Abstract

This project's primary objective is to develop a precise predictive model using Jupyter Notebook that can estimate house prices with a high level of accuracy. To achieve this goal, we employ advanced regression methods and a systematic data-driven approach.This project holds significant value for the Surprise Housing market, as it provides a powerful tool for making data-driven decisions in the real estate sector.

The solution is divided into the following sections:

- Data understanding and exploration
- Data cleaning
- Data preparation
- Model building and evaluation

OBJECTIVE

The objective of the "Advanced Regression - Surprise Housing" project is to develop a robust and accurate predictive model for estimating housing prices in the Surprise Housing market. This project aims to leverage advanced regression techniques and data analysis using Jupyter Notebook to achieve the following goals:

Predictive Accuracy: Create a regression model that accurately predicts house prices based on a set of relevant features and attributes. Evaluate and fine-tune the model to minimize prediction errors, ensuring that it provides reliable price estimates to assist home buyers and sellers.

Data Exploration and Analysis: Perform comprehensive data exploration and analysis to gain insights into the Surprise Housing market.Visualize key trends, relationships, and patterns in the data to inform feature selection and model interpretation.

Model Evaluation: Employ appropriate evaluation metrics, such as mean squared error (MSE), root mean squared error (RMSE), and R-squared (R^2), to assess the model's performance. Utilize cross-validation techniques to validate the model's generalization capabilities and detect overfitting.

By successfully achieving these objectives, this project aims to provide a valuable predictive tool for potential home buyers, sellers, and real estate professionals in the Surprise Housing market. Additionally, it serves as an educational resource for understanding advanced regression modeling techniques in data science

INTRODUCTION

The "**Advanced Regression - Surprise Housing**" project is a comprehensive data science endeavor aimed at harnessing the power of advanced regression techniques to accurately predict housing prices in the Surprise Housing market. The world of real estate is marked by a multitude of variables and factors that influence property values, making it a prime candidate for the application of sophisticated data analysis and predictive modeling.

Key Project Components:

- Data Exploration and Understanding
- Feature Engineering
- Regression Modeling
- Model Evaluation
- Model Interpretability
- Deployment
- Documentation and Reporting

By undertaking this project, we aim to not only provide a valuable tool for those involved in the Surprise Housing market but also contribute to the broader field of data science by demonstrating the application of advanced regression techniques. Through this journey, we seek to uncover the intricate factors influencing housing prices and provide actionable insights into this dynamic market.

METHOLOGY

1. **Data Collection and Understanding:** Begin by collecting the Surprise Housing dataset, which includes information on various housing attributes such as size, location, condition, and historical pricing data. Perform an initial data exploration to gain a comprehensive understanding of the dataset's structure, features, and any missing or erroneous data.

2. **Data Preprocessing and Cleaning:** Address missing data by imputing or removing entries as appropriate, ensuring data quality. Encode categorical variables using techniques such as one-hot encoding or label encoding.

3. **Data Visualization and Exploration:** Utilize data visualization libraries (e.g., Matplotlib and Seaborn) to generate plots and charts, providing insights into relationships between variables. Analyze data distribution, correlations, and outliers to inform feature selection and model building.

4. **Model Selection:**

Experiment with a variety of regression models, including but not limited to:

- Linear Regression
- Decision Trees
- Random Forest
- Gradient Boosting
- Lasso Regression

5. **Model Training and Evaluation:** Split the dataset into training and testing subsets to train and evaluate model performance. Utilize appropriate evaluation metrics, such as MSE, RMSE, and R^2, to assess the models' predictive accuracy. Employ cross-validation techniques to validate the model's generalization capability and detect overfitting.

By following this systematic methodology, we aim to develop an accurate and interpretable regression model that can effectively predict house prices in the Surprise Housing market.

CODE

jupyter surprise_housing (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

Let's now have a look at the dataset and understand the size, attribute names etc. in it.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

# To Scale our data
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import os
from sklearn.metrics import r2_score


# hide warnings
import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 100)


%matplotlib inline
```

In [2]:
```
# reading the dataset
housing = pd.read_csv("train.csv")
housing.head()
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Conditi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	N
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	N
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	N
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	N
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	N

In [3]:
```
#check missing percentage
percent_missing = (housing.isnull().sum() / housing.isnull().count()*100).sort_values(ascending = False)
percent_missing.head(15)
```

Out[3]:
```
PoolQC          99.520548
MiscFeature     96.301370
Alley           93.767123
Fence           80.753425
FireplaceQu     47.260274
LotFrontage     17.739726
GarageYrBlt      5.547945
GarageCond       5.547945
GarageType       5.547945
GarageFinish     5.547945
GarageQual       5.547945
BsmtFinType2     2.602740
BsmtExposure     2.602740
BsmtQual         2.534247
BsmtCond         2.534247
dtype: float64
```

In [4]:
```
# Here we are dropping Columns with high missing values ,above 40%

housing.drop(['PoolQC','MiscFeature','Alley','Fence','FireplaceQu'],axis=1,inplace=True)
```

In [5]:
```
housing[['LotFrontage','MasVnrArea','GarageYrBlt']].describe()
```

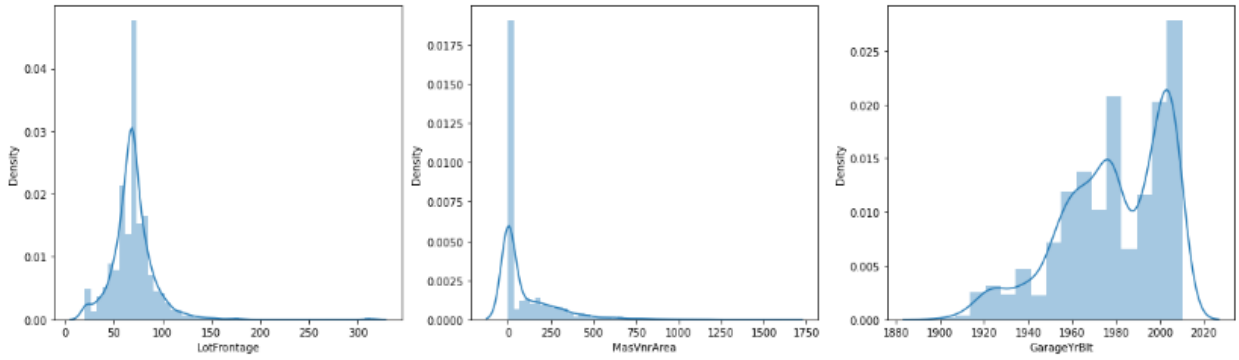
Out[5]:

	LotFrontage	MasVnrArea	GarageYrBlt
count	1201.000000	1452.000000	1379.000000
mean	70.049958	103.685262	1978.506164
std	24.284752	181.066207	24.689725
min	21.000000	0.000000	1900.000000
25%	59.000000	0.000000	1961.000000
50%	69.000000	0.000000	1980.000000
75%	80.000000	166.000000	2002.000000
max	313.000000	1600.000000	2010.000000

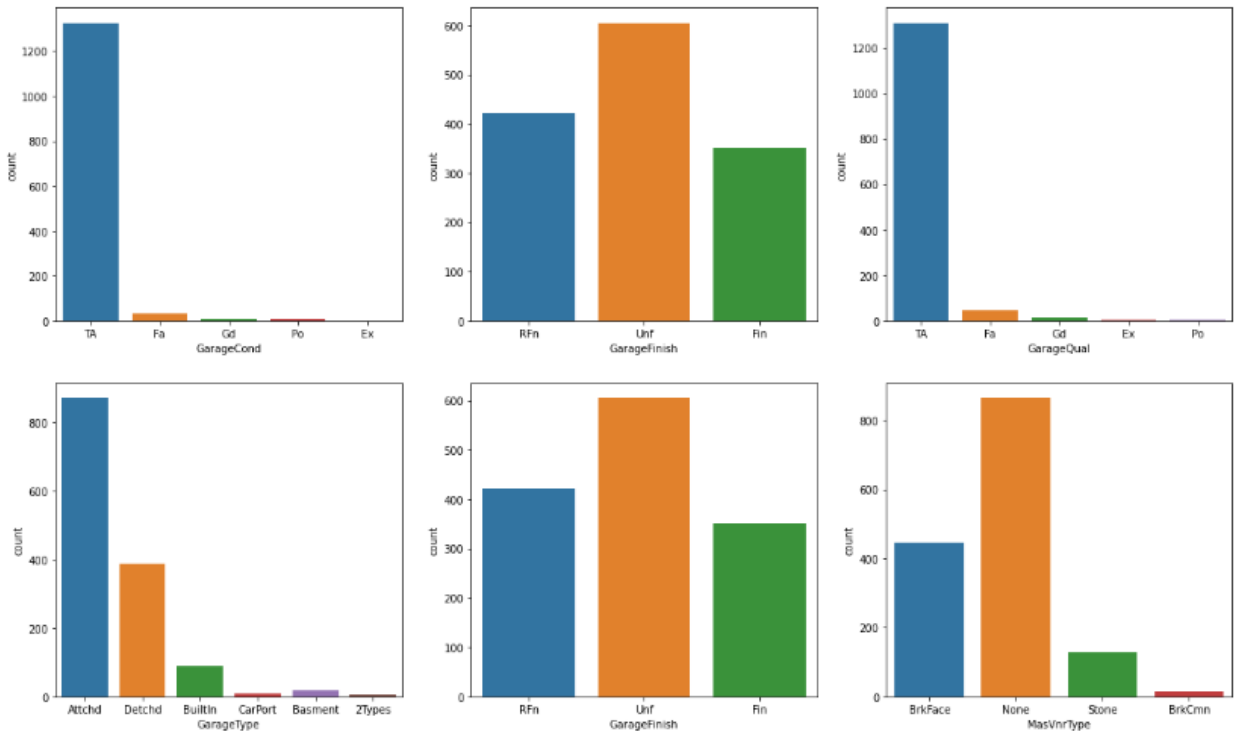
In [6]:
```
 #Treaing missing values by imputing for columns with missing values less than or equal to 40%

housing['LotFrontage']= housing.LotFrontage.fillna(housing.LotFrontage.median()) #Can see a presence of outlier so imputing the m
housing['MasVnrArea']= housing.MasVnrArea.fillna(housing.MasVnrArea.median())    # Can see presence of outlier
housing['GarageYrBlt']= housing.GarageYrBlt.fillna(housing.GarageYrBlt.mean())  # it looks stable with no outlier presence so we
```

In [7]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.distplot(housing['LotFrontage'])
plt.subplot(2,3,2)
sns.distplot(housing['MasVnrArea'])
plt.subplot(2,3,3)
sns.distplot(housing['GarageYrBlt'])
```

Out[7]: <AxesSubplot:xlabel='GarageYrBlt', ylabel='Density'>



In [8]:
```
#Here we'll be visualising the variables wih missing values

plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'GarageCond', data = housing)

plt.subplot(2,3,2)
sns.countplot(x = 'GarageFinish',  data = housing)

plt.subplot(2,3,3)
sns.countplot(x = 'GarageQual', data = housing)
plt.subplot(2,3,4)
sns.countplot(x = 'GarageType', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'GarageFinish',  data = housing)
plt.subplot(2,3,6)
sns.countplot(x = 'MasVnrType', data = housing)
plt.show()
```



In [9]:
```
# The cases when house doesn't have the garrage so replacing null with No Garrage

housing['GarageType'] = housing['GarageType'].replace(np.nan, 'No Garage')
housing['GarageFinish'] = housing['GarageFinish'].replace(np.nan, 'No Garage')
housing['GarageCond'] = housing['GarageCond'].replace(np.nan, 'No Garage')
housing['GarageQual'] = housing['GarageQual'].replace(np.nan, 'No Garage')
housing['MasVnrType'] = housing['MasVnrType'].replace(np.nan, 'None') # replacing nan with the top option of this field
housing['Electrical'] = housing['Electrical'].replace(np.nan, 'SBrkr') # replacing nan with the top option of this field
```

In [9]:
```
# The cases when house doesn't have the garrage so replacing null with No Garrage

housing['GarageType'] = housing['GarageType'].replace(np.nan, 'No Garage')
housing['GarageFinish'] = housing['GarageFinish'].replace(np.nan, 'No Garage')
housing['GarageCond'] = housing['GarageCond'].replace(np.nan, 'No Garage')
housing['GarageQual'] = housing['GarageQual'].replace(np.nan, 'No Garage')
housing['MasVnrType'] = housing['MasVnrType'].replace(np.nan, 'None') # replacing nan with the top option of this field
housing['Electrical'] = housing['Electrical'].replace(np.nan, 'SBrkr') # replacing nan with the top option of this field
```

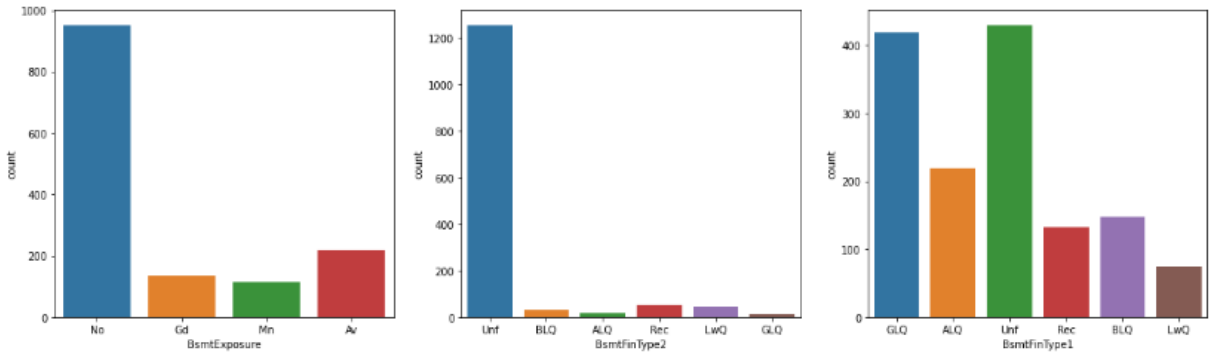
In [10]:
```
#changing the num to categorical so as to form these as dummy variables
housing['MSSubClass']=housing['MSSubClass'].replace({20:'1-STORY 1946 & NEWER ALL STYLES',30:'1-STORY 1945 & OLDER',40:'1-STORY W
        45:'1-1/2 STORY - UNFINISHED ALL AGES',
        50:'1-1/2 STORY FINISHED ALL AGES',
        60:'2-STORY 1946 & NEWER',
        70:'2-STORY 1945 & OLDER',
        75:'2-1/2 STORY ALL AGES',
        80:'SPLIT OR MULTI-LEVEL',
        85:'SPLIT FOYER',
        90:'DUPLEX - ALL STYLES AND AGES',
       120:'1-STORY PUD (Planned Unit Development) - 1946 & NEWER',
       150:'1-1/2 STORY PUD - ALL AGES',
       160:'2-STORY PUD - 1946 & NEWER',
       180:'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER',
       190:'2 FAMILY CONVERSION - ALL STYLES AND AGES'})

housing['OverallQual']=housing['OverallQual'].replace({ 10:'Very Excellent',
       9:'Excellent',
       8:'Very Good',
       7:'Good',
       6:'Above Average',
       5:'Average',
       4:'Below Average',
       3:'Fair',
       2:'Poor',
       1:'Very Poor'})
housing['OverallCond']=housing['OverallCond'].replace({ 10:'Very Excellent',
       9:'Excellent',
       8:'Very Good',
       7:'Good',
       6:'Above Average',
       5:'Average',
       4:'Below Average',
       3:'Fair',
       2:'Poor',
       1:'Very Poor'})
```

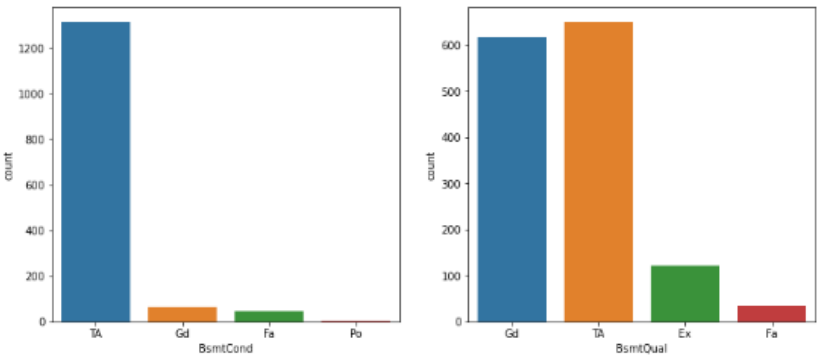
In [11]:
```
housing.head()
```

Out[11]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	B
0	1	2-STORY 1946 & NEWER	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	
1	2	1-STORY 1946 & NEWER ALL STYLES	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	
2	3	2-STORY 1946 & NEWER	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	
3	4	2-STORY 1945 & OLDER	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	
4	5	2-STORY 1946 & NEWER	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	

In [12]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'BsmtExposure', data = housing)

plt.subplot(2,3,2)
sns.countplot(x = 'BsmtFinType2',  data = housing)

plt.subplot(2,3,3)
sns.countplot(x = 'BsmtFinType1', data = housing)
plt.subplot(2,3,4)
sns.countplot(x = 'BsmtCond', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'BsmtQual',  data = housing)

plt.show()
```





In [13]:
```
# The cases when house doesn't have the basement so replacing null with No Basement

housing['BsmtExposure'] = housing['BsmtExposure'].replace(np.nan, 'No Basement')
housing['BsmtFinType2'] = housing['BsmtFinType2'].replace(np.nan, 'No Basement')
housing['BsmtFinType1'] = housing['BsmtFinType1'].replace(np.nan, 'No Basement')
housing['BsmtCond'] = housing['BsmtCond'].replace(np.nan, 'No Basement')
housing['BsmtQual'] = housing['BsmtQual'].replace(np.nan, 'No Basement')
```

In [14]:
```
#check missing percentage
percent_missing = (housing.isnull().sum() / housing.isnull().count()*100).sort_values(ascending = False)
percent_missing.head(10)
```

Out[14]:
```
Id              0.0
FullBath        0.0
Fireplaces      0.0
Functional      0.0
TotRmsAbvGrd    0.0
KitchenQual     0.0
KitchenAbvGr    0.0
BedroomAbvGr    0.0
HalfBath        0.0
BsmtHalfBath    0.0
dtype: float64
```

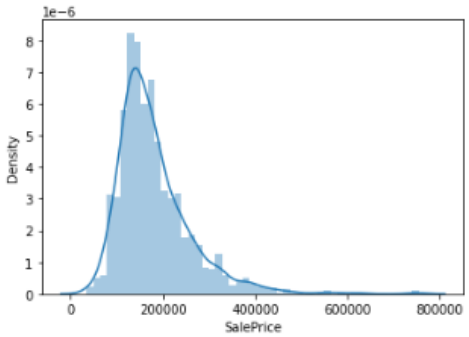
All the missing values has been treated

In [15]:
```
#Let's check the dependent variable i.e SalePrice

#A descriptive form of summary statistucs
housing['SalePrice'].describe()
```

Out[15]:
```
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
```

In [16]:
```
sns.distplot(housing['SalePrice']) #it's skewed
```

Out[16]:
```
<AxesSubplot:xlabel='SalePrice', ylabel='Density'>
```



In [17]:
```
#skewness and kurtosis
print("Skewness: %f" % housing['SalePrