FINAL REPORT

Making Houses More Appealing to Buyers

Abstract

Exploratory Data Analysis and Machine Learning Algorithms on a Housing Dataset to support which home improvement may positively affect the Selling Price

Rafael Silveira

rafaelmoreirasilveira@gmail.com

Table of Contents

Introduction

Background

The housing market is the target of several studies because it affects an important part of the society. Moreover, it involves large amounts of money, especially if we are dealing with a family's entire life's worth of savings or future earnings in the form of loans.

Every owner wants to make a good deal when selling their properties. The most significant actions an owner can take to get a better evaluation of their houses are performing house upgrades. But there are so many possible enhancements available, wouldn't it be great if we could identify which services would impact the selling price the most?

Proposal

This study is the first two parts of a more ambitious study that would recommend homeowners and home brokers which home improvements would greatly impact the selling price. The outcome of the complete version of the study would be a "portfolio" of possible home upgrades, alongside with the cost and duration of the project, and the estimated increase in the selling price. This would be presented to homeowners and home brokers, giving them the opportunity to either sell a "harder-to-sell" house faster or to maximize their profits.

As far as this study is concerned, the first part would be to identify which house features are immutable, like location, type of dwelling, etc. and which features can be affected by house projects. The immutable features will be used in the future to classify a house, therefore they will be called "classifying variables". The second group are the ones that the owners can act upon with house projects, therefore they are the "actionable variables".

Our final deliverable is a Machine Learning model capable of providing good predictions for a random house. This way, we could apply the model to a specific house and provide a list of "actionable variables" that have the most positive impact on the selling price.

Out of the scope of this study, but vital to the success of this business, is to identify which home project will affect one or more of the "studied variables". The next step would be to get quotes and time estimates for these home projects. For that, services from a company like Homestars can be used, where home professionals are easily found, as well as quotes and time estimates for home projects.

Some of the "classifying variables" can be used for acquiring the estimates. Once this part is done, the portfolio will be complete and it will be possible to recommend which home improvement will better impact the selling price of a house, taking the time and resources available for home projects into consideration. This is the reason why this part is crucial: it is possible to find that the cost of the upgrade may be greater than the impact on the selling price, voiding the findings of this study.

Dataset

The dataset used for this study is the Ames Housing Dataset, which presents 79 explanatory variables describing several aspects of residential homes in Ames, Iowa, United States. The dataset can be found following the link below:

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

Once this study is finalized, with minor adjustments, the algorithms generated may be applied for different regions.

It is also important to note that a description of the columns of the dataset is presented in the appendix.

Data Wrangling

Checking for errors on categorical columns

The document "data_description.txt" shows all the allowed values for each categorical data. Let us list all unique values on each categorical column to look for mistyped data. IMPORTANT: there are 3 categorical columns that are represented by numbers ('MSSubClass','OverallQual','OverallCond'). Please note the highlighted line where we added those 3 columns.

```
import pandas as pd
df = pd.read_csv('house_price.csv, header=0, index_col='Id')
dfnumbers = df._get_numeric_data()
catcols = set(df.columns) - set(dfnumbers.columns)
catcols = catcols.union(set(['MSSubClass','OverallQual','OverallCond']))
for col in catcols:
    print(col, ' - ',df[col].unique())
```

Here is a part of the output

```
GarageQual  -  ['TA' 'Fa' 'Gd' nan 'Ex' 'Po']
Foundation  -  ['PConc' 'CBlock' 'BrkTil' 'Wood' 'Slab' 'Stone']
CentralAir  -  ['Y' 'N']
ExterQual  -  ['Gd' 'TA' 'Ex' 'Fa']
BsmtExposure  -  ['No' 'Gd' 'Mn' 'Av' nan]
MSZoning  -  ['RL' 'RM' 'C (all)' 'FV' 'RH']
RoofStyle  -  ['Gable' 'Hip' 'Gambrel' 'Mansard' 'Flat' 'Shed']
```

With a quick inspection, we notice some NaNs that are not "missing data". The highlighted nan, for example, means "no garage", but it could also mean "missing data" in other columns. We will investigate that later.

Comparing the output with the data description, we noticed small errors that are quite easy to miss if we inspect manually. So, we created a .csv file called "AllPossibleValuesForCatCols.csv", using the data_description.txt as the source, to list all possible values those categorical columns may have. We will import this file as a Dataframe and match the actual values found on the dataset with the possible values for that column. This way we can find typos easier. This is the code to find all typos:

```
df = pd.read_csv('train.csv', header=0, index_col='Id')
dfAll = pd.read_csv('AllPossibleValuesForCatCols.csv', header=0)
dfAll = dfAll.applymap(lambda x: x.strip() if isinstance(x, str) else x)
diffs = {}
for col in dfAll.columns:
    diffs[col] = set(df[col]) - set(dfAll[col])
for key, value in diffs.items():
    if(len(value)>0): print(key, ' - ',value)
```

Analyzing the output,

```
MSZoning  -  {'C (all)'}
Neighborhood  -  {'NAmes'}
BldgType  -  {'Duplex', '2fmCon', 'Twnhs'}
Exterior2nd  -  {'Wd Shng', 'CmentBd', 'Brk Cmn'}
```

we noticed some small variations on the text of some columns, compared to the data_description data. To match the values exactly, we changed the following, by running this code:

```
df.BldgType = df.BldgType.replace('Twnhs', 'TwnhsI')
df.BldgType = df.BldgType.replace('Duplex', 'Duplx')
df.BldgType = df.BldgType.replace('2fmCon', '2FmCon')
df.MSZoning = df.MSZoning.replace('C (all)', 'C')
df.Exterior2nd = df.Exterior2nd.replace('Wd Shng', 'WdShing')
df.Exterior2nd = df.Exterior2nd.replace('CmentBd', 'CemntBd')
df.Exterior2nd = df.Exterior2nd.replace('Brk Cmn', 'BrkComm')
df.Neighborhood = df.Neighborhood.replace('NAmes', 'Names')
```

- Column **BldgType**: "Duplex" should be "Duplx" and "2fmCon" should be "2FmCon". "Twnhs" should be "TwnhsI".
- Column **MSZoning:** "C (all)" should be "C"
- Column **Exterior2nd:** "Wd Shng" should be "WdShing" because wecan see "Wd Shng" on the data. "CmentBd" should be "CemntBd" and "Brk Cmn" should be "BrkComm"
- Column **Neighborhood**: "NAmes" should be "Names" (meaning North Ames)

If we run the same code again to check typos, the output is empty, meaning the code ran successfully.

Back to the unique values from all columns, a few other things that caught our attention. Below are the modifications we performed on the columns **MasVnrType** and **MasVnrArea,** performed by the code below:

```
df = df.replace({'MasVnrType':{np.nan : 'None'}, 'MasVnrArea': {np.nan : 0}}, value = None)
print(df[['MasVnrType', 'MasVnrArea']][(df.MasVnrType == 'None') & (df.MasVnrArea > 0)])
df.MasVnrArea = df.MasVnrArea.replace(1.0,0.0)
print(df[['MasVnrType', 'MasVnrArea']][df.MasVnrArea > 0].MasVnrType.value_counts())
for index, row in df.iterrows():
    if((row['MasVnrType'] == 'None') & (row['MasVnrArea'] > 0)):
        df.loc[index, 'MasVnrType'] = 'BrkFace'
```

There are 8 values that are "None" values and NaN on the MasVnrType column. Replace NaN with 'None' on column MasVnrType and replace NaN with 0 on the MasVnrArea column.

When the MasVnrType is None, MasVnrArea should be zero. Analysing this table,

Id	MasVnrType	MasVnrArea
625	None	288.0
774	None	1.0
1231	None	1.0
1301	None	344.0
1335	None	312.0

generated by the code above, we find values different than zero, which is not what we expected. To deal with that, the two records with Areas = 1.0 will be replaced by Area = 0.0, and the other three values will be replaced by the most common Masonry veneer type, which is type = 'BrkFace'.

This type of consistency will be checked for every column later. For example, if the house has no garage, all columns related to "garage" should show NA.

Cleaning NaN from categorical columns

We already know that missing values are stored as NaN. The code below

```
nulls = df.isnull().sum()
catcolswithnan = set(dfAll.columns) & set(nulls[nulls>0].index)
for col in catcolswithnan:
    print(col, ' - ',df[col].unique())
```

lists the categorical columns with their unique values. Analyzing the output

```
PoolQC  -  [nan 'Ex' 'Fa' 'Gd']
Alley  -  [nan 'Grvl' 'Pave']
BsmtQual  -  ['Gd' 'TA' 'Ex' nan 'Fa']
BsmtCond  -  ['TA' 'Gd' nan 'Fa' 'Po']
Electrical  -  ['SBrkr' 'FuseF' 'FuseA' 'FuseP' 'Mix' nan]
BsmtExposure  -  ['No' 'Gd' 'Mn' 'Av' nan]
GarageCond  -  ['TA' 'Fa' nan 'Gd' 'Po' 'Ex']
BsmtFinType2  -  ['Unf' 'BLQ' nan 'ALQ' 'Rec' 'LwQ' 'GLQ']
GarageType  -  ['Attchd' 'Detchd' 'BuiltIn' 'CarPort' nan 'Basment' '2Types']
GarageQual  -  ['TA' 'Fa' 'Gd' nan 'Ex' 'Po']
GarageFinish  -  ['RFn' 'Unf' 'Fin' nan]
Fence  -  [nan 'MnPrv' 'GdWo' 'GdPrv' 'MnWw']
FireplaceQu  -  [nan 'TA' 'Gd' 'Fa' 'Ex' 'Po']
BsmtFinType1  -  ['GLQ' 'ALQ' 'Unf' 'Rec' 'BLQ' nan 'LwQ']
MiscFeature  -  [nan 'Shed' 'Gar2' 'Othr' 'TenC']
```

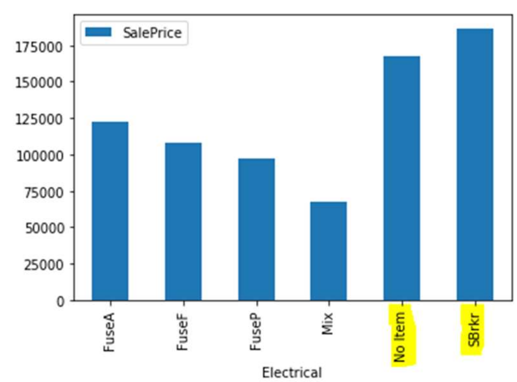
we notice that, for all columns, it makes sense to replace nan with “No Item”, except for the column “Electrical”, because It does not make sense not to have an electrical system.

After replacing all NaN from the above columns with “No Item”, this code

```
print(df[df.Electrical == 'No Item']['Electrical'].count())
#plot code 6
df[['Electrical', 'SalePrice']].groupby('Electrical').mean().plot(kind = 'bar')
plt.show()

df.Electrical = df.Electrical.replace('No Item', 'SBrkr')
```

performs the following operations on the “Electrical” column:

- Finds the number of records we are dealing with
- Checks how the price of this house stands against the other types of electrical systems.



- Replace “No Item” by “SBrkr”

Cleaning NaN from numerical columns

The code below

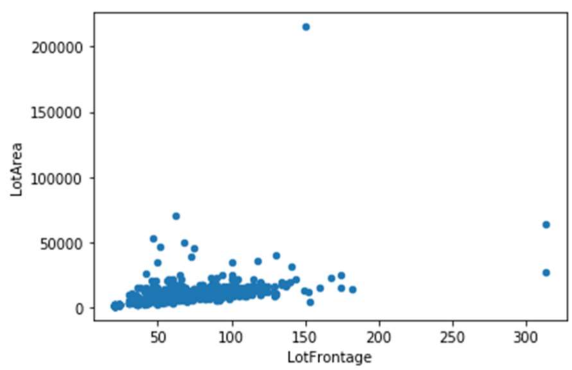
```
dfNumericCols = set(df.columns) - set(dfAll.columns)
dfNumeric = df[list(dfNumericCols)]
print(dfNumeric.isnull().sum()[dfNumeric.isnull().sum()>0])
print(df[['GarageType', 'GarageYrBlt']][df.GarageYrBlt.isnull()].GarageType.unique())
df[['LotFrontage', 'LotArea']].plot(x = 'LotFrontage', y = 'LotArea', kind = 'scatter')
df[['LotFrontage', 'LotArea']].plot(x = 'LotFrontage', y = 'LotArea',
                                    xlim=(0,150), ylim=(0,20000), kind = 'scatter')
```

performs the following operations:

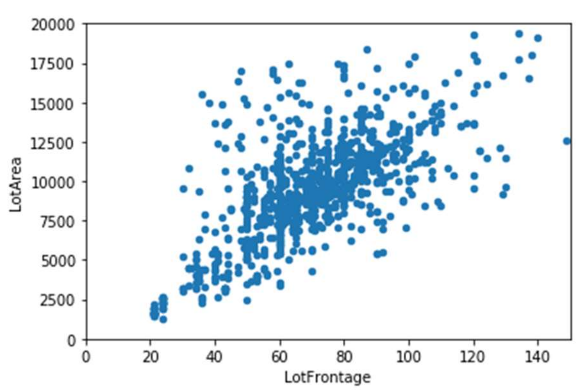
- Create a DataFrame with only the numeric columns;
- Inspect the NaN with this table, generated by code above

```
GarageYrBlt     81
LotFrontage    259
```

  - Column **GarageYrBlt**: Hopefully, those NaN are for houses without a igarage. Check the unique values of the column GarageType when GarageYrBlt is NaN
  - Column **LotFrontage**: those are legitimate NaNs, too many to discard. Inspecting the columns, we found a **LotArea.** Maybe a scatter plot



    will show a relationship between those two variables
    - We can easily identify the presence of outliers on both dimensions. To better identify the relationship, let us “zoom in” in the plot



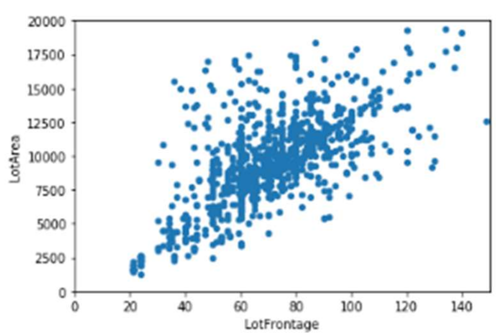
- As suspected, there is a big correlation between those two variables.

Filling those 259 NaN from LotFrontage with values that are proportional to the column LotArea makes sense. To do that, we will find the equation that best represents the linear regression between these two columns and apply it to replace the NaNs. We used LinearRegression() from scikit learn to find that equation, as shown below:

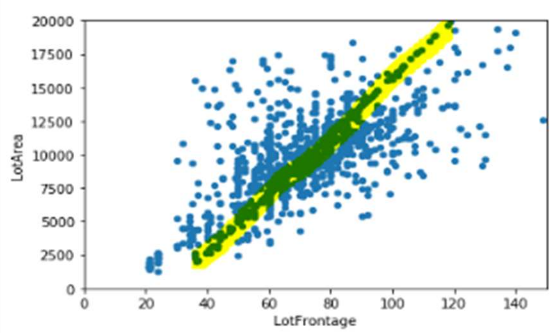
```
#filter out NaN
dfLotTrain = df[np.logical_not(np.isnan(df['LotFrontage']))]
dfLotTest = df[np.isnan(df['LotFrontage'])]

#apply method
model = LinearRegression()
model.fit(dfLotTrain['LotArea'].values.reshape(-1, 1), dfLotTrain['LotFrontage'].values.reshape(-1, 1))
aux = model.predict(dfLotTest['LotArea'].values.reshape(-1, 1))
dfLotTest = dfLotTest.assign(LotFrontage=aux)
```

Here is the before and after this procedure:

BEFORE



AFTER



Consistency among columns

If a house has no garage, all columns related to “garage” should reflect this fact. We have already done this for MasVnrType column on item 1 of this report, now let us take the same approach for garage, Basement, fireplace, pool, and miscellaneous features. Please check the long code below:

```python
#Garage
collist = ['GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond']
df.GarageType== 'No Item'
myDict = {}

for col in collist:
    myDict[col] = []

for index, row in df.iterrows():
    if(df.loc[index, 'GarageType'] == 'No Item'):
        for col in collist:
            myDict[col].append(row[col])

dfGarage = pd.DataFrame(myDict)

for col in dfGarage.columns:
    print(col, '- ', dfGarage[col].unique())

#Basement
collist = ['BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2',
                'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath']
df.BsmtQual == 'No Item'
myDict = {}

for col in collist:
    myDict[col] = []

for index, row in df.iterrows():
    if(df.loc[index, 'BsmtQual'] == 'No Item'):
        for col in collist:
            myDict[col].append(row[col])

dfBsmt = pd.DataFrame(myDict)

for col in dfBsmt.columns:
    print(col, '- ', dfBsmt[col].unique())

#Fireplace
 print(df[['Fireplaces', 'FireplaceQu']][(df.FireplaceQu == 'No Item') & (df.Fireplaces != 0 )])

#Pool
print(df[['PoolArea', 'PoolQC']][(df.PoolQC == 'No Item') & (df.PoolArea != 0 )])

#miscellaneous
print(df[['MiscFeature', 'MiscVal']][(df.MiscFeature == 'No Item') & (df.MiscVal != 0 )])
```

which resulted in empty outputs only, meaning no action needs to be taken.

Checking for outliers

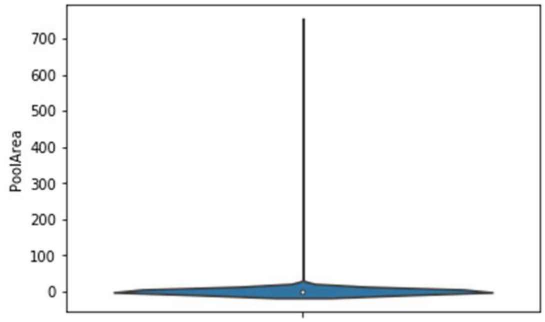
In this section, we are going through a few relevant numerical columns to inspect outliers and decide whether to keep or discard them.

- Violin plots to help inspect the variables

We used this code

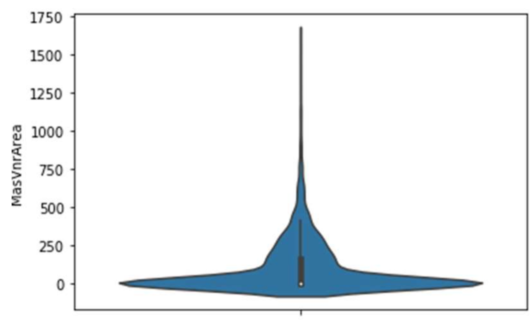
```python
for col in dfNumericCols:
    ax = sns.violinplot(y=col, data=df)
    plt.show()
```

to plot violin plots to spot outliers on the other variables. Inspecting each one of the curves, the following called our attention:



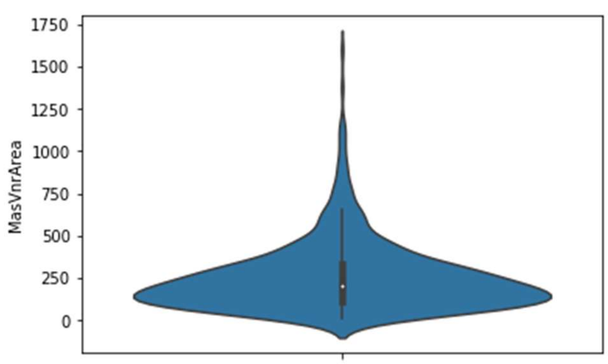
Looking back into the data, this behavior is expected as we only have 7 values different than 0. Maybe this data can help predict house selling prices for specific cases, so we will keep this column as-is.

The same argument goes to columns 3SsdPorch, BsmtFinSF2, LowQualFinSF, MiscVal, ScreenPorch and BsmtHalfBath.



This variable has more than 500 samples different than zero, so let us see if we can find outliers in a violin distribution that ignores the zeros. Here is the code and output

```
ax = sns.violinplot(y='MasVnrArea', data = df[df.MasVnrArea > 0])
```



Now this distribution has only a few outliers that can be useful for some cases and may have a combined effect with other variables, so we will keep it as is.

Exploratory data Analysis - EDA

Our EDA is mainly presented on a Jupyter Notebook called “Story Telling.ipynb”, which is found in the same Github repository. We will show our findings below, but please refer to the original file for more details.

