Regression - Final Project: House Prices Prediction

Tommy Tran - Thomas de Mareuil

12/22/2019

I. Introduction

The goal of this project is to build a model to predict house prices based on several input variables, both quantitative and qualitative. For training, we use a pre-processed dataset containing 1460 àbservations of 68 variables, among which information about sale prices, location, house characteristics.

Our research hypothesis is that sale prices can be estimated with a certain accuracy based on such a set of variables. We will first go through exploratory data analysis to understand our dataset and check how variables behave (compared to sale price and to each other), before building and evaluating regression models.

Considering the large number of covariates in our dataset, an important step will be to select the most relevant ones. For numerical variables, we will analyse correlation in the EDA part. For categorical variables, we will evaluate significance with Anova/Ancova in the Modeling part. We will then try several models, select features, check postulates, and keep the best-performing model.

Load the data

```
data = read.csv('train_preprocessed.csv', header=TRUE, row.names="X")
```

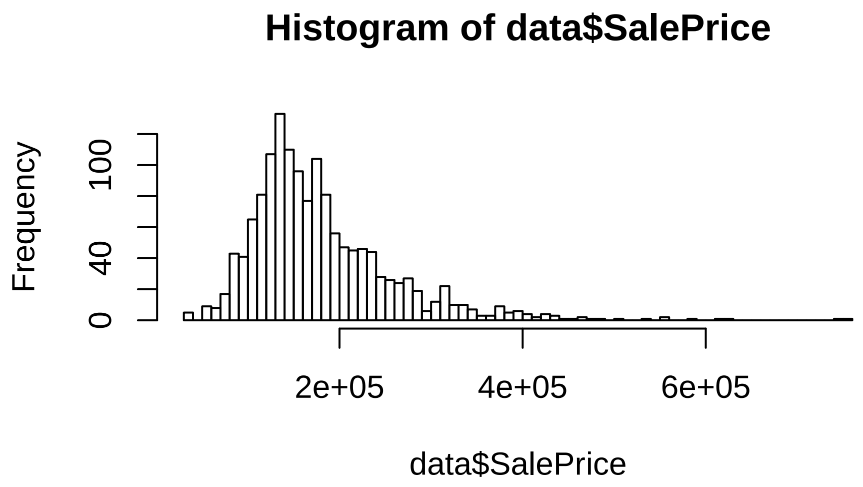
II. Exploratory Data Analysis

In our preprocessed dataset, we have a total of 66 covariates, of which 28 are quantitative and 38 are qualitative:

```
quantitative = names(dplyr::select_if(data, is.numeric))
qualitative = names(dplyr::select_if(data, is.object))
```

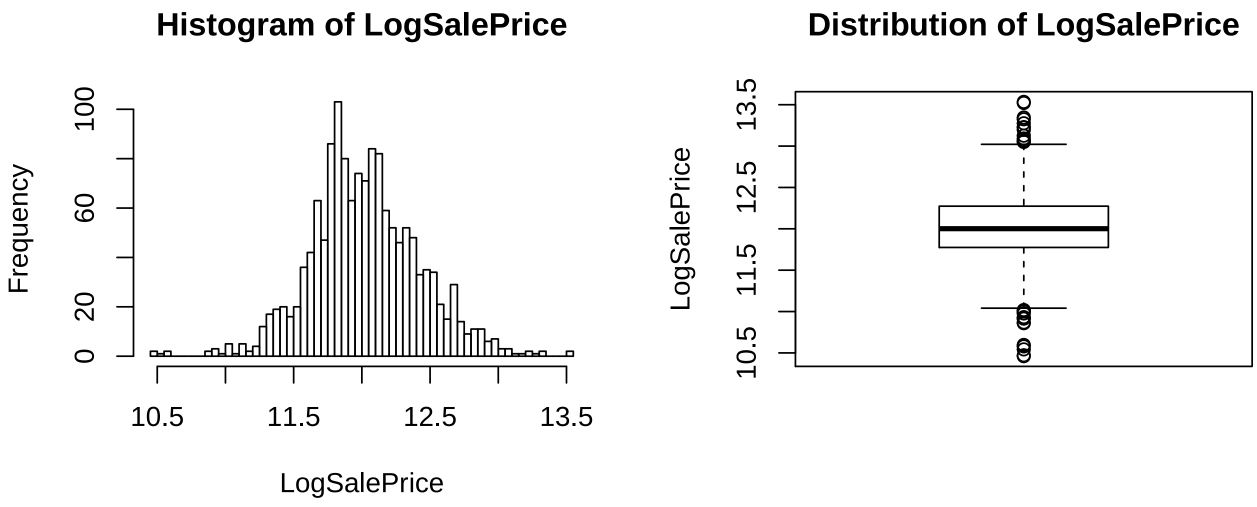
Sale Price distribution

```
hist(x = data$SalePrice, breaks = 100)
```



The `SalePrice` distribution is skewed (i.e. most house are sold at intermediate prices but a significant number of houses are very expensive). We'll scale prices by applying a log transformation before performing regression.

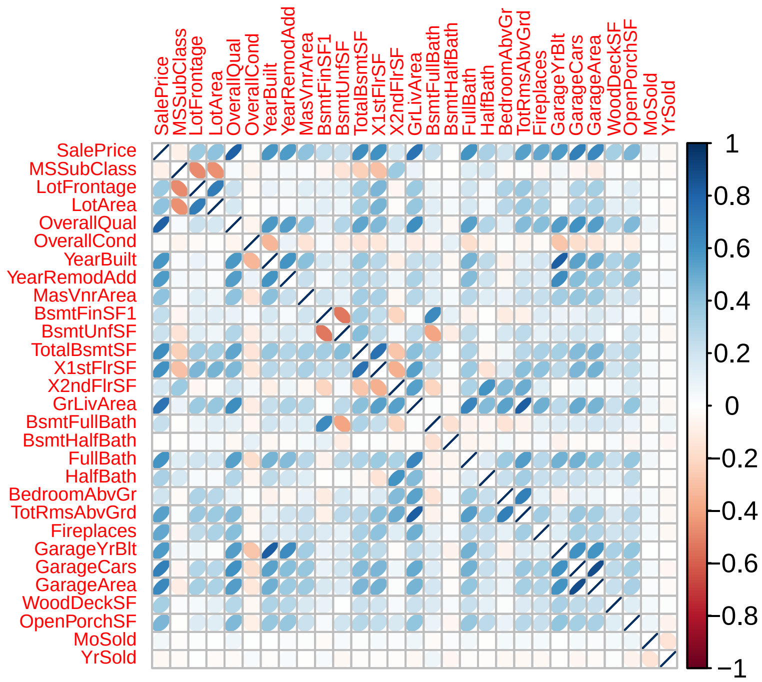
```
data$SalePrice <- log(data$SalePrice)
par(mfrow = c(1,2))
hist(x = data$SalePrice, breaks = 50, xlab = "LogSalePrice", main = "Histogram of LogSalePrice")
boxplot(data$SalePrice, ylab = "LogSalePrice", main = "Distribution of LogSalePrice")
```



Correlation analysis of numerical variables

To plot linear correlation relationships between variables, we could use pair plots or correlation plots. Considering the large number of variables, let's plot the more visual `corrplot`:

```
M <- cor(data[quantitative], method = c("pearson", "kendall", "spearman"))
corrplot(M, method='ellipse', tl.cex=0.6)
```



The variables less correlated to SalePrice appear to be: `MSSubClass`, `OverallCond`, `BsmtFinSF1`, `BsmtUnfSF`, `X2ndFlrSF`, `BsmtFullBath`, `BsmtHalfBath`, `HalfBath`, `BedroomAbvGr`, `MoSold`, `YrSold`. `MSSubClass` is a number corresponding to the house type, with higher numbers not representing better quality (no order), therefore we should better cast this variable as categorical. For the other ones, we will try a model with and without them to check if removing variables less correlated to SalePrice improves predictions.

```
data$MSSubClass <- as.factor(data$MSSubClass)
quantitative = names(dplyr::select_if(data, is.numeric))
qualitative = names(dplyr::select_if(data, is.object))
```

We also observe multicolinearity between our numerical covariates. Let's take a deeper look by computing VIF (Variance Inflation Factors), which measures how much the variance of a regression coefficient is inflated due to multicollinearity in the model. The smallest possible value of VIF is one (absence of multicollinearity). As a rule of thumb, a VIF value that exceeds 5 or 10 indicates a problematic amount of collinearity (James et al. 2014). Multicollinearity implies that the information that this variable provides about the response is redundant in the presence of the other variables.

```
VIF(data[quantitative] %>% dplyr::select(-SalePrice), data$SalePrice)
```

