House Price Prediction in Ames

Zhiguang GUAN (Jackey)

Supervisors: Prof. William Au & Prof. David Parent

August 2022

Abstract

This study investigates the data on home prices in Ames and uses common machine learning techniques, such as decision trees and linear regression, to forecast the sale price of homes. In comparison to the other single models, the stacked model of linear regression, random forest, and XGBoost produces the best accurate predictions as measured by error metrics and R-square.

Contents

1 Introduction

1.1 Background and Problem Statement

A place to call home is among a person's most basic needs, along with other items like food, water, and many other things. As people's living standards climbed over time, so did the need for housing. The majority of people acquire homes for occupancy or as a source of support, however some people buy them as investments or as real estate. Housing markets are advantageous for a country's currency, which is an important measure of the health of the overall economy. Homeowners will purchase products like furniture and household equipment to satisfy their housing needs, and homebuilders or contractors will purchase building supplies, which is an indication of the economic wave impact brought on by the new housing supply. Customers also have the money to make a big investment, and a nation's plentiful home supply is a sign that the construction industry is doing well.

With the development of the real estate industry, all countries, societies, and individuals have become more concerned about the price of housing. After experiencing the 2008 financial crisis and the 2020 Covid-19 pandemic, people have doubtless become more price sensitive and constantly look for the best possible deals when making an important investment such as a real estate purchase. With the increasing degree of transparency and availability of real-estate data, things like house price prediction that was difficult - if not impossible - in the past have become more possible than ever.

Besides macro-economic factors such as inflation or regulation interventions, there are many more factors that can potentially affect the price of housing, among which are factors such as lighting, house type, neighborhood, etc. It is important to take these factors into account because they offer greater details that help quantify the characteristics of a given real-estate project.

1.2 Objective

Leveraging the power of big data and machine learning, the aim of this project is to develop a model that decreases information asymmetries between individuals and real estate agents and could be easily used by individuals as a reference point when evaluating a quote. Ultimately, this tool will also help to reduce the cases where certain real-estate projects are irregularly high or low and help to maintain a healthy real estate market in the long run.

Given this consideration, an extensive data set from Kaggle that includes 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa is used as the basis of this project. By exploring the data set thoroughly and applying various machine learning models in Python, the final outcome of this project will be a house price prediction tool for the Ames region.

1.3 Analytics Plan

1.3.1 Business Opportunity Brief

This project focuses on enhancing the use of data and information related to housing pricing and makes accurate forecasting and evaluation for all potential house buyers and sellers. House price predictions may be made using a variety of machine learning prediction models, such as support vector regression. The house-price model benefits real estate investors, home buyers, and home builders in a number of ways. This project will provide knowledge and information to home buyers, investors in real estate, and home builders, including an evaluation of current market housing prices that will help in determining house pricing. In the meanwhile, this model can help prospective purchasers choose the attributes of a home that correspond to their budget. Previous studies mainly focused on identifying the independent influences on home price and predicting house price using a machine learning model.

1.3.2 Supporting Insights

House price forecasting is an important topic of real estate. Tryolab is a famous AI pricing company in the current market. Tryolab takes advantage of AI-generated insights and suggestions based on historical data or starts controlled experimentation where data lacks. But the disadvantages are the models Tryolab has applied are not available for retail buyers and sellers and the consultation fees are very high.

Furthermore, when asked to describe their ideal home, purchasers are unlikely to start with the basement ceiling height or the house's closeness to an east-west railroad. However, the research shows that factors other than the number of beds or a white picket fence have a significantly greater impact on price discussions. It would be simpler to create models for house price forecasts if there were 79 explanatory variables that could be used to describe (nearly) every feature of residential dwellings in Ames, Iowa.

1.3.3 Project Gains

The research makes an effort to extract relevant information from historical data on real estate markets. In order to find models that are helpful to home buyers and sellers, machine learning techniques are used to examine previous real estate transactions. maintaining consumer openness and making comparisons with ease using the models The consumer has the option to reject a home if the price shown on a certain website is higher than the price indicated by the model.

If we do nothing, all the individual buyers and sellers will not trade houses at proper prices. And property agents can take advantage of asymmetric infor-

mation to rip off customers. The property markets will be disorganized without some price references.

Note: Completion of the following sections is possible only after a careful assessment and triage of the Ask. This is required to determine scope, resource, time, priority, and data availability.

1.3.4 Methodology and Approach

Type of Analysis: Regression, Random Forest

Methodology: This project starts with data exploration of the original house dataset such as exploring the distribution of each feature. And I will clean the data such as removing outliers, handling null values, and deleting duplicates. Then I will fit the data into different machine learning models. Next, I will tune the hyperparameters and check the related metrics. Finally, I will analyze the results of different models and compare them to select the best one.

Output: The output is house prices.

1.3.5 Population, Variable Selection, considerations

Audience/population selection: house buyers and sellers

Observation window: not applicable

Inclusions: data like Electrical system, Central air conditioning, Pool quality

Exclusions: macro data like house pricing index

Data Sources: https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

Audience Level: not applicable

Variable Selection: will rank the features and choose not correlated features

Derived Variables: will transform categorical data into dummy variables

Assumptions and data limitations: current data doesn't include any macro data such as the current economic situation and CPI

2 Data Source and Exploration

Package Importing

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm,lognorm
from sklearn.preprocessing import StandardScaler
from scipy import stats
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

1. **Understand the problem**. I'll look at each variable and do a philosophical analysis about their meaning and importance for this problem.
2. **Univariable study**. I'll just focus on the dependent variable ('SalePrice') and try to know a little bit more about it.
3. **Multivariate study**. I'll try to understand how the dependent variable and independent variables relate.
4. **Basic cleaning**. I'll clean the dataset and handle the missing data, outliers and categorical variables.
5. **Test assumptions**. I'll check if the data meets the assumptions required by most multivariate techniques.

The data set in use contains 1460 entries of different real-estate transactions and 81 columns including one ID column, one pricing column, and 79 feature columns (among which 36 are quantitative and 43 are categorical). Table 1 summarizes all columns and their attributes accordingly.

Table 1: Data set column description

Number	Variable Name	Description	Characteristics
1	MSSubClass	The building class	Quantitative
2	MSZoning	The general zoning classification	Categorical
3	LotFrontage	Linear feet of street connected to property	Quantitative
4	LotArea	Lot size in square feet	Quantitative
5	Street	Type of road access	Categorical
6	Alley	Type of alley access	Categorical
7	LotShape	General shape of property	Categorical
8	LandContour	Flatness of the property	Categorical
9	Utilities	Type of utilities available	Categorical
10	LotConfig	Lot configuration	Categorical
11	LandSlope	Slope of property	Categorical
12	Neighborhood	Physical locations within Ames city limits	Categorical
13	Condition1	Proximity to main road or railroad	Categorical
14	Condition2	Proximity to main road or railroad (if present)	Categorical
15	BldgType	Type of dwelling	Categorical
16	HouseStyle	Style of dwelling	Categorical

Continued on next page

Table 1 – continued from previous page

Number	Variable Name	Description	Characteristics
17	OverallQual	Overall material and finish quality	Quantitative
18	OverallCond	Overall condition rating	Quantitative
19	YearBuilt	Original construction date	Quantitative
20	YearRemodAdd	Remodel date	Quantitative
21	RoofStyle	Type of roof	Categorical
22	RoofMatl	Roof material	Categorical
23	Exterior1st	Exterior covering on house	Categorical
24	Exterior2nd	Exterior covering on house (if present)	Categorical
25	MasVnrType	Masonry veneer type	Categorical
26	MasVnrArea	Masonry veneer area in square feet	Quantitative
27	ExterQual	Exterior material quality	Categorical
28	ExterCond	Present condition of the material on the exterior	Categorical
29	Foundation	Type of foundation	Categorical
30	BsmtQual	Height of the basement	Categorical
31	BsmtCond	General condition of the basement	Categorical
32	BsmtExposure	Walkout or garden level basement walls	Categorical
33	BsmtFinType1	Quality of basement finished area	Categorical
34	BsmtFinSF1	Type 1 finished square feet	Quantitative
35	BsmtFinType2	Quality of second finished area (if present)	Categorical
36	BsmtFinSF2	Type 2 finished square feet	Quantitative
37	BsmtUnfSF	Unfinished square feet of basement area	Quantitative
38	TotalBsmtSF	Total square feet of basement area	Quantitative
39	Heating	Type of heating	Categorical
40	HeatingQC	Heating quality and condition	Categorical
41	CentralAir	Central air conditioning	Categorical
42	Electrical	Electrical system	Categorical
43	1stFlrSF	First Floor square feet	Quantitative
44	2ndFlrSF	Second floor square feet	Quantitative
45	LowQualFinSF	Low quality finished square feet (all floors)	Quantitative
46	GrLivArea	Above grade (ground) living area square feet	Quantitative
47	BsmtFullBath	Basement full bathrooms	Quantitative
48	BsmtHalfBath	Basement half bathrooms	Quantitative
49	FullBath	Full bathrooms above grade	Quantitative
50	HalfBath	Half baths above grade	Quantitative
51	BedroomAbvGr	Number of bedrooms above basement level	Quantitative
52	KitchenAbvGr	Number of kitchens	Quantitative
53	KitchenQual	Kitchen quality	Categorical
54	TotRmsAbvGrd	Total rooms above grade (does not include bathrooms)	Quantitative
55	Functional	Home functionality rating	Categorical
56	Fireplaces	Number of fireplaces	Quantitative
57	FireplaceQu	Fireplace quality	Categorical
58	GarageType	Garage location	Categorical
59	GarageYrBlt	Year garage was built	Quantitative

