033156e+03
YearBuilt             3.278001e+02
YearRemodAdd          1.035831e+02
RoofStyleGable        .
RoofStyleGambrel      2.243278e+03
RoofStyleHip          .
RoofStyleMansard      .
RoofStyleShed         .
RoofMatlCompShg       .
RoofMatlMetal         1.468620e+04
RoofMatlTar&Grv      -4.111737e+03
RoofMatlWdShake       .
RoofMatlWdShngl       2.682910e+04
Exterior1stAsphShn    .
Exterior1stBrkComm   -2.292312e+04
Exterior1stBrkFace    1.362765e+04
Exterior1stCBlock     .
Exterior1stCemntBd    8.846067e+02
Exterior1stHdBoard   -1.464303e+03
Exterior1stImStucc   -2.783190e+04
Exterior1stMetalSd    5.399708e+01
Exterior1stPlywood   -6.991049e+02
Exterior1stStone      .
```



```
Exterior1stStucco     7.940680e+02
Exterior1stVinylSd    .
Exterior1stWd Sdng   -1.859370e+02
Exterior1stWdShing    .
Exterior2ndAsphShn    .
Exterior2ndBrk Cmn    .
Exterior2ndBrkFace    .
Exterior2ndCBlock     .
Exterior2ndCmentBd    .
Exterior2ndHdBoard   -1.018801e+03
Exterior2ndImStucc    2.891035e+04
Exterior2ndMetalSd    .
Exterior2ndPlywood   -2.340462e+03
Exterior2ndStone      .
Exterior2ndStucco     .
Exterior2ndVinylSd    .
Exterior2ndWd Sdng    .
Exterior2ndWd Shng    .
MasVnrTypeBrkFace    -2.706387e+03
MasVnrTypeNone        .
MasVnrTypeStone       6.943292e+02
MasVnrArea            1.981019e+01
ExterQualFa           .
ExterQualGd          -1.599990e+02
ExterQualTA          -3.054787e+03
ExterCondFa           1.941899e+03
ExterCondGd           .
ExterCondPo           .
ExterCondTA           .
FoundationCBlock     -8.616560e+02
FoundationPConc       1.501698e+03
FoundationSlab        .
FoundationStone      -4.289174e+03
FoundationWood       -1.696439e+04
BsmtQualFa           -2.600784e+03
BsmtQualGd           -1.726512e+04
BsmtQualNo            .
BsmtQualTA           -1.147685e+04
BsmtCondGd            .
BsmtCondNo            .
BsmtCondPo            9.460725e+03
BsmtCondTA            2.042352e+03
BsmtExposureGd        1.144316e+04
BsmtExposureMn        .
BsmtExposureNo       -4.590387e+03
BsmtFinType1BLQ       .
BsmtFinType1GLQ       1.509283e+03
BsmtFinType1LwQ      -2.708689e+03
```



```
BsmtFinType1No          .
BsmtFinType1Rec        -1.159197e+03
BsmtFinType1Unf         .
BsmtFinSF1              1.769337e+01
BsmtFinType2BLQ         .
BsmtFinType2GLQ         2.599009e+03
BsmtFinType2LwQ        -5.217373e+02
BsmtFinType2No          .
BsmtFinType2Rec         .
BsmtFinType2Unf         .
BsmtFinSF2              .
BsmtUnfSF               .
TotalBsmtSF             1.585571e+01
HeatingGasW             .
HeatingGrav             .
HeatingOthW            -2.208507e+04
HeatingWall             6.874268e+03
HeatingQCFa             .
HeatingQCGd            -2.508107e+03
HeatingQCPo             .
HeatingQCTA            -2.186081e+03
CentralAirY             .
ElectricalFuseF         .
ElectricalFuseP         .
ElectricalMix           .
ElectricalSBrkr         .
X1stFlrSF               .
X2ndFlrSF               1.558138e+00
LowQualFinSF           -1.361714e+01
GrLivArea               5.726247e+01
BsmtFullBath            6.767735e+02
BsmtHalfBath           -3.418904e+03
FullBath                1.481590e+03
HalfBath                .
BedroomAbvGr           -3.625844e+03
KitchenAbvGr           -1.114539e+04
KitchenQualFa          -7.457374e+03
KitchenQualGd          -1.614345e+04
KitchenQualTA          -1.452449e+04
TotRmsAbvGrd            6.914744e+02
FunctionalMaj2         -2.987353e+03
FunctionalMin1          .
FunctionalMin2          .
FunctionalMod           .
FunctionalSev          -4.590934e+04
FunctionalTyp           1.406924e+04
Fireplaces              1.580426e+03
GarageTypeAttchd       -1.203721e+02
```