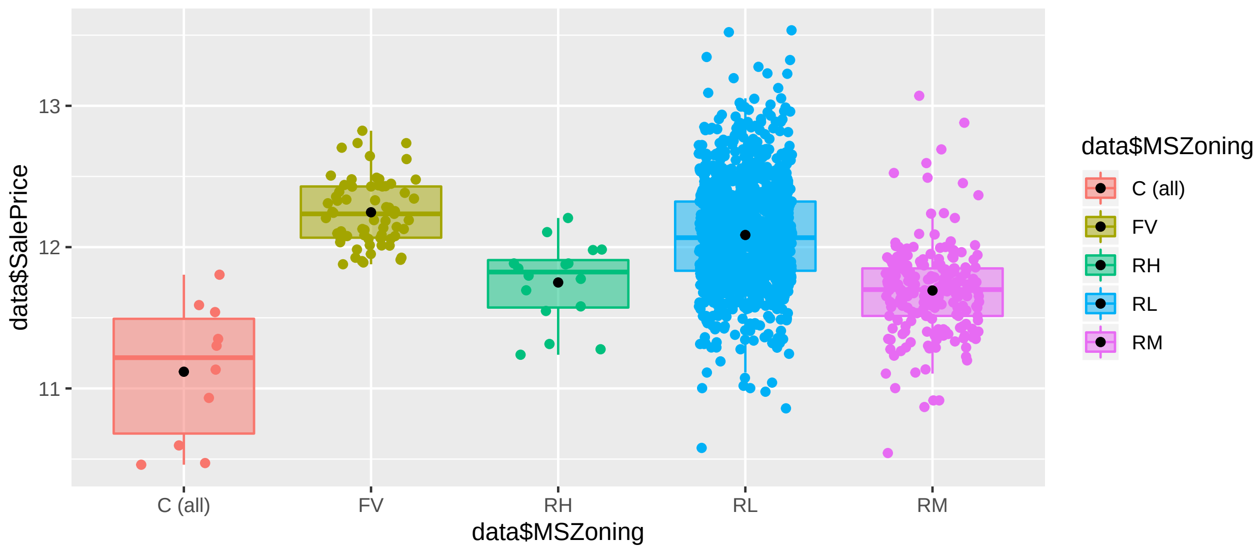
```
##
##  LotFrontage      LotArea  OverallQual  OverallCond    YearBuilt YearRemodAdd
##      2.19008      2.26466      3.15670      1.50353      4.97836      2.37319
##   MasVnrArea   BsmtFinSF1    BsmtUnfSF  TotalBsmtSF    X1stFlrSF    X2ndFlrSF
##      1.41287      3.53495      3.96039      5.54290     13.55890     12.75480
##    GrLivArea BsmtFullBath BsmtHalfBath     FullBath     HalfBath BedroomAbvGr
##     22.38990      2.16030      1.15419      2.83787      2.29295      2.39606
## TotRmsAbvGrd   Fireplaces  GarageYrBlt   GarageCars   GarageArea   WoodDeckSF
##      4.51449      1.54744      4.59220      5.66850      5.64882      1.23414
##  OpenPorchSF       MoSold       YrSold
##      1.43146      1.03972      1.04443
##
##   Mean: 4.34014
```

Here, the mean VIF is below 5, therefore there doesn't seem to be a significant problem with multicolinearity. However 3 variables (`X1stFlrSF`, `X2ndFlrSF` and `GrLiveArea`) have VIF above 10. We will try a model without these varibales to check if it performs better (- knowing that `GrLiveArea` corresponds to living area surface: it sounds very relevant to explain sale prices! We'll see in the modeling part).

