House Prices - Advanced Regression Techniques

Alvaro Ronquillo - Karan Padhiyar - Paola Toscano - Patricio Helfer - Santiago Crane \ Business Challenge #2 \ Proffesor: Thomas Kurnicki \ February 26th, 2023

Overview

The real estate market in the US has been affected by a variety of factors in recent years, including the COVID-19 pandemic, economic fluctuations, and changing consumer preferences. However, a home's price is determined by many factors beyond just the number of bedrooms and a white-picket fence. According to a dataset from a playground competition hosted by Kaggle, the final price of a home is influenced by 79 explanatory variables, such as the height of the basement ceiling and proximity to an east-west railroad. The dataset challenges participants to predict the final price of residential homes in Ames, Iowa, using advanced regression techniques like random forest and gradient boosting.

In this analysis, our team will provide our top three actionable insights based on our predictive model, along with a conclusion, credible sources, analysis outputs, and data visualizations. We will strive to make our analysis accessible and meaningful to a non-technical audience.

Insights

Investment Oportunities

Even while it isn't perfect, the model may provide a fairly accurate assessment of a home's value, which can then be used to find possibilities for a quick, low-risk return on investment. For example based on the characteristics of the property and the state of the market, we can use the forecasting model to estimate the sale prices of various homes in a given neighborhood. An investor can then select properties that are inexpensive in comparison to nearby properties of a similar caliber or properties with a strong growth potential as a result of shifting market dynamics or neighborhood growth by studying the predictions. This could be very useful for new real estate agents who are not yet familiar with the local market, as they could rely on the model to establish themselves as good agents, without having to rely on the expertise of others, which could lead to a reduction in their commission fees.

High Value Features

We can determine which aspects of a home—other than its size and number of bedrooms—buyers value most. For example, if the model predicts that a properties with high proximity to schools are in demand, an investor can target properties with those features to increase the potential for a high return. By understanding which features have the most impact on the property valuation, investors and potential homeowners can make informed decisions about where to prioritize the investments, whether it be buying a

built property, building one from the ground or making any kind of renovation to an existing property. Moreover, investors can be informed about which properties to purchase and for the right price. By identifying undervalued or high-growth potential properties, investors can increase their chances of earning a high return on their investment.

Landlord – Accurate Pricing

When one rents a property, they earn income from it in the form of rent. However, at the same time due to renting a landlord is missing out on the opportunity of selling it and fetching a lumpsum amount. Hence, it is really important for a landlord to be able to set the rent amount in such a way that it covers for the opportunity cost in the form of sales price of the house. This is possible if the landlord can determine the fair value of the property. The fair value of the property is nothing but the amount the property will fetch when sold in a market. Our model helps in predicting the sales price of the property given a certain features in the property. Generally, rent amount is set at a certain percentage of the fair value of the property. However, care must be taken to not use this as the only source of determining the rent. There are a few variables which can affect the rent of the property, such as: 1) Consider the demand and supply of the property. If the demand is high the higher rent could be set and if the demand is low lower rent would be required to be set. 2) One should also consider the rent of similar properties in the immediate vicinity of the property.

Assumptions

Before evaluating the model, it is important to acknowledge the assumptions made regarding the data and results. Regarding the data, it is important to note that the timestamp of data generation or any other information is not available. Therefore, we assume that the data is from the current year, which would enable our model to consider the latest trends in housing valuations. This assumption can be seen in our code, as we create a variable for the Age of the House and we take 2023 as the current year. Additionally, we assume that the database contains all relevant features that real estate properties may have. On the other hand, the model lacks a location variable due to its absence from the database. Therefore, we assume that the model can be applied generally to any real estate market, regardless of the specific location.

Limitations

1) Given the number of outliers, some of this data may be wrong, but we just had to assume it wasn't. One of the limitations for this dataset was that there was no way to verify it.

2) There were too few data points, which in the end meant that in a lot of cases some of the independent variables didn't have all their possible values represented in a statistically significant way if at all. For example the amount of neighborhoods was incredibly limited which means that one of the most important variables, location, was really lacking. This means that the model is not generalizable.

3) Another limitation from the data could be that even though 79 columns seems like a lot, many very important factors such as how good are the nearby schools, or interest rates from banks are left out from

the dataset. This factor greatly influence housing prices but are left out of the data.

4) Other problem is that a lot of the variables were not normally distributed, in the end it was important to handle those variables by transforming them into normally distributed data but that transformation sometimes introduces errors that could mess with the accuracy of the model.

Secunadry Research

What is affecting the real estate market in the US.?

Real estate is known for being a market that fluctuates and is continuously changing, not just in the US but also globally. The following factors are those that have the most impact on this market (Nguyen, 2021):

- Economic expansion
- Demographics
- Governmental directives
- Rates of interest

Changes in interest rates can have a substantial impact on a person's capacity to purchase a residential home. This is primarily due to the fact that getting a mortgage to buy a property will become less expensive when interest rates decline. The demand for real estate will increase as a result, raising prices. Demographics are therefore a significant role as well. They outline the preferences of the populace and what they seek in terms of housing kind and price, having an impact both now and in the future. Economic expansion also has an impact on the market. The primary factors influencing housing value are the GDP, employment, and prices for goods and services (Nguyen, 2021).

Real Estate Market in Boston and Cambridge

Sales of single-family homes and condominiums were down both annually and monthly in October compared to the same time last year, signaling a cooling in the Greater Boston housing market. Sales of single-family homes fell by 16.3% year over year while those of condominiums fell by 22.9%. Although the monthly median selling price of a single-family home has decreased 6.7% from its peak in June and the monthly median price of a condominium fell 0.8% from its high in April, both the median selling prices for single-family homes and condominiums hit new record highs for October. (Santarelli, 2022)

Trends in the real state business REIT

Several trends in the real estate industry have gained popularity during the past year. As stated by Real estate experts are looking at properties long-term despite cyclical difficulties including rising interest rates, a weakening economy, and less deal volume. They want to survive the current recession and prepare their companies for a new phase of steady growth and good profits, therefore they are cautiously optimistic. While certain areas of the industry are returning to normal, the pandemic caused long-term changes in others. Companies must be flexible and sensitive to changes in the market. (2023 Carlock)

Conclusion

Collinearity refers to a high degree of correlation between two or more independent variables in a regression model. It's standard procedure to remove this values as it makes interpreting the coefficients difficult and removes statistical significance for each independent. However, sometimes the goal of a regression model is not to understand the relationship between the independent variables and the target variable, but rather to simply make accurate predictions.

Given the nature of this challenge there was no need to remove collinearity from the model, especially when doing so lead to a decrease in the model's predictive accuracy reflected by our Kaggle score when we removed some correlated independent variables from the model. This suggested that those variables were actually providing useful information for the model's prediction accuracy, despite their collinearity with other variables.

Therefore, when the goal is to build a predictive model and the collinearity among independent variables does not lead to any major issues with the model's performance, there is no need to remove collinearity. It is important to keep in mind, however, that if the goal of the model is to understand the relationship between the independent variables and the dependent variable, then collinearity should be addressed in order to obtain reliable and interpretable coefficients.

Data analysis (EDA + Graphs + feature engineering)

Intalations and imports

In [1]:
```
# Installing required modules/models
%pip install lightgbm
%pip install catboost
%pip install xgboost


# Importing libraries
import numpy as np
import pandas as pd

from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from scipy.stats import skew

from sklearn.linear_model import Ridge, ElasticNet,  Lasso
from sklearn.kernel_ridge import KernelRidge
from sklearn.ensemble import GradientBoostingRegressor
from catboost import CatBoostRegressor
from datetime import datetime
import lightgbm
from lightgbm import LGBMRegressor
import xgboost as xg

import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: lightgbm in c:\users\santi\appdata\roaming\python\python39\site-p
ackages (3.3.5)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from lightgb
m) (1.9.1)
Requirement already satisfied: wheel in c:\programdata\anaconda3\lib\site-packages (from lightgb
m) (0.37.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from lightgb
m) (1.21.5)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\programdata\anaconda3\lib\site-package
s (from lightgbm) (1.0.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-package
s (from scikit-learn!=0.22.0->lightgbm) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\santi\appdata\roaming\python\python39\si
te-packages (from scikit-learn!=0.22.0->lightgbm) (1.2.0)
Note: you may need to restart the kernel to use updated packages.
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: catboost in c:\users\santi\appdata\roaming\python\python39\site-p
ackages (1.1.1)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from ca
tboost) (3.5.2)
Requirement already satisfied: plotly in c:\programdata\anaconda3\lib\site-packages (from catboo
st) (5.9.0)
Requirement already satisfied: pandas>=0.24.0 in c:\programdata\anaconda3\lib\site-packages (fro
m catboost) (1.4.4)
Requirement already satisfied: numpy>=1.16.0 in c:\programdata\anaconda3\lib\site-packages (from
catboost) (1.21.5)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from catboost)
(1.16.0)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from catboos
t) (1.9.1)
Requirement already satisfied: graphviz in c:\users\santi\appdata\roaming\python\python39\site-p
ackages (from catboost) (0.20.1)
Requirement already satisfied: pytz>=2020.1 in c:\users\santi\appdata\roaming\python\python39\si
te-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-packa
ges (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages
(from matplotlib->catboost) (4.25.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\programdata\anaconda3\lib\site-packages (f
rom matplotlib->catboost) (3.0.9)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from
matplotlib->catboost) (0.11.0)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (fr
om matplotlib->catboost) (21.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages
(from matplotlib->catboost) (1.4.2)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from
matplotlib->catboost) (9.2.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (fr
om plotly->catboost) (8.0.1)
Note: you may need to restart the kernel to use updated packages.
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: xgboost in c:\programdata\anaconda3\lib\site-packages (1.7.4)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboos
t) (1.21.5)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboos
t) (1.9.1)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
# Data import and df creation
data = pd.read_csv('train.csv')
```

```
test = pd.read_csv('test.csv')
data_b = data.drop(['Id'], axis=1)
test_b = test.drop(['Id'], axis=1) # We are removing the ID Index
```

EDA

In [3]:
```
data_b.head()
```

Out[3]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...

