House Prices Analysis

Gregoire Mansio

28/07/2021

This data-science project has been conducted for the EdX Capstone Part 2

Introduction

This pdf document is a data analysis of house prices in Ames, Iowa, USA. The dataset has been gathered by Dean De Cock and is available on Kaggle at the following link: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

Otherwise, the provided script automatically download the required files from my github page and load them into the Rstudio environment.

The purpose of the present data science analysis is to predict a continuous outcome: house sale prices. We first make use of visualization tools before applying two different models, a standard linear regression and then a random forest algorithm.

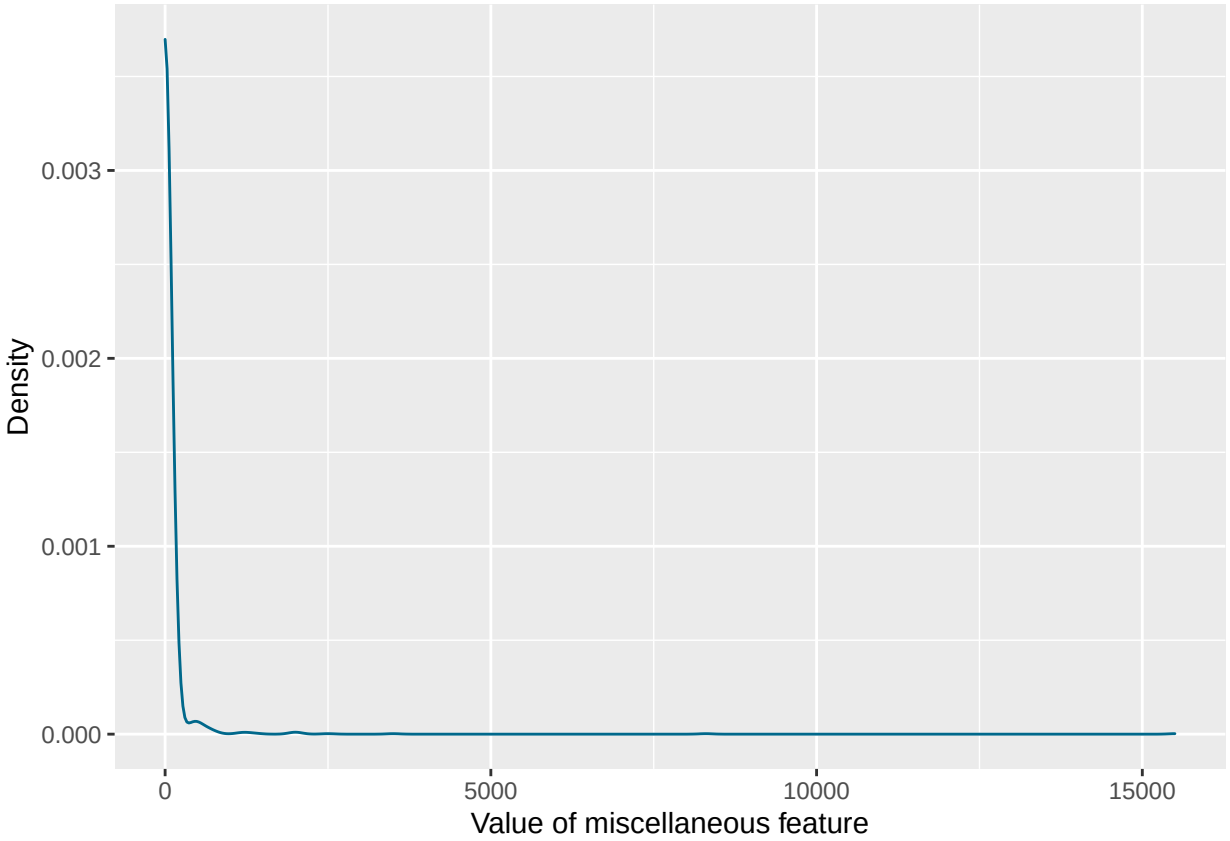
The dataset contains 79 variables and 2919 observations that have been collected between 2006 and 2010, time span of sells. Because the number of variables is already large, we won't create new variables from combinations of existing ones.