Categorical variables

For categorical variables, we could plot boxplots to visualize the distribution of saleprice based on different modalities. We could also print `SalePrice` means per modality. See example with `MSZoning` below, showing that `SalePrice` is impacted by the zoning classification of the sale (different means per category). As we have a lot of categorical variables, we will not conduct it for all variables, but we will select the most relevant ones using anova and step by step selection in the next section.

```
ggplot(data, aes(x = data$MSZoning, y = data$SalePrice, colour=data$MSZoning, fill=data$MSZoning)) +
  geom_boxplot(alpha=0.5, outlier.alpha=0) + geom_jitter(width=0.25) +
  stat_summary(fun.y=mean, colour="black", geom="point")
```



```
Tmean=tapply(data$SalePrice,list(MSZoning=data$MSZoning),mean);Tmean
```

```
## MSZoning
##  C (all)       FV       RH       RL       RM
## 11.11826 12.24662 11.74984 12.08589 11.69289
```

Let's now move on to building and evaluating different regression models.

III. Modeling and Diagnostics

Linear regression models

Train/Test Split

First, we split the data into train and test sets.

```
set.seed(2019)
inTrain <- createDataPartition(y = data$SalePrice, p = 0.85, list = FALSE)
train <- data[inTrain, ]
test <- data[-inTrain, ]
```

Full Model

As a first try, let's put all variables into a simple linear regression model.

```
reg1 <-lm(SalePrice  ~., data=data)
summary(reg1)
```

Output:
Residual standard error: 0.1018 on 1224 degrees of freedom
Multiple R-squared: 0.9455, Adjusted R-squared: 0.9351
F-statistic: 90.45 on 235 and 1224 DF, p-value: < 2.2e-16

Model evaluation:

```
predicted = predict(reg1, test %>% dplyr::select(-SalePrice))
predicted_train = predict(reg1, train %>% dplyr::select(-SalePrice))
```

```r
residuals_train=train$SalePrice - predicted_train
residuals =test$SalePrice - predicted

RMSE_train = sqrt(mean(residuals_train^2))
RMSE = sqrt(mean(residuals^2))

y_test_mean = mean(test$SalePrice)
tss = sum((test$SalePrice - y_test_mean)^2)
rss = sum(residuals^2)
rsq = 1 - (rss/tss)
radjust=1-(1044*rss)/((1045-260)*tss)
radjust2=1-(1-rsq)*(1044/(1045-259-1))

message("train RMSE: ", RMSE_train)
```

```
## train RMSE: 0.0929705857294483
```

```r
message("test RMSE: ", RMSE)
```

```
## test RMSE: 0.0943606034650498
```

```r
message("test adjusted R square: ", radjust)
```

```
## test adjusted R square: 0.922375732903414
```

This first model is already performant, but many variables aren't considered as significant based on t-test in the model summary output. Potential overfitting occurs when including all variables. Let's select the most relevant variables.

Model based on EDA correlation analysis

When removing the variables spotted in EDA as multicolinear or less related to SalePrice, we obtained the following results.

```r
reg2 <-lm(SalePrice  ~ . - OverallCond - BsmtFinSF1 - BsmtUnfSF - X2ndFlrSF -
            BsmtFullBath - BsmtHalfBath - HalfBath - BedroomAbvGr - MoSold -
            YrSold - X2ndFlrSF, data=data)
summary(reg2)
```

Output:
train RMSE: 0.0972971292224848
test RMSE: 0.103917924636335
test adjusted R square: 0.90585505133741

This model yields a slightly higher mean square error on the test set and a slighlty lower adjusted R square than the full model. Therefore we decided not to discard the covariates we had spotted and to try other feature selection methods.

Feature selection with AIC

We performed AIC (using the forward, backward and both method) to select the most significant features. We chose to use AIC and not BIC because of the high number of covariates. The 3 methods yielded very similar results, the backward method being slightly more performant. It selected 40 covariates:

```r
#stepAIC(reg1, ~., trace=TRUE, direction=c("backward"))
```

```r
reg3 <- lm(formula = SalePrice ~ MSSubClass + MSZoning + LotArea + LotConfig +
    LandSlope + Neighborhood + Condition1 + Condition2 + OverallQual +
    OverallCond + YearBuilt + YearRemodAdd + RoofMatl + Exterior1st +
    MasVnrType + Foundation + BsmtQual + BsmtExposure + BsmtFinSF1 +
    BsmtUnfSF + TotalBsmtSF + Heating + HeatingQC + CentralAir +
    GrLivArea + BsmtFullBath + FullBath + HalfBath + KitchenQual +
    Functional + Fireplaces + GarageType + GarageYrBlt + GarageCars +
    GarageArea + GarageQual + GarageCond + WoodDeckSF + SaleType +
    SaleCondition, data = data)
```

Output:
train RMSE: 0.0955541308687478
test RMSE: 0.0976432548916017
test adjusted R square: 0.916880946456593

This new model is equivalent to the full model in terms of mean square error and adjusted R2, but we chose to keep it as it includes less covariates, which are all significant.