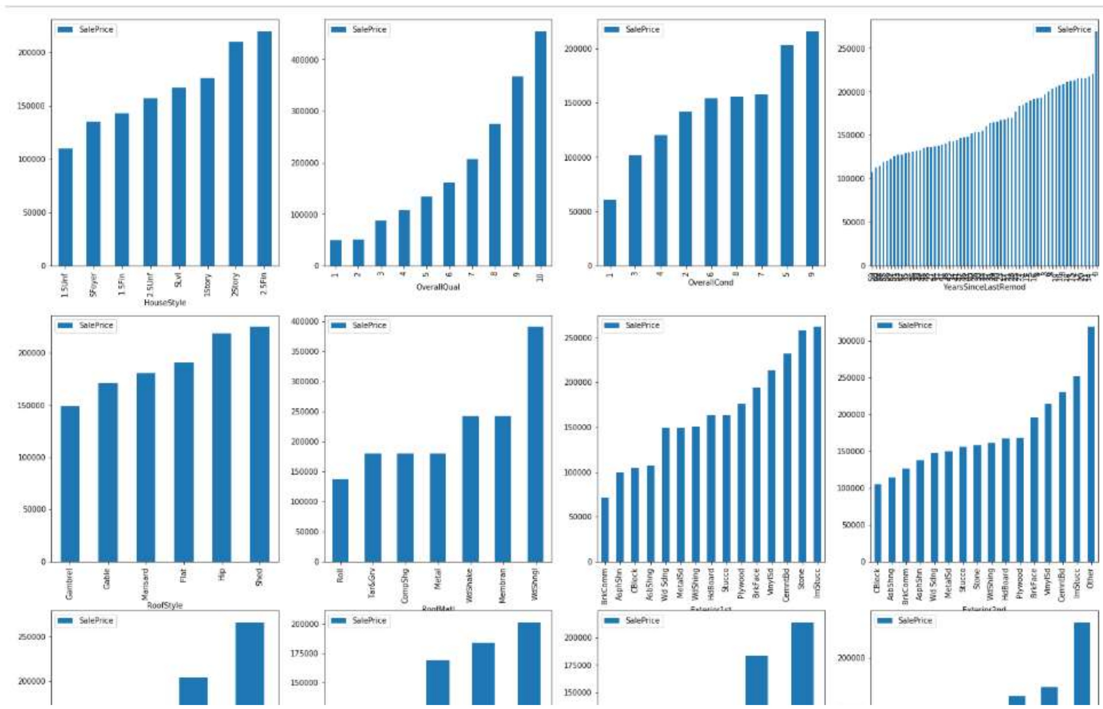
Actionable variables X Selling Price

With the following codes

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math

df = pd.read_csv("DatasetTreated.csv", index_col='Id', header=0)
varList = ['HouseStyle','OverallQual','OverallCond','YearsSinceLastRemod','RoofStyle','RoofMatl','Exterior1st','Exterior2nd',
           'MasVnrType','ExterCond','BsmtCond','BsmtFinType1','BsmtFinType2','HeatingQC','CentralAir','KitchenQual',
           'Functional','PavedDrive','PoolQC','Fence']

fig, axes = plt.subplots(nrows=5, ncols=4)
i=0
j=0
for col in varList:
    df[[col, 'SalePrice']].groupby(col).mean().sort_values('SalePrice').plot(kind="bar", ax=axes[i,j], figsize=(25,35))
    j+=1
    if(j==4):
        j=0
        i+=1

plt.show()
```

```
fig, axes = plt.subplots(nrows=5, ncols=4)
i=0
j=0
for col in varList:
    ax = df[col].value_counts().plot(kind='bar', ax=axes[i,j], figsize=(25,35),title = col)
    for p in ax.patches:
        # get_x pulls left or right; get_height pushes up or down
        ax.text(p.get_x()+.12, p.get_height()+3,
            str(p.get_height()))

    j+=1
    if(j==4):
        j=0
        i+=1

plt.show()
```

we generated several charts at once. This way, we were able to visualize the association of all variables with the Selling Price. You may check part of the output below:



Let us analyse the generated outputs.

a. Roof Style and Roof Material

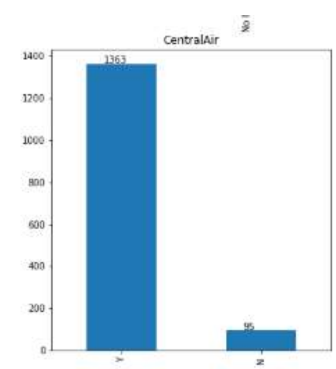
Roof styles are predominantly "Gable" with material "standard Shingle". As we have very few observations of the other types of Roof and types of material, we cannot draw statistically relevant conclusions. As a result, this variable will not be among those we will suggest being improved.

b. Pool Quality

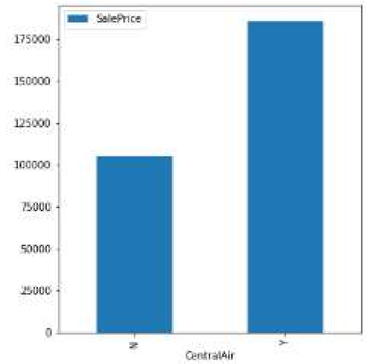
Not enough houses with pools to draw significant conclusions. Improving the quality of the pool will not be suggested to owners.

c. Central Air

Most of the houses on this sample have central air, as shown below:

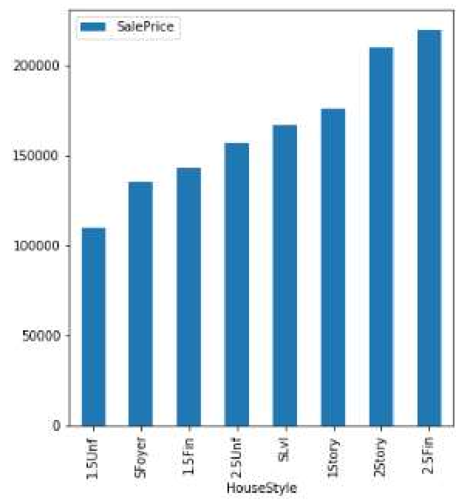


which means that it is possible to find houses with several combinations of the other variables that have that feature. In other words, a buyer will either look for another house or make a low offer. We understand it is not a simple service but, comparing a house with central air with a similar one without central air, we notice an increment of $80,000 on average, as seen below.



d. Price based on number of stories

We were not expecting the average selling price of a 1 story house being more expensive than a one and a half story (finished or unfinished) and more expensive than a two stories house with 2nd level unfinished.

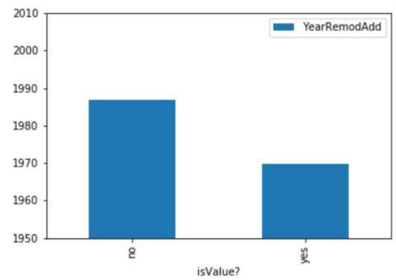


We investigated this and found that:

- 1.5 and 2 stories houses with 2nd level unfinished are located in "poor" neighborhoods

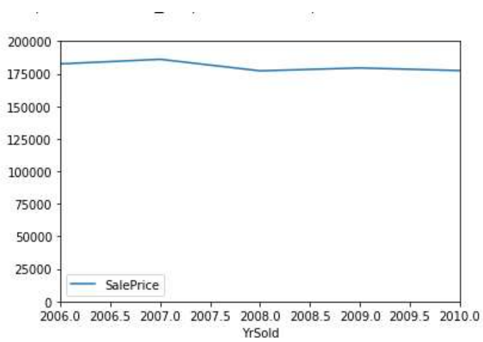
```
 STYLE  - % ON POOR NEIGHBORHOODS
1.5Unf  -  71.0%
1.5Fin  -  73.0%
2.5Unf  -  91.0%
```

- 1.5 and 2 stories houses with 2nd level unfinished are older



e. Selling prices do not increase along the years

We do not need to take into consideration the inflation over the years, as the average selling price does not change significantly

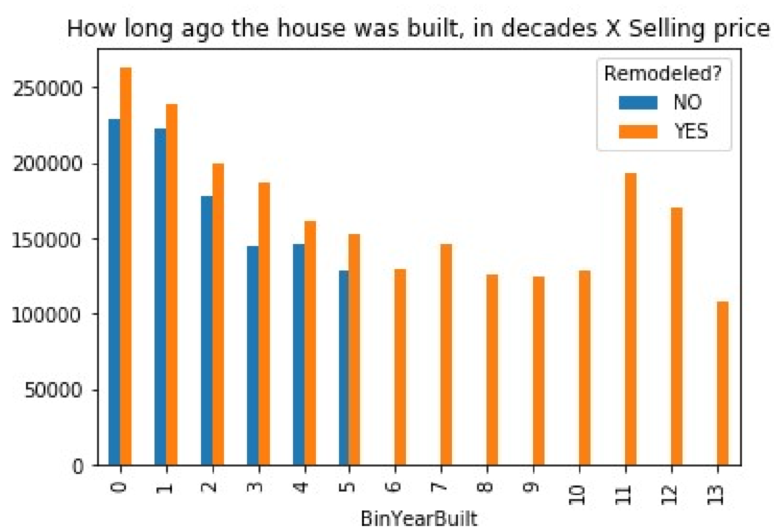


f. Basement improvement

The jump on the average selling price when the basement finished area goes from any category to Good Living Quarters is enormous. It is almost $100,000.00. Which means: if the owner is improving their basement, they better make a great job. Otherwise, the basement may fall into an average quality and they will lose the investment.

g. Difference in price between remodelled houses X non-remodelled houses

We were able to prove that the difference in price of remodelled houses is indeed more expensive than non-remodelled houses, which is the cornerstone of our project. The following plot helps us see that:



Another interesting fact this chart shows is that every house built more than 60 years ago has undergone a house improvement.

Inferential Statistics

The last chart presented in the previous section is by far the most important we produced in this report because it makes the whole project viable. With that in mind, this section will apply hypothesis test to confirm what the data is showing us.

Considering that the smallest difference between the prices was shown for houses in bin 4 (built between 40 - 49 years before selling), if we can prove that the difference in the mean price for houses that were remodelled and houses that were not is statistically significant, the other houses, that fit other bins, will also be significant.

Let us find the p-value associated with the following hypothesis test (1-sided):

For houses built between 40 to 49 years ago:

H0: Mean(remodelled) – Mean(not-remodelled) = 0

H1: Mean(remodelled) – Mean(not-remodelled) > 0

We will estimate the mean and the standard deviation from the sample and use t-statistic.

Using python code,

