Machine Learning Engineer Nanodegree

Capstone Project

Guilherme Augusto Kater Marson
October 29st, 2018

I. Definition

This project will define a model to better evaluate the sell price of houses in Ames, Iowa. The data can be found here. It's important to mention that this dataset is part of a Kaggle competition, so it will be simple to compare the model performance to other model's performance.

Project Overview

There are three values for any home on the market: What the seller thinks it's worth, what the buyer thinks it's worth and what a professional appraiser will think it's worth. The seller wants as much money as possible for his house. The buyer wants to pay as low as possible and the professional appraiser will gather some data from the house, together with his background experience to come with a price. Seller and buyer usually trust the professional appraiser, but how accurate is his price? What if he has personal interest in completing this deal? Does he have all the data and knowledge to accurately evaluate that specific house? And the last point: isn't he biased?

Problem Statement

There are some ways to evaluate houses but the most used is the sales comparison method, that compares the house to be sold with similar properties in the same locality. With this in mind and assuming that there is a database with explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, it's possible to create a model to accurately evaluate house's price. The model will try to regress the explanatory variables to the fairest price possible, removing from the equation the integrity and knowledge of the professional appraiser.

Metrics

The root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSE represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset and not between datasets, as it is scale-dependent. RMSE is always non-negative, and a value of 0 (almost never achieved in practice) would indicate a perfect fit to the data. In general, a lower RMSE is better than a higher one. However, comparisons across different types of data would be invalid because the measure is dependent on the scale of the numbers used. RMSE is the square root of the average of squared errors. The effect of each error on RMSE is proportional to the size of the

squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers. The logarithm will be applied to both observed and predicted sales price so that the error predicting expensive and cheap houses will affect the result equally.

II. Analysis

Data Exploration

There are 2 datasets available to solve this problem. The first one is the train dataset and the second is the test dataset. The mais difference of them is that the test dataset does not have the target variables (SalePrice) because it's defined to be used as a submission to Kaggle website to evaluate the model.

Train Dataset

- Rows: 1460
- Columns: 81
- Data Types: 38 numerical columns and 43 textual columns #### Test Dataset
    - Rows: 1459
- Columns: 80
- Data Types: 37 numerical columns and 43 textual columns

Data Sample

The dataset has more than 80 columns, so instead of showing all columns in this document, we will present only the first 10.

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
194	160	RM	24.0	2522	Pave	NaN	Reg	Lvl	AllPub
1157	80	RL	85.0	9350	Pave	NaN	Reg	Lvl	AllPub
825	20	FV	81.0	11216	Pave	NaN	Reg	Lvl	AllPub
993	60	RL	80.0	9760	Pave	NaN	Reg	Lvl	AllPub
27	20	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub

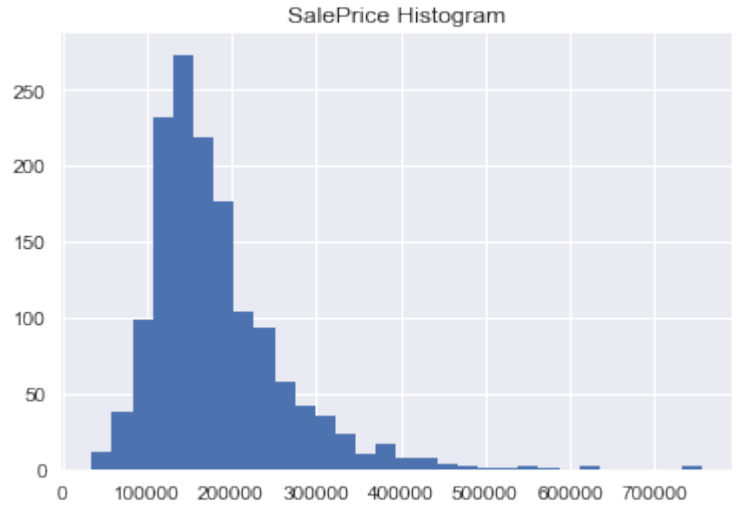
For a more detailed description on the variables, there is a file named variable_description.xlsx that holds the following information about each variable:

- Name: the name of the variable
- Type: numerical or categorical (textual)
- Convert to Categorical: If it´s a good idea to convert the numerical to a categorical variable
- Description: detailed description about the variable
- Expectation: my personal expectation about the variable. It will be used to guide the data exploration priorization

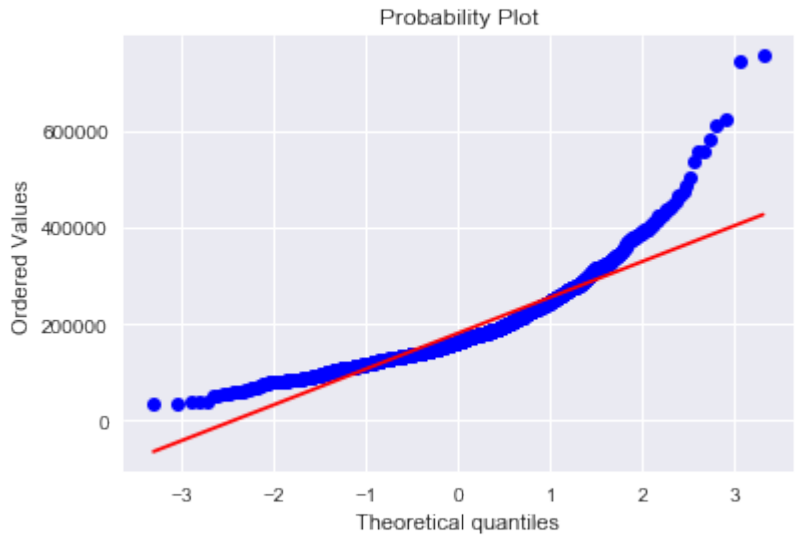
Exploratory Visualization

Target Analysis

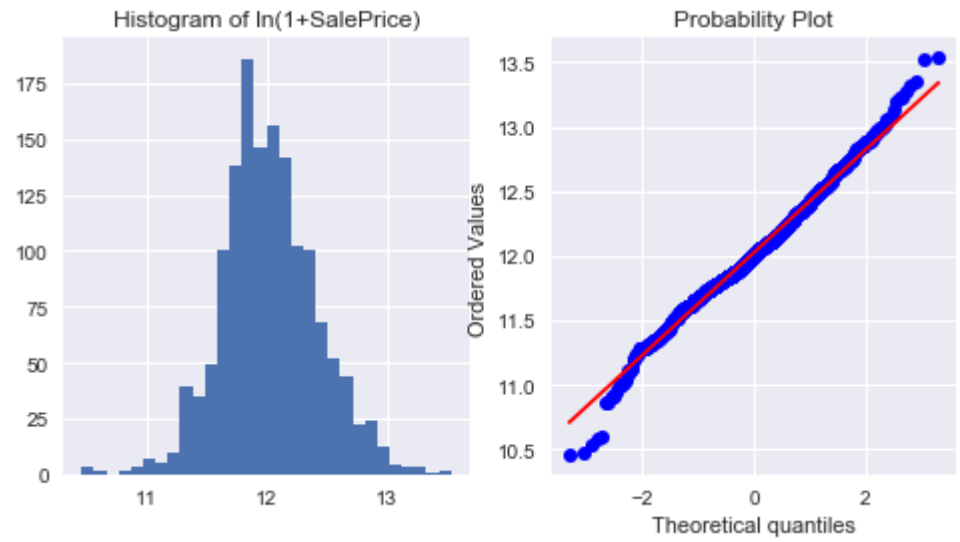
The target is a continuous variable, so the first step is to analyze its distribution:



As we can see, the distribution is right skewed. To illustrate a bit more, we will also analyze the Probability plot:



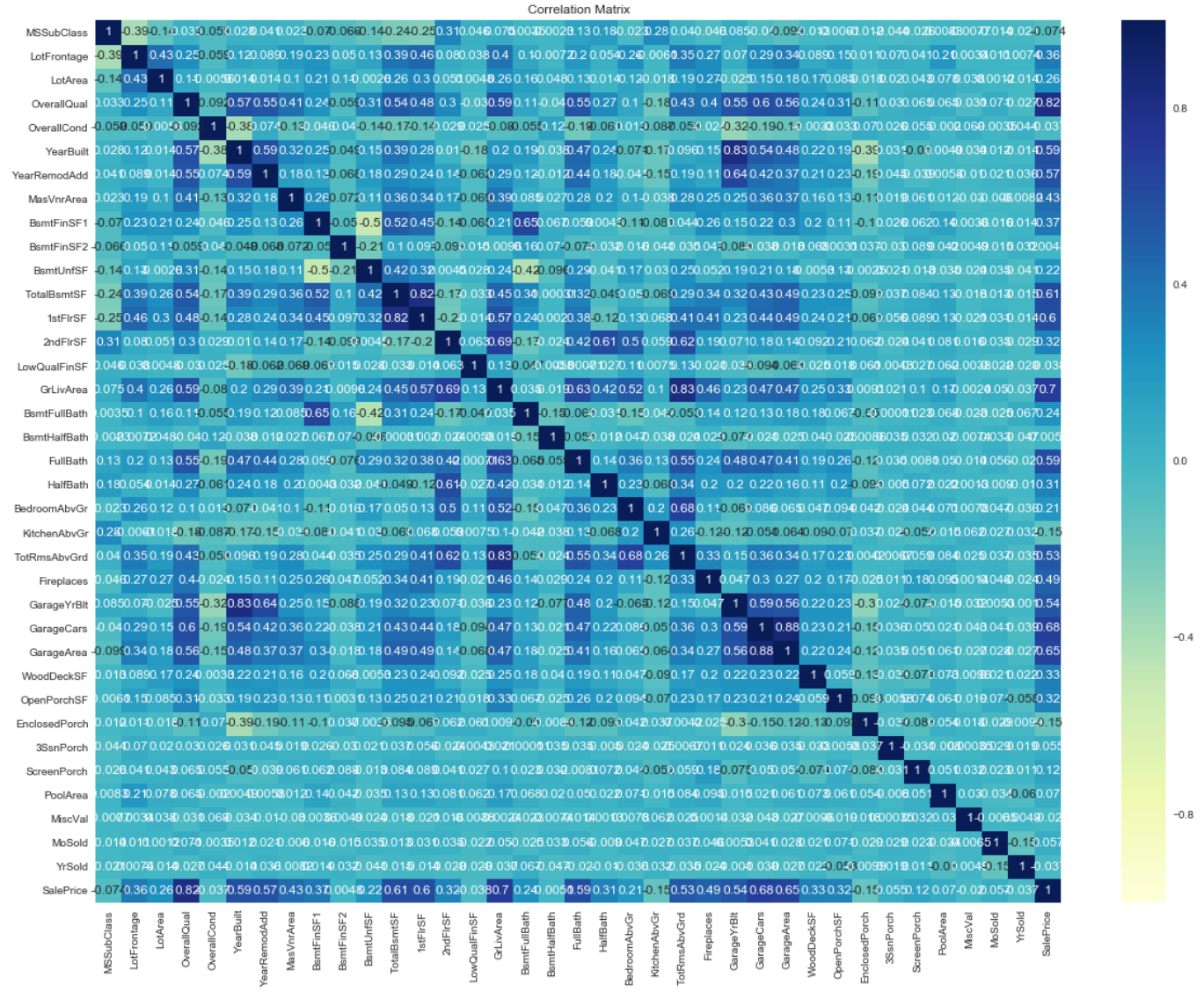
Usually predictive models work better on normally distributed targets. A log transformation on right skewed targets tends to be sufficient to normalyze it. Although we have already validated that there is no SalePrice that is 0 or negative, as a good practice we will use the ln(x+1) transformation. Below we can see the target´s distribution and probability plots after the log transformation:



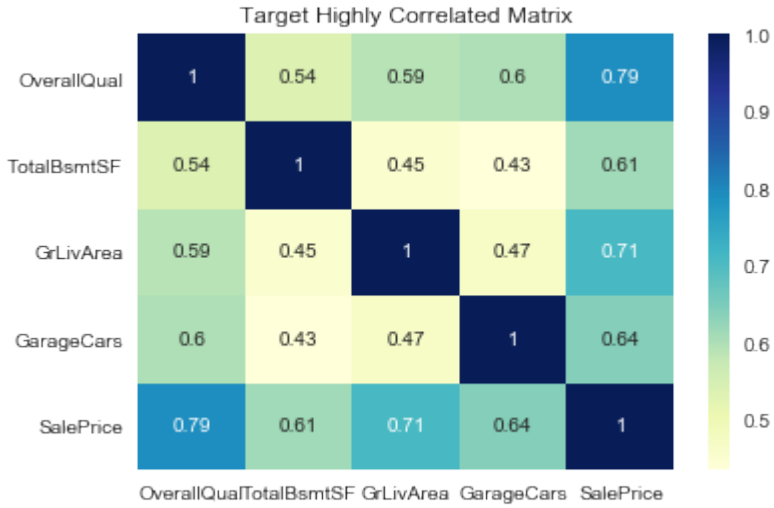
As we can see above, after the transformation the target is almost normal, so we will keep the log transformation on target and continue analysing the rest of the variables.

Numerical Variable Analysis

The first analysis on numerical variables is the Correlation Matrix, which brings the correlation between all numerical variables. We are going to use only the train dataset here because we want to take a closer look at how the variables correlates with the target (SalePrice)



Although we have some interesting correlations that needs to be investigated deeper, we will firstly focus on the variables that are highly correlated with SalesPrice. * OverallQual * TotalBsmtSF * GrLivArea * GarageCars



Let's take a look at the scatter plots to have a better idea of the correlations between the variables and the target:

OverallQual: This numerical variable will be transformed to an ordered categorical variable. But it´s interesting to note that it has an almost linear correlation with the price **TotalBsmtSF** This variable has a positive relationship with price. There is an interesting outlier that is a building with a very big basement, but I will keep it **GrLivArea** and **GarageCars** are very correlated with the price as well with no outlier to be removed

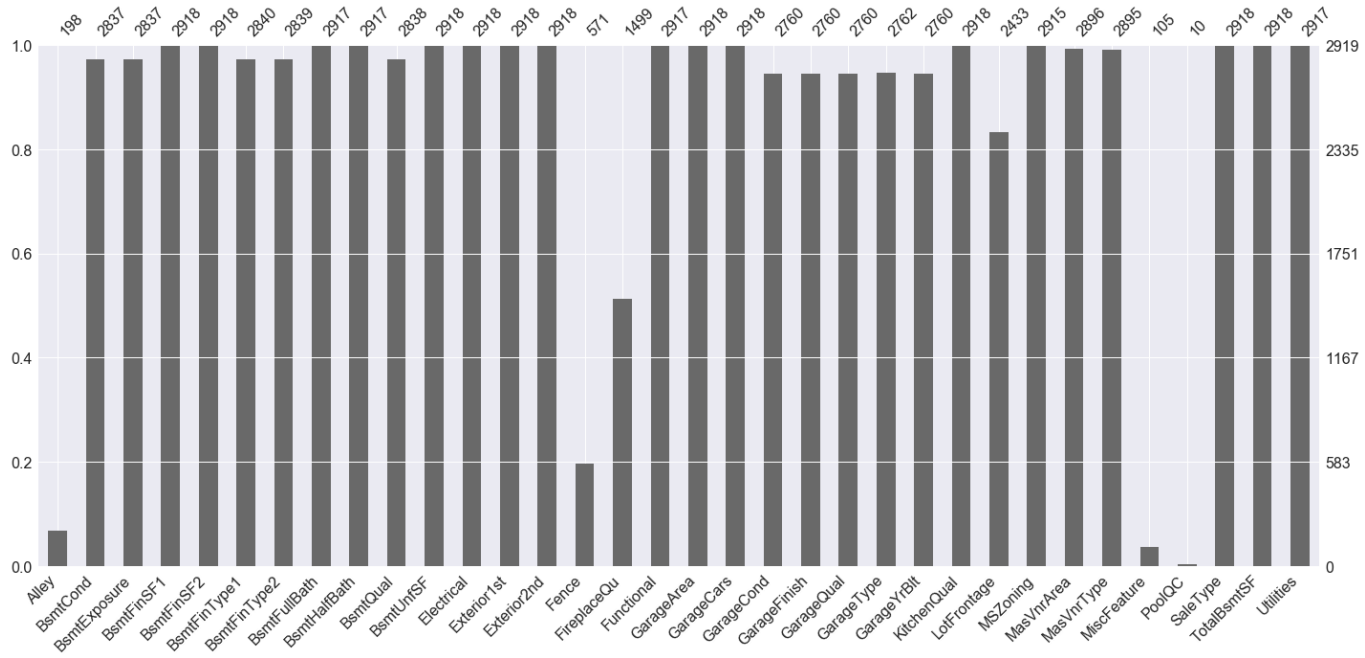
From the above, only OverallQual and GarageCars are in my list of High Importance vars on "variable_description.xlsx" file. It´s important to mention that I ranked the variables in the spreadsheet prior to analysing the data, so it's only a personal feeling on the variable importance. The next step is to analyze variables that are highly correlated with itselves, not necessarily with the target. To avoid multicolinearity, the most correlated with the target will be kept and the other one will be dropped: - GarageArea Vs GarageCars - GarageArea dropped - YearBuilt Vs GarageYrBlt - GarageYrBlt dropped - GrLivArea Vs TotRmsAbvGrd - TotRmsAbvGrd dropped - 1stFlrSF Vs TotalBsmtSF - 1stFlrSF dropped

Before advancing to categorical variables, let´s take a look at nulls

Null Variable Analysis

The null analysis is a very important step on data analysis. In this part we will find out if we have some variables with such a high amount of nulls that they can´t even be used. In this step we will also define the rules for null imputation for all the variables. Although the act of using test data to analyze data and create null imputation rules can be considered data leakage, it's a common practice in Kaggle competitions in order to achieve better results. In real world modelling it would

not be used, but for this specific case we are going to use. The graph below shows bars with the number of no null observations per variable. Each bar represents the number of not null observartion for each variable (higher bars are better). We will present only the variables that have at least 1 null observation:

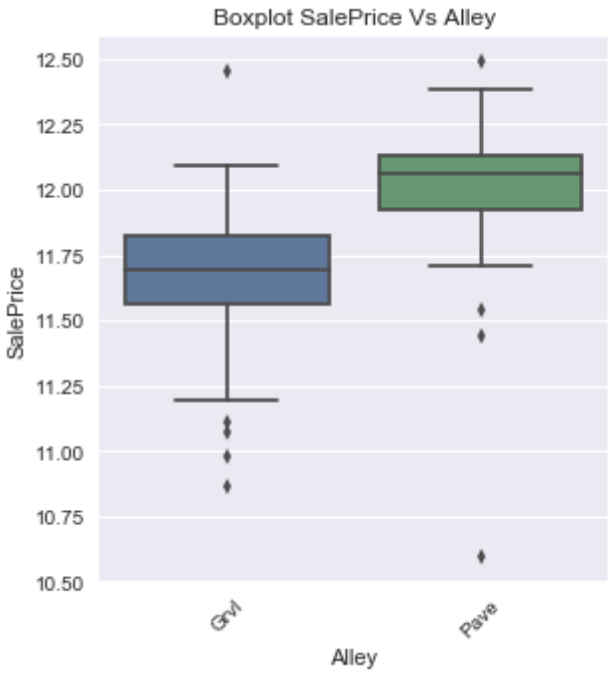


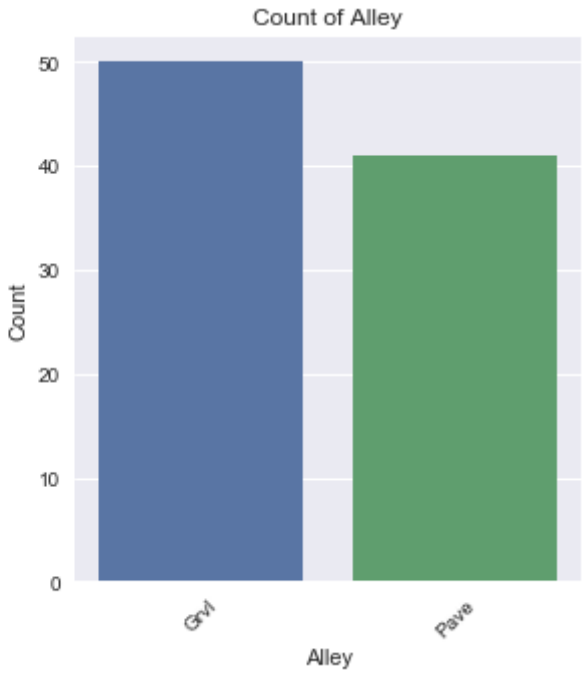
As we can see there is much work to be done on null imputation but the rules will be covered later on Data Preprocessing section.

Categorical Variables Analysis

The objective of the catagorical variable analysis is to identify variables that can explain the target. Also, it´s very important to identify if the data is not concentrated on only one category or if the categorical has only one category. Since the number of categorical variables is too high and all of the analysis is in the data_exploration.ypnb file, we will cover only the some findings about the categorical variables in this section:

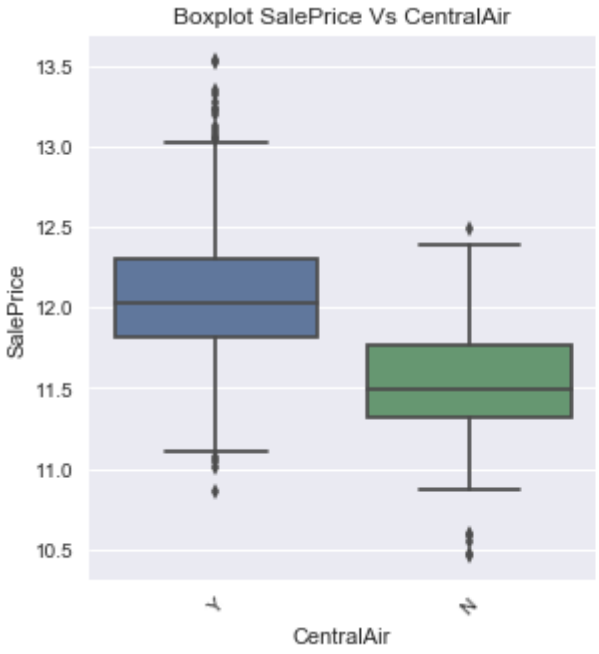
Alley Analysis

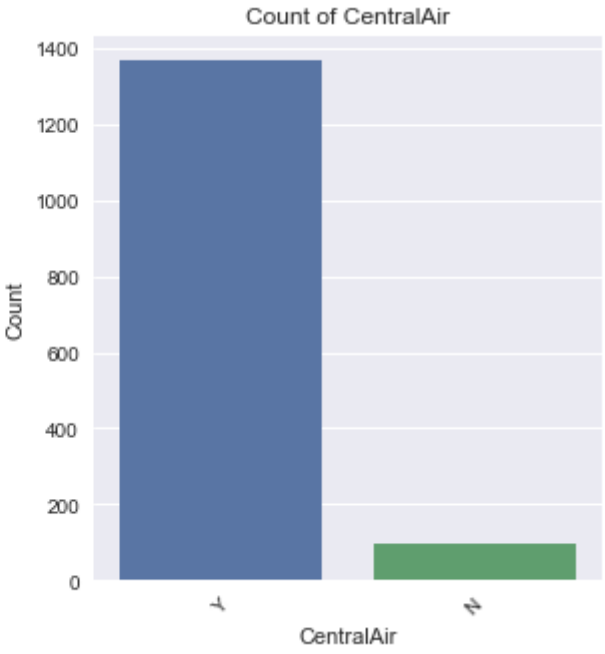




It´s clear that houses with paved alley have higher prices

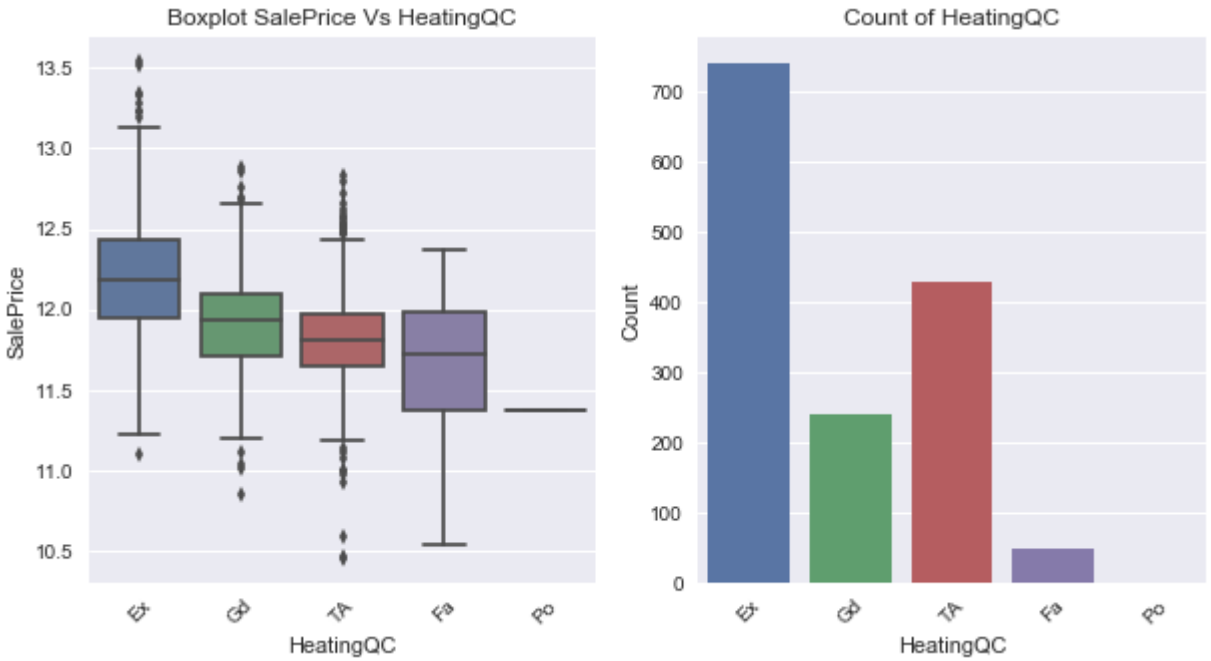
Central Air Analysis





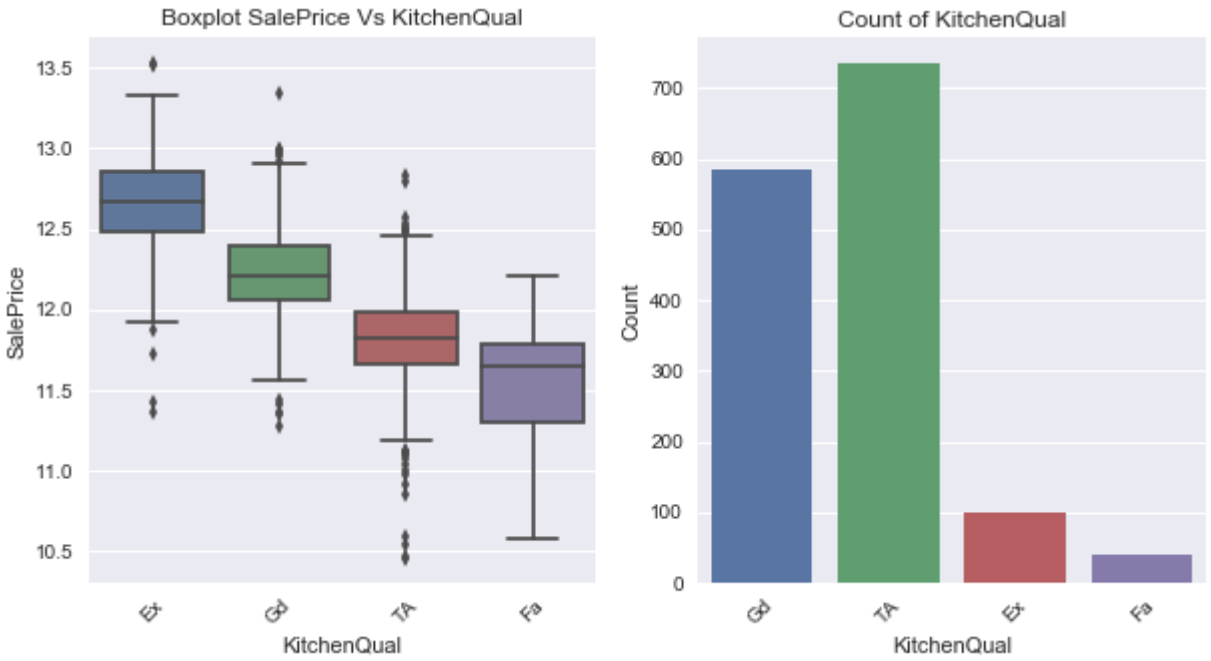
The same can be said about houses with central air conditioning

Heating Quality Analysis



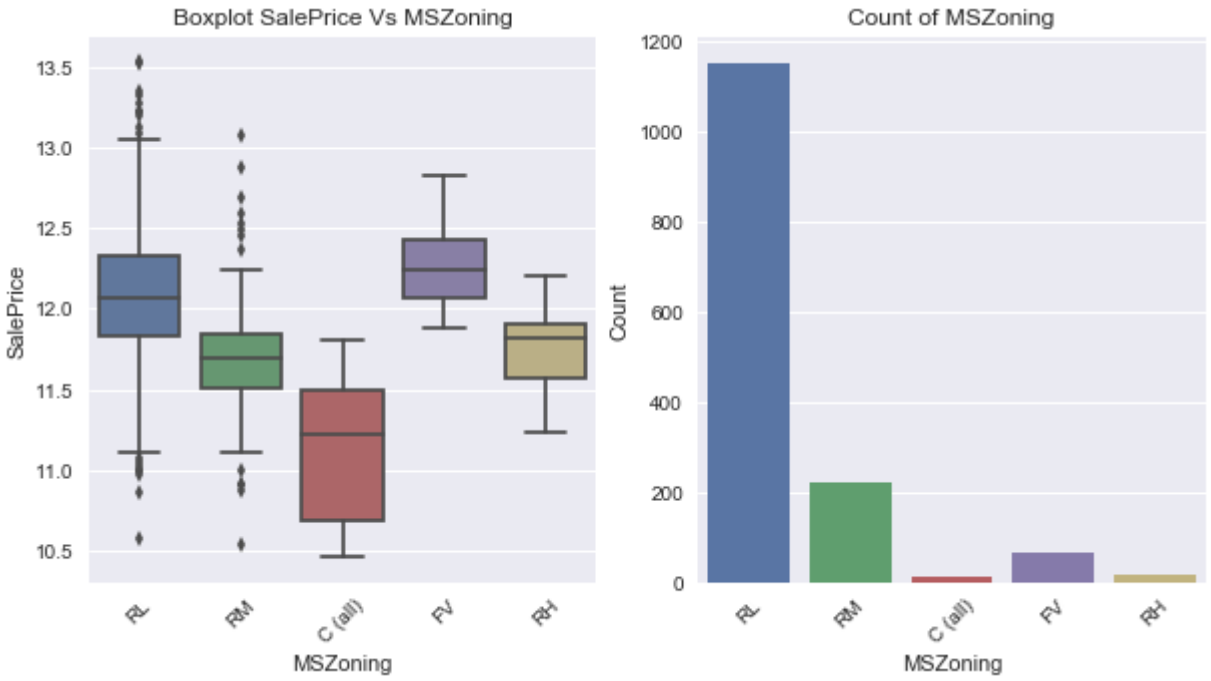
There is a clear relationship between Heating Quality and sale price. The higher the heating quality, the higher the sale price

Kitchen Quality



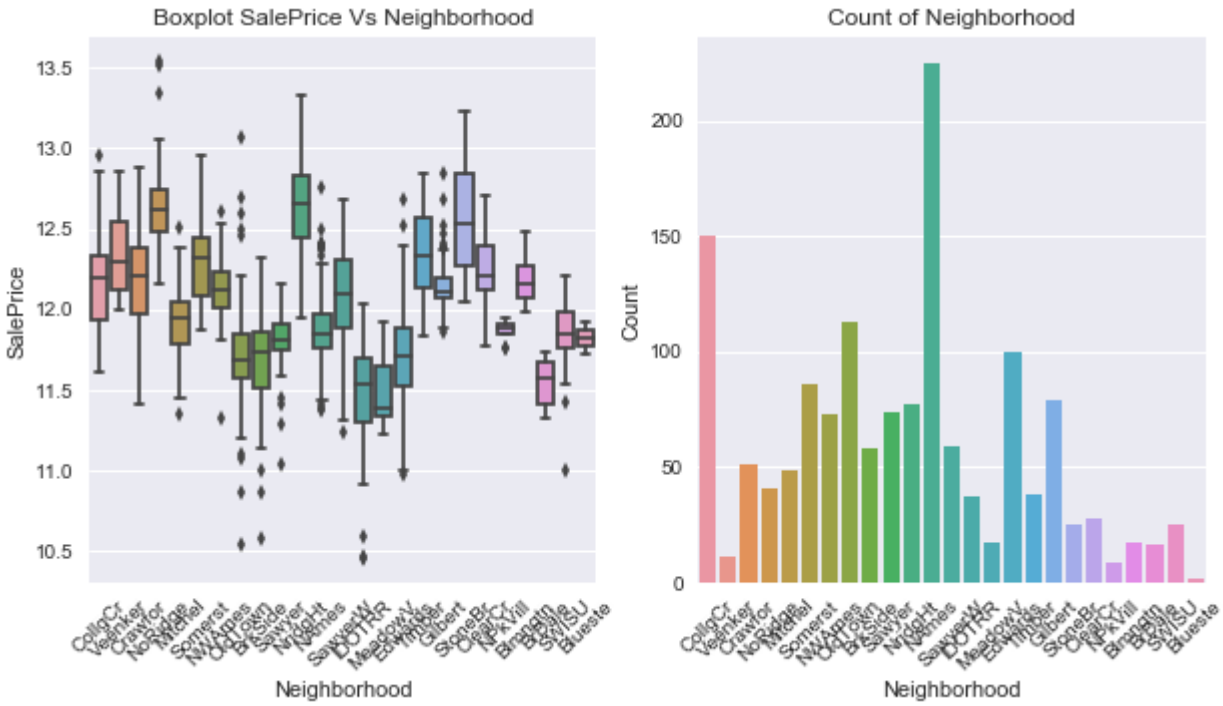
The same can be said about the kitchen quality