Continued on next page

Table 1 – continued from previous page

Number	Variable Name	Description	Characteristics
60	GarageFinish	Interior finish of the garage	Categorical
61	GarageCars	Size of garage in car capacity	Quantitative
62	GarageArea	Size of garage in square feet	Quantitative
63	GarageQual	Garage quality	Categorical
64	GarageCond	Garage condition	Categorical
65	PavedDrive	Paved driveway	Categorical
66	WoodDeckSF	Wood deck area in square feet	Quantitative
67	OpenPorchSF	Open porch area in square feet	Quantitative
68	EnclosedPorch	Enclosed porch area in square feet	Quantitative
69	3SsnPorch	Three season porch area in square feet	Quantitative
70	ScreenPorch	Screen porch area in square feet	Quantitative
71	PoolArea	Pool area in square feet	Quantitative
72	PoolQC	Pool quality	Categorical
73	Fence	Fence quality	Categorical
74	MiscFeature	Miscellaneous feature not covered in other categories	Categorical
75	MiscVal	Dollar Value of miscellaneous feature	Quantitative
76	MoSold	Month Sold	Quantitative
77	YrSold	Year Sold	Quantitative
78	SaleType	Type of sale	Categorical
79	SaleCondition	Condition of sale	Categorical
80	SalePrice	the property's sale price in dollars	Quantitative

In [5]:
```
quantitative = [f for f in df_train.columns if df_train.dtypes[f] != 'object']
quantitative.remove('SalePrice')
quantitative.remove('Id')
print(f"Quantitative Data:\n{quantitative}")

qualitative = [f for f in df_train.columns if df_train.dtypes[f] == 'object']
print(f"Qualitative Data:\n{qualitative}")
```

```
Quantitative Data:
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF
2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'Enclo
sedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold']
Qualitative Data:
['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition
2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature', 'Sale
Type', 'SaleCondition']
```

There are 1460 instances of training data and 1460 of test data. Total number of attributes equals 81, of which 36 is quantitative, 43 categorical + Id and SalePrice.

Quantitative: 1stFlrSF, 2ndFlrSF, 3SsnPorch, BedroomAbvGr, BsmtFinSF1, BsmtFinSF2, BsmtFullBath, BsmtHalfBath, BsmtUnfSF, EnclosedPorch, Fireplaces, FullBath, GarageArea, GarageCars, GarageYrBlt, GrLivArea, HalfBath, KitchenAbvGr, LotArea, LotFrontage, LowQualFinSF, MSSubClass, MasVnrArea, MiscVal, MoSold, OpenPorchSF, OverallCond, OverallQual, PoolArea, ScreenPorch, TotRmsAbvGrd, TotalBsmtSF, WoodDeckSF, YearBuilt, YearRemodAdd, YrSold

Qualitative: Alley, BldgType, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, BsmtQual, CentralAir, Condition1, Condition2, Electrical, ExterCond, ExterQual, Exterior1st, Exterior2nd, Fence, FireplaceQu, Foundation, Functional, GarageCond, GarageFinish, GarageQual, GarageType, Heating, HeatingQC, HouseStyle, KitchenQual, LandContour, LandSlope, LotConfig, LotShape, MSZoning, MasVnrType, MiscFeature, Neighborhood, PavedDrive, PoolQC, RoofMatl, RoofStyle, SaleCondition, SaleType, Street, Utilities,

In [6]:
```
sum(df_train[qualitative + quantitative].duplicated())
```

Out[6]: 0

There is no duplicate data in the training set

Before any feature engineering or exploratory data analysis, a data sanity check is performed. There is no duplication in the data set, but several features have a high percentage of missing values. There are in total 19 attributes that have missing values, and 5 of them are missing more than 50% of all data (see Figure below). However, most of the time, these values are missing because the physical feature of the subject described by this feature is missing. For example, if a house doesn't have a pool or basement, it will lead to missing values for the columns corresponding to the pool or basement description. Hence, one should remember that some of the missing values are not a data collection or sanity problem, but rather a truthful reflection of the actual features of the records. We will revisit this issue after the exploratory data analysis is done.

In [7]:
```
missing = df_train.isnull().sum().sort_values(ascending=False)
df_missing = pd.DataFrame({'Total': missing})
df_missing['Percentage'] = df_missing.Total/len(df_train)
df_missing = df_missing[df_missing.Total>0]
display(df_missing)
df_missing.Total.plot.bar()
```

	Total	Percentage
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685

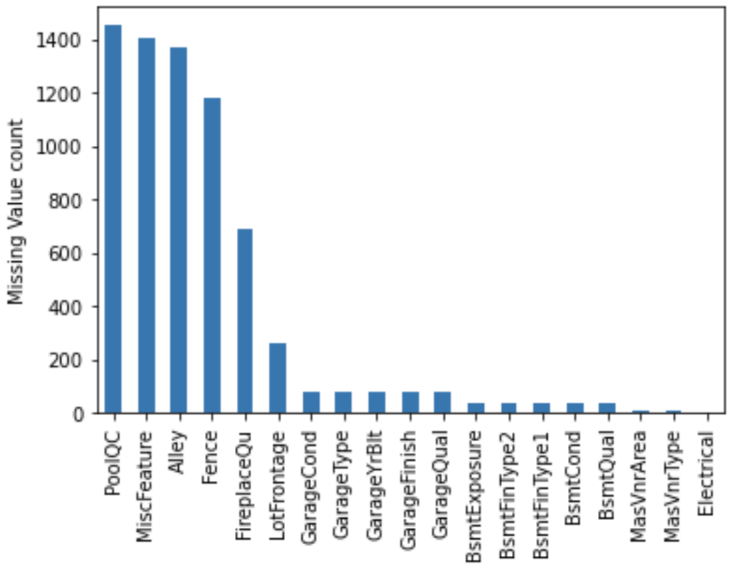

Figure 1: Missing value counts

2.1 Univariable Study

```
In [8]: df_train['SalePrice'].describe()
```

```
Out[8]: count      1460.000000
        mean     180921.195890
        std       79442.502883
        min       34900.000000
        25%      129975.000000
        50%      163000.000000
        75%      214000.000000
        max      755000.000000
        Name: SalePrice, dtype: float64
```

The minimum sale price is larger than zero

```
In [9]: print(f"Skewness: {df_train['SalePrice'].skew()}")
        print(f"Kurtosis: {df_train['SalePrice'].kurt()}")
```

```
Skewness: 1.8828757597682129
Kurtosis: 6.536281860064529
```

In this section, the target variable (dependent variable) SalePrice will be analyzed. From the below distribution plot, we can see the distribution is positively skewed. The skewness is 1.88. And the kurtosis is 6.54, which means the dataset has heavier tails than normal distribution.

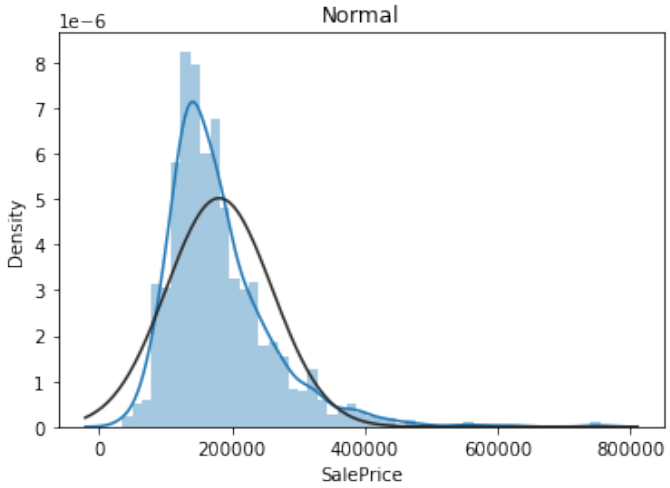

Figure 2: Normal Distribution Plot for SalePrice

The log-normal distribution plot shows that the blue line and black line align well with each other, which means the SalePrice is supposed to be log-transformed in order to get a normal distribution for better model fitting.

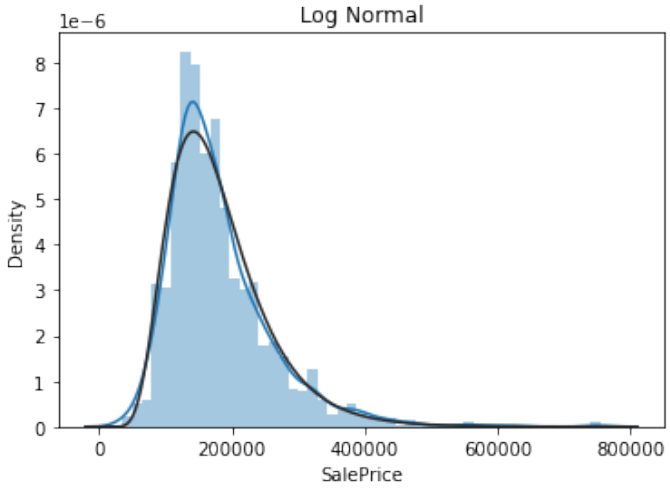

Figure 3: Log Normal Distribution Plot for SalePrice

2.2 Multivariable Study

2.2.1 Quantitative Data

To gain a better understanding of the distribution of value for each quantitative data and determine how to handle them in the later steps, a distribution plot is created for each quantitative feature (see Figure below).

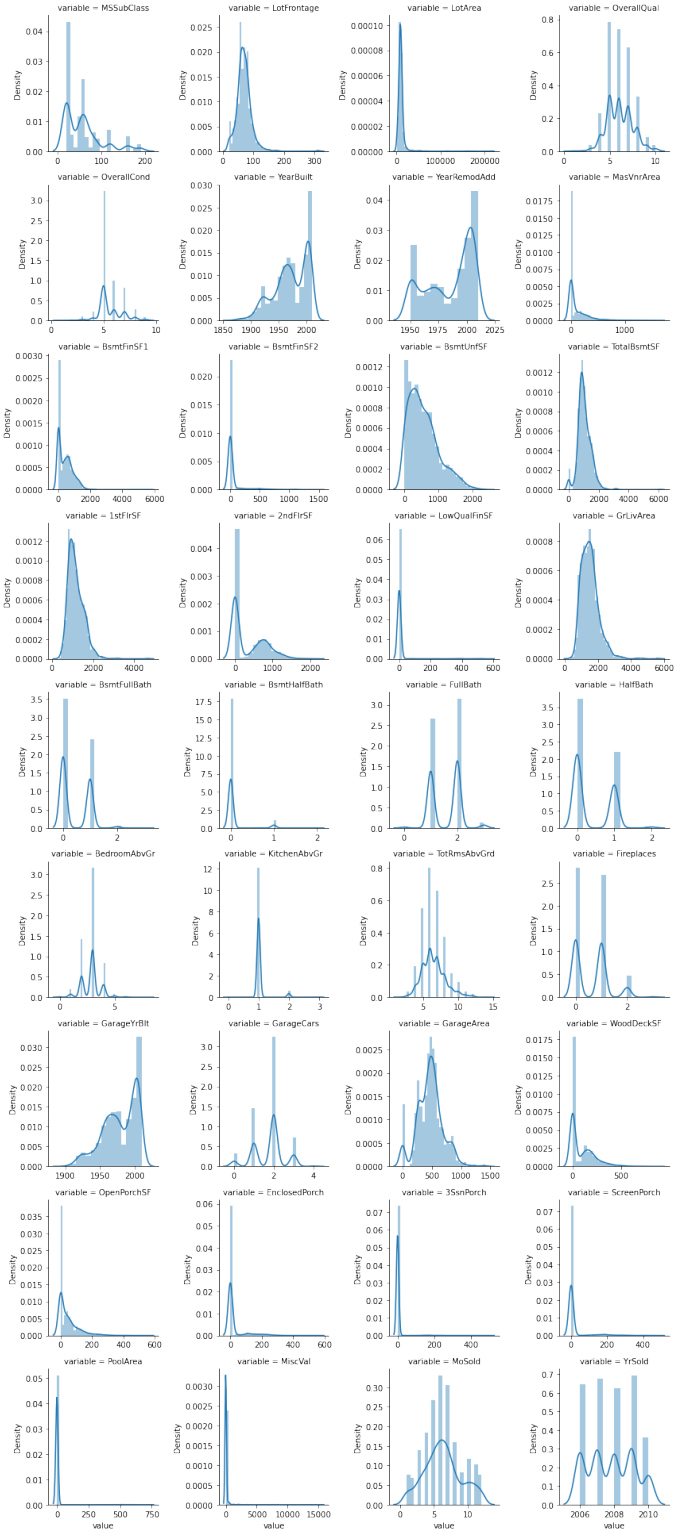

Figure 4: Distribution plot for quantitative variables

From the distribution graph summary, one can see that some independent variables (e.g., TotalBsmtSF, KitchenAbvGr, LotFrontage, LotArea) could be good candidates for log transformation. For other variables such as 2ndFlrSF, given the disproportionately large percentage of 0 values, it is important to consider smoothing out these irregularities with regression transformation. However, these variables are good candidates for feature construction and should still be included in the later steps.

2.2.2 Categorical Data

The analysis and interpretation of categorical data are comparably more difficult than that of quantitative data. However, given that almost half of the columns in the data set, there are mainly two methods that can be used to understand categorical data, namely by checking the distribution of SalePrice (the dependent variable) with respect to variable values of each independent categorical variable and enumerate them or by creating a dummy variable for each possible category. However, if taking the second option, the number of dummy variables created will be too large and will not be informative enough to be interpreted. For example, the variable "Exterior1st" alone could take 17 different values (AsbShng, AsphShn, BrkComm, etc.), which will become 16 dummy variables once engineered. And if the same is done for all other variables, we could easily end up with more than 300 dummy variables, which could make the following analysis more difficult, less conclusive, and even less precise. Hence, the first method is chosen and the figure below is produced.

In [13]:
```
for col in qualitative:
    df_train[col] = df_train[col].astype('category')
    if df_train[col].isnull().any():
        df_train[col] = df_train[col].cat.add_categories(['MISSING'])
        df_train[col] = df_train[col].fillna('MISSING')
```

In [14]:
```
def boxplot(x, y, **kwargs):
    sns.boxplot(x=x, y=y)
    x=plt.xticks(rotation=90)
```

In [15]:
```
f = pd.melt(df_train, id_vars=['SalePrice'], value_vars=qualitative)
g = sns.FacetGrid(f, col="variable",  col_wrap=4, sharex=False, sharey=False,size=4)
g = g.map(boxplot, "value", "SalePrice")
```

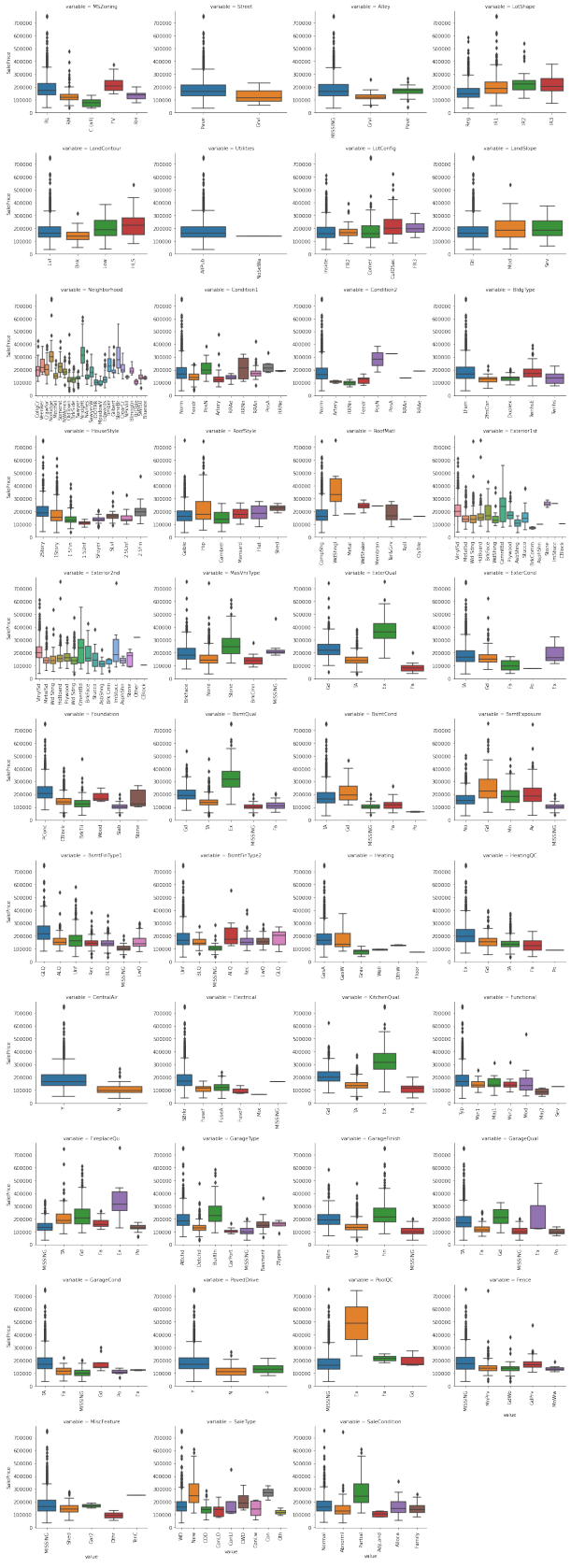

Figure 5: Distribution plot for qualitative variables

One can see that some categories seem too diverse more with respect to SalePrice than others. "Neighborhood", for example, has a big impact on house prices, and having a pool on the property seems to improve prices substantially at the first glance. There are also differences in variability between category values that need to be considered in the later steps.

2.3 Correlation Analysis

In this section, both the Pearson and the Spearman correlation are computed. The two approaches vary in that Spearman correlation uses monotonic correlations, i.e., based on the ranking value of each variable, whereas Pearson correlation uses a linear relationship between two variables based on the raw data directly.[2]. For this project, Spearman correlation is better to work with because it picks up relationships between variables even when they are nonlinear. The python code and the results are shown in the figures below.

```
def rank_category(df, col):
    df_ = (df[[col, 'SalePrice']].copy()
            .groupby(col).mean()
            .sort_values('SalePrice')
            .reset_index()
            .reset_index()
            .rename(columns={'index':col+'_Order','SalePrice':col+'_SalePriceMean'}))
    return df_

qualitative_ranked = []
for col in qualitative:
    df_ = rank_category(df_train, col)
    df_train = pd.merge(df_train,df_,left_on=col,right_on=col,how='inner')
    qualitative_ranked.append(col+'_Order')
```

```
corr_all = (df_train[quantitative + qualitative_ranked + ['SalePrice']]
        .corr(method='spearman')
        .sort_values('SalePrice')
        .reset_index())
plt.figure(figsize=(6, len(corr_all)/4))
sns.barplot(data=corr_all, y='index', x='SalePrice')
```