```
GarageTypeBasment       .
GarageTypeBuiltIn       3.370889e+03
GarageTypeCarPort       .
GarageTypeDetchd        .
GarageTypeNo            2.337501e+03
GarageYrBlt             .
GarageFinishNo          .
GarageFinishRFn        -1.654308e+03
GarageFinishUnf         .
GarageCars              4.425056e+03
GarageArea              1.417114e+01
GarageQualFa           -3.788390e+03
GarageQualGd            .
GarageQualNo            .
GarageQualPo            .
GarageQualTA           -1.046122e+01
GarageCondGd            .
GarageCondNo            .
GarageCondPo           -5.378031e+03
GarageCondTA            .
PavedDriveP            -3.080476e+03
PavedDriveY             4.668097e+02
WoodDeckSF              1.667577e+01
OpenPorchSF             1.679287e+01
EnclosedPorch          -7.169102e+00
X3SsnPorch              .
ScreenPorch             2.442810e+01
PoolArea                7.710002e+01
MiscVal                 .
MoSold                  .
YrSold                  .
SaleTypeCon             1.769781e+04
SaleTypeConLD           4.016510e+02
SaleTypeConLI           .
SaleTypeConLw           .
SaleTypeCWD             2.593103e+04
SaleTypeNew             2.049228e+04
SaleTypeOth             1.172806e+04
SaleTypeWD              .
SaleConditionAdjLand    .
SaleConditionAlloca     6.649743e+03
SaleConditionFamily     .
SaleConditionNormal     3.229225e+03
SaleConditionPartial    .
```

Now we can proceed to predict the values of `data_test`, which is our final goal.
Before doing so, remember to delete NA values from `data_test` in the same way we did with `data_train`

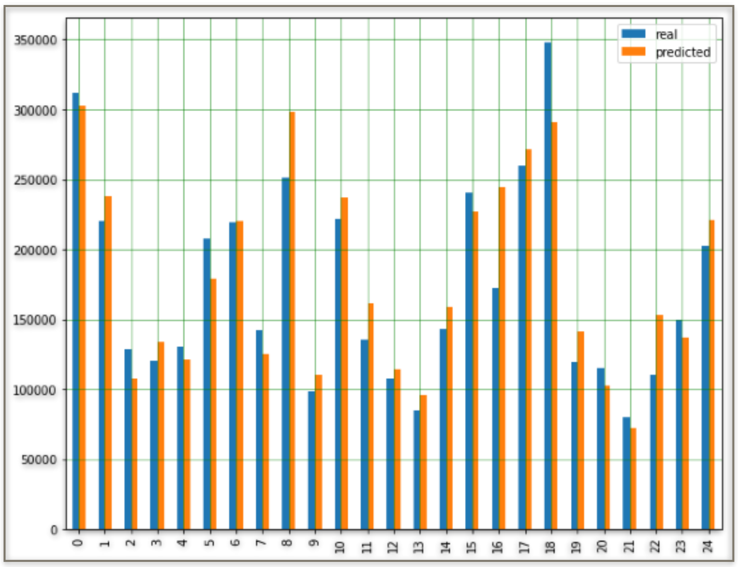
Conclusions

We were given a large dataset with many predictors, and we managed to restrict our regression analysis to only a few, which are the most significant ones, in order to predict the response SalePrice.

In this way, we obtained a linear model that seems to show a pretty good level of accuracy, as can be visualized by the following graph:

- The blue bars represent the real values of the train set
- The orange bars represent the predicted values of train set

It's clear to see that the two bars of each observation are pretty similar in height.


real
predicted