General Zoning Classification of the house Analysis



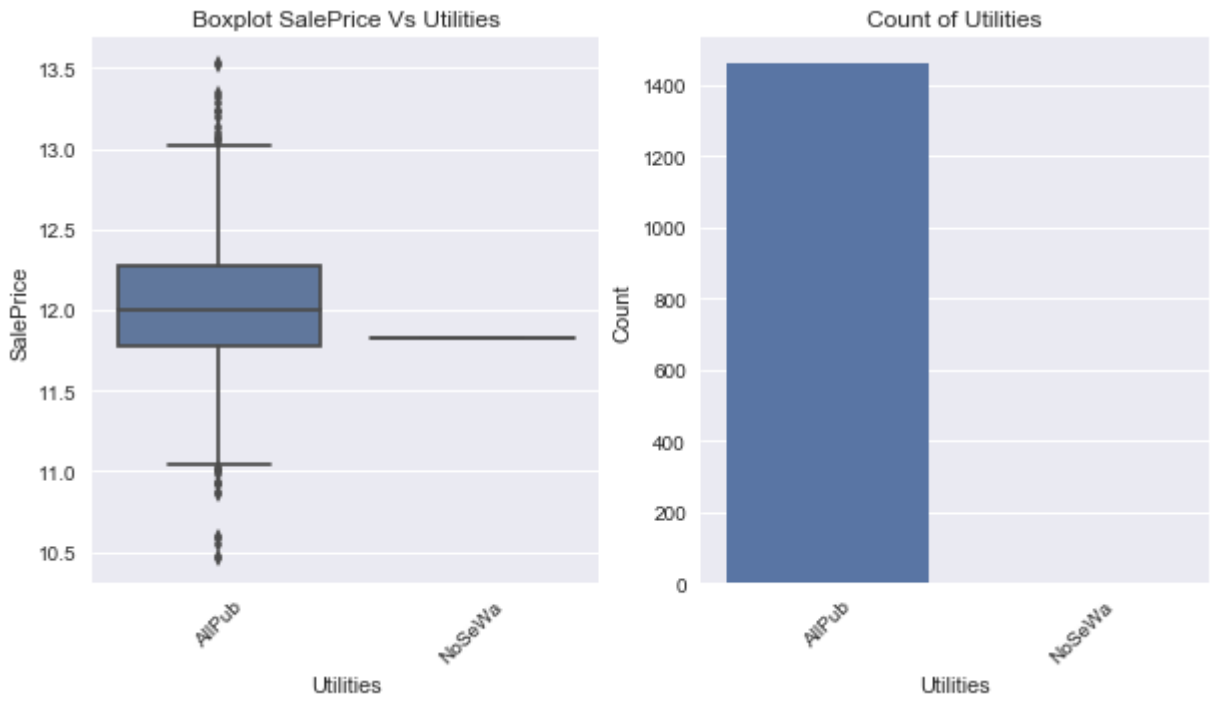
Commercial have lower prices than others. Residential Low Density and fluvial have slightly higher prices

Neighborhood Analysis



As expected some Neighborhoods have higher/lower prices

Utilities Analysis



This categorical have a high concentration on only one category so it will be removed

Algorithms and Techniques

The selected model is a **Random Forest**, because althogh it´s not as robust as a Xtreme Gradient Boosting, it's very easy to understand and much more robust than a simple decision tree. A Random Forest starts with a decision tree which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets. The random forest takes this notion to the next level by combining trees in an ensemble. Thus, in ensemble terms, the trees are weak learners and the random forest is a strong learner. Here is how such a system is trained; for some number of trees T: 1. Sample N cases at random with replacement to create a subset of the data 2. At each node: I. For some number m , m predictor variables are selected at random from all the predictor variables. II. The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node. III . At the next node, choose another m variables at random from all predictor variables and do the same. 3. After the T trees are trained the response is obtained by the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set

To avoid overfitting, the model will be trainned using ***k*-fold cross-validation** Cross-validation is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model’s ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset .

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k - 1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

One more important technique that will be used is the **RandomizedSearchCV**:

RandomizedSearchCV is a hyperparameter tunning technique that tries a fixed number of hyperparameter settings sampled from specified probability distributions. In contrast to GridSearchCV, not all parameter values are tried out, so it´s not so computationally expensive.

Benchmark

The random forest model will be evaluated using Root Mean Squared Error against a Gradient Boosting Model and against a TPOT generated model. All models will be evaluated using root mean squared error applied to the test database using Kaggle submission platform.

III. Methodology

Data Preprocessing

Scale Transformation

The first step on data preprocessing was to perform a log transformation (ln(x+1)) on the target to make its distribution closer to a normal distribution.

Numerical to Categorical Transformation

Next, some variables were converted from numerical to categorical:

- "MSSubClass" that is the type of dwelling involved in the sale, so it does not make sense to be numerical
- "OverallQual" that rates the overall material and finish of the house was converted to a ordered categorical variable. The order does not influence on the model, but helps to understand the categories.
- "OverallCond" that rates the overall condition of the house was converted to a ordered categorical variable. The order does not influence on the model, but helps to understand the categories.
- "MoSold" that is the month sold was also converted to an ordered categorical variable

Null Inputation

According to the data_description file, some nulls makes sense, so the variables that had at least 1 null row were analyzed and the rule chosen for null imputation is described below: * Alley : NA means No Alley access * BsmtQual : NA means No basement * BsmtCond : NA means No basement * BsmtExposure: NA means No basement * BsmtFinType1: NA means No basement * BsmtFinType2 :

NA means No basement * BsmtExposure : NA means No basement * BsmtFinType1 : NA means No basement * BsmtFinType2 : NA means No basement * FireplaceQu : NA means No fireplace * GarageType : NA means No garage * GarageFinish : NA means No garage * GarageQual : NA means No garage * GarageCond : NA means No garage * PoolQC : NA means No pool * Fence : NA means No fence * MiscFeature : NA means None * MasVnrType: NA means None * Electrical: Mode fill * KitchenQual: Mode fill * MSZoning,SaleType: Mode fill * Exterior1st: Mode fill * Exterior2nd: Mode fill * GarageYrBlt: Replacing missing data with 0 (Since No garage = no cars in such garage.) * GarageArea and: Replacing missing data with 0 (Since No garage = no cars in such garage.) * GarageCars : Replacing missing data with 0 (Since No garage = no cars in such garage.) * MasVnrArea: NA means 0 * BsmtFinSF1: NA means 0 * BsmtFinSF2: NA means 0 * BsmtUnfSF: NA means 0 * TotalBsmtSF: NA means 0 * BsmtFullBath: NA means 0 * BsmtHalfBath: NA means 0 * Functional : NA means typical * LotFrontage : Group by neighborhood and fill in missing value by the median LotFrontage of all the neighborhood * LGarageYrBlt : All garages that do not have a Year Built, does not exist (has 0 size in cars), so we will fill nulls with 0

Dimension Reduction

In statistics, multicollinearity (also collinearity) is a phenomenon in which one predictor variable in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy. In this situation the coefficient estimates of the multiple regression may change erratically in response to small changes in the model or the data. Multicollinearity does not reduce the predictive power or reliability of the model as a whole, at least within the sample data set; it only affects calculations regarding individual predictors. That is, a multivariate regression model with collinear predictors can indicate how well the entire bundle of predictors predicts the outcome variable, but it may not give valid results about any individual predictor, or about which predictors are redundant with respect to others. Knowing that, we decided to remove variables that have a correlation with other variable of the model greater than 0.8 or -0.8. This is because we want a good model but we also want to be able to explain how each variable is contributing to the target prediction. For all pairs of multicolinear variables, we kept the one with a bigger correlation with the target variable. The variables that were removed in this step are:

- 'GarageArea',
- 'GarageYrBlt',
- 'TotRmsAbvGrd',
- '1stFlrSF'

Other problem that can happen : sometimes the categorical variables have only 1 category, or have 2 categories with more than 99% of the concentration on one of te categories. These kinds of variables have no or almost no predictive power, and because of this they will be removed:

- 'Utilities'
- 'Street'
- 'Condition2'
- 'PoolQC'

LabelEncoder

We could have used one-hot-encoding on categorical variables, but as we are going to use tree-based models, we decided to go with Label encoding to keep the dimension of the dataset smaller and accelerate the trainning time. As all categorical variables were encoded, they will not be listed here. All preprocessing steps can be found on data_pipeline.ipynb file.