Figure 6: Python code to calculate the Spearman correlation

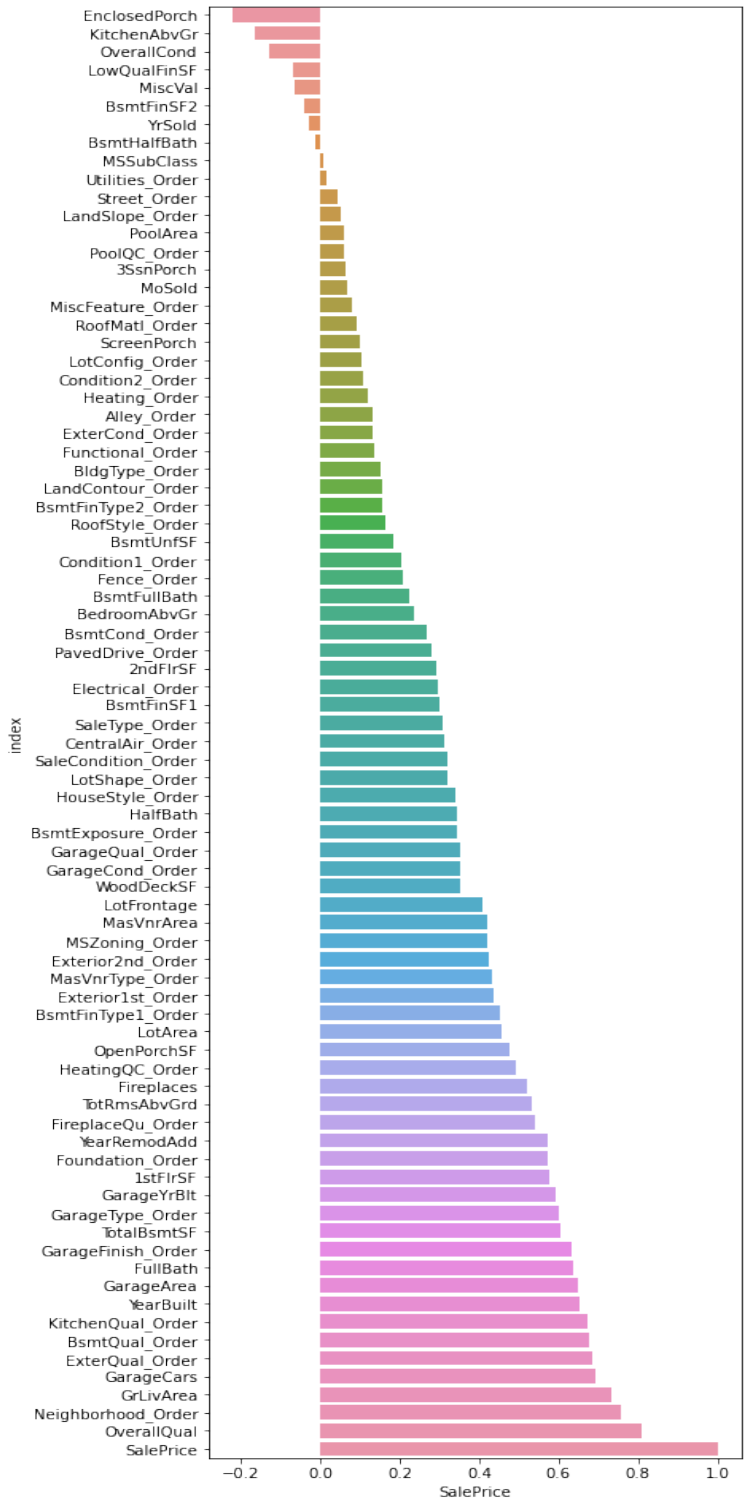

Figure 7: Spearman correlation result

As we can see clearly, OverallQual is the main criterion influencing the sales price. Neighborhood seems to be the second strongest factor, which has some

implications that are worth addressing. First, the importance could partially be that certain neighborhoods have some intrinsic value in themselves, but we should also note that houses in the same neighborhood tend to share similar characteristics (e.g., similar house style, similar distance to the railway, etc.) that might lead to similar valuations. Therefore, correlation heat maps are made to further examine if any confounding/linearity is present among different features.

```
In [18]: plt.figure(figsize=(15,10))
         corr_ = df_train[quantitative+['SalePrice']].corr()
         sns.heatmap(corr_)
         plt.figure(figsize=(15,10))
         corr_ = df_train[qualitative_ranked+['SalePrice']].corr()
         sns.heatmap(corr_)
```

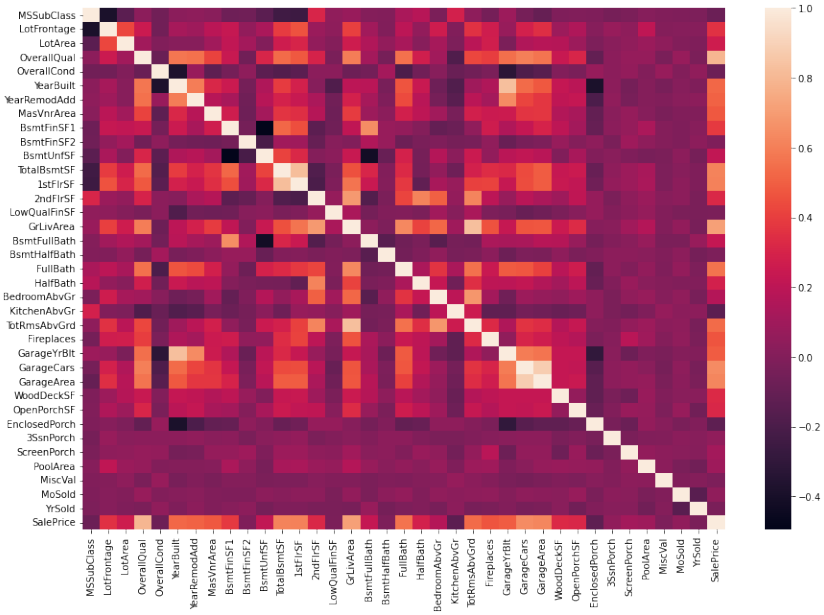

Figure 8: Heat-map for quantitative features

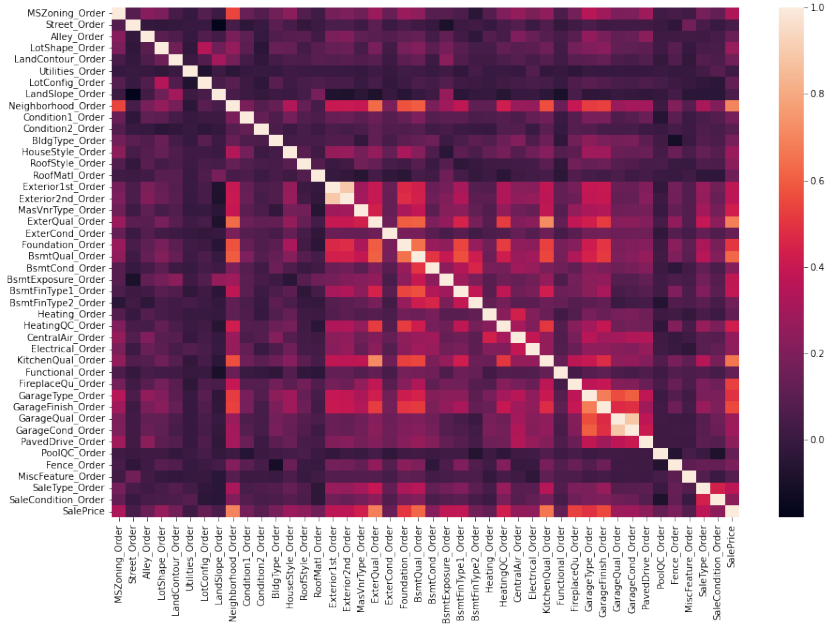

Figure 9: Heat-map for categorical features

Looking at the heat-map for quantitative features, two light-colored squares stand out. The first pair corresponds to the variables "TotalBsmtSF" and "1stFlrSF," and the second pair to the variables "GarageArea" and "GarageCars." In both instances, there is a very strong correlation between these variables, one that is so substantial as to suggest multicollinearity. In reality, if we consider these factors experimentally, we can really say that they provide information that is quite similar. For example, if we think of an average house, the basement usually shares the same floor plan with (part of) the ground floor, hence the strong correlation between these two variables; and the high correlation between the garage area and how many cars fit in the garage is even easier to explain - logically, the bigger the garage, the more cars that will fit in. Similarly, in the heat-map for categorical features, the high correlation between "Exterior1st_Order" and "Exterior2nd_Order" as well as between "GarageQual_Order" and "GarageCond_Order" can be explained. As such, we can conclude that multicollinearity does occur, and we should avoid including "duplicating" features in the models.

To select the most relevant, a clearer heat map is made for the top 15 features that have the largest influence on the sales price. Similar to the issue from the heat maps just now, I will exclude "GarageYrBlt" and "GarageArea" for the model given their "twins" ("YearBlt" and "GarageCars" have a higher correlation with the sales price.

In [19]:
```
#saleprice correlation matrix
k = 15 #number of variables for heatmap
cols = corr_all.iloc[-k:]['index']
corr_ = corr_all.iloc[-k:][list(cols) + ['index']].copy().set_index('index')
plt.figure(figsize=(len(corr_), len(corr_)))
sns.set(font_scale=1.25)
hm = sns.heatmap(corr_, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10})
plt.show()
```

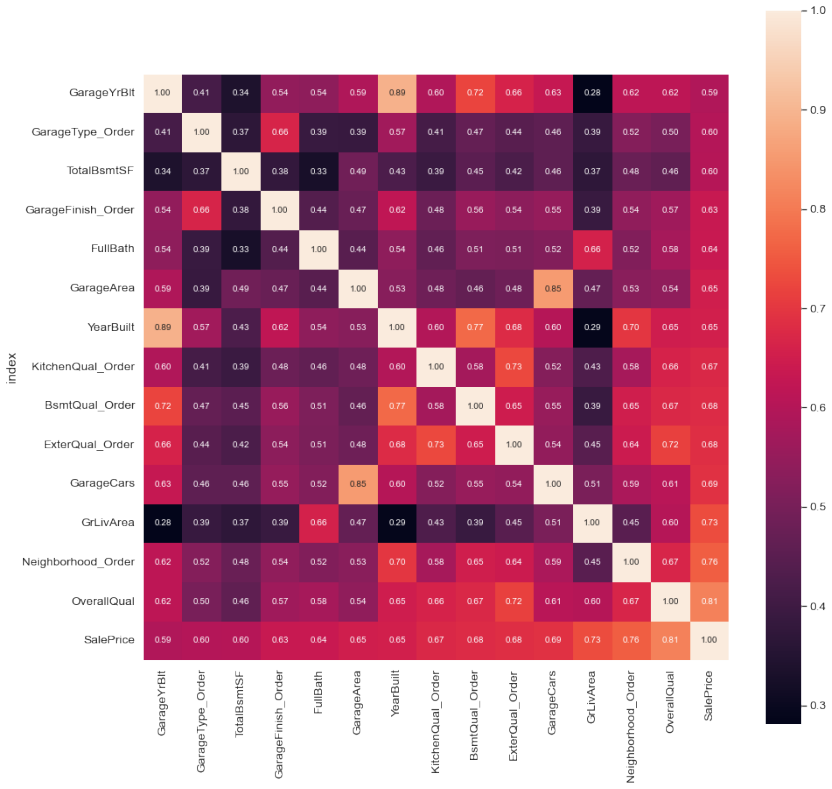

Figure 10: Heat-map for top 15 features

2.4 Missing Data

Thank to the extensive EDA performed in the last few sections, we now have a deeper understanding of the data set and have more confidence in knowing how to deal with missing data. The figure below revisits the missing data component of the data set.