```
dfBin4 = df[df.BinYearBuilt == 4]
groupRemod = dfBin4[dfBin4['Remodeled?']=='yes']['SalePrice']
groupNoRemod = dfBin4[dfBin4['Remodeled?']=='no']['SalePrice']

import scipy.stats as stats
stats.ttest_ind(a=groupRemod,b=groupNoRemod)
```

```
Ttest_indResult(statistic=2.61985113389767, pvalue=0.009547840858910668)
```

we notice that the p-value is less then 1%, confirming that we should reject our null hypotheses and accept that remodelled houses tend to be sold for more.

Machine Learning Models

To confidently suggest home improvements to homeowners, we need to predict the future value of their property after the job is done. In this section, we are analysing a few machine learning models to help us in this crucial matter. It is important to note that not all models tested are shown in the table for the sake of simplicity, because a lot of parameters tweaking was done to generate this table.

For all models:

- We used all columns: Categorical, numeric and the ones created during visual analysis;
- To use the categorical data, we first added the data from the “test.csv” file to the training data, then we generated dummy columns. Later we removed the test data and trained the model. This way, we guarantee that the columns on both training and testing dataset are the same.

Below is a table that summarizes all our efforts. There are links to the codes in the Appendix, a quick description of each model and their respective scores*.

ID	Model Family	Description	Kaggle Score*	Local Score**
1	LinearRegression()	- The column “LotFrontage” had the missing values filled by an equation determined on Excel; - Data not normalized	0.19883	-
2	LinearRegression()	- The column “LotFrontage” had the missing values filled by an equation determined on Excel; - Data normalized	1.55232	-
3	LinearRegression()	- Column “LotFrontage” calculated with LinearRegression(); - Data not normalized	0.19921	0.09799
4	LassoCV()	CV = 3 Alpha = 100	0.15103	0.87345
5	LassoCV()	CV = 5 Alpha = 110	0.15297	-
6	SVC()	Did not work. Predicted all rows with the same values	-	-
7	LinearSVC()	No parameters tweaking	0.40867	
8	ElasticNet()	-Pipeline with StandardScaler and GridSearchCV - L1_ratio = 0.74	0.15345	0.84387
9	ElasticNet()	- Without StandardScaler - Execution cancelled after 2 hours	-	-
10	RandomForest()	- 'RF__max_depth': 3	0.32349	-
11	RandomForest()	- 'RF__max_depth': 4 GridSearchCV → cv = 5	0.30828	-
12	RandomForest()	- 'RF__max_depth': 7, - 'RF__n_estimators': 110 - GridSearchCV → cv = 5	0.25614	-

13	[RandomForest()]()	- 'RF__max_depth': 8, - 'RF__n_estimators': 200 - GridSearchCV → cv = 5	0.25075	0.01027
14	GradientBoosting()	All parameters default	0.13650	-
15	[GradientBoosting()]()	- 'grad__learning_rage': 0.05 - 'grad__n_estimators': 1000	0.13327	0.90036

*Score obtained by submitting the predictions to the Kaggle competition. The Benchmark score is 0.40890. For the testing dataset, we do not have access to the actual values of the properties selling prices.

**R square scores calculated from the training dataset, where 20% of the data was used as the testing dataset, to compare the predictions with the actual selling prices. Here is the code we used to split the data:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Apr 15 19:47:24 2020
4 
5 @author: Rafael"""
6 
7 import pandas as pd
8 from sklearn.linear_model import LassoCV
9 from sklearn.model_selection import train_test_split
10 
11 #read training file
12 df = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
13 
14 #Separate last column (target)
15 X = df.iloc[:, 0:-1]
16 y = df.iloc[:, -1]
17 
18 #generate dummy columns
19 X = pd.get_dummies(X, drop_first = True)
20 
21 #train test split
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23 
24 #run the model
25 #MY MODEL HERE
26 reg.fit(X_train, y_train)
27 
28 #calculate Score
29 print(reg.score(X_test, y_test))
```

* We replaced line 25 by the models being evaluated to generate the column of local errors.

Exploring the best model

Considering the scores column, our best model is the GradientBoostingRegressor(), ID=15. In the same GitHub repository, there is a file called "Complete Code.py", which goes from reading the csv data until the preparation of the file to be submitted to Kaggle. This code uses this model to perform the predictions.

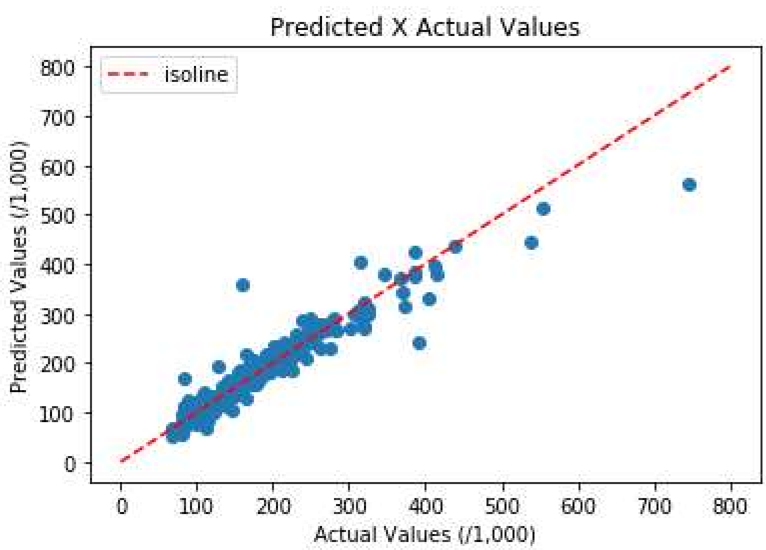
Predictions X Actual values plot

Even though we calculated the R square and the log loss score, it is easier to verify the quality of the predictions with a plot. Considering that we do not have access to the actual values of the testing dataset (Kaggle does not make that data public), we will separate 20 % of the training data to measure the quality of the predictions.