Data exploration, analyses and methods

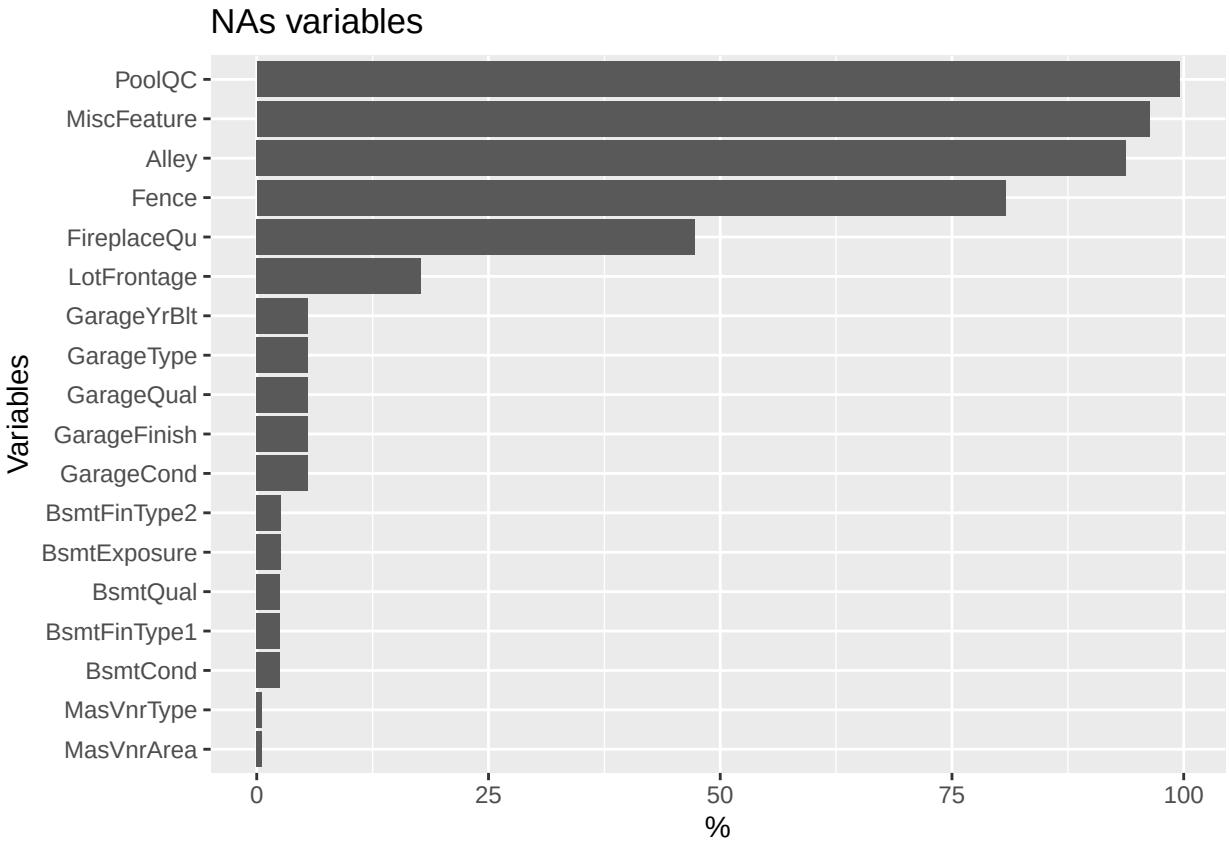
Visualisation

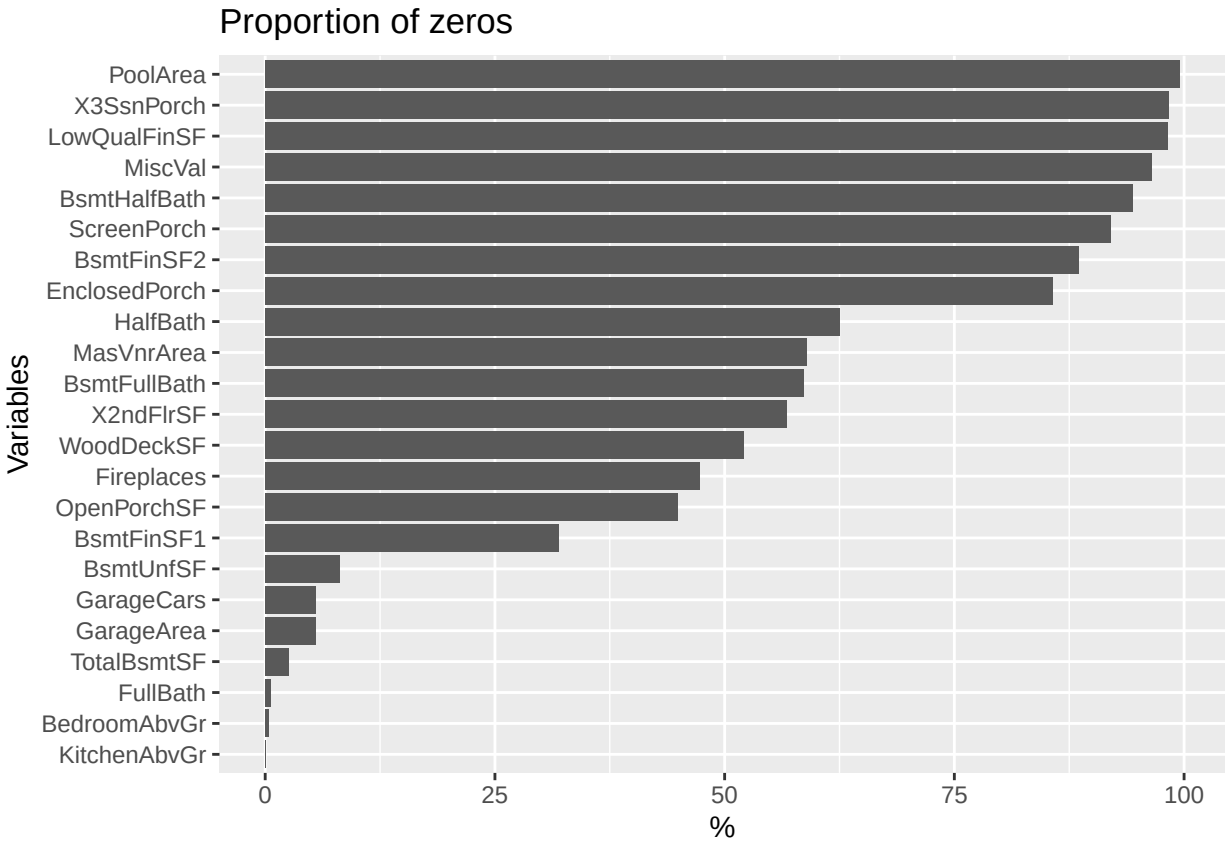
A considerable number of features are of character type, and some won't be used for this first analysis. Moreover, many of those characters features, and several numerical ones have a single value weight very high, which means that they have a very low variance, and therefore aren't really usefull to do modelisation. I choose to exclude them completely.

```
zerovar_names <- nearZeroVar(train_set, names=TRUE) # Checking for low variance variables
zerovar_index <- nearZeroVar(train_set)

miscval_dens <- ggplot(train_set, aes(x =train_set$ MiscVal)) +
  geom_density(colour = "deepskyblue4") +
  xlab("Value of miscellaneous feature") +
  ylab("Density")

miscval_dens
```



The above density graph shows for example that the 'MiscVal' feature has almost only 0 as value. Also, we have some features with many missing values. We will exclude those as well. The second histogram below also shows that the variance of PoolArea can not be useful for us as almost 100% of the observations are zeros. This coincide with what the nearZeroVar function previously found.