	Total	Percentage
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685

Figure 11: Missing data count and percentage

A rule of thumb commonly used is that when more than 15% of data is missing for a certain feature, we should delete the corresponding feature. This implies that in certain situations, we won't use any tricks to fill in the missing data. This suggests that we should remove a number of variables, including "PoolQC," "MiscFeature," "Alley," and others. From the EDA analysis, none of these factors don't seem to stand out. Given that hardly every house sold would have a pool or fireplace in the first place, we might also claim that variables like "PoolQC," "MiscFeature," and "FireplaceQu" are excellent candidates for outliers. Since these variables do not have a strong correlation with the sales price and are not representative of all houses sold, we can simply delete them.

Moving on to the next set of variables, all garage-related variables have the same number of missing data - very likely stemming from the same set of observations. Since "GarageCars" expresses the most crucial information about garages (as seen from earlier analysis) and considering that only 5% of total records have these variables missing. We can then garage-related variables on the list. The same logic applies to all basement-related variables as well.

We can assume that "MasVnrArea" and "MasVnrType" are not necessary variables. They also strongly correlate with "YearBuilt" and "OverallQual," which are previously recognized as crucial attributes and don't have a missing data problem. Thus, if we remove "MasVnrArea" and "MasVnrType," we won't lose a lot of data.

Finally, "Electrical" is lacking one observation. We'll maintain the variable and remove this observation as there was just one made. In conclusion, we will only remove the observation with missing "Electrical" data and will not delete any of the characteristics with missing data other than the variable "Electrical".

2.5 Potential Features

Based on the above analysis, we select 6 highly correlated features.

1. Quantitative:

   GrLivArea (Above grade (ground) living area square feet)

   GarageArea (Size of garage in square feet)

   TotalBsmtSF (Total square feet of basement area)

2. Qualitative/Categorical:

   OverallQual (Rates the overall material and finish of the house)

   Neighborhood (Physical locations within Ames city limits)

   YearBuilt (Original construction date)

The ground living area plot shows there is an approximately linear relationship between the house sale price and the ground living area.

In [21]:
```
potential_features = ['OverallQual','GrLivArea','GarageArea','YearBuilt','TotalBsmtSF','Neighborhood']
```

In [22]:
```
df_train.plot.scatter(x='OverallQual', y='SalePrice', ylim=(0,800000),figsize = (25,5))
df_train.plot.scatter(x='GrLivArea', y='SalePrice', ylim=(0,800000),figsize = (25,5))
df_train.plot.scatter(x='GarageArea', y='SalePrice', ylim=(0,800000),figsize = (25,5))
df_train.plot.scatter(x='YearBuilt', y='SalePrice', ylim=(0,800000),figsize = (25,5))
df_train.plot.scatter(x='TotalBsmtSF', y='SalePrice', ylim=(0,800000),figsize = (25,5))
df_train.plot.scatter(x='Neighborhood', y='SalePrice', ylim=(0,800000),figsize = (25,5))
```

Figure 12: Scatter Plot of SalePrice v.s. GrLivArea

We can notice a sizable number of observations with 0 values (properties without a basement or garage) in the two graphs below. a significant issue because the log conversions cannot be performed with the value zero. We'll make a variable that can reflect whether or not there is a basement in order to apply a log transformation in this case (binary variable). Then, excluding observations containing zeros, we will apply a log transformation to all the non-zero data. In this manner, we may modify data without losing the impact of a basement's presence or absence.

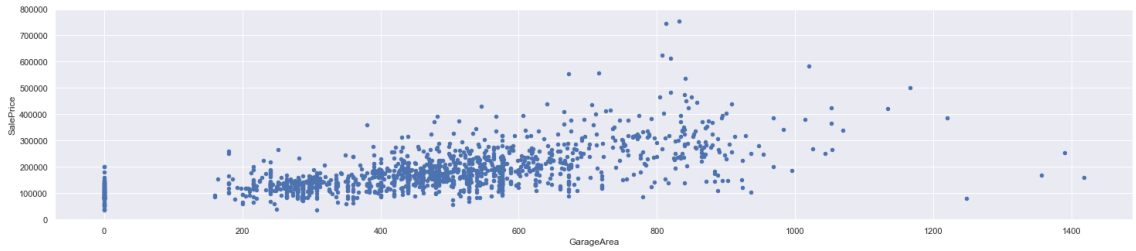

Figure 13: Scatter Plot of SalePrice v.s. GarageArea

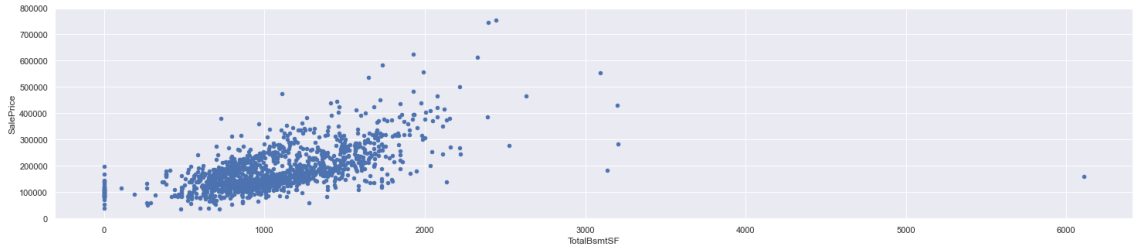

Figure 14: Scatter Plot of SalePrice v.s. TotalBsmtSF

For the following three categorical data, there are big differences among different groups of each variable, which means these variables provide useful information for the later regression task.

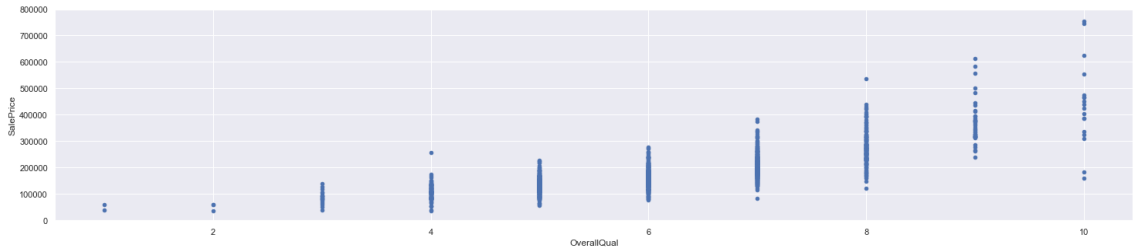

Figure 15: Scatter Plot of SalePrice v.s. OverallQual

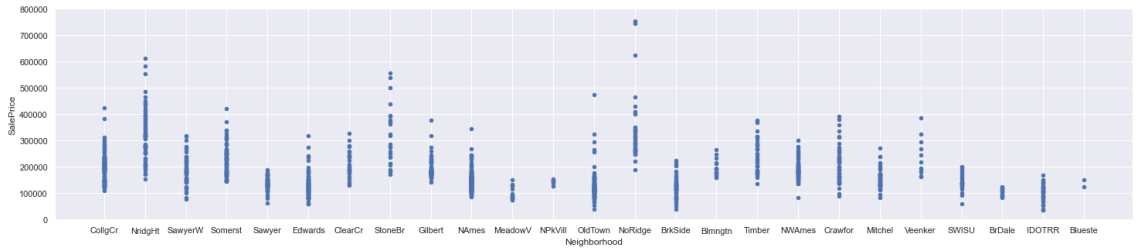

Figure 16: Scatter Plot of SalePrice v.s. Neighborhood

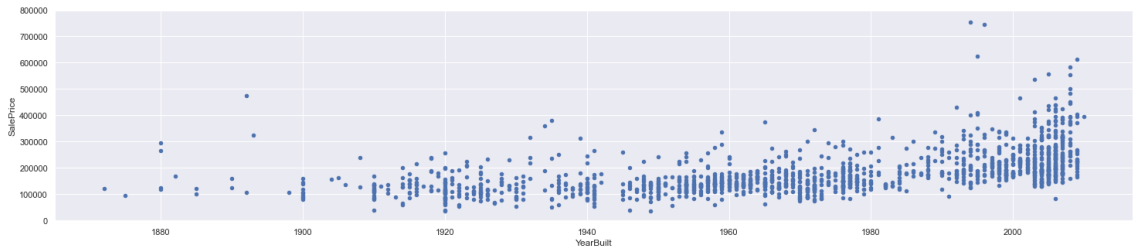

Figure 17: Scatter Plot of SalePrice v.s. YearBuilt

3 Data Preparation and Feature Engineering

Through the above analysis, we have finalized 6 independent variables and the dependent variable. In this section, we focus on these 7 variables and clean them only.