Implementation

The main model implemented was a random forest. The algorithm is part of scikit-learn python package, which is a machine learning open sorce package. As the datasets were ready to use because all the steps that were performed on the previews phases, the only data manipulations needed prior to trainning the model were to drop the ID column and separate the target variable (SalePrice) from the predictors. Following the same steps a gradient boosting model was implemented too. To select the third model I didn´t pick a specific algorithm. Instead I chose to use an automated machine learning package called TPOT. This package test lots of models trying to optimize the metric chosen. The best model that was created by TPOT was a Gradient Boosting too. One interesting point here is that TPOT does not accept RMSE (root mean squared error) as a metric to optimize, but it accepts NMSE (negative mean squared error), so to keep the model results comparable, I trained and refined all the models using NMSE, but validated the results of the three models with RMSE, using Kaggle´s submission system, as proposed.

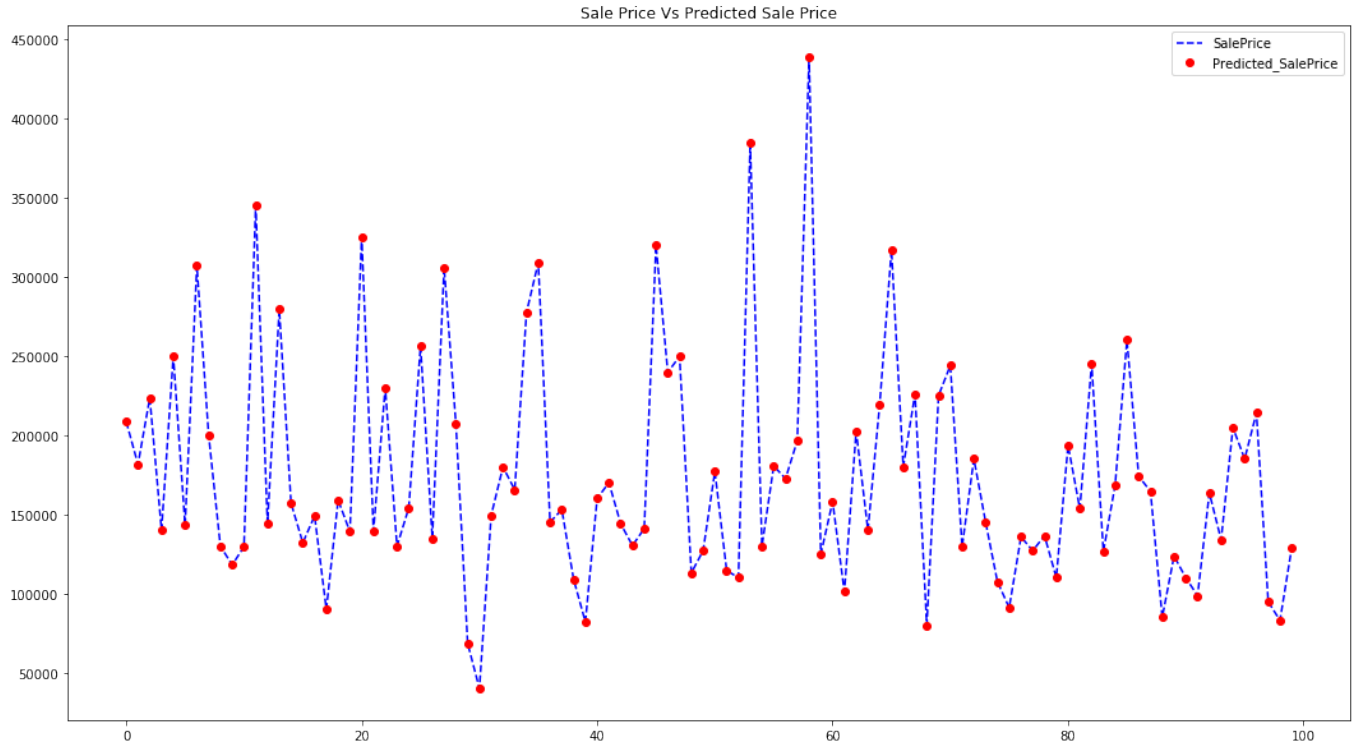
Refinement

The refinement of the model via Hyperparameter tunning was done using **RandomizedSearchCV**, which tries a fixed number of parameter settings sampled from the specified distributions passed to the function. A dictionary was created with some of the Random Forest and Gradient Boosting models hyperparameters and this dictionary was passed to the RandomizedSearchCV function to sample and test the possibilities. TPOT does this part automatically .

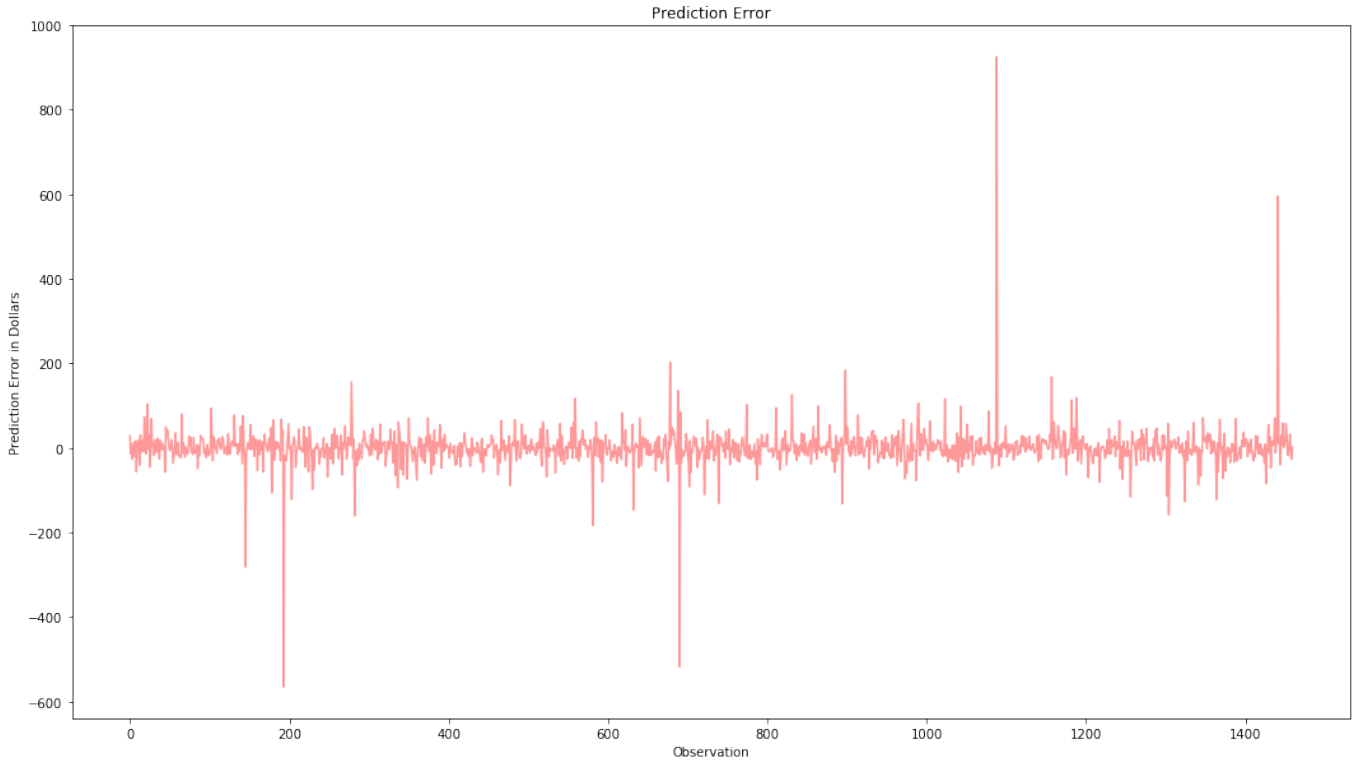
IV. Results

Model Evaluation and Validation

All three models were trainned using k-fold cross validation, so, in theory, they generalized very well to unseen data and it was making sense because the better, the cross validation score during trainning, the better the test score on Kaggle. I started to doubt when I plotted real sale price vs predicted sale price for the first 100 observations. Although predicting observations that we used to train the model can return results better than predicting out of sample results, my objective was to verify if something was wrong:

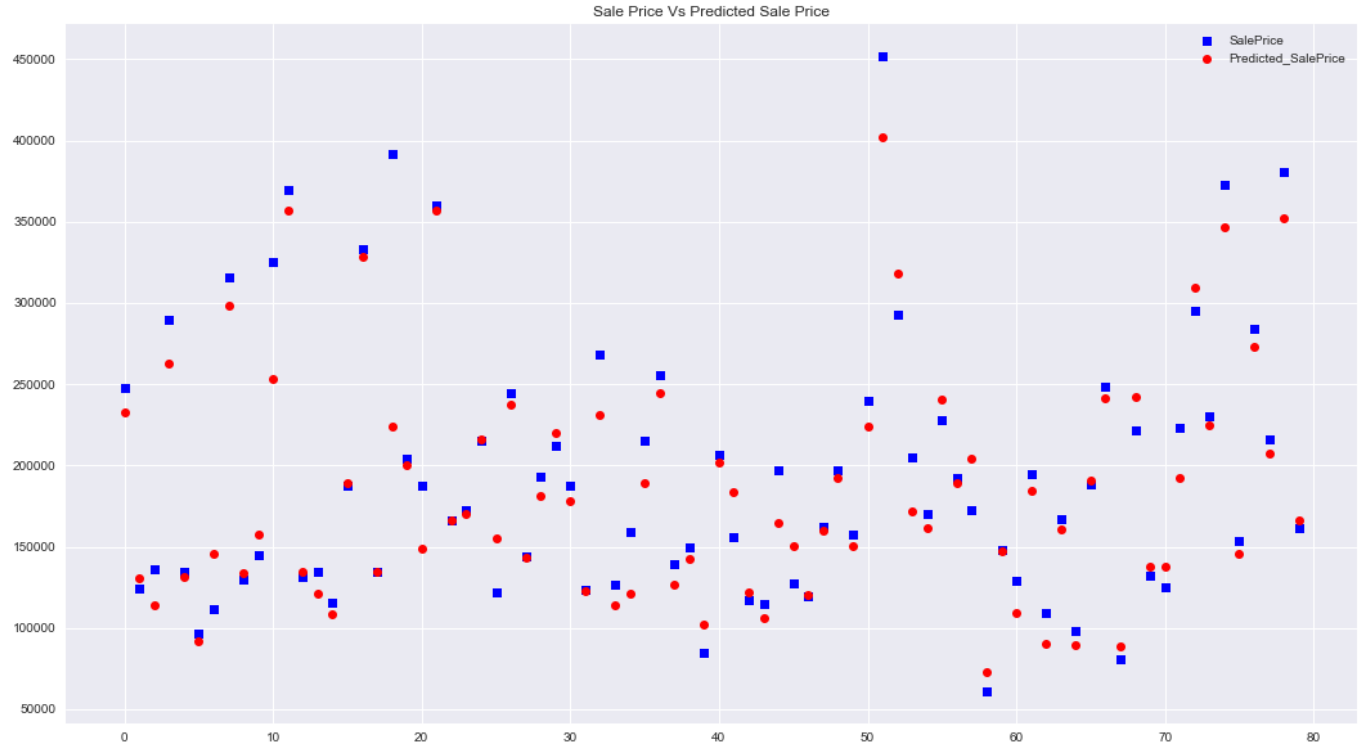


That was an almost perfect model or an overfitted one. To continue my investigation, I calculated the r2 score for all observations, because an r2 of 1 means a perfect (overfitted in most cases) model. The result was: 0.9999996 To better visualize how close the predictions were, I plotted the differences between the real sale price and the predicted one for all observations:

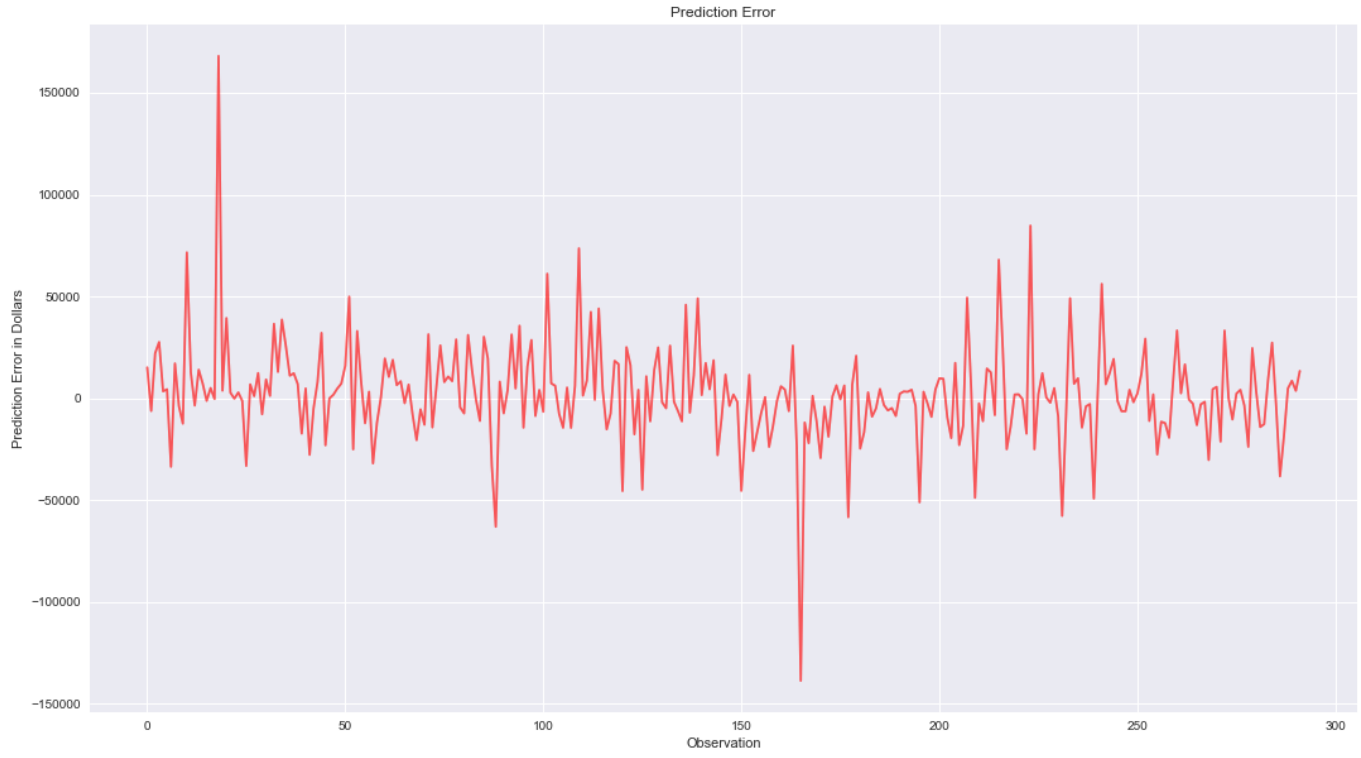


The mean sale price of the houses is almost 200.000 dollars and the error was between -200 and 200 dollars for almost all observations . All those points were more than enough to convince me of the overfit. To overcome this problem I decided to split the dataset between train and test to garantee generalization. The train part of the dataframe was trainned in the same way, including k-fold cross validation, but the model was validated predicting the sale price on the test (out of sample)

dataframe:



And below is the error plot:



Now I have a good model that can generalize and predict prices with error ranging from -50.000 to +50.000 on a sample of observations that were not used to train the model.