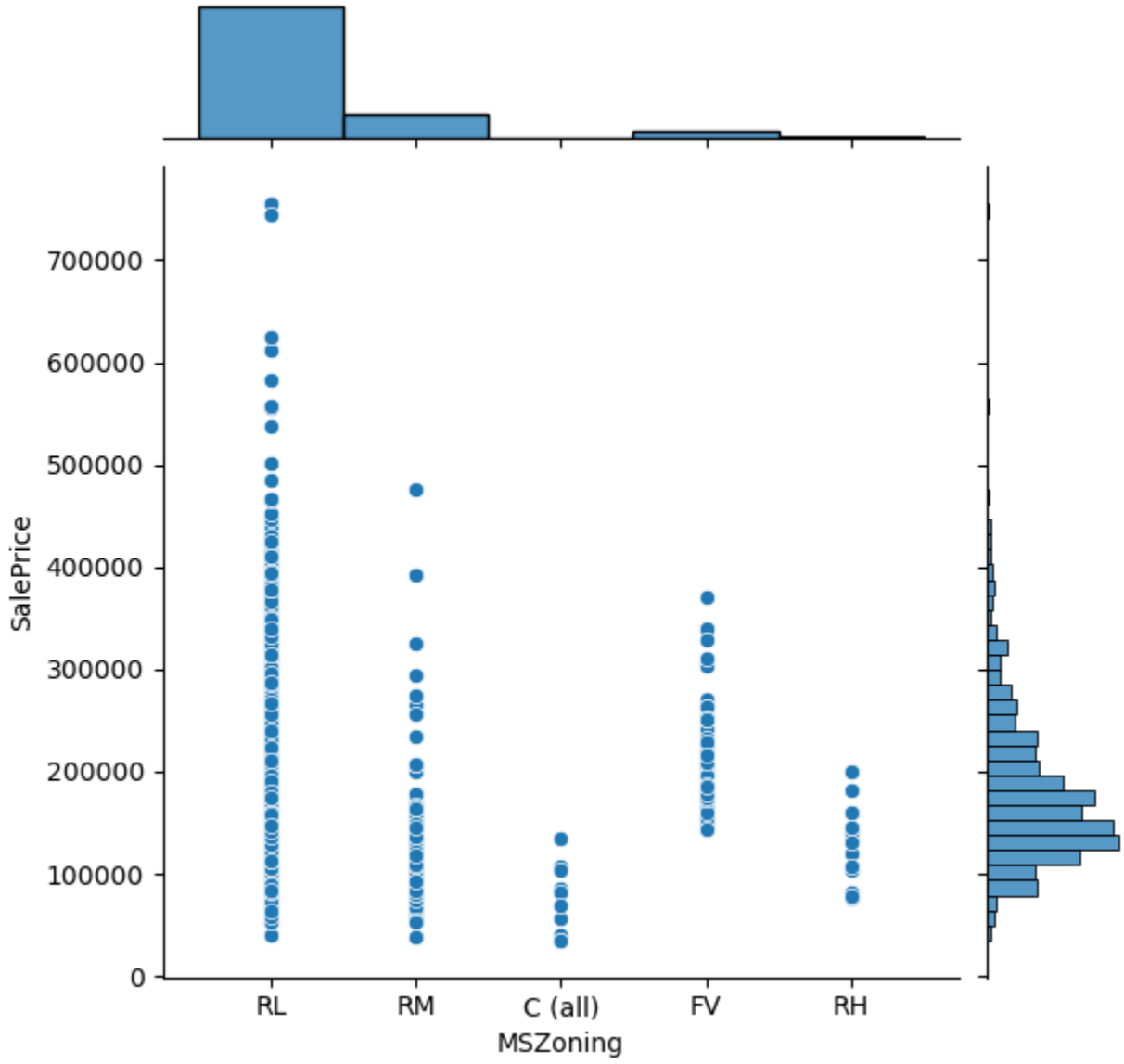
5 rows × 80 columns

In [4]:
```
# Analyse missing Values

data_b.isnull().sum().sort_values(ascending=False).head(20)
```

Out[4]:
```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageYrBlt       81
GarageCond        81
GarageType        81
GarageFinish      81
GarageQual        81
BsmtExposure      38
BsmtFinType2      38
BsmtCond          37
BsmtQual          37
BsmtFinType1      37
MasVnrArea         8
MasVnrType         8
Electrical         1
MSSubClass         0
dtype: int64
```

In [5]:
```
# We analysed all features by user difined functions
# Plotting a joint plot to see outliers

def analyse_columns(feature_name):

    sns.jointplot(data=data_b, y="SalePrice", x=feature_name)
    plt.show()
    print("\n****INFO****")
    print(data_b[feature_name].describe())
    print("\n****VALUE COUNTS****")
    print(data_b[feature_name].value_counts())
    print("\n****VALUE AVG SALE PRICE****")
```

```
    print(data_b.groupby(feature_name)['SalePrice'].mean())
    if data_b[feature_name].dtype!="O":
        print("\nSkewness:",str(skew(data_b[feature_name])))

    print("\n****TEST INFO****")
    print(test_b[feature_name].describe())
    print("\n****VALUE COUNTS****")
    print(test_b[feature_name].value_counts())

    print("\nOnly in Train: "+str(list(set(data_b[feature_name].value_counts().index.values) - s
    print("Only in Test: "+ str(list(set(test_b[feature_name].value_counts().index.values) - set

analyse_columns("MSZoning")
```



```
****INFO****
count     1460
unique       5
top         RL
freq      1151
Name: MSZoning, dtype: object

****VALUE COUNTS****
RL         1151
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64

****VALUE AVG SALE PRICE****
MSZoning
C (all)     74528.000000
FV         214014.061538
RH         131558.375000
RL         191004.994787
RM         126316.830275
Name: SalePrice, dtype: float64

****TEST INFO****
count     1455
unique       5
top         RL
freq      1114
Name: MSZoning, dtype: object

****VALUE COUNTS****
RL         1114
RM          242
FV           74
C (all)      15
RH           10
Name: MSZoning, dtype: int64

Only in Train: []
Only in Test: []
```

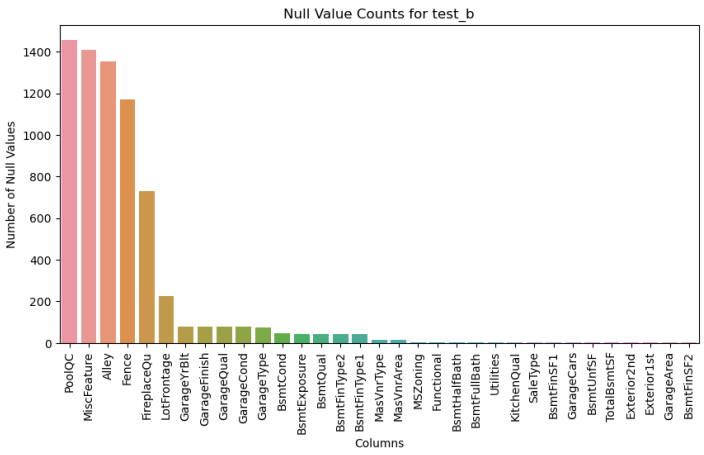
Replacing Null Values

Null or missing values in a dataset can adversely affect the accuracy of machine learning models. This is because many machine learning algorithms cannot handle missing data and may fail to run if null values are present in the dataset.

Furthermore, if null values are not handled properly, they can bias the results of the analysis and lead to incorrect conclusions. For example, if a large number of null values are present in a specific column, this may cause the algorithm to ignore that column altogether, even though it may contain important information.

Therefore, it is essential to fill the null values with appropriate values before training the models. In this chapter, the null values will be filled with either categorical or numerical values, depending on the nature of the column. For categorical columns, null values will be replaced with the mode value, which is the most frequently occurring value in the column. For numerical columns, null values will be replaced with the median value, which is the middle value of the column.

In [6]:
```
null_counts = test_b.isnull().sum().sort_values(ascending=False)
null_counts = null_counts[null_counts != 0]  # remove columns with zero null values
plt.figure(figsize=(10, 5))
sns.barplot(x=null_counts.index, y=null_counts.values)
plt.xticks(rotation=90)
plt.xlabel('Columns')
plt.ylabel('Number of Null Values')
plt.title('Null Value Counts for test_b')
#Printed text to me input on the side of the plot
plt.text(0.95, 0.01, """
The given dataframe has a total of 1460 entries and 80 columns. \n
Among these columns, the columns "LotFrontage", "Alley", "MasVnrType",\n
"MasVnrArea", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", \n
"BsmtFinType2", "Electrical", "FireplaceQu", "GarageType", "GarageYrBlt",\n
"GarageFinish", "GarageQual", "GarageCond", "PoolQC", and "Fence" contain\n
null values. The total number of null values in the dataframe is 6965, which\n
is 5.8% of the total number of entries in the dataframe. The highest number\n
of null values are in the "PoolQC" column, with only 7 non-null entries.

The null values in the other columns can be handled by using appropriate techniques\n
such as imputation, deletion or replacement. The presence of null values in the\n
dataset can impact the quality of the analysis, so it is important to handle \n
them appropriately.\n
""",fontsize=13, transform=plt.gcf().transFigure)
plt.show()
```



The given dataframe has a total of 1460 entries and 80 columns.
Among these columns, the columns "LotFrontage", "Alley", "MasVnrType", "MasVnrArea", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2", "Electrical", "FireplaceQu", "GarageType", "GarageYrBlt", "GarageFinish", "GarageQual", "GarageCond", "PoolQC", and "Fence" contain null values. The total number of null values in the dataframe is 6965, which is 5.8% of the total number of entries in the dataframe. The highest number of null values are in the "PoolQC" column, with only 7 non-null entries.
The null values in the other columns can be handled by using appropriate techniques such as imputation, deletion or replacement. The presence of null values in the dataset can impact the quality of the analysis, so it is important to handle them appropriately.

In [7]:
```
# FILL MISSING VALUES AFTER ANALYSING EACH COLUMN
test_b['MSZoning']=test_b['MSZoning'].fillna("C (all)")
test_b['GarageCars']=test_b['GarageCars'].fillna(0)
test_b['GarageArea']=test_b['GarageArea'].fillna(0)
test_b['Functional']=test_b['Functional'].fillna("Typ")
test_b['SaleType']=test_b['SaleType'].fillna("WD")
test_b['SaleCondition']=test_b['SaleCondition'].fillna("Normal")
data_b['Fence']=data_b['Fence'].fillna("None")
test_b['Fence']=test_b['Fence'].fillna("None")
data_b['Electrical'] = data_b['Electrical'].fillna("SBrkr")
data_b['FireplaceQu'] = data_b['FireplaceQu'].fillna("None")
test_b['FireplaceQu'] = test_b['FireplaceQu'].fillna("None")
data_b['GarageType'] = data_b['GarageType'].fillna("None")
test_b['GarageType'] = test_b['GarageType'].fillna("None")
data_b['GarageQual'] = data_b['GarageQual'].fillna("None")
test_b['GarageQual'] = test_b['GarageQual'].fillna("None")
data_b['GarageCond'] = data_b['GarageCond'].fillna("None")
test_b['GarageCond'] = test_b['GarageCond'].fillna("None")
```

```
data_b['GarageFinish'] = data_b['GarageFinish'].fillna("None")
test_b['GarageFinish'] = test_b['GarageFinish'].fillna("None")
test_b['Exterior1st'] = test_b['Exterior1st'].fillna("VinylSd")
test_b['Exterior2nd']= test_b['Exterior2nd'].fillna("VinylSd")
data_b['MasVnrType'] = data_b['MasVnrType'].fillna("None")
test_b['MasVnrType'] = test_b['MasVnrType'].fillna("None")
data_b['MasVnrArea'] = data_b['MasVnrArea'].fillna(0)
test_b['MasVnrArea'] = test_b['MasVnrArea'].fillna(0)
test_b['BsmtHalfBath']=test_b['BsmtHalfBath'].fillna(0)
test_b['BsmtFullBath']=test_b['BsmtFullBath'].fillna(0)
test_b['KitchenQual']=test_b['KitchenQual'].fillna("Gd")
test_b['TotalBsmtSF']=test_b['TotalBsmtSF'].fillna(0)
test_b['BsmtUnfSF']=test_b['BsmtUnfSF'].fillna(0)
test_b['BsmtFinSF1']=test_b['BsmtFinSF1'].fillna(0)
test_b['BsmtFinSF2']= test_b['BsmtFinSF2'].fillna(0)
test_b['BsmtQual']=test_b['BsmtQual'].fillna("None")
data_b['BsmtQual']=data_b['BsmtQual'].fillna("None")
test_b['BsmtCond']= test_b['BsmtCond'].fillna("None")
data_b['BsmtCond']= data_b['BsmtCond'].fillna("None")
test_b['BsmtExposure']= test_b['BsmtExposure'].fillna("None")
data_b['BsmtExposure']= data_b['BsmtExposure'].fillna("None")
test_b['Utilities']= test_b['Utilities'].fillna("AllPub")
data_b['GarageYrBlt'] = data_b['GarageYrBlt'].fillna(2000)
test_b['GarageYrBlt'] = test_b['GarageYrBlt'].fillna(2000)
```

In [8]:

```
test_b.describe().T
```

Out[8]:

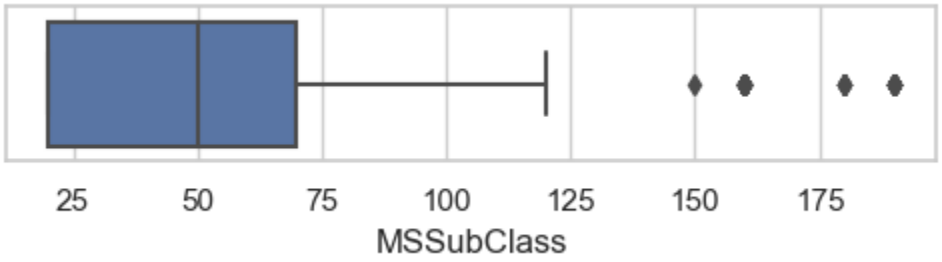
	count	mean	std	min	25%	50%	75%	max
MSSubClass	1459.0	57.378341	42.746880	20.0	20.0	50.0	70.0	190.0
LotFrontage	1232.0	68.580357	22.376841	21.0	58.0	67.0	80.0	200.0
LotArea	1459.0	9819.161069	4955.517327	1470.0	7391.0	9399.0	11517.5	56600.0
OverallQual	1459.0	6.078821	1.436812	1.0	5.0	6.0	7.0	10.0
OverallCond	1459.0	5.553804	1.113740	1.0	5.0	5.0	6.0	9.0
YearBuilt	1459.0	1971.357779	30.390071	1879.0	1953.0	1973.0	2001.0	2010.0
YearRemodAdd	1459.0	1983.662783	21.130467	1950.0	1963.0	1992.0	2004.0	2010.0
MasVnrArea	1459.0	99.673749	177.001792	0.0	0.0	0.0	162.0	1290.0
BsmtFinSF1	1459.0	438.902673	455.257119	0.0	0.0	350.0	752.0	4010.0
BsmtFinSF2	1459.0	52.583276	176.698671	0.0	0.0	0.0	0.0	1526.0
BsmtUnfSF	1459.0	553.915010	437.351324	0.0	219.0	460.0	797.5	2140.0
TotalBsmtSF	1459.0	1045.400960	443.592976	0.0	784.0	988.0	1304.0	5095.0
1stFlrSF	1459.0	1156.534613	398.165820	407.0	873.5	1079.0	1382.5	5095.0
2ndFlrSF	1459.0	325.967786	420.610226	0.0	0.0	0.0	676.0	1862.0
LowQualFinSF	1459.0	3.543523	44.043251	0.0	0.0	0.0	0.0	1064.0
GrLivArea	1459.0	1486.045922	485.566099	407.0	1117.5	1432.0	1721.0	5095.0
BsmtFullBath	1459.0	0.433859	0.530527	0.0	0.0	0.0	1.0	3.0
BsmtHalfBath	1459.0	0.065113	0.252307	0.0	0.0	0.0	0.0	2.0
FullBath	1459.0	1.570939	0.555190	0.0	1.0	2.0	2.0	4.0
HalfBath	1459.0	0.377656	0.503017	0.0	0.0	0.0	1.0	2.0
BedroomAbvGr	1459.0	2.854010	0.829788	0.0	2.0	3.0	3.0	6.0
KitchenAbvGr	1459.0	1.042495	0.208472	0.0	1.0	1.0	1.0	2.0
TotRmsAbvGrd	1459.0	6.385195	1.508895	3.0	5.0	6.0	7.0	15.0
Fireplaces	1459.0	0.581220	0.647420	0.0	0.0	0.0	1.0	4.0
GarageYrBlt	1459.0	1978.912269	26.198603	1895.0	1960.5	1982.0	2001.0	2207.0
GarageCars	1459.0	1.764907	0.777056	0.0	1.0	2.0	2.0	5.0
GarageArea	1459.0	472.444825	217.326902	0.0	317.5	480.0	576.0	1488.0
WoodDeckSF	1459.0	93.174777	127.744882	0.0	0.0	0.0	168.0	1424.0
OpenPorchSF	1459.0	48.313914	68.883364	0.0	0.0	28.0	72.0	742.0
EnclosedPorch	1459.0	24.243317	67.227765	0.0	0.0	0.0	0.0	1012.0
3SsnPorch	1459.0	1.794380	20.207842	0.0	0.0	0.0	0.0	360.0
ScreenPorch	1459.0	17.064428	56.609763	0.0	0.0	0.0	0.0	576.0
PoolArea	1459.0	1.744345	30.491646	0.0	0.0	0.0	0.0	800.0
MiscVal	1459.0	58.167923	630.806978	0.0	0.0	0.0	0.0	17000.0
MoSold	1459.0	6.104181	2.722432	1.0	4.0	6.0	8.0	12.0

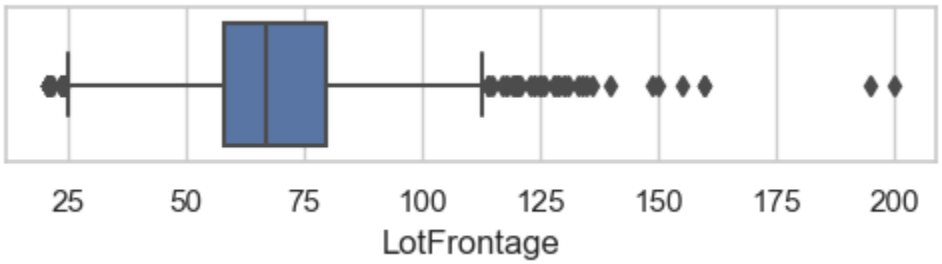
	count	mean	std	min	25%	50%	75%	max
YrSold	1459.0	2007.769705	1.301740	2006.0	2007.0	2008.0	2009.0	2010.0