NAs variables
PoolQC
MiscFeature
Alley
Fence
FireplaceQu
LotFrontage
GarageYrBlt
GarageType
GarageQual
GarageFfinish
GarageCond
BsmtFinType2
BsmtExposure
BsmtQual
BsmtFinType1
BsmtCond
MasVnrType
MasVnrArea
Variables
0
25
50
75
100
%



Transformation

First thing to do is remove the variables with too many missing values and those with a really low variance

After that, we have to take care of other variables presenting NAs in acceptable proportions. This is a very tedious operation that requires an inspection of features one by one. But this operation can also be automated with a degree of approximation. For example, I created a function to find the most common value for a given variable, and then used this value to replace the NAs.

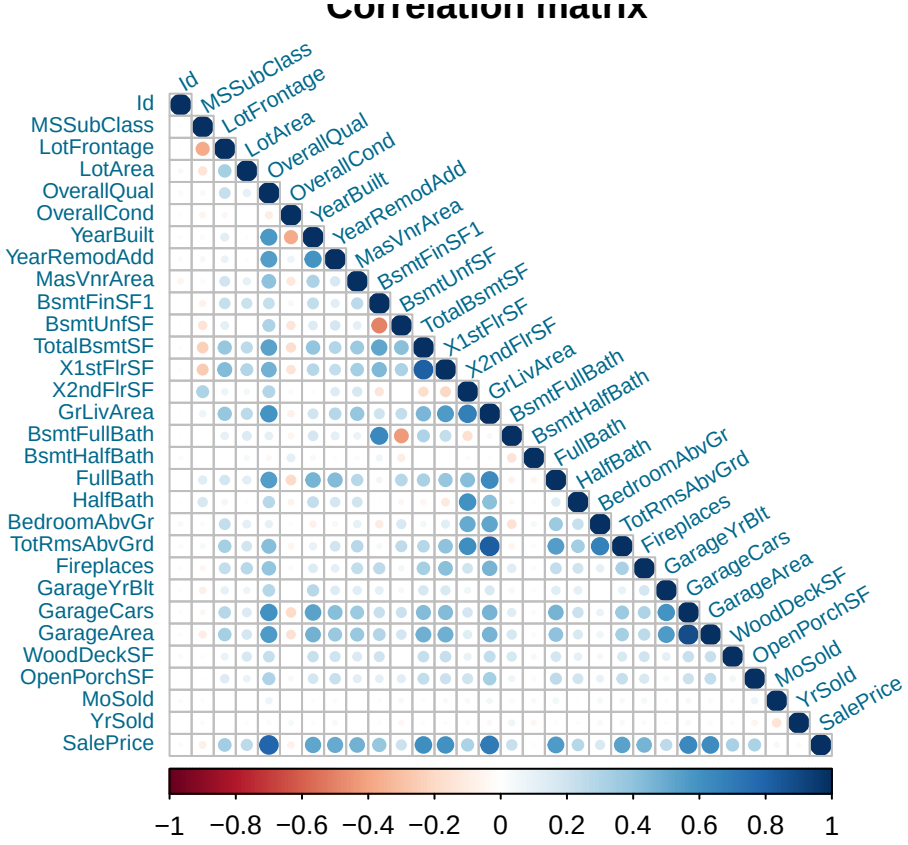
```
Mostcommon <- function(x) {
  sort(table(x), decreasing = TRUE)[1]
}

train_set[is.na(KitchenQual), KitchenQual := Mostcommon(train_set$KitchenQual)]
```

Other operations can be done to recode characters variables with levels into ordinal factor variables that are usable in models. Here a snippet of the code I used to do so. It is quite convenient to transform character variables effortlessly, but some problems can arise because of this operation. For example it seems that there is a typo in the 'Electrical' variable, which requires further inspection.

```
train_set[,KitchenQual:=ordered(KitchenQual, levels = c("Po","Fa","TA","Gd","Ex"))]
train_set[,GarageFinish:=ordered(GarageFinish, levels = c("None","Unf","RFn","Fin"))]
train_set[,ExterQual:=ordered(ExterQual, levels = c("Po","Fa","TA","Gd","Ex"))]
```

Now that we have cleaned the datasets, let's have some final looks at the data and build the models.