3.1 Outliers

Establishing the right threshold helps us identify what sales price data is an outlier and what data is not is at the core of this section. The starting point of

this analysis will be to standardize all data to have a mean of 0 and a standard deviation of 1 to get rid of the effects from scale.

```
saleprice_scaled = StandardScaler().fit_transform(df_train['SalePrice'][:,np.newaxis])
low_range = saleprice_scaled[saleprice_scaled[:,0].argsort()][:10]
high_range= saleprice_scaled[saleprice_scaled[:,0].argsort()][-10:]
print('outer range (low) of the distribution:')
print(low_range)
print('outer range (high) of the distribution:')
print(high_range)
```

```
outer range (low) of the distribution:
[[-1.83870376]
 [-1.83352844]
 [-1.80092766]
 [-1.78329881]
 [-1.77448439]
 [-1.62337999]
 [-1.61708398]
 [-1.58560389]
 [-1.58560389]
 [-1.5731    ]]
outer range (high) of the distribution:
[[3.82897043]
 [4.04098249]
 [4.49634819]
 [4.71041276]
 [4.73032076]
 [5.06214602]
 [5.42383959]
 [5.59185509]
 [7.10289909]
 [7.22881942]]
```

Figure 18: Outlier Detection

The two outer range values that are greater than 7 are concerning. Hence, a scatter plot is made for GrLivArea vs. Sales Price to visualize them better (see Figure below). Surprisingly, the two points in the top right corner that corresponds to the properties with the > 7 sales price after standardization actually seem to follow the group trendline. Whereas the two points in the bottom right corner seem to be out of place - with a very large living area but a very low price. Although it could be speculated that these might be properties in the countryside, but since we can't be sure of this assumption and these two points are not representative of an average property sold in Ames, we can proceed with deleting them.

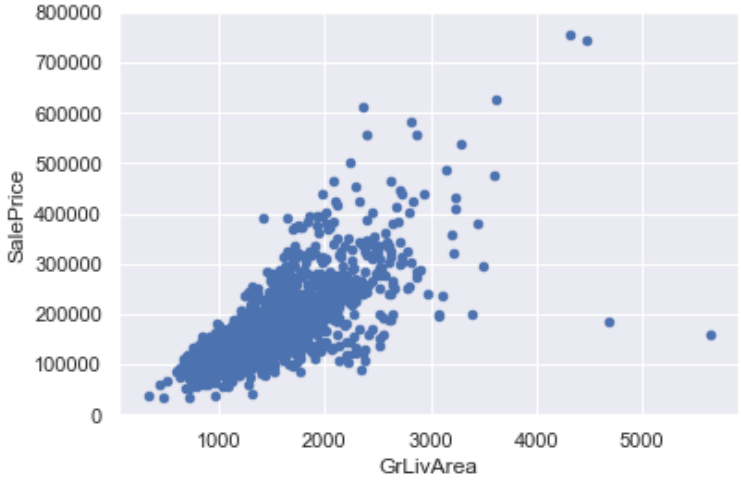

Figure 19: Outlier Visualization

3.2 One-Hot Encoding

For categorical data which are in text form instead of numerical numbers, we need to convert them into dummy variables or enumerate them. In this research, one-hot encoding is applied to map text categories into their corresponding numerical categories. For example, the Neighborhood will be transformed into Neighborhood_Order.

In [26]:
```
col = 'Neighborhood'
def categorical_data_transform(df,col):
    df_ = df[[col]].copy().groupby(col).mean().reset_index().reset_index().rename(columns={'index':col+'_Order'})
    df = pd.merge(df,df_,left_on=col,right_on=col,how='inner').drop(columns=[col])
    return df
df_train = categorical_data_transform(df_train,col)
df_train
```

Out[26]:

	Id	OverallQual	GrLivArea	GarageArea	YearBuilt	TotalBsmtSF	SalePrice	Neighborhood_Order
0	1	7	1710	548	2003	856	208500	5
1	3	7	1786	608	2001	920	223500	5
2	14	7	1494	840	2006	1494	279500	5
3	23	8	1795	534	2002	1777	230000	5
4	33	8	1234	484	2007	1234	179900	5
...	...	...	...	...	...	...	...	...
1453	1361	5	2601	621	1921	612	189000	18
1454	1377	6	790	160	1930	768	91000	18
1455	1400	6	1608	216	1925	976	137450	18
1456	600	6	1556	452	1980	716	151000	1
1457	957	6	1229	462	1980	561	124000	1

1458 rows × 8 columns

3.3 Transformation

Since there are zero values in GrLivArea and TotalBsmtSF, we may need to exclude them first, then do log-transformation."It is recommended that log

transformed analysis should frequently be preferred to untransformed analysis"[1]. In addition, GrLivArea and SalePrice will be directly applied log-normal transformation.

In [27]:
```
def features_handling(df):
    df['GrLivArea_Log'] = np.log(df['GrLivArea'])

    df['HasGarage'] = pd.Series(len(df['GarageArea']), index=df.index)
    df['HasGarage'] = 0
    df.loc[df['GarageArea']>0,'HasGarage'] = 1
    df.loc[df['HasGarage']==1,'GarageArea_Log'] = np.log(df['GarageArea'])
    df.loc[df['HasGarage']==0,'GarageArea_Log'] = 0

    df['HasBsmt'] = pd.Series(len(df['TotalBsmtSF']), index=df.index)
    df['HasBsmt'] = 0
    df.loc[df['TotalBsmtSF']>0,'HasBsmt'] = 1
    df.loc[df['HasBsmt']==1,'TotalBsmtSF_Log'] = np.log(df['TotalBsmtSF'])
    df.loc[df['HasBsmt']==0,'TotalBsmtSF_Log'] = 0
    return df
```

In [28]:
```
df_train['SalePrice_Log'] = np.log(df_train['SalePrice'])
df_train = features_handling(df_train)
df_train
```

Out[28]:

a	GarageArea	YearBuilt	TotalBsmtSF	SalePrice	Neighborhood_Order	SalePrice_Log	GrLivArea_Log	HasGarage	GarageArea_Log	HasBsmt	TotalBsmtSF_Log
0	548	2003	856	208500	5	12.247694	7.444249	1	6.306275	1	6.752270
6	608	2001	920	223500	5	12.317167	7.487734	1	6.410175	1	6.824374
4	840	2006	1494	279500	5	12.540758	7.309212	1	6.733402	1	7.309212
5	534	2002	1777	230000	5	12.345835	7.492760	1	6.280396	1	7.482682
4	484	2007	1234	179900	5	12.100156	7.118016	1	6.182085	1	7.118016
..	...	...	...	...	...	...	...	...	...	...	...
1	621	1921	612	189000	18	12.149502	7.863651	1	6.431331	1	6.416732
0	160	1930	768	91000	18	11.418615	6.672033	1	5.075174	1	6.643790
8	216	1925	976	137450	18	11.831015	7.382746	1	5.375278	1	6.883463
6	452	1980	716	151000	1	11.925035	7.349874	1	6.113682	1	6.573680
9	462	1980	561	124000	1	11.728037	7.113956	1	6.135565	1	6.329721

4 Model Exploration

4.1 Linear Regression

When modeling the relationship between dependent and independent variables in statistics, multivariate linear regression would be the first considered choice. Given a data set $\{y_i, x_{i1}, \ldots, x_{i_p}\}_{i=1}^n$ of n records, linear regression assumes the there is a linear relationship between dependent variable y and p-vector independent variables x. This relationship is modeled through a random variable . Thus the model takes the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{i_p} + \epsilon_i = \mathbf{x}_i^{\mathsf{T}} \beta + \epsilon_i, \qquad i = 1, \ldots, n$$

And the linear model can be written in matrix notation as

$$\mathbf{y} = X\beta + \epsilon$$

Give a data set, the linear regression model is fitted by estimating the regression coefficients β such that the error term $\epsilon = \mathbf{y} - X\beta$ can be minimized. And in practice, the sum of squared errors $||\epsilon||_2^2$ as a measure of ϵ are used for minimization, which is

$$\min_{\beta} ||\mathbf{y} - X\beta||_2^2$$

.

Linear Regression

```
In [10]: from sklearn.linear_model import LinearRegression

linear_regressor = LinearRegression(normalize=True)
linear_regressor.fit(X_train, y_train)

train_pred = linear_regressor.predict(X_train)
test_pred = linear_regressor.predict(X_test)
```

4.2 Lasso Regression

Compared to linear regression, lasso regression performs regularization as well so that it can improve the predicting accuracy. The L1 regularization process used by lasso regression results in a penalty proportional to the absolute magnitude of the coefficients. Larger penalties can provide coefficient values that are closer to zero, which is great for creating more straightforward models. And the minimization method is denoted as

$$\min_{\beta} \frac{1}{N} ||\mathbf{y} - X\beta||_2^2 + \lambda ||\beta||_1$$

.

Lasso Regression

```
In [12]: from sklearn.linear_model import Lasso

lasso_regressor = Lasso(alpha=0.1, precompute=True, positive=True, selection='random',random_state=0)
lasso_regressor.fit(X_train, y_train)
train_pred = lasso_regressor.predict(X_train)
test_pred = lasso_regressor.predict(X_test)
```

4.3 Decision Tree

Decision tree is a supervised learning algorithm, which can be used for regression topics since there is a hierarchical tree structure consisting of the root node, internal node as well as leaf node. "In most of the cases, the discrete splitting functions are univariate. Univariate means that an internal node is split according to the value of a single attribute. Consequently, the inducer searches for the best attribute upon which to split."[3]

Normally, information gain and Gini impurity can be served as the splitting criterion. Entropy values range from 0 to 1. Entropy will be equal to zero if the data set S contains only samples that fall into a single class. Entropy will be at its peak at 1 if half the samples are assigned to one class and the other half to a different class. The characteristic with the least degree of entropy should be utilized to choose the best feature to split on and identify the best decision

tree. Entropy before and after a split on a particular property is different, and information gain is that difference. The characteristic that performs the best at categorizing the training data in accordance with its goal will result in the best split.

Decision Tree Regressor

```
In [13]: from sklearn.tree import DecisionTreeRegressor
         dt_Regressor = DecisionTreeRegressor()
         dt_Regressor.fit(X_train, y_train)
         train_pred = dt_Regressor.predict(X_train)
         test_pred = dt_Regressor.predict(X_test)
```

4.4 Random Forest

Random forests are an ensemble learning technique that builds a large number of decision trees for classification and regression. For regression tasks, the average forecast of each individual tree is returned. Random forests correct decision trees' tendency to overfit their training set. Building many distinct trees while only considering a subset of the input may increase generalization accuracy.

Random forest training algorithm uses the typical bootstrap aggregation approach (bagging). Every time, a random training sample is chosen, and the sample is fitted into the decision tree algorithm. "Each tree is built on a different bootstrap sample. Each bootstrap sample randomly leaves out about one-third of the observations."[5]. After training, the predictions will be the average value generated by all the trained decision trees.