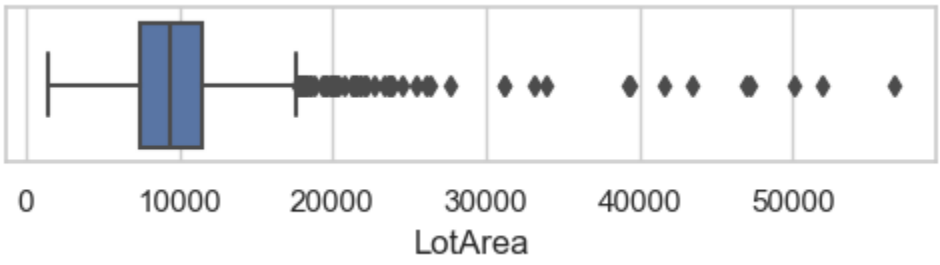
Outliers

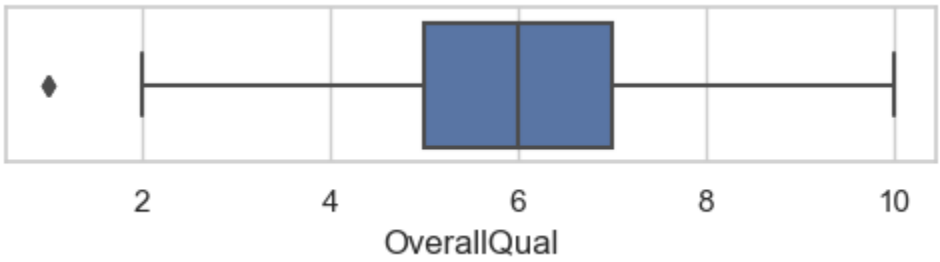
We decided to try and identify the outliers in the model as they can have a significant impact on the estimated regression coefficients, which could lead to biased and unreliable results. To do this we decided to use boxplot graphs, to quickly identify which variables had outliers way out of the quartiles of the whiskers of the boxplot. To identify outliers using a boxplot graph, you need to look for points that lie outside of the whiskers. Since removing the outliers could affect the results of our analysis (as we assumed that this data points were not measurement errors, data entry or poor sampling), after identifying the outliers using a boxplot graph, we decided that instead of taking the data points out, we created new data that basically signified a “larger than”, that replaced all the outliers of a specific variable into a single new value. For example for the total lot area we replaced the values larger than 40 thousand square feet, which were just 14 values in total which varied way to much, and changed them all to 40 thousand.

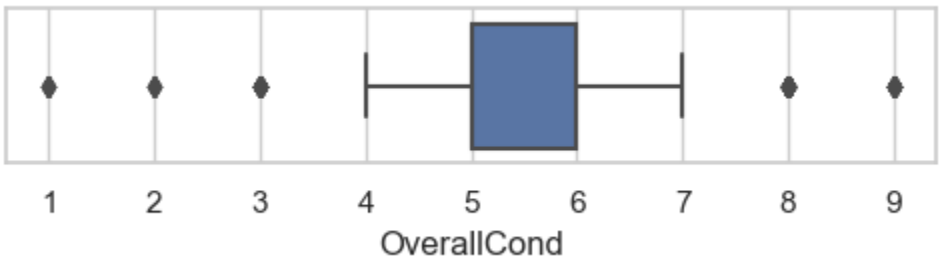
In [29]:
```
numerical_df = test.select_dtypes(exclude=['object'])
#numerical_df = numerical_df.drop(["Id"], axis=1)
for column in numerical_df:
    plt.figure(figsize=(6, 1))
    sns.set_theme(style="whitegrid")
    sns.boxplot(numerical_df[column])
```

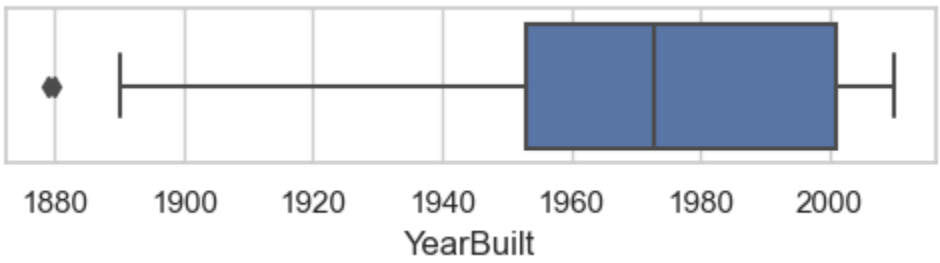


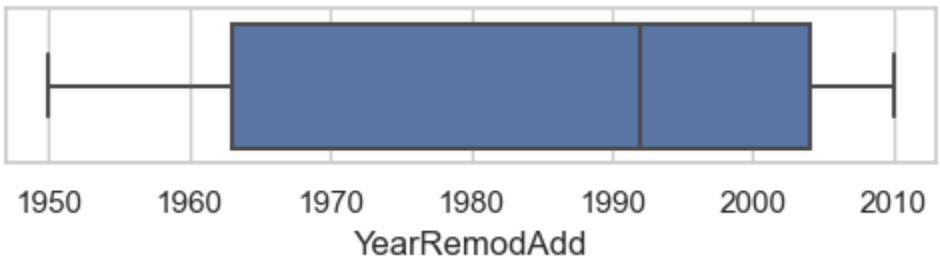


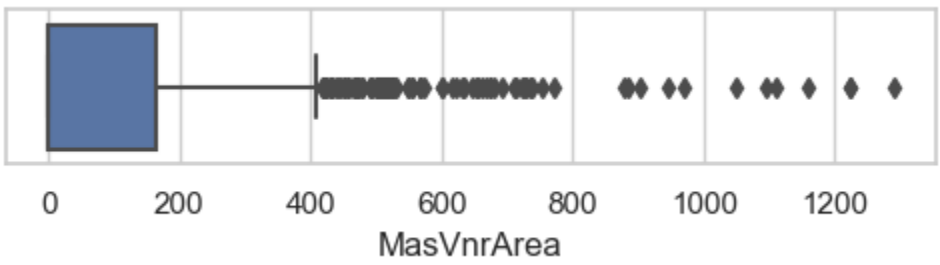

0
10000
20000
30000
40000
50000
LotArea

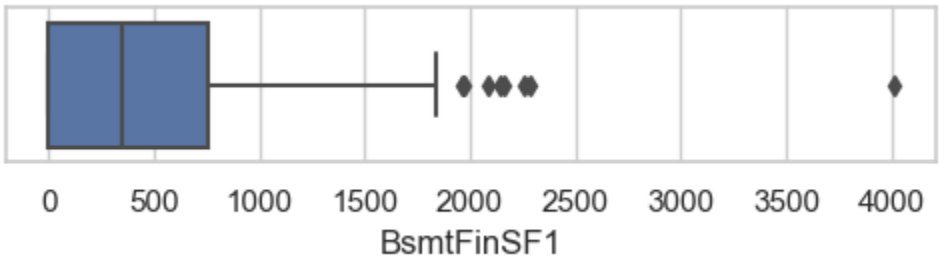

2
4
6
8
10
OverallQual

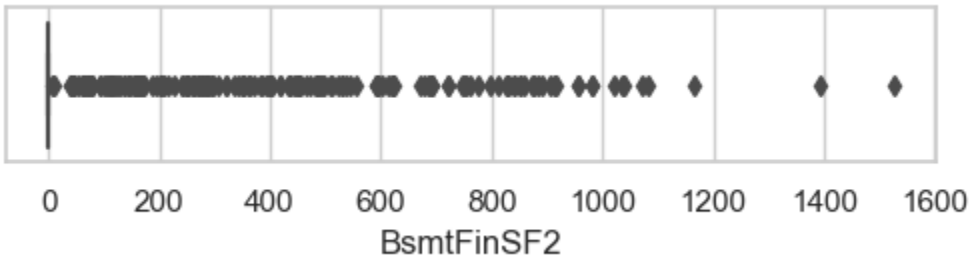

1
2
3
4
5
6
7
8
9
OverallCond

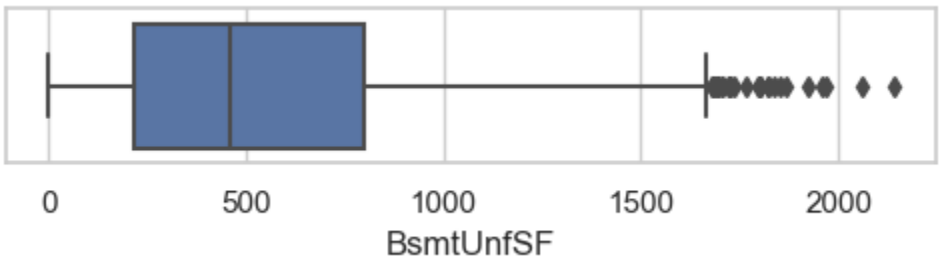

1880
1900
1920
1940
1960
1980
2000
YearBuilt

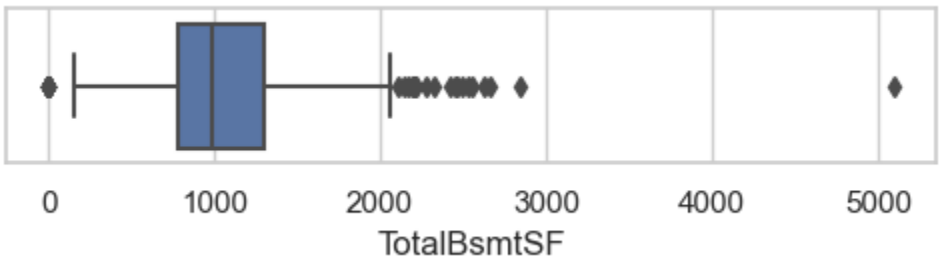

1950
1960
1970
1980
1990
2000
2010
YearRemodAdd

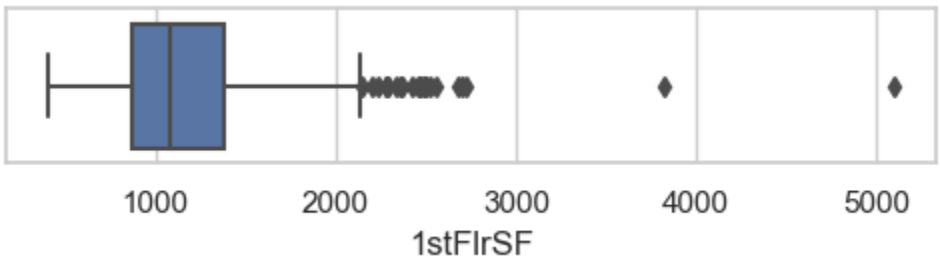

0
200
400
600
800
1000
1200
MasVnrArea

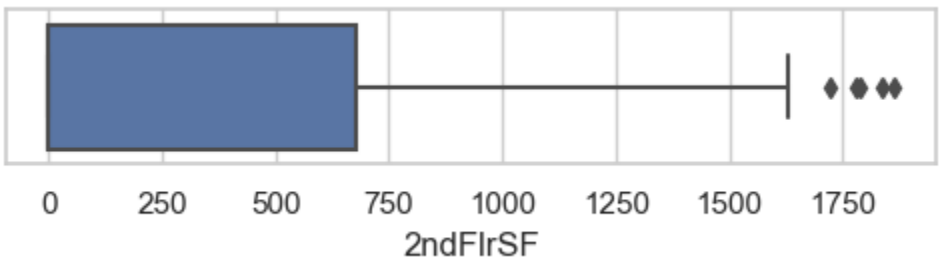

0
500
1000
1500
2000
2500
3000
3500
4000
BsmtFinSF1


0
200
400
600
800
1000
1200
1400
1600
BsmtFinSF2

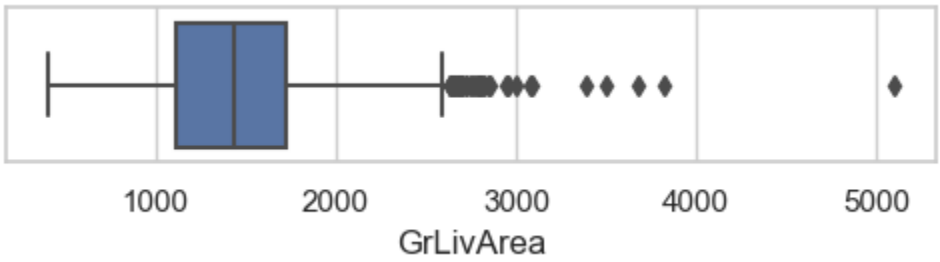

0
500
1000
1500
2000
BsmtUnfSF

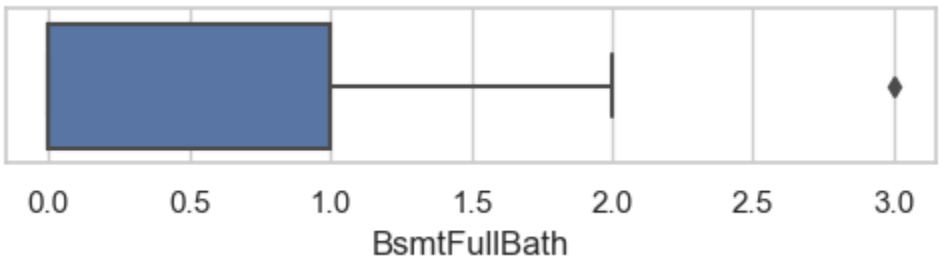

0
1000
2000
3000
4000
5000
TotalBsmtSF

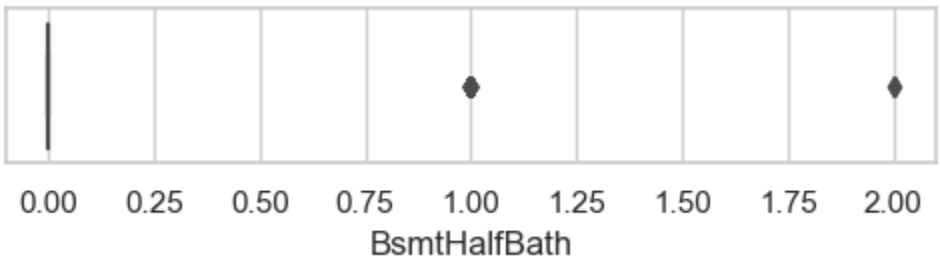

1000
2000
3000
4000
5000
1stFlrSF

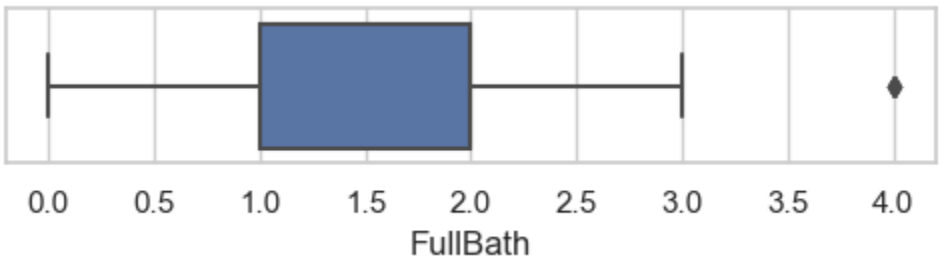

0
250
500
750
1000
1250
1500
1750
2ndFlrSF

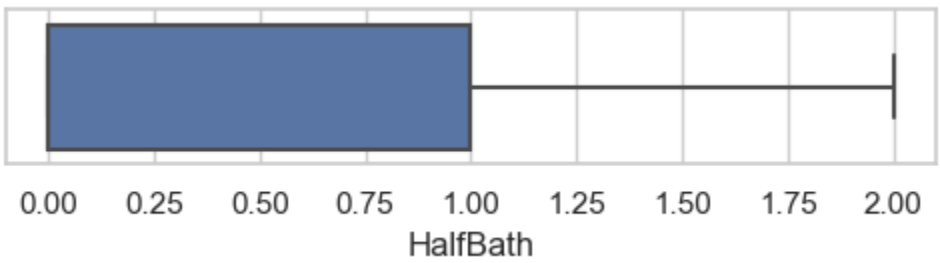

0
200
400
600
800
1000
LowQualFinSF

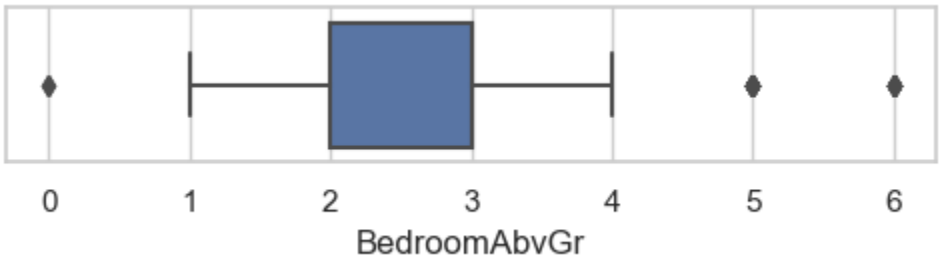

1000
2000
3000
4000
5000
GrLivArea


0.0
0.5
1.0
1.5
2.0
2.5
3.0
BsmtFullBath

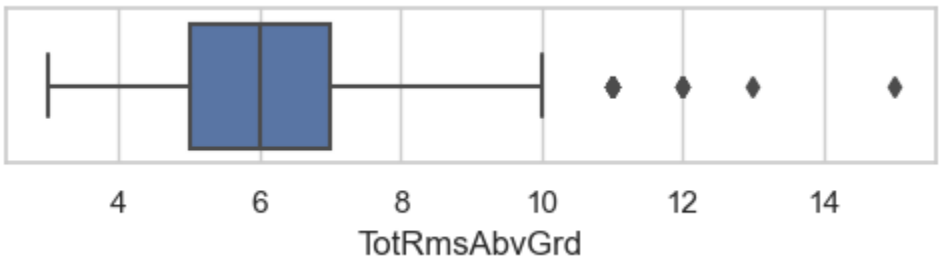

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
BsmtHalfBath

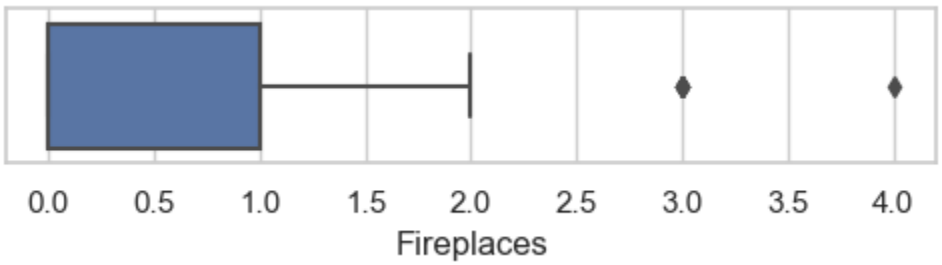

0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
FullBath

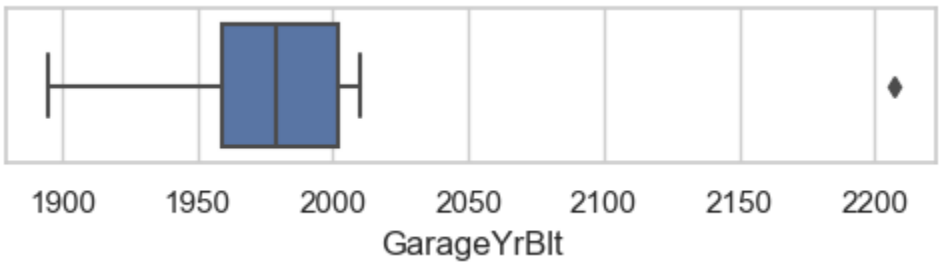

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
HalfBath

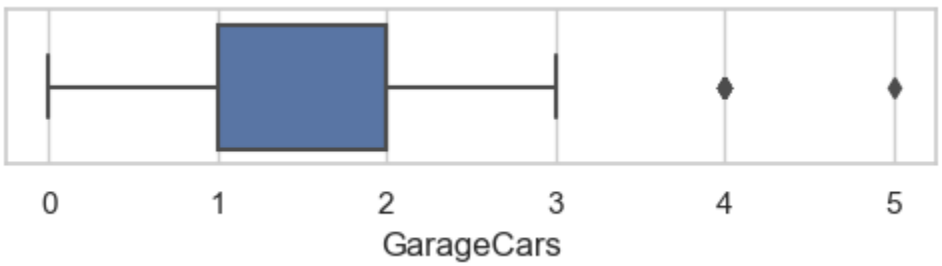

0
1
2
3
4
5
6
BedroomAbvGr

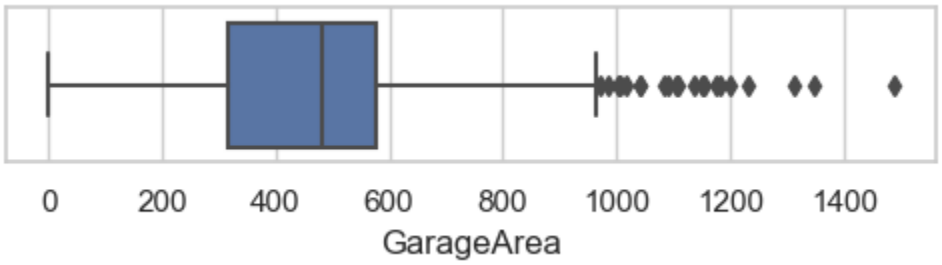

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
KitchenAbvGr


4
6
8
10
12
14
TotRmsAbvGrd

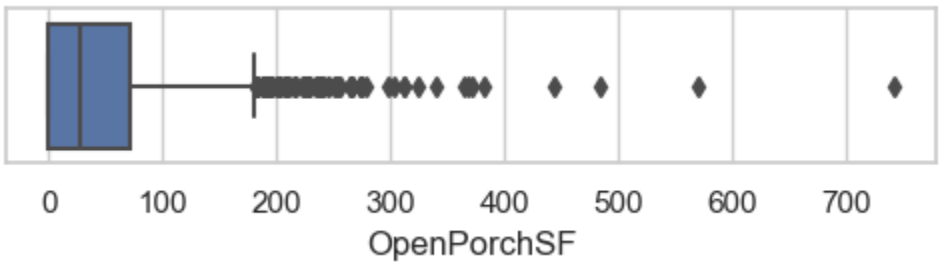

0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Fireplaces


1900
1950
2000
2050
2100
2150
2200