Keeping all modifications made by the data wrangling code, we use the following code to make our plot:

```
#split data
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 0:-1],
                                   df.iloc[:, -1], test_size=0.2, random_state=0)

#steps for the pipeline
steps = [('scaler', StandardScaler()),
('gradient', GradientBoostingRegressor(random_state=0, n_estimators=1000, learning_rate=0.05))]

#Assemble pipeline
pipeline = Pipeline(steps)

#fit the model
pipeline.fit(X_train, y_train)

#scatter plot----------------
#prediction
y_pred = pipeline.predict(X_test)

#scatter
plt.scatter(y_test/1000, y_pred/1000)

#iso line
plt.plot([0,800],[0,800], '--', color='red', label='isoline')

#decorations
plt.xlabel('Actual Values (/1,000)')
plt.xticks(np.arange(0,800,100))
plt.ylabel('Predicted Values (/1,000)')
plt.yticks(np.arange(0,800,100))
plt.title('Predicted X Actual Values')
plt.legend()

plt.show()
```

Which generates the following plot:



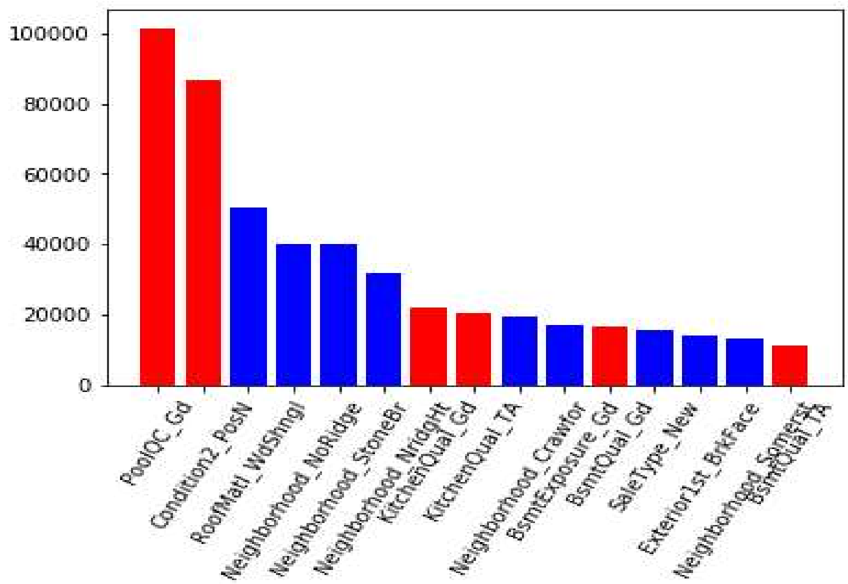
The X axis represents the Actual values and the Y axis represents the Predicted values. The closer the points are to the isoline (red dotted line), the better. Except for the most expensive houses, our predictions are satisfactory.

Most relevant features

We are now interested in checking which features impact the selling price the most. Using this code

```python
df_feat = pd.DataFrame(index=X_aux.columns, data={'Reg_Coef':reg.coef_})
df_feat['Signal'] = df_feat.apply(lambda x: 'Pos' if(x['Reg_Coef']>0) else 'Neg', axis=1)
df_feat['Abs'] = df_feat.apply(lambda x: np.abs(x['Reg_Coef']), axis=1)
df_feat = df_feat.sort_values(by=['Abs'], ascending=False).head(15)

plt.bar(df_feat.index, df_feat.Abs, color=df_feat.Signal.map({'Pos':'blue', 'Neg':'red'}))
plt.xticks(rotation=60)
plt.figure(figsize=(20,40))
plt.show()
```

We generate the following bar chart:



Red and blue columns are the features that affect negatively and positively the Selling Price, respectively.

For our purposes, we can confidently apply this model to potential homeowners willing to sell their property and suggest which home improvement would better increase their selling price.

Conclusion

With the help of charts, we were able to Explore our dataset and expose some interesting findings, the most important of them being the increment on the Selling price of a house after improvement. We were able to prove with 99% confidence that houses that undergo home improvements tend to be sold for more, validating our hypothesis and our project.

Finally, we built a robust Machine Learning model, capable of supporting decisions to homeowners. The model may be applied in specific cases. For example, by simply imputing data about a house, the owner will be able to estimate how much the value of his house will increase in case they finish remodelling their basement, for example. With this increment in mind, the owner may take a proper decision regarding hiring home improvement services to finish their basement.

Ideally, the second part of this project would be to contact home improvement service providers, get a quote for generic jobs, and offer the owners a list of home improvements along with their cost and the expected increment on the final Selling Price. As stated on the introduction, this is not part of the scope of this project.

Appendix

Please find the codes used to test different Machine Learning models

Machine Leaning Codes

3 – LinearRegression()

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df_train = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
df_test = pd.read_csv('DatasetTreated_test.csv', index_col=0, header=0)

X_aux = df_train.iloc[:, 0:-1]
y_train = df_train.iloc[:, -1]

#save the number of the last row in the training dataset. Useful to unappend later
lastRow = X_aux.shape[0]

#append test dataset, so when the dummy columns are generated, any categorical data from the testing dataset will alse be present
X_aux = X_aux.append(df_test)

#generate dummy columns
X_aux = pd.get_dummies(X_aux, drop_first = True)
#X_aux.to_csv("cols_train_all.csv", index=False)

#split back to train/test
X_train = X_aux.iloc[:lastRow,:]
X_test = X_aux.iloc[lastRow:,:]

#generate csvs to check
#X_train.to_csv("cols_train.csv", index=False)
#X_test.to_csv("cols_test.csv", index=False)

reg = LinearRegression()
reg.fit(X_train, y_train)

#print(X_train.columns)
#print(X_test.columns)
#print(reg.predict(X_test))

result = pd.DataFrame({'Id':X_test.index, 'SalePrice': reg.predict(X_test)})
result.to_csv("submission_Rafael3.csv", index=False)
```

5 – LassoCV()

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LassoCV

df_train = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
df_test = pd.read_csv('DatasetTreated_test.csv', index_col=0, header=0)

X_aux = df_train.iloc[:, 0:-1]
y_train = df_train.iloc[:, -1]

#save the number of the last row in the training dataset. Useful to unappend later
lastRow = X_aux.shape[0]

#append test dataset, so when the dummy columns are generated, any categorical data from the testing dataset will alse be present
X_aux = X_aux.append(df_test)

#generate dummy columns
X_aux = pd.get_dummies(X_aux, drop_first = True)
#X_aux.to_csv("cols_train_all.csv", index=False)

#split back to train/test
X_train = X_aux.iloc[:lastRow,:]
X_test = X_aux.iloc[lastRow:,:]

#generate csvs to check
#X_train.to_csv("cols_train.csv", index=False)
#X_test.to_csv("cols_test.csv", index=False)

#reg = LassoCV(cv=5, alphas=[10, 60, 100, 200, 500, 1000], max_iter=10000, random_state=0).fit(X_train, y_train)
reg = LassoCV(cv=5, alphas=[110], max_iter=10000, random_state=0).fit(X_train, y_train)

result = pd.DataFrame({'Id':X_test.index, 'SalePrice': reg.predict(X_test)})
result.to_csv("submission_Rafael_LassoCVBest.csv", index=False)
```