Model including impact of interaction between regressors

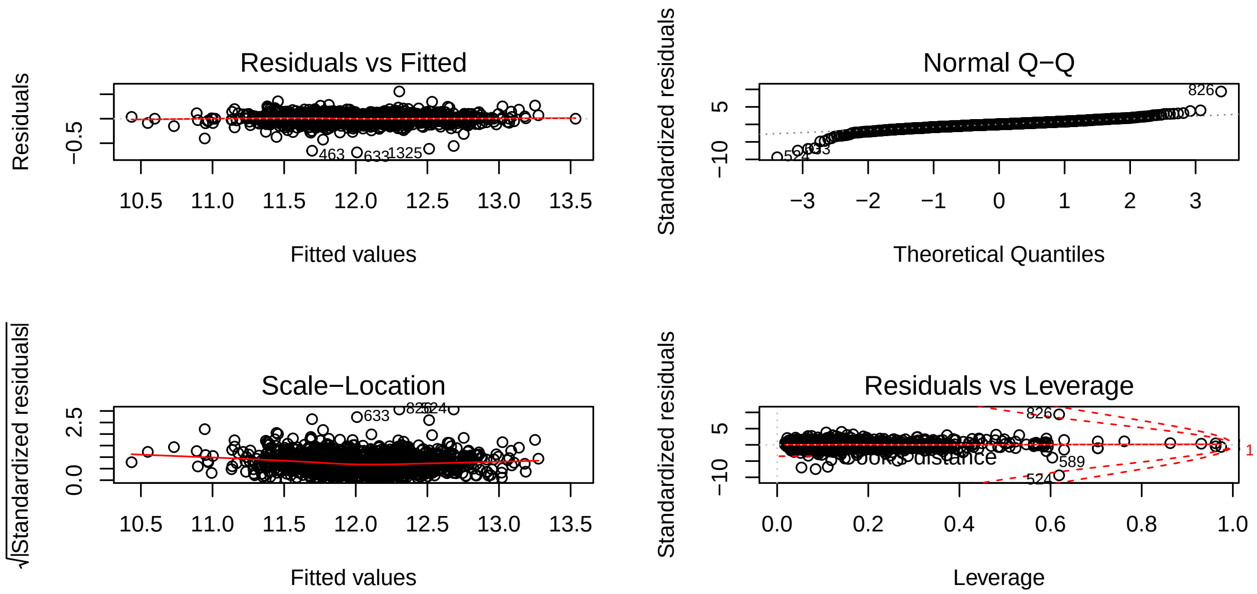
We finally used ANCOVA (as we have a mix of numerical and categorical variables) to assess the impact of interaction between regressors on the predictions. We tested the most relevant combinations based on our understanding of the different variables. When we spotted interactions impacting significantly our predictions, we dropped the corresponding single variables and kept the interaction in order to obtain a more efficient model. The best model we could obtain this way was finally:

```r
reg4 <- lm(SalePrice ~ MSSubClass * MSZoning + LotArea * Neighborhood +
    LotConfig + LandSlope + Condition1 + Condition2 + OverallQual * GrLivArea +
    OverallCond * YearBuilt + RoofMatl * Exterior1st +
    Foundation + BsmtQual * BsmtExposure + YearRemodAdd + MasVnrType + BsmtFinSF1 +
    TotalBsmtSF + BsmtUnfSF + Heating + HeatingQC + CentralAir +
    BsmtFullBath + HalfBath + KitchenQual + MasVnrType +
    Fireplaces + GarageYrBlt + GarageCars + Functional +
    GarageYrBlt:GarageCars:GarageArea:GarageQual:GarageCond + SaleType +
    SaleCondition, data = data)
anova(reg4)
```

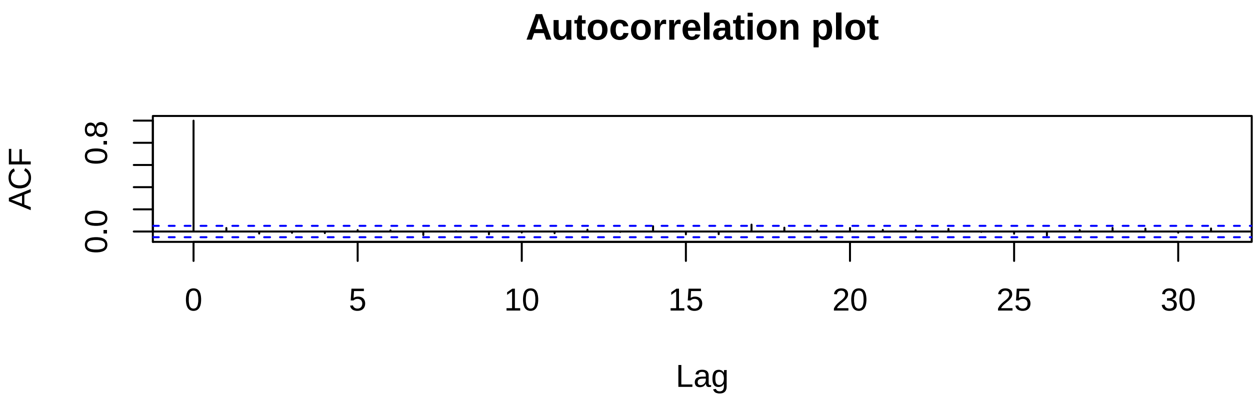
Output:
train RMSE: 0.0879503124367945
test RMSE: 0.0908861822990065
test adjusted R square: 0.927986848769291

Residuals Analysis

```r
layout(matrix(c(1,2,3,4,5), 2, 2, byrow = TRUE))
plot(reg4)
```



```r
acf(residuals(reg4), main = "Autocorrelation plot")
```



- Errors are approximately centered around 0.
- The QQ-plot and the Shapiro-Wilk test below reveal that the Gaussianity assumption isn't totally met, but we already performed logtransformation of the output variable and this assumption isn't the most necessary one, so for now we'll stick with our model as it is.
- Homoscedasticity seems questionable, so we ran a Breush-Pagan test (see below). The p-value is above 0.05, so we do not reject the homescedasticity assumption.
- The autocorrelation plot tells us that the postulate of uncorrelated residuals is reasonable since none of the bars after the first bar exceeds the threshold, which is confirmed by the Durbin-Watson test below (p-value above 0.05).

```r
ncvTest(reg4) # Breush-Pagan test for homscedasticity
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 1.569393, Df = 1, p = 0.2103
```

```
shapiro.test(residuals((reg4))) # Shapiro-Wilk test for Gaussianity
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals((reg4))
## W = 0.91835, p-value < 2.2e-16
```

```
durbinWatsonTest(reg4) # Durbin-Watson test for autocorrelation
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      0.03192335      1.935951   0.218
##  Alternative hypothesis: rho != 0
```

The last diagnostic plot, Residuals vs. Leverage, revealed no point with a Cook distance above 1. However, diagnostice plots show a few potential outliers that we will study in the next section.

Search for outliers

Let's check if the potential outliers should be removed from our model by looking at their Bonferroni p-values:

```
outlierTest(reg4)
```

```
##       rstudent unadjusted p-value Bonferroni p
## 826   9.738926         1.2073e-21   1.7229e-18
## 524  -9.738926         1.2073e-21   1.7229e-18
## 633  -7.654974         3.8814e-14   5.5388e-11
## 463  -7.138506         1.6056e-12   2.2913e-09
## 1325 -6.904710         8.0277e-12   1.1456e-08
## 969  -4.934430         9.1368e-07   1.3038e-03
## 1454 -4.778260         1.9802e-06   2.8258e-03
## 1433 -4.227709         2.5348e-05   3.6171e-02
```

We found 8 outliers with Bonferroni p-values below 0.05. We tried to compare our model including these outliers and a model without the outliers. The model without outilers yielded significantly best results, so we will run final diagnostic plots and select it as our final model.