GarageYrBlt


0
1
2
3
4
5
GarageCars

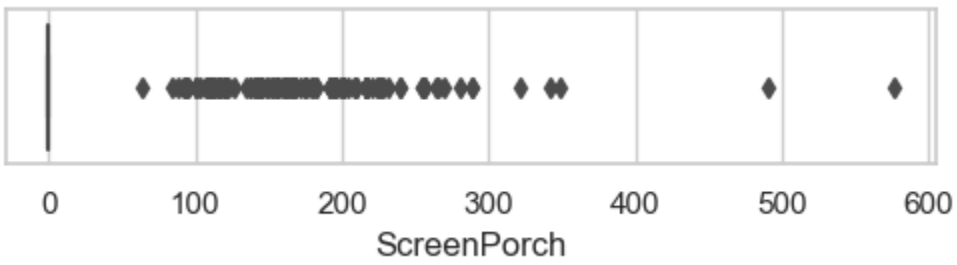

0
200
400
600
800
1000
1200
1400
GarageArea

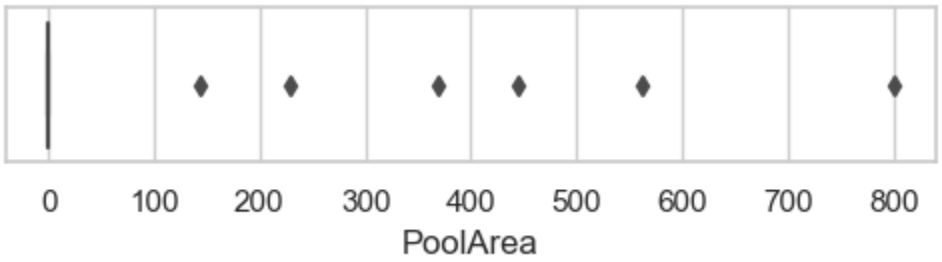

0
200
400
600
800
1000
1200
1400
WoodDeckSF

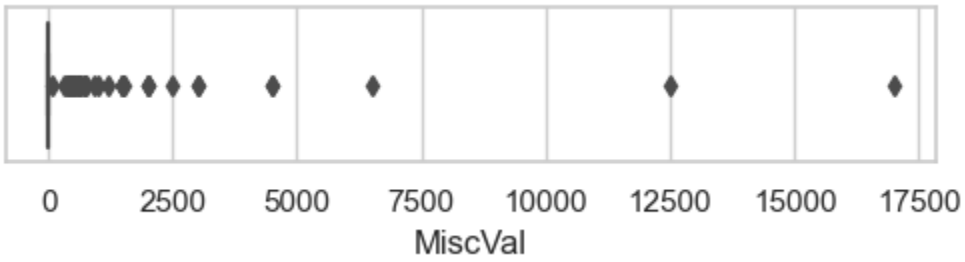

0
100
200
300
400
500
600
700
OpenPorchSF

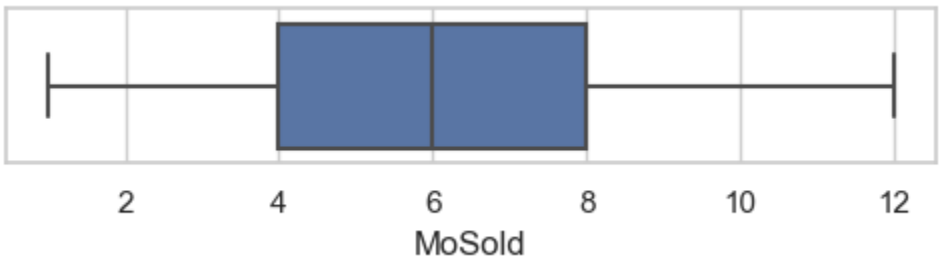

0
200
400
600
800
1000
EnclosedPorch

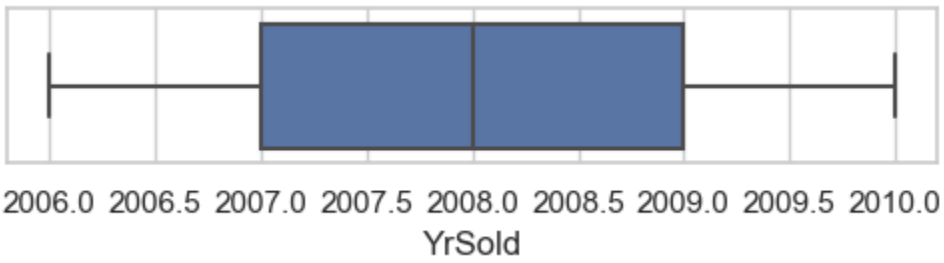

0
50
100
150
200
250
300
350
3SsnPorch


0
100
200
300
400
500
600
ScreenPorch









The values for GarageYrBlt, LotArea, LotFrontage, 1stFlrSF, GrLivArea, TotRmsAbvGrd, BsmtFinSF1, TotalBsmtSF, GarageCars, and GarageArea were changed to more sensible numbers in this instance. For instance, since it was probably an error, the value of GarageYrBlt 2207 was adjusted to 2005. Likewise, LotArea larger than 40000's value was modified to 40000. The missing values for LotFrontage were then filled up using Simple Linear Regression. The used code applies values to the pertinent features in the test dataset using the.loc method.

Test data

In [9]:
```
#Normalizing values of test data

test_b.loc[test.GarageYrBlt==2207,'GarageYrBlt'] = 2005  # mode value
test_b.loc[test_b.LotArea==1533,'LotFrontage'] = 21 # Minimum Value
test_b.loc[test_b.LotFrontage>130,'LotFrontage'] = 130
test_b.loc[test_b.LotArea>40000,'LotArea'] = 40000
test_b.loc[test_b["1stFlrSF"]>2200,'1stFlrSF'] = 2200
test_b.loc[test_b["GrLivArea"]>3000,'GrLivArea'] = 3000
test_b.loc[test_b["TotRmsAbvGrd"]>10,'TotRmsAbvGrd'] = 10
test_b.loc[test_b["BsmtFinSF1"]>2100,'BsmtFinSF1'] = 2100
test_b.loc[test_b["TotalBsmtSF"]>2500,'TotalBsmtSF'] = 2500
test_b.loc[test_b["GarageCars"]>3,'GarageCars'] = 3
test_b.loc[test_b["GarageArea"]>1000,'GarageArea'] = 1000
```

```
# Making a linear regression to fill missin LotFrontage values

test_b.loc[test_b["LotFrontage"].isnull(),["LotFrontage"]]=(test_b.loc[test_b["LotFrontage"].isn
```