Justification

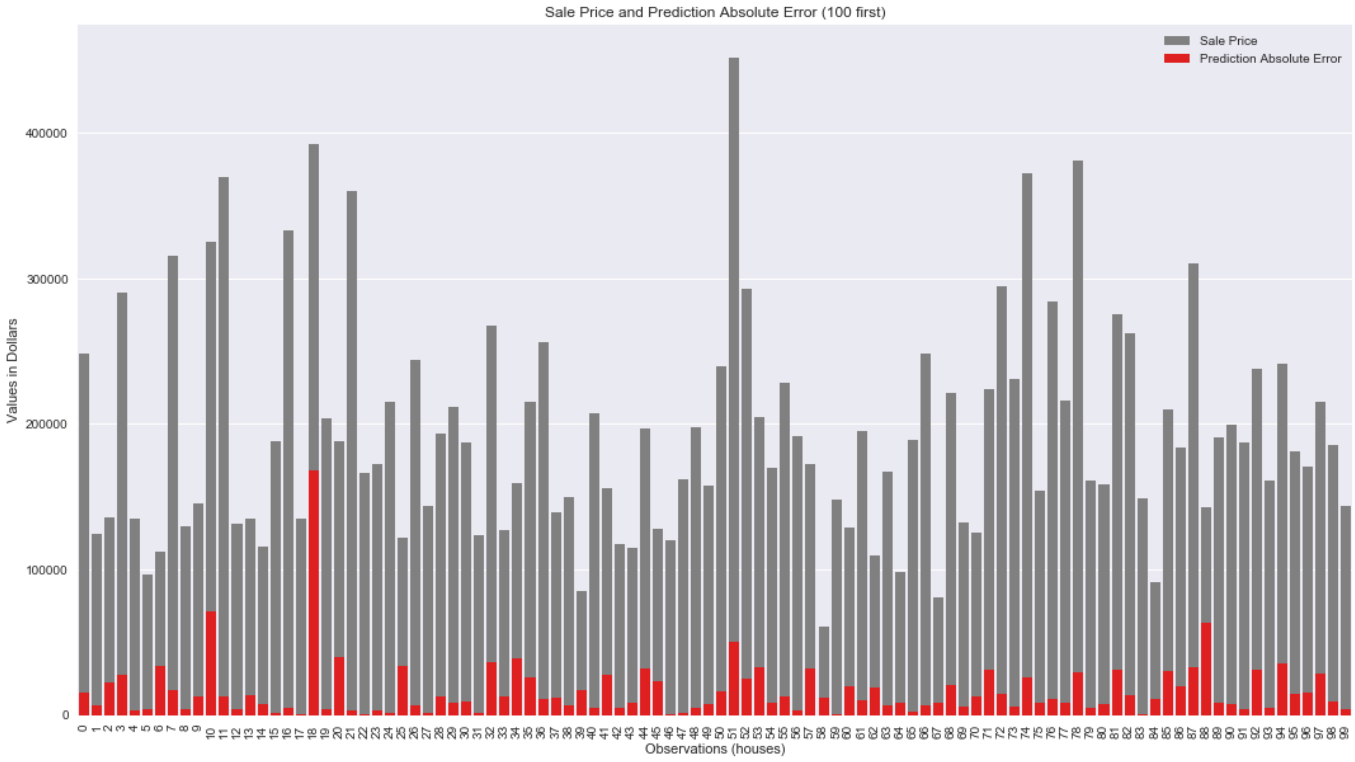
The last model brought the security that I was persuing. It performed very well using negative means squared error and r2. It also proved that it could generalize well, as demonstrated above. The

last step that was needed in order to prove that this was the best model, was to submit the predictions to Kaggle, and it proved to be the best, because it ranked 45 positions better than my TPOT model submission, that was my best submission until this moment.

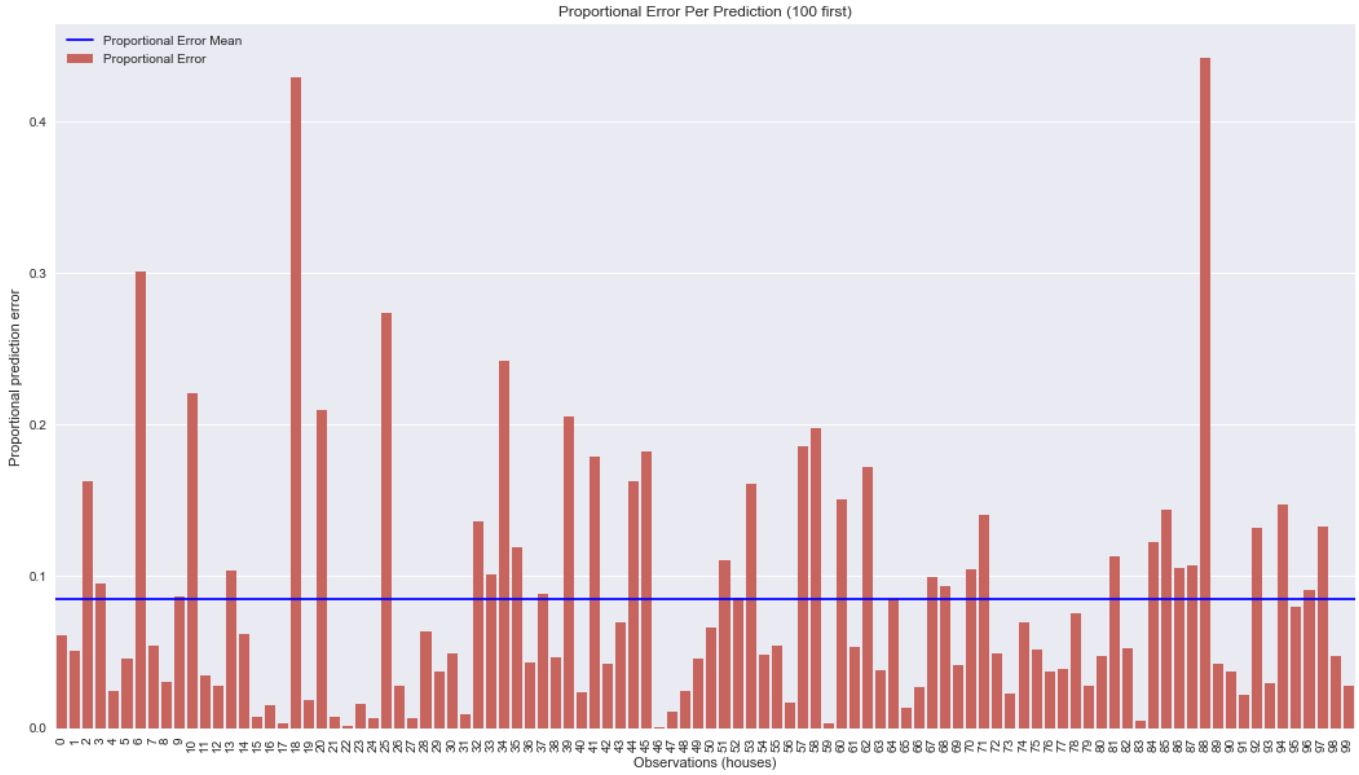
V. Conclusion

Free-Form Visualization

A good form to analyze if the model is solving the problem in an easy to understand way is to look at the real sale price of a house and compare it with the error of the predicted price. To be a bit more clear, the error is the difference between the real sale price and the predicted sale price. In the graph below we are going to compare the real sale price of the house with the prediction error. For this specific graph, we will use the absolute value of the error, because we want to see the how big the error is, compared to the sale price:



As we can see, for the first 100 rows, the error is very small compared to the sell price of the houses with some few exceptions, but to have a better idea, the graph below will show us the proportion between the error and the sale price:



With this visualization it's clear that we have some spikes, but most part of the predictions have a difference smaller than 10% when compared to the real sale price. This is something awesome because we can predict a price that is very close to the real one. Besides that, the spikes, can be specific cases that the seller got a very good price or had to sell it very fast, so he had to lower the price to accomplish that.

Reflection

The first big challenge on this project was to decide which dataset and problem to try to solve. There are lots of possibilities, but I tried to find one that I had some prior knowledge. After deciding the dataset and problem, the next big challenge was to understand all the variables. The dataset has a considerable number of variables and each one was explored and analyzed. This part took a lot of time and effort. The last challenge was to prove that the model was overfitted, prove and eliminate the problem. All the steps taken to overcome this were covered on the previous section.

Improvement

There is a clear improvement to be done on the pre-processing step. I could have used one-hot-encoding on the categorical variables instead of label encoding because it would open space to non-tree models. One other improvement that could have been done was to transform skewed numerical variables to make them closer to normal. With these 2 improvements I think I would reached better results

References

https://en.wikipedia.org/wiki/Cross-validation_(statistics)
https://en.wikipedia.org/wiki/Random_forest

http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics
https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d