IV. Final model

```
data_out_removed = data[-c(826,524,633,463,1325,969,1454,1433),]

final_reg <- lm(SalePrice ~ MSSubClass * MSZoning + LotArea * Neighborhood +
    LotConfig + LandSlope + Condition1 + Condition2 + OverallQual * GrLivArea +
    OverallCond * YearBuilt + RoofMatl * Exterior1st +
    Foundation + BsmtQual * BsmtExposure + YearRemodAdd + MasVnrType + BsmtFinSF1 +
    TotalBsmtSF + BsmtUnfSF + Heating + HeatingQC + CentralAir +
    BsmtFullBath + HalfBath + KitchenQual + MasVnrType +
    Fireplaces + GarageYrBlt + GarageCars + Functional +
    GarageYrBlt:GarageCars:GarageArea:GarageQual:GarageCond + SaleType +
    SaleCondition, data = data_out_removed)
```

Output (Final Model):
Residual standard error: 0.08381 on 1229 degrees of freedom
Multiple R-squared: 0.962, Adjusted R-squared: 0.9551
F-statistic: 140.1 on 222 and 1229 DF, p-value: < 2.2e-16

train RMSE: 0.0765191020162539
test RMSE: 0.0804040574197461
test adjusted R square: 0.947069947387763

This final model yields better results in terms of adjusted R sqaure and mean square error than all other models.

Based on new diagnostic plots and tests, postulates are still validated - except for the Gaussianity assumption that still isn't perfectly met, but as this model is the best-performing one we will consider it as our final model.

Confidence intervals

As a last step in our analysis, let's compute confidence intervals for our sale price estimates:

```
ICconf = predict(final_reg, interval = "confidence", level = 0.95)
head(ICconf)
```

```
##        fit      lwr      upr
## 1 12.24018 12.21619 12.26417
## 2 12.11597 12.03174 12.20020
## 3 12.29096 12.25860 12.32332
## 4 12.04390 11.98595 12.10184
## 5 12.58497 12.54286 12.62708
## 6 11.88234 11.78067 11.98401
```

Other Models we tried

In this additional sub-section we shortly present some of the other models we tried in order to predict the Sale prices - none of them yielded better result than our linear models, so we did not keep them.

XGBoost

Before performing `XGBoost`, we encoded the data with one-hot encoding, we split our dataset between a train and a test set, and we selected parameters through `GridSearchCV`.

Model training

```
params <- list(booster = "gbtree",  objective = "reg:squarederror", eta=0.1,
               gamma=0, max_depth=3, min_child_weight=1, subsample=1, colsample_bytree=0.9)

xgb_model <- xgb.train( params = params, data = X_train, nrounds = 2000,
                        nfold = 5, showsd = T, stratified = T, print.every.n = 10,
                        early.stop.round = 20, watchlist = list(val=X_test, train=X_train),
                        maximize = F)
```

Output:
RMSE: 0.108650317957415
adjusted R square: 0.897268082348635

Lasso

```
set.seed(1012)
lasso=train(SalePrice~., train, method='glmnet',
            tuneGrid=expand.grid(alpha=1,lambda=seq(0.01,0.1,length=10)))
```

Output:
train RMSE: 0.129390145987898
test RMSE: 0.113431110315135
test adjusted R square: 0.887762307368666

Ridge

```
set.seed(1012)
ridge=train(SalePrice~.,train, method='glmnet',
            tuneGrid=expand.grid(alpha=0,lambda=seq(13,15,length=10)))
```

Output:
train RMSE: 0.300306365242122
test RMSE: 0.298312154569017
test adjusted R square: 0.223723817913419

→ All these models performed less well than our custom linear regression.

V. Discussion

The main difficulty we found in this analysis was to deal with a large number of variables, and identify which ones we more relevant to predict sale prices. For future improvement, we could for example talk with an industry expert, who could provide valuable insights regarding how to select the most relevant variables. Such industry knowledge could greatly improve our understanding of the data and help us build a more performant model.

Our final model doesn't significantly overfit, but to improve robustness we could also go through an additional cross-validation step.

Last, residuals in our final model don't exactly follow a Gaussian distribution. This doesn't invalidate our model, but it implies that it might not be the best one to understand our data. Maybe we miss here some relationships between the predictors and the outcome (maybe non-linear relationships that we don't capture?), or maybe we did not include some other variables that could play an important role, or maybe didn't identify some bias in our data… Our model still performs relatively well at predicting real estate sale prices, so we will stick with it for now - but this project shows that dealing with a large amount of data always leaves room for improvement!