In [10]:
```
data_b.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
MSSubClass	1460.0	56.897260	42.300571	20.0	20.00	50.0	70.00	190.0
LotFrontage	1201.0	70.049958	24.284752	21.0	59.00	69.0	80.00	313.0
LotArea	1460.0	10516.828082	9981.264932	1300.0	7553.50	9478.5	11601.50	215245.0
OverallQual	1460.0	6.099315	1.382997	1.0	5.00	6.0	7.00	10.0
OverallCond	1460.0	5.575342	1.112799	1.0	5.00	5.0	6.00	9.0
YearBuilt	1460.0	1971.267808	30.202904	1872.0	1954.00	1973.0	2000.00	2010.0
YearRemodAdd	1460.0	1984.865753	20.645407	1950.0	1967.00	1994.0	2004.00	2010.0
MasVnrArea	1460.0	103.117123	180.731373	0.0	0.00	0.0	164.25	1600.0
BsmtFinSF1	1460.0	443.639726	456.098091	0.0	0.00	383.5	712.25	5644.0
BsmtFinSF2	1460.0	46.549315	161.319273	0.0	0.00	0.0	0.00	1474.0
BsmtUnfSF	1460.0	567.240411	441.866955	0.0	223.00	477.5	808.00	2336.0
TotalBsmtSF	1460.0	1057.429452	438.705324	0.0	795.75	991.5	1298.25	6110.0
1stFlrSF	1460.0	1162.626712	386.587738	334.0	882.00	1087.0	1391.25	4692.0
2ndFlrSF	1460.0	346.992466	436.528436	0.0	0.00	0.0	728.00	2065.0
LowQualFinSF	1460.0	5.844521	48.623081	0.0	0.00	0.0	0.00	572.0
GrLivArea	1460.0	1515.463699	525.480383	334.0	1129.50	1464.0	1776.75	5642.0
BsmtFullBath	1460.0	0.425342	0.518911	0.0	0.00	0.0	1.00	3.0
BsmtHalfBath	1460.0	0.057534	0.238753	0.0	0.00	0.0	0.00	2.0
FullBath	1460.0	1.565068	0.550916	0.0	1.00	2.0	2.00	3.0
HalfBath	1460.0	0.382877	0.502885	0.0	0.00	0.0	1.00	2.0
BedroomAbvGr	1460.0	2.866438	0.815778	0.0	2.00	3.0	3.00	8.0
KitchenAbvGr	1460.0	1.046575	0.220338	0.0	1.00	1.0	1.00	3.0
TotRmsAbvGrd	1460.0	6.517808	1.625393	2.0	5.00	6.0	7.00	14.0
Fireplaces	1460.0	0.613014	0.644666	0.0	0.00	1.0	1.00	3.0
GarageYrBlt	1460.0	1979.698630	24.494189	1900.0	1962.00	1984.5	2001.00	2010.0
GarageCars	1460.0	1.767123	0.747315	0.0	1.00	2.0	2.00	4.0
GarageArea	1460.0	472.980137	213.804841	0.0	334.50	480.0	576.00	1418.0
WoodDeckSF	1460.0	94.244521	125.338794	0.0	0.00	0.0	168.00	857.0
OpenPorchSF	1460.0	46.660274	66.256028	0.0	0.00	25.0	68.00	547.0
EnclosedPorch	1460.0	21.954110	61.119149	0.0	0.00	0.0	0.00	552.0
3SsnPorch	1460.0	3.409589	29.317331	0.0	0.00	0.0	0.00	508.0
ScreenPorch	1460.0	15.060959	55.757415	0.0	0.00	0.0	0.00	480.0
PoolArea	1460.0	2.758904	40.177307	0.0	0.00	0.0	0.00	738.0
MiscVal	1460.0	43.489041	496.123024	0.0	0.00	0.0	0.00	15500.0
MoSold	1460.0	6.321918	2.703626	1.0	5.00	6.0	8.00	12.0

	count	mean	std	min	25%	50%	75%	max
YrSold	1460.0	2007.815753	1.328095	2006.0	2007.00	2008.0	2009.00	2010.0
SalePrice	1460.0	180921.195890	79442.502883	34900.0	129975.00	163000.0	214000.00	755000.0

Train Data

In [11]:
```
#Normalizing values of train data

data_b.loc[data_b.LotArea>40000,'LotArea'] = 40000
data_b.loc[data_b.LotFrontage>150,'LotFrontage'] = 150
data_b.loc[data_b["1stFlrSF"]>3000,'1stFlrSF'] = 3000
data_b.loc[data_b["GrLivArea"]>4000,'GrLivArea'] = 4000
data_b.loc[data_b["TotRmsAbvGrd"]>10,'TotRmsAbvGrd'] = 10
data_b.loc[data_b["BsmtFinSF1"]>2200,'BsmtFinSF1'] = 2200
data_b.loc[data_b["TotalBsmtSF"]>2500,'TotalBsmtSF'] = 2500
data_b.loc[data_b["GarageCars"]>3,'GarageCars'] = 2 # mode value
data_b.loc[data_b["GarageArea"]>1000,'GarageArea'] = 1000


# Making a linear regression to fill missin LotFrontage values

data_b.loc[data_b["LotFrontage"].isnull(),["LotFrontage"]]=(data_b.loc[data_b["LotFrontage"].isn
```

In [12]:
```
# This line of code sets a variable cur_year to the current year using the datetime module
cur_year = datetime.now().year

#These two lines of code create a new feature called AllSF by adding the square footage of the b
data_b['AllSF'] = data_b['TotalBsmtSF'] + data_b['GrLivArea']
test_b['AllSF'] = test_b['TotalBsmtSF'] + test_b['GrLivArea']

#These two lines of code create a new feature called RatioConstArea by calculating the ratio of
#The resulting values are rounded to 2 decimal places.
data_b['RatioConstArea'] = (data_b['AllSF'] / data_b['LotArea']).round(2)
test_b['RatioConstArea'] = (test_b['AllSF'] / test_b['LotArea']).round(2)

#These two lines of code create a new feature called Age_of_house by subtracting the year the ho
data_b["Age_of_house"] = cur_year - data_b["YearBuilt"]
test_b["Age_of_house"] = cur_year - test_b["YearBuilt"]

#These two lines of code create two new features called RemodAfter and AgeRemodAdd.
#RemodAfter calculates the number of years between the year the house was remodeled (YearRemodAd
#AgeRemodAdd calculates the number of years between the year the house was remodeled and the cur
data_b["RemodAfter"] = data_b["YearRemodAdd"] - data_b["YearBuilt"]
data_b["AgeRemodAdd"] = cur_year - data_b["YearRemodAdd"]
```

Year Feature Ingeneering

The codes create new features based on the existing features of the dataset. For example, the 'AllSF' feature is created by adding the 'TotalBsmtSF' and 'GrLivArea' features. This new feature represents the total living area of the house, including the basement. Similarly, the 'RatioConstArea' feature is created by dividing the 'AllSF' feature by the 'LotArea' feature, to capture the ratio of constructed area to the lot area.

Other new features include the age of the house, the age of the house at the time of remodeling, the age of the garage, and the age of the garage at the time of sale. The total area of the porch, the total number of bathrooms, and the time taken to sell the house are also new features.

By creating these new features, the codes aim to capture more information about the houses and thus improve the performance of machine learning models that will be trained on this dataset.

In [ ]:
```
#These three lines of code create two new features called RemodAfter and AgeRemodAdd for the tes
#RemodAfter calculates the number of years between the year the house was remodeled (YearRemodAd
#However, there is an additional line of code that sets any negative values of RemodAfter to 0.
#AgeRemodAdd calculates the number of years between the year the house was remodeled and the cur
test_b["RemodAfter"] = test_b["YearRemodAdd"] - test_b["YearBuilt"]
test_b.loc[test_b.RemodAfter<0,'RemodAfter'] = 0
test_b["AgeRemodAdd"] = cur_year - test_b["YearRemodAdd"]

#These three lines of code create a new feature called Age_Sold by subtracting the year the hous
#There is also an additional line of code that sets any negative values of Age_Sold in the test
data_b["Age_Sold"] = data_b["YrSold"] - data_b["YearBuilt"]
test_b["Age_Sold"] = test_b["YrSold"] - test_b["YearBuilt"]
test_b.loc[test_b.Age_Sold<0,'Age_Sold'] = 0


data_b["Sold_before"] = cur_year - data_b["YrSold"]
test_b["Sold_before"] = cur_year - test_b["YrSold"]

data_b['Age_Garage'] = cur_year - data_b['GarageYrBlt']
test_b['Age_Garage'] = cur_year - test_b['GarageYrBlt']
data_b['Age_Garage_Sold'] = data_b["YrSold"] - data_b['GarageYrBlt']
test_b['Age_Garage_Sold'] = test_b["YrSold"] - test_b['GarageYrBlt']

data_b['TotalPorchArea']=data_b['EnclosedPorch'] + data_b['OpenPorchSF'] + data_b['ScreenPorch']
test_b['TotalPorchArea']=test_b['EnclosedPorch'] + test_b['OpenPorchSF'] + test_b['ScreenPorch']