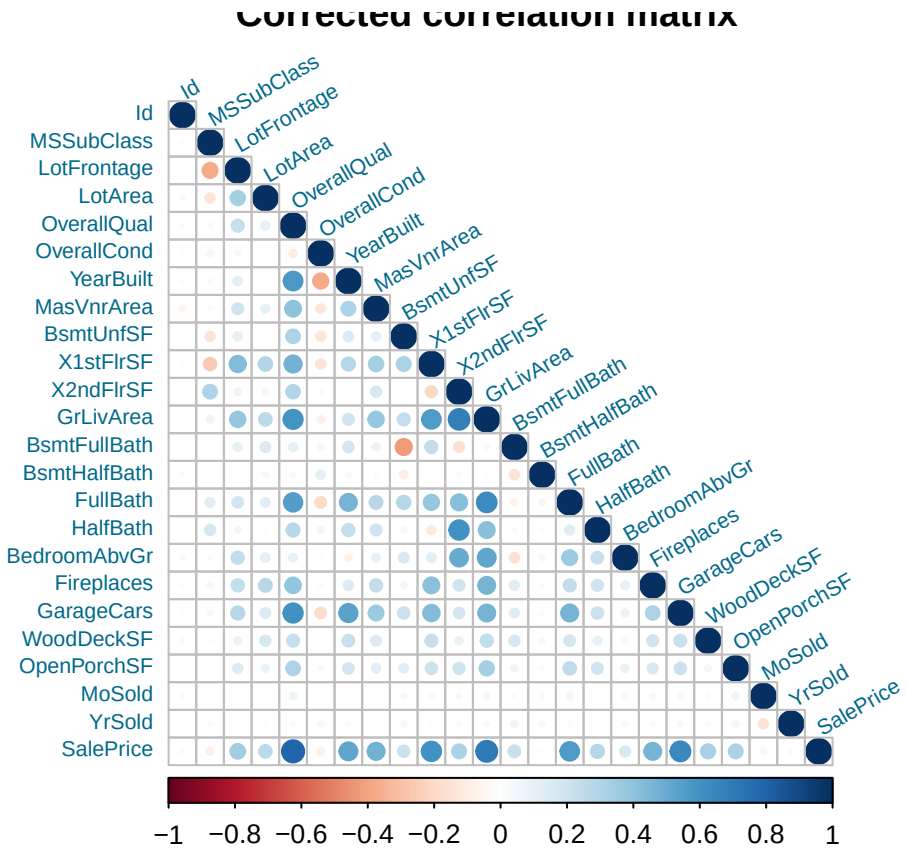
Correlations



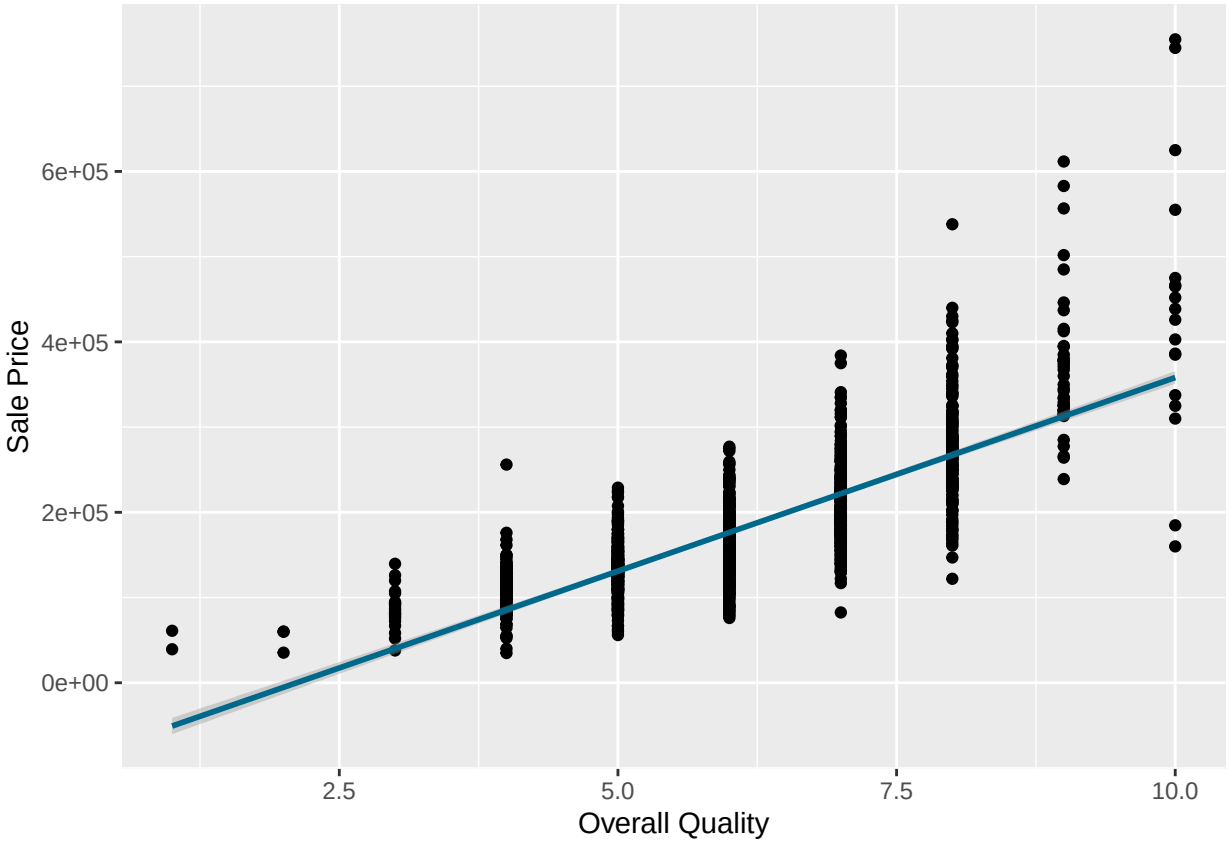
We can see on the plot below that some variable are highly correlated like 'Total basement squared ft' and 'First floor squared ft' (which make sense) and that they are almost perfectly colinear, which is absolutely not recommended in regression. I choose to purely remove one of two variables when they show multicolinearity.

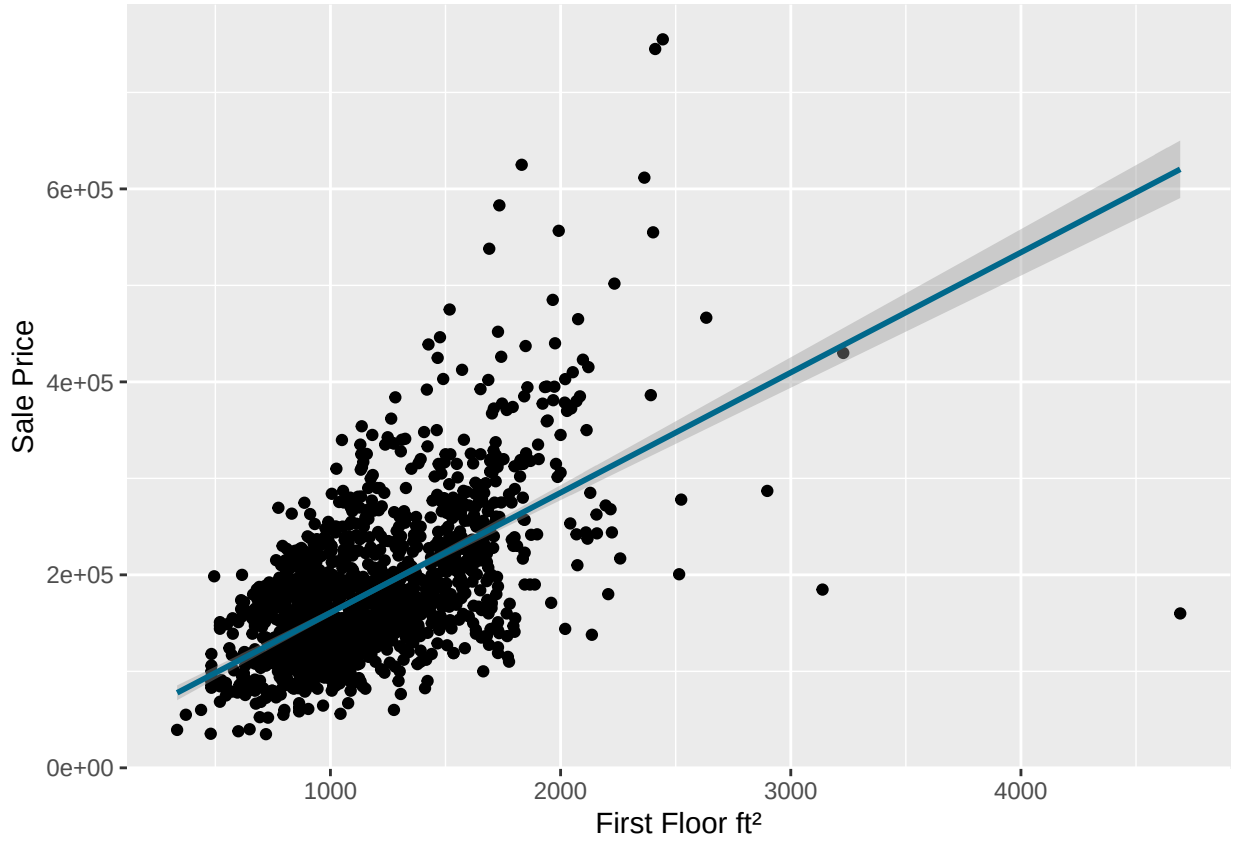
```
train_set <- train_set %>% select(-TotRmsAbvGrd, -TotalBsmtSF, -GarageArea, -YearRemodAdd, -GarageYrBlt
test_set <- test_set %>% select(-TotRmsAbvGrd, -TotalBsmtSF, -GarageArea, -YearRemodAdd, -GarageYrBlt, -
```

After this final removal operation, we can take a last look at the corrected correlation matrix which synthesizes the most important variables, presumed to be the most significant ones in the upcoming algorithms.



Moreover, plotting 'Sale Price' against 'Overall Quality', as well as against 'First floor Squared ft' indeed indicates that those features are part of the most important ones, and that the model should perform well on those features.