6 - SVC()

```python
@author: Rafael
"""
import pandas as pd
from sklearn.svm import LinearSVC

df_train = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
df_test = pd.read_csv('DatasetTreated_test.csv', index_col=0, header=0)

X_aux = df_train.iloc[:, 0:-1]
y_train = df_train.iloc[:, -1]

#save the number of the last row in the training dataset. Useful to unappend later
lastRow = X_aux.shape[0]

#append test dataset, so when the dummy columns are generated, any categorical data f
X_aux = X_aux.append(df_test)

#generate dummy columns
X_aux = pd.get_dummies(X_aux, drop_first = True)
#X_aux.to_csv("cols_train_all.csv", index=False)

#split back to train/test
X_train = X_aux.iloc[:lastRow,:]
X_test = X_aux.iloc[lastRow:,:]

#generate csvs to check
#X_train.to_csv("cols_train.csv", index=False)
#X_test.to_csv("cols_test.csv", index=False)

reg = LinearSVC(max_iter=10000).fit(X_train, y_train)

result = pd.DataFrame({'Id':X_test.index, 'SalePrice': reg.predict(X_test)})
result.to_csv("submission_Rafael5.csv", index=False)
```

9 - ElasticNet()

```python
@author: Rafael
"""
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import ElasticNet

df_train = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
df_test = pd.read_csv('DatasetTreated_test.csv', index_col=0, header=0)

X_aux = df_train.iloc[:, 0:-1]
y_train = df_train.iloc[:, -1]

#save the number of the last row in the training dataset. Useful to unappend later
lastRow = X_aux.shape[0]

#append test dataset, so when the dummy columns are generated, any categorical data from the testing dataset will alse be present
X_aux = X_aux.append(df_test)

#generate dummy columns
X_aux = pd.get_dummies(X_aux, drop_first = True)
#X_aux.to_csv("cols_train_all.csv", index=False)

#split back to train/test
X_train = X_aux.iloc[:lastRow,:]
X_test = X_aux.iloc[lastRow:,:]

#generate csvs to check
#X_train.to_csv("cols_train.csv", index=False)
#X_test.to_csv("cols_test.csv", index=False)

steps = [('scaler', StandardScaler()), ('elasticnet', ElasticNet(max_iter=100000))]
pipeline = Pipeline(steps)

parameters = {'elasticnet__l1_ratio':np.linspace(0,1,100)}
#parameters = {'l1_ratio':np.linspace(0,1,100)}

reg = GridSearchCV(pipeline, param_grid=parameters)
#reg = GridSearchCV(ElasticNet(max_iter=100000), param_grid=parameters)
reg.fit(X_train, y_train)

#print(X_train.columns)
#print(X_test.columns)
#print(reg.predict(X_test))

result = pd.DataFrame({'Id':X_test.index, 'SalePrice': reg.predict(X_test)})
result.to_csv("submission_Rafael7.csv", index=False)
```

13 - RandomForest()

```python
@author: Rafael
"""
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier

df_train = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
df_test = pd.read_csv('DatasetTreated_test.csv', index_col=0, header=0)

X_aux = df_train.iloc[:, 0:-1]
y_train = df_train.iloc[:, -1]

#save the number of the last row in the training dataset. Useful to unappend later
lastRow = X_aux.shape[0]

#append test dataset, so when the dummy columns are generated, any categorical data from the testing dataset will alse be present
X_aux = X_aux.append(df_test)

#generate dummy columns
X_aux = pd.get_dummies(X_aux, drop_first = True)
#X_aux.to_csv("cols_train_all.csv", index=False)

#split back to train/test
X_train = X_aux.iloc[:lastRow,:]
X_test = X_aux.iloc[lastRow:,:]

#generate csvs to check
#X_train.to_csv("cols_train.csv", index=False)
#X_test.to_csv("cols_test.csv", index=False)

steps = [('scaler', StandardScaler()), ('RF', RandomForestClassifier(random_state=0))]
pipeline = Pipeline(steps)

parameters = {'RF__max_depth':np.arange(2,10), 'RF__n_estimators':np.arange(100,600,100)}
#parameters = {'l1_ratio':np.linspace(0,1,100)}

reg = GridSearchCV(pipeline, param_grid=parameters, cv=5)
#reg = GridSearchCV(ElasticNet(max_iter=100000), param_grid=parameters)
reg.fit(X_train, y_train)

print(reg.best_params_)

#print(X_train.columns)
#print(X_test.columns)
#print(reg.predict(X_test))

result = pd.DataFrame({'Id':X_test.index, 'SalePrice': reg.predict(X_test)})
result.to_csv("submission_Rafael8diff.csv", index=False)
```

15 - GradientBoosting()

```python
@author: Rafael
"""
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingRegressor


df_train = pd.read_csv('DatasetTreated.csv', index_col=0, header=0)
df_test = pd.read_csv('DatasetTreated_test.csv', index_col=0, header=0)

X_aux = df_train.iloc[:, 0:-1]
y_train = df_train.iloc[:, -1]

#save the number of the last row in the training dataset. Useful to unappend later
lastRow = X_aux.shape[0]

#append test dataset, so when the dummy columns are generated, any categorical data from the testing dataset will alse be present
X_aux = X_aux.append(df_test)

#generate dummy columns
X_aux = pd.get_dummies(X_aux, drop_first = True)
#X_aux.to_csv("cols_train_all.csv", index=False)

#split back to train/test
X_train = X_aux.iloc[:lastRow,:]
X_test = X_aux.iloc[lastRow:,:]

#generate csvs to check
#X_train.to_csv("cols_train.csv", index=False)
#X_test.to_csv("cols_test.csv", index=False)
steps = [('scaler', StandardScaler()), ('gradient', GradientBoostingRegressor(random_state=0, learning_rate=0.05, n_estimators=1000))]
pipeline = Pipeline(steps)

#parameters = {'gradient__learning_rate':[0.01, 0.05, 0.1], 'gradient__n_estimators':[500, 1000, 1500, 2000]}
#parameters = {'l1_ratio':np.linspace(0,1,100)}

#reg = GridSearchCV(pipeline, param_grid=parameters, cv=5, scoring='r2')
pipeline.fit(X_train, y_train)

#print(reg.score(X_test, y_test))

#print(X_train.columns)
#print(X_test.columns)
#print(reg.predict(X_test))

result = pd.DataFrame({'Id':X_test.index, 'SalePrice': pipeline.predict(X_test)})
result.to_csv("submission_Rafael_gradBoostCV.csv", index=False)
```

Columns description

MSSubClass: Identifies the type of dwelling involved in the sale.

20 1-STORY 1946 & NEWER ALL STYLES
30 1-STORY 1945 & OLDER
40 1-STORY W/FINISHED ATTIC ALL AGES
45 1-1/2 STORY - UNFINISHED ALL AGES
50 1-1/2 STORY FINISHED ALL AGES
60 2-STORY 1946 & NEWER
70 2-STORY 1945 & OLDER
75 2-1/2 STORY ALL AGES
80 SPLIT OR MULTI-LEVEL
85 SPLIT FOYER
90 DUPLEX - ALL STYLES AND AGES
120 1-STORY PUD (Planned Unit Development) - 1946 & NEWER
150 1-1/2 STORY PUD - ALL AGES
160 2-STORY PUD - 1946 & NEWER
180 PUD - MULTILEVEL - INCL SPLIT LEV/FOYER

190 2 FAMILY CONVERSION - ALL STYLES AND AGES

MSZoning: Identifies the general zoning classification of the sale.

A	Agriculture
C	Commercial
FV	Floating Village Residential
I	Industrial
RH	Residential High Density
RL	Residential Low Density
RP	Residential Low Density Park
RM	Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

GrvlGravel
Pave	Paved

Alley: Type of alley access to property

GrvlGravel
Pave	Paved
NA	No alley access

LotShape: General shape of property

Reg	Regular
IR1	Slightly irregular
IR2	Moderately Irregular
IR3	Irregular

LandContour: Flatness of the property

Lvl	Near Flat/Level
Bnk	Banked - Quick and significant rise from street grade to building
HLS	Hillside - Significant slope from side to side
LowDepression

Utilities: Type of utilities available

AllPub	All public Utilities (E,G,W,& S)
NoSewr	Electricity, Gas, and Water (Septic Tank)
NoSeWa	Electricity and Gas Only
ELO	Electricity only

LotConfig: Lot configuration

Inside	Inside lot
Corner	Corner lot
CulDSac	Cul-de-sac
FR2	Frontage on 2 sides of property
FR3	Frontage on 3 sides of property

LandSlope: Slope of property

Gtl	Gentle slope
Mod	Moderate Slope
Sev	Severe Slope

Neighborhood: Physical locations within Ames city limits

Blmngtn	Bloomington Heights
Blueste	Bluestem
BrDale	Briardale
BrkSide	Brookside
ClearCr	Clear Creek
CollgCr	College Creek
Crawfor	Crawford
Edwards	Edwards
Gilbert	Gilbert
IDOTRR	Iowa DOT and Rail Road
MeadowV	Meadow Village
Mitchel	Mitchell
Names	North Ames
NoRidge	Northridge
NPkVill	Northpark Villa
NridgHt	Northridge Heights
NWAmes	Northwest Ames
OldTown	Old Town
SWISU	South & West of Iowa State University
Sawyer	Sawyer
SawyerW	Sawyer West
Somerst	Somerset
StoneBr	Stone Brook
Timber	Timberland
Veenker	Veenker

Condition1: Proximity to various conditions

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal

RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

BldgType: Type of dwelling

1Fam	Single-family Detached
2FmCon	Two-family Conversion; originally built as one-family dwelling
Duplx	Duplex
TwnhsE	Townhouse End Unit
TwnhsI	Townhouse Inside Unit

HouseStyle: Style of dwelling

1Story	One story
1.5Fin	One and one-half story: 2nd level finished
1.5Unf	One and one-half story: 2nd level unfinished
2Story	Two story
2.5Fin	Two and one-half story: 2nd level finished
2.5Unf	Two and one-half story: 2nd level unfinished
SFoyer	Split Foyer
SLvl	Split Level

OverallQual: Rates the overall material and finish of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average
3	Fair

2 Poor
1 Very Poor

OverallCond: Rates the overall condition of the house

10 Very Excellent
9 Excellent
8 Very Good
7 Good
6 Above Average
5 Average
4 Below Average
3 Fair
2 Poor
1 Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

Flat Flat
Gable Gable
Gambrel Gabrel (Barn)
Hip Hip
Mansard Mansard
Shed Shed

RoofMatl: Roof material

ClyTile Clay or Tile
CompShg Standard (Composite) Shingle
Membran Membrane
Metal Metal
Roll Roll
Tar&Grv Gravel & Tar
WdShake Wood Shakes
WdShngl Wood Shingles

Exterior1st: Exterior covering on house

AsbShng Asbestos Shingles
AsphShn Asphalt Shingles
BrkComm Brick Common
BrkFace Brick Face
CBlock Cinder Block
CemntBd Cement Board

HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles
BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

MasVnrType: Masonry veneer type

BrkCmn	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
None	None
Stone	Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair

Po Poor

ExterCond: Evaluates the present condition of the material on the exterior

Ex Excellent
Gd Good
TA Average/Typical
Fa Fair
Po Poor

Foundation: Type of foundation

BrkTil Brick & Tile
CBlock Cinder Block
PConc Poured Contrete
Slab Slab
Stone Stone
Wood Wood

BsmtQual: Evaluates the height of the basement

Ex Excellent (100+ inches)
Gd Good (90-99 inches)
TA Typical (80-89 inches)
Fa Fair (70-79 inches)
Po Poor (<70 inches
NA No Basement

BsmtCond: Evaluates the general condition of the basement

Ex Excellent
Gd Good
TA Typical - slight dampness allowed
Fa Fair - dampness or some cracking or settling
Po Poor - Severe cracking, settling, or wetness
NA No Basement

BsmtExposure: Refers to walkout or garden level walls

Gd Good Exposure
Av Average Exposure (split levels or foyers typically score average or above)
Mn Mimimum Exposure
No No Exposure
NA No Basement

BsmtFinType1: Rating of basement finished area

GLQ Good Living Quarters

ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

Floor	Floor Furnace
GasA	Gas forced warm air furnace
GasW	Gas hot water or steam heat
Grav	Gravity furnace
OthW	Hot water or steam heat other than gas
Wall	Wall furnace

HeatingQC: Heating quality and condition

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

CentralAir: Central air conditioning

N	No
Y	Yes

Electrical: Electrical system

SBrkr Standard Circuit Breakers & Romex
FuseA Fuse Box over 60 AMP and all Romex wiring (Average)
FuseF 60 AMP Fuse Box and mostly Romex wiring (Fair)
FuseP 60 AMP Fuse Box and mostly knob & tube wiring (poor)
Mix Mixed

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

Ex Excellent
Gd Good
TA Typical/Average
Fa Fair
Po Poor

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Typ Typical Functionality
Min1 Minor Deductions 1
Min2 Minor Deductions 2
Mod Moderate Deductions
Maj1 Major Deductions 1
Maj2 Major Deductions 2
Sev Severely Damaged
Sal Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

Ex Excellent - Exceptional Masonry Fireplace
Gd Good - Masonry Fireplace in main level
TA Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement
Fa Fair - Prefabricated Fireplace in basement
Po Poor - Ben Franklin Stove
NA No Fireplace

GarageType: Garage location

2Types More than one type of garage
Attchd Attached to home
Basment Basement Garage
BuiltIn Built-In (Garage part of house - typically has room above garage)
CarPort Car Port
Detchd Detached from home
NA No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

Fin Finished
RFn Rough Finished
Unf Unfinished
NA No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

Ex Excellent
Gd Good
TA Typical/Average
Fa Fair
Po Poor
NA No Garage

GarageCond: Garage condition

Ex Excellent
Gd Good
TA Typical/Average

Fa	Fair
Po	Poor
NA	No Garage

PavedDrive: Paved driveway

Y	Paved
P	Partial Pavement
N	Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

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

Fence: Fence quality

GdPrv	Good Privacy
MnPrv	Minimum Privacy
GdWo	Good Wood
MnWw	Minimum Wood/Wire
NA	No Fence

MiscFeature: Miscellaneous feature not covered in other categories

ElevElevator
Gar2	2nd Garage (if not described in garage section)
Othr	Other
Shed	Shed (over 100 SF)
TenC	Tennis Court
NA	None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

WD	Warranty Deed - Conventional
CWD	Warranty Deed - Cash
VWD	Warranty Deed - VA Loan
New	Home just constructed and sold
COD	Court Officer Deed/Estate
Con	Contract 15% Down payment regular terms
ConLw	Contract Low Down payment and low interest
ConLI	Contract Low Interest
ConLD	Contract Low Down
Oth	Other

SaleCondition: Condition of sale

Normal	Normal Sale
Abnorml	Abnormal Sale - trade, foreclosure, short sale
AdjLand	Adjoining Land Purchase
Alloca	Allocation - two linked properties with separate deeds, typically condo with a garage unit
Family	Sale between family members
Partial	Home was not completed when last assessed (associated with New Homes)