data_b['TotalBathNos'] = data_b['FullBath']+ data_b['HalfBath']+ data_b['BsmtFullBath']+ data_b[
test_b['TotalBathNos'] = test_b['FullBath']+ test_b['HalfBath']+ test_b['BsmtFullBath']+ test_b[
```

Prediction Model

In this chapter, we'll look at how to build a prediction model using the stacking ensemble learning technique. A robust technique called stacking combines the advantages of various models to produce a prediction that is more reliable and accurate. In order to prevent any particular variables that differ between the models from having an impact on the stacking process, we will first standardize the datasets that were used. This process is essential for assembling a consistent dataset that can be applied to all models.

Once the datasets are uniform, we will deploy a number of models and use a diagnostic model to assess how well they work. We can assess which models are operating well and which models may need to be altered or replaced using the diagnostic model's results for each of the installed models.

Lastly, we will use the stacking ensemble learning technique to combine all the outcomes from the various models and provide the optimum response. Each of the models that have been applied will produce scores, and the stacking model will use those values to produce a final prediction. This final forecast combines the qualities of various models to provide a more dependable conclusion, making it more accurate and robust than any single model alone.

Overall, the stacking ensemble learning technique is a potent tool for developing strong, accurate, and reliable prediction models. This method allows us to combine the benefits of various models and produce

forecasts that are more precise than those produced by any one model alone.

Standarizing Datasets

In [13]:
```
#Replacing values that are outliers in both datasets
test_b['MSSubClass'] = test_b['MSSubClass'].replace(["150"], ["160"])
data_b['HouseStyle'] = data_b['HouseStyle'].replace(["2.5Fin"], ["2Story"])
data_b['Exterior1st'] = data_b['Exterior1st'].replace(['Stone', 'ImStucc'],['CemntBd', 'Stucco']
data_b['Electrical'] = data_b['Electrical'].replace(["Mix","FuseF","FuseP"], ["SBrkr","FuseA","F
test_b['Electrical'] = test_b['Electrical'].replace(["Mix","FuseF","FuseP"], ["SBrkr","FuseA","F
test_b['BsmtCond'] = test_b['BsmtCond'].replace(["Ex"], ["Gd"])
data_b['GarageQual'] = data_b['GarageQual'].replace(["Ex"], ["Gd"])
data_b['GarageCond'] = data_b['GarageCond'].replace(["Ex"], ["Gd"])
test_b['GarageCond'] = test_b['GarageCond'].replace(["Ex"], ["Gd"])
```

In [14]:
```
#creating the Y variable for the analysis
y = data_b.pop("SalePrice")
```

In [15]:
```
#After the changes in the data, with this we will verify the null values
data_b.isnull().sum().sort_values(ascending=False).head(10)
```

Out[15]:
```
PoolQC          1453
MiscFeature     1406
Alley           1369
BsmtFinType2      38
BsmtFinType1      37
MSSubClass         0
PavedDrive         0
GarageCond         0
GarageQual         0
GarageArea         0
dtype: int64
```

In [16]:
```
#After the changes in the data, with this we will verify the null values
test_b.isnull().sum().sort_values(ascending=False).head(10)
```

Out[16]:
```
PoolQC          1456
MiscFeature     1408
Alley           1352
BsmtFinType2      42
BsmtFinType1      42
MSSubClass         0
PavedDrive         0
GarageCond         0
GarageQual         0
GarageArea         0
dtype: int64
```

In [17]:
```
#dropping the unnecessary columns that have nulls
data_b = data_b.dropna(axis=1)
test_b = test_b.dropna(axis=1)
```

In [18]:
```
#dropping the columns that were used for the creation of the new variables
#steps before this
data_b = data_b.drop(["YearBuilt","YearRemodAdd","Street","RoofMatl","Condition2","LowQualFinSF"
                      "3SsnPorch","ScreenPorch","PoolArea","MiscVal","YrSold","Utilities","Heating
test_b = test_b.drop(["YearBuilt","YearRemodAdd","Street","RoofMatl","Condition2","LowQualFinSF"
                      "3SsnPorch","ScreenPorch","PoolArea","MiscVal","YrSold","Utilities","Heating
```

In [19]:
```
#Loop that splits the numerical from the categorical. This to create dummies for the categorical
cols = data_b.columns
numeric_columns, categorical_columns = [], []
for i in range(len(cols)):
    if data_b[cols[i]].dtypes == 'O':
        categorical_columns.append(cols[i])
    else:
        numeric_columns.append(cols[i])
```

In [20]:
```
# getting dummies for both datasets
data_b = pd.get_dummies(data_b)
test_b = pd.get_dummies(test_b)
```

In [21]:
```
# Skewing all features (Log level transform)
skewed_feats = data_b.apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)

#deep copying data into new dataframe
data_b_skewed = data_b.copy()
test_b_skewed = test_b.copy()

skewness = skewness[abs(skewness) > 1]
print("There are {} skewed numerical features to transform".format(skewness.shape[0]))

#converting actual values into log values in column for each columns of the data frame
#and storting it into data and test
skewed_features = skewness.index
for feat in skewed_features:
    data_b_skewed[feat] = np.log1p(data_b_skewed[feat])
    test_b_skewed[feat] = np.log1p(test_b_skewed[feat])
```

```
There are 225 skewed numerical features to transform
```

In [22]:
```
#Determining if there are swede values in the data
print("Only in Train: "+ str(list(set(data_b_skewed.columns) - set(test_b_skewed.columns))))
print("Only in Test: "+ str(list(set(test_b_skewed.columns) - set(data_b_skewed.columns))))
```

```
Only in Train: []
Only in Test: []
```

In [23]:
```
# converting the actual values of sales price into log values
y_log = np.log1p(y)
```

In [24]:
```
#Converting the -INF values into 0
test_b_skewed=test_b_skewed.replace(-np.Inf, 0)
```

Model Score diagnosis

The main purpose of this diagnosis is to evaluate the performance of several regression models using cross-validation. The code defines a function called run_cvs which takes in two arguments, the input data and the target variable. The function then initializes several regression models including ElasticNet, Ridge, Lasso, KernelRidge, LGBMRegressor, XGBRegressor, CatBoostRegressor, and GradientBoostingRegressor.

The cross_val_score method from Scikit-learn is used to perform cross-validation on each model with 10-fold cross-validation. The average score of each model is printed to the console, giving an idea of how well

each model performs on the data. Finally, the function is called with the input data data_b_skewed and the target variable y_log.

In [25]:
```
#running cross valuation scores for different regression models
def models(X,y):

    #ElasticNet regression
    baseline = ElasticNet(random_state=0,max_iter=10e7,alpha=0.0003)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("ENet avg:",np.mean(baseline_score))

    #Ridge Regression
    baseline = Ridge(alpha = 1, random_state=0)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("Ridge avg:",np.mean(baseline_score))

    #Lasso regression
    baseline = Lasso(alpha = 0.0001,random_state=0)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("Lasso avg:",np.mean(baseline_score))

    #KernelRidge regression
    baseline = KernelRidge(alpha=0.1)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("KRR avg:",np.mean(baseline_score))

    #LGBMR  regression
    baseline = LGBMRegressor(learning_rate=0.01,num_leaves=4,n_estimators=2000, random_state=0)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("LGBM avg:",np.mean(baseline_score))

    #XGB Regression
    baseline = xg.XGBRegressor(learning_rate=0.01,n_estimators=2000, subsample=0.7,colsample_byt
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("XGB avg:",np.mean(baseline_score))

    #Catboost regression
    baseline = CatBoostRegressor(random_state=0,verbose=0)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("CatB avg:",np.mean(baseline_score))

    #Gradient Boost regression
    baseline = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.02,max_depth=4, max_
                min_samples_leaf=15, min_samples_split=50,loss='huber', random_state = 0)
    baseline_score = cross_val_score(baseline, X, y, cv=10)
    print("GBR avg:",np.mean(baseline_score))

models(data_b_skewed,y_log)
```

```
ENet avg: 0.9081423769455832
Ridge avg: 0.9030750344201939
Lasso avg: 0.9054133981153164
KRR avg: 0.8946640870310489
LGBM avg: 0.9005330771094776
XGB avg: 0.9139168117227585
CatB avg: 0.914869479102849
GBR avg: 0.9087245630873504
```

Prediction Model: Stacking Ensable Learning

Stacking is a type of ensemble learning technique that combines multiple models to achieve better performance than any individual model. In a two-level hierarchy of models, the first level consists of several base models, such as ridge regression, Lasso regression, CatBoost, XGBoost, LightGBM, and Gradient Boosting Regression (GBR). These base models are trained on the same dataset, but with different hyperparameters and features. In the second level, a meta-model is trained using the predictions made by the base models as input features. The meta-model can be any algorithm, such as linear regression, logistic regression, or neural networks.

Each base model is designed to capture different aspects of the dataset, and the meta-model is trained to learn the optimal way of combining these models to achieve better predictions. Ridge regression and Lasso regression are linear models that are useful for feature selection and regularization. CatBoost, XGBoost, and LightGBM are gradient boosting models that are widely used for tabular data. GBR is another gradient boosting model that is useful for regression problems. By combining these models, we can leverage the strengths of each model and minimize their weaknesses. Stacking ensemble learning is a powerful technique that can help us achieve better results in many machine learning problems.

In [26]:
```
#using 6 models with the highest values in the previous step
def submission(X_train, y_train, X_test):
    sub_df = pd.read_csv('sample_submission.csv', index_col = "Id")

    ridge = Ridge(alpha = 1, random_state=0).fit(X_train,y_train)
    ridge_preds_log=ridge.predict(X_test)

    lasso = Lasso(alpha = 0.0001,random_state=0).fit(X_train,y_train)
    lasso_preds_log=lasso.predict(X_test)

    catB = CatBoostRegressor(random_state=0,verbose=0).fit(X_train,y_train)
    catB_preds_log=catB.predict(X_test)

    xgb = xg.XGBRegressor(learning_rate=0.01,n_estimators=2000, subsample=0.7,colsample_bytree=0
    xgb_preds_log=xgb.predict(X_test)

    lgbm = LGBMRegressor(n_estimators=2000,learning_rate=0.01,subsample=0.7,random_state=0).fit(
    lgbm_preds_log=lgbm.predict(X_test)

    gbr = GradientBoostingRegressor(n_estimators=2000, learning_rate=0.01,subsample=0.7,random_s
    gbr_preds_log=gbr.predict(X_test)

    catb_xbr_lasso_ridge_mean_preds_log=(catB_preds_log+ridge_preds_log+lasso_preds_log+xgb_pred
    sub_df['SalePrice'] = np.exp(catb_xbr_lasso_ridge_mean_preds_log)-1
    sub_df.to_csv("Team_12_HULT_Boston_submission.csv")

submission(data_b_skewed,y_log,test_b_skewed)
```