Sale Price
6e+05
4e+05
2e+05
0e+00
2.5
5.0
7.5
10.0
Overall Quality



First model and results: Regression

Before any modeling, it is important to notice that we don't have any SalePrice observation in the test_set downloadable on Kaggle. Therefore, to test the performance of our algorithm, we have to split the train_set into a train_subset and a test_subset to have an idea of the performance of the algorithms and choose the final one.

```
train_index <- createDataPartition(train_set$Id, times = 1, p = 0.9, list=FALSE) # First, don't forget
train_subset <- train_set[train_index,]
test_subset <- train_set[-train_index,]
```

```
Linear_matrix <- train_subset %>% dplyr::select(where(is.numeric)) # The first model is a simple regres
lm_fit <- lm(data = Linear_matrix, formula = SalePrice ~ .)
summary(lm_fit) # R² of 0.80 is a decent but perfectible score.
```

```
##
## Call:
## lm(formula = SalePrice ~ ., data = Linear_matrix)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -435276  -17818   -1113   14092  284196
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.002e+06  1.533e+06   0.654  0.51352
```

```
## Id           -1.699e+00  2.367e+00  -0.718  0.47303
## MSSubClass   -2.276e+02  2.796e+01  -8.141 9.15e-16 ***
## LotFrontage  -7.688e+01  5.522e+01  -1.392  0.16409
## LotArea       4.248e-01  1.064e-01   3.992 6.91e-05 ***
## OverallQual   1.957e+04  1.233e+03  15.869  < 2e-16 ***
## OverallCond   5.490e+03  9.980e+02   5.501 4.56e-08 ***
## YearBuilt     3.693e+02  5.546e+01   6.658 4.09e-11 ***
## MasVnrArea    3.610e+01  6.304e+00   5.727 1.27e-08 ***
## BsmtUnfSF    -1.542e+00  3.066e+00  -0.503  0.61498
## X1stFlrSF     1.687e+01  2.074e+01   0.814  0.41599
## X2ndFlrSF     1.014e+01  2.071e+01   0.490  0.62440
## GrLivArea     4.662e+01  2.026e+01   2.301  0.02154 *
## BsmtFullBath  1.448e+04  2.550e+03   5.678 1.69e-08 ***
## BsmtHalfBath  2.660e+03  4.390e+03   0.606  0.54461
## FullBath      6.159e+03  2.995e+03   2.057  0.03991 *
## HalfBath     -7.437e+02  2.863e+03  -0.260  0.79508
## BedroomAbvGr -7.561e+03  1.600e+03  -4.727 2.53e-06 ***
## Fireplaces    4.498e+03  1.869e+03   2.406  0.01625 *
## GarageCars    1.117e+04  1.836e+03   6.081 1.57e-09 ***
## WoodDeckSF    2.507e+01  8.584e+00   2.920  0.00356 **
## OpenPorchSF   3.891e+00  1.613e+01   0.241  0.80944
## MoSold       -2.099e+02  3.737e+02  -0.562  0.57452
## YrSold       -8.954e+02  7.601e+02  -1.178  0.23905
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 35950 on 1292 degrees of freedom
## Multiple R-squared:  0.8029, Adjusted R-squared:  0.7994
## F-statistic: 228.8 on 23 and 1292 DF,  p-value: < 2.2e-16
```

The linear model obtains gives the most importance to 9 variables, among which 'Overall Quality'. Surprisingly the surface of the floors isn't significant, but their effect can be captured by 'LotArea', representing the overall surface of the lot. It could be interesting to assess whether or not the 'LotArea' is positively correlated with the house surface, which would invite us to remove some of those overlapping features as we did before.

Still, the performance of the model is quite good, as we end up with an RMSE of 0.1459. Let's now try the random forest.

```
## [1] 0.1458844
```

Second model and results: Random forest

Thanks to the caret package, the random forest model is very simple to implement as long as a proper data cleaning has been made, which is the case.

```
random_forest <- randomForest(SalePrice ~ ., data = train_subset)

prediction_rf <- predict(random_forest, test_subset)
RMSE(log(test_subset$SalePrice), log(prediction_rf)) # 0.105 is now a very interesting result!
```

```
## [1] 0.1050817
```

The RMSE is now down to 0.104 which is very good for a 1459 observations dataset! I won't present the last use of the random forest model (on the full train_set) because it's exactly the same methodology as the one just above, and because we aren't able to test the precision of the full model. Indeed, the test_set has no SalePrice indicated.

Conclusion

The dataset is of good quality and provides us with many options. Working only with numeric variables, it already shows significant results, even if it requires some work to be really effective. Thanks to our two simple algorithms, we are able to predict house prices with an impressive precision. The linear regression model already shows a good R^2 and most importantly a good RMSE when used for prediction. Moreover, the random forest improve the RMSE by approx 30% which is unexpected but very conclusive.

Further improvements could be made. First, we could use the character variables like Neighborhood which surely influences SalePrice, by recoding those character variables with hypotheses definitions and numerically-leveled groups. Also, we could improve the linear regression with regularization and cross-validation in order to better improve the precision power. Finally what could be of particular interest is a XGBoost algorithm as the random forest is particularly effective, and because it seems to be a very popular and effective ML technique.