Random Forest Regressor

```
In [14]: from sklearn.ensemble import RandomForestRegressor
         rf_Regressor = RandomForestRegressor()
         rf_Regressor.fit(X_train, y_train)
         train_pred = rf_Regressor.predict(X_train)
         test_pred = rf_Regressor.predict(X_test)
```

4.5 KNN

K-nearest neighbors algorithm (KNN) is also a non-parametric supervised learning algorithm. The projected value in KNN regression is the average of the values of the k closest neighbors. Vectors with class labels in a multidimensional feature space make up the training samples. During the training phase of the

method, just the feature vectors and class labels of the training examples are kept.

The first step to use KNN regressor is to determine the distance calculation method. There are several ways to figure out this distance, but the three that are most widely used are the Euclidian, Manhattan (for continuous), and Hamming distances (for categorical). Selecting the k value is the next step. This specifies how many neighbors we take a look at before giving each new observation a value.

KNN Regressor

In [15]:
```
from sklearn.neighbors import KNeighborsRegressor
knn_regressor=KNeighborsRegressor()
knn_regressor.fit(X_train,y_train)

train_pred = knn_regressor.predict(X_train)
test_pred = knn_regressor.predict(X_test)
```

4.6 XGBoost[6]

XGBoost(eXtreme Gradient Boosting) is an open-source library that provides a regularizing gradient boosting framework. A second-order Taylor approximation is utilized in the loss function to create the connection to the Newton Raphson technique, which is how XGBoost operates in function space as opposed to gradient boosting, which operates in function space as gradient descent.

There are mainly four inputs in the model: a training set, a differentiable loss function, a number of weak learners, and a learning rate. The first step is to initialize the model with a constant value:

$$\hat{f}_{(0)}(x) = \min_{\theta} \sum_{i=1}^{N} L(y_i, \theta)$$

For m = 1 to M:

1. Compute gradients and hessians:

$$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}$$

$$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}$$

2. Fit a weak learner by solving the optimization problem below:

$$\hat{\phi}_m = \min_{\phi} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)\right]^2$$

$$\hat{f}_m(x) = \alpha\hat{\phi}_m(x)$$

3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x)$$

The predicted value is also the average value shown below:

$$\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$$

XGBoost Regressor

```
In [16]: from xgboost import XGBRegressor
         xgb_regressor = XGBRegressor()
         xgb_regressor.fit(X_train, y_train)

         train_pred = xgb_regressor.predict(X_train)
         test_pred = xgb_regressor.predict(X_test)
```

4.7 Stacked Model

The stacked regressor is a bagging method of Ensemble Learning. Some well-performed models will be combined together to generate a better result. This stacked method can also avoid the over-fitting problem since it takes the average predictions of all the effective models.

5 Metrics

In regression, accuracy cannot be utilized since it is designed for classification tasks. The accuracy of a regression model's predictions must be given as an error. There are three error metrics as well as R squared used in this paper.

5.1 MSE

Mean Squared Error (MSE) is calculated as the average of squared differences between predictions and real values. It is always a positive number that becomes smaller as the error gets closer to zero since it is derived from the square of the Euclidean distance.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

5.2 RMSE

Root Mean Squared Error (RMSE) is an extension of MSE. The square root of the error indicates that the RMSE's units are the same as the original units of the forecasted target value.

The mean error score is inflated by MSE and RMSE, which penalize larger errors more severely than smaller errors. The MAE does not give distinct sorts of mistakes more or less weight; instead, the scores rise linearly as the number of errors rises.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

5.3 MAE

Mean Absolute Error (MAE) is the absolute value of the differences. The error's units are the same as the target value's units, much like RMSE. The changes in MAE are linear and straightforward, in contrast to RMSE.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

5.4 R^2

R^2 is the coefficient of determination, which is the proportion of variation in the dependent variable. The variance of one variable's explanation for the variance of the second variable is measured by R-squared[4]. The sum of squares of residuals:

$$SS_{residual} = \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n}\epsilon_i^2$$

The total sum of squares (proportional to the variance of the data):

$$SS_{total} = \sum_{i=1}^{n}(Y_i - \bar{Y})^2$$

The general form of the coefficient of determination:

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

Model Training

In [7]:
```
features = ['OverallQual','GrLivArea_Log','GarageArea_Log','YearBuilt','TotalBsmtSF_Log','Neighborhood_Order']
target = 'SalePrice_Log'
X = df_train[features].copy()
y = df_train[target].copy()
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

In [9]:
```
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

6 Model Selection and Performance

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm,lognorm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import dataframe_image as dfi
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
df_train_original = pd.read_csv('train.csv')
potential_features = ['OverallQual','GrLivArea','GarageArea','YearBuilt','TotalBsmtSF','Neighborhood']
df_train  = df_train_original[['Id'] + potential_features + ['SalePrice']]
df_train
```

Out[2]:

	Id	OverallQual	GrLivArea	GarageArea	YearBuilt	TotalBsmtSF	Neighborhood	SalePrice
0	1	7	1710	548	2003	856	CollgCr	208500
1	2	6	1262	460	1976	1262	Veenker	181500
2	3	7	1786	608	2001	920	CollgCr	223500
3	4	7	1717	642	1915	756	Crawfor	140000
4	5	8	2198	836	2000	1145	NoRidge	250000
...	...	...	...	...	...	...	...	...
1455	1456	6	1647	460	1999	953	Gilbert	175000
1456	1457	6	2073	500	1978	1542	NWAmes	210000
1457	1458	7	2340	252	1941	1152	Crawfor	266500
1458	1459	5	1078	240	1950	1078	NAmes	142125
1459	1460	5	1256	276	1965	1256	Edwards	147500

1460 rows × 8 columns

In [4]:
```
col = 'Neighborhood'
def categorical_data_transform(df,col):
    df_ = df[[col]].copy().groupby(col).mean().reset_index().reset_index().rename(columns={'index':col+'_Order'})
    df = pd.merge(df,df_,left_on=col,right_on=col,how='inner').drop(columns=[col])
    return df
df_train = categorical_data_transform(df_train,col)
df_train
```

Out[4]:

	Id	OverallQual	GrLivArea	GarageArea	YearBuilt	TotalBsmtSF	SalePrice	Neighborhood_Order
0	1	7	1710	548	2003	856	208500	5
1	3	7	1786	608	2001	920	223500	5
2	14	7	1494	840	2006	1494	279500	5
3	23	8	1795	534	2002	1777	230000	5
4	33	8	1234	484	2007	1234	179900	5
...	...	...	...	...	...	...	...	...
1453	1361	5	2601	621	1921	612	189000	18
1454	1377	6	790	160	1930	768	91000	18
1455	1400	6	1608	216	1925	976	137450	18
1456	600	6	1556	452	1980	716	151000	1
1457	957	6	1229	462	1980	561	124000	1

1458 rows × 8 columns

In [5]:
```
def features_handling(df):
    df['GrLivArea_Log'] = np.log(df['GrLivArea'])

    df['HasGarage'] = pd.Series(len(df['GarageArea']), index=df.index)
    df['HasGarage'] = 0
    df.loc[df['GarageArea']>0,'HasGarage'] = 1
    df.loc[df['HasGarage']==1,'GarageArea_Log'] = np.log(df['GarageArea'])
    df.loc[df['HasGarage']==0,'GarageArea_Log'] = 0

    df['HasBsmt'] = pd.Series(len(df['TotalBsmtSF']), index=df.index)
    df['HasBsmt'] = 0
    df.loc[df['TotalBsmtSF']>0,'HasBsmt'] = 1
    df.loc[df['HasBsmt']==1,'TotalBsmtSF_Log'] = np.log(df['TotalBsmtSF'])
    df.loc[df['HasBsmt']==0,'TotalBsmtSF_Log'] = 0
    return df
```

In [6]:
```
df_train['SalePrice_Log'] = np.log(df_train['SalePrice'])
df_train = features_handling(df_train)
df_train
```

Out[6]:

a	GarageArea	YearBuilt	TotalBsmtSF	SalePrice	Neighborhood_Order	SalePrice_Log	GrLivArea_Log	HasGarage	GarageArea_Log	HasBsmt	TotalBsmtSF_Log
0	548	2003	856	208500	5	12.247694	7.444249	1	6.306275	1	6.752270
6	608	2001	920	223500	5	12.317167	7.487734	1	6.410175	1	6.824374
4	840	2006	1494	279500	5	12.540758	7.309212	1	6.733402	1	7.309212
5	534	2002	1777	230000	5	12.345835	7.492760	1	6.280396	1	7.482682
4	484	2007	1234	179900	5	12.100156	7.118016	1	6.182085	1	7.118016
..	...	...	...	...	...	...	...	...	...	...	...
1	621	1921	612	189000	18	12.149502	7.863651	1	6.431331	1	6.416732
0	160	1930	768	91000	18	11.418615	6.672033	1	5.075174	1	6.643790
8	216	1925	976	137450	18	11.831015	7.382746	1	5.375278	1	6.883463
6	452	1980	716	151000	1	11.925035	7.349874	1	6.113682	1	6.573680
9	462	1980	561	124000	1	11.728037	7.113956	1	6.135565	1	6.329721

In the model training, the independent variables are OverallQual, YearBuilt, the one-hot transformation of Neighborhood, and log transformations of GrLivArea, GarageArea, TotalBsmtSF. The target variable is log-transformation of SalePrice. This research splits 30% of the data set into testing set and the remaining 70% into training set.

As for Python packages applied to construct machine learning algorithms, this research mainly uses sklearn and xgboost for better model fitting:

1. Linear Regression: sklearn.linear_model.LinearRegression()
2. Lasso Regression: sklearn.linear_model.Lasso()
3. Decision Tree: sklearn.tree.DecisionTreeRegressor()
4. Random Forest: sklearn.ensemble.RandomForestRegressor()
5. XGBoost: xgboost.XGBRegressor()

6. Stacked Model: Linear Regression, Random Forest, and XGBoost

Stacked Regressor

In [17]:
```
regressor_ls = [linear_regressor,rf_Regressor,xgb_regressor]

train_pred = np.mean([regressor.predict(X_train) for regressor in regressor_ls],axis=0)
test_pred = np.mean([regressor.predict(X_test) for regressor in regressor_ls],axis=0)
result_stacked_train = ["Stacked Regression Train", *evaluate(y_train, train_pred)]
result_stacked_test = ["Stacked Regression Test", *evaluate(y_test, test_pred)]
results_df = pd.DataFrame(data=[result_stacked_train,result_stacked_test], columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df
```

Out[17]:

	Model	MAE	MSE	RMSE	R2 Square
0	Stacked Regression Train	0.056559	0.006080	0.077977	0.961578
1	Stacked Regression Test	0.109084	0.021793	0.147626	0.866155

6.1 Overall Metrics

The following graphs display there are significant overfitting problems in Decision Tree since R squared of training set is much higher than testing set. Moreover, linear regression is the most stable one since the error metrics and R squared is quite similar between training set and testing set. And lasso regression is worse than linear regression even though lasso regression includes more punishments.

Model	MAE	MSE	RMSE	R2 Square
Linear Regression Train	0.122996	0.028135	0.167735	0.822212
Lasso Regression Train	0.172885	0.055715	0.236041	0.647933
Decision Tree Regression Train	0.000775	0.000035	0.005908	0.999779
Random Forest Regression Train	0.041677	0.003470	0.058907	0.978072
KNN Regression Train	0.129632	0.033511	0.183061	0.788241
XGBoost Regression Train	0.017632	0.000630	0.025101	0.996019
Stacked Regression Train	0.056444	0.005976	0.077306	0.962236

Figure 20: Training Set Metrics

Model	MAE	MSE	RMSE	R2 Square
Linear Regression Test	0.129638	0.028841	0.169826	0.822871
Lasso Regression Test	0.176100	0.056548	0.237798	0.652709
Decision Tree Regression Test	0.151209	0.043788	0.209255	0.731076
Random Forest Regression Test	0.113284	0.025044	0.158254	0.846190
KNN Regression Test	0.161574	0.052813	0.229811	0.675645
XGBoost Regression Test	0.112859	0.022999	0.151655	0.858749
Stacked Regression Test	0.108803	0.021748	0.147472	0.866434

Figure 21: Testing Set Metrics

6.2 Error Metrics

According to the below error metrics (MSE, RMSE, and MAE), the stacked method outperform the other models since all the error metrics of it are the lowest. In addition, lasso regression, decision tree and KNN have shown poor predicting power, which means they are not quite useful in evaluating house sale prices in the current setting.

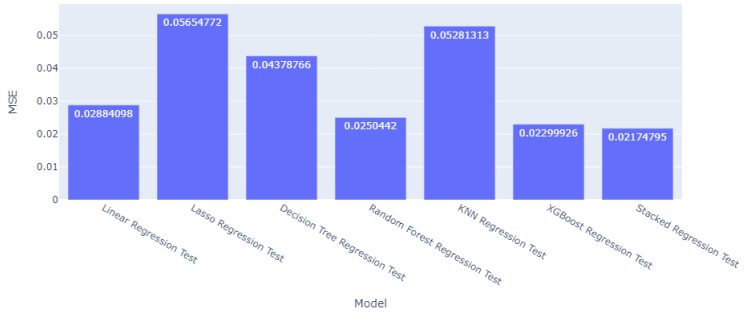

Figure 22: Testing Set MSE

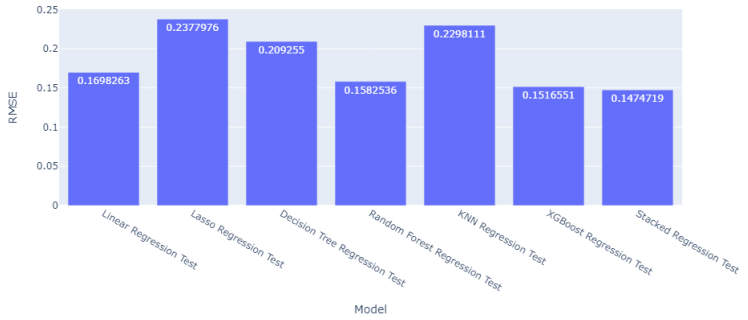

Figure 23: Testing Set RMSE

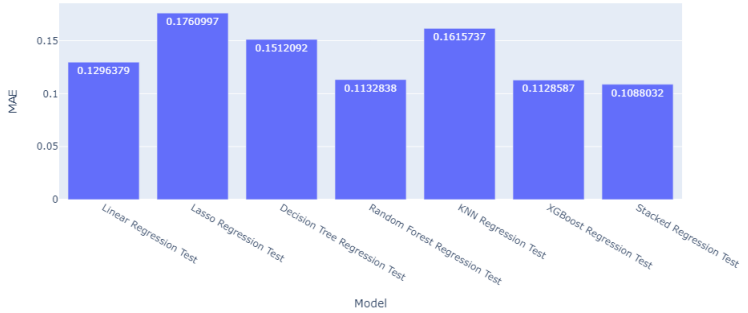

Figure 24: Testing Set MAE

6.3 R^2

In terms of R^2, the results align with the above error metrics that the stacked regression achieves the highest R squared. And 0.866 means the stacked regression explains 86.6% variance of the whole data set, which is relatively high. Thus, in the future house price evaluation, the stacked model will be applied as the major tool.

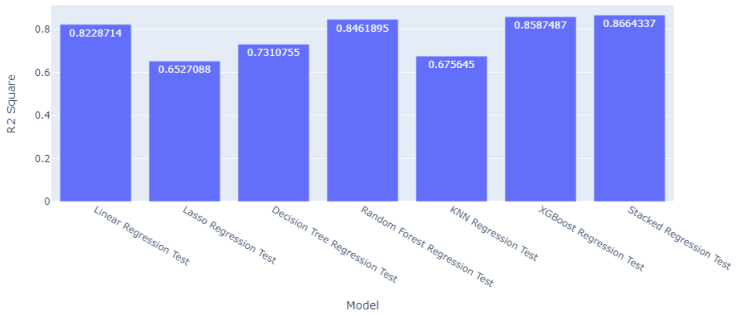

Figure 25: Testing Set R squared

7 Validation and Governance

Model governance is a framework that establishes how a project will carry out policies, manage model access, and keep track of activity. It's comparable to corporate governance, but it only pertains to machine learning models, therefore model risk management should be seen as a subset of it. Its goals are to increase productivity, decrease risks, and either lessen or completely remove the financial effect of any potential dangers.

Data and analytics governance should be in line with business goals. Governance initiatives must to be closely related to project aims and strategies. It is better to align governance standards and rules with project goals and process indicators to better support business results. I need to put the governance charter's focus on business value and prioritized goals, with explicit project criteria for success.

Since model risk is the possibility of unfavorable outcomes from choices made based on the wrong model choice, application, or use, to make machine learning model trustworthy, we need to clarify potential risk and harm. In order to monitor model health, stability and drift, this research has already adopted R-square, MSE, RMSE, and MAE. In addition, since this is a regression task, Area Under the Curve (AUC) and Receiver Operator Characteristics (ROC), Gini, may not be applicable.

Moreover, this project assumes the models can only be used to price houses in Ames, Iowa due to the data limitation. And another potential risk is that inflation/deflation will affect prices every year. It is better not to parallelly or directly compare house prices across different years.

In summary, it's crucial to build strict governance mechanisms with established operational controls on inputs and output that can rapidly recognize when an machine learning project starts to fail.

8 Discussion and Future Plan

In future research, more data sets of different cities should be included in order to make our model unbiased. Otherwise, the model cannot offer any advice to other cities except for Ames.

For the stacked model itself, the current method is using bagging method to get the average of predictions of three base models. Future research should also test whether boosting method can generate a better result and explore more possible techniques and machine learning algorithms such as LightGBM.

9 Conclusion

This research explores the house price-related data in Ames and selects key variables like overall quality and neighborhood. And mainstream machine learning algorithms such as linear regression and decision tree have been used on house sale price prediction. Measured by error metrics and R-square, the stacked model of linear regression, random forest and XGBoost achieve the most accurate predictions compared to the other single models. In later practice, the stacked model is supposed to be used for house price prediction.

References

[1] Oliver N. Keene. "The log transformation is special". In: *Statistics in Medicine* 14.8 (1995), pp. 811–819. DOI: https://doi.org/10.1002/sim.4780140810. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.4780140810. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.4780140810.

[2] Leann Myers and Maria J. Sirois. "Spearman Correlation Coefficients, Differences between". In: *Encyclopedia of Statistical Sciences*. John Wiley Sons, Ltd, 2006. ISBN: 9780471667193. DOI: https://doi.org/10.1002/0471667196.ess5050.pub2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471667196.ess5050.pub2. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471667196.ess5050.pub2.

[3] Lior Rokach and Oded Maimon. "Decision Trees". In: vol. 6. Jan. 2005, pp. 165–192. DOI: 10.1007/0-387-25465-X_9.

[4] Valentin Rousson and Nicoleta Francisca Goşoniu. "An R-square coefficient based on final prediction error". In: *Statistical Methodology* 4.3 (2007), pp. 331–340. ISSN: 1572-3127. DOI: https://doi.org/10.1016/j.stamet.2006.11.004. URL: https://www.sciencedirect.com/science/article/pii/S1572312706000529.

[5] Matthias Schonlau and Rosie Yuyan Zou. "The random forest algorithm for statistical learning". In: *The Stata Journal* 20.1 (2020), pp. 3–29. DOI: 10.1177/1536867X20909688. eprint: https://doi.org/10.1177/1536867X20909688. URL: https://doi.org/10.1177/1536867X20909688.

[6] Wikipedia. *XGBoost — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=XGBoost&oldid=1100542461. [Online; accessed 07-August-2022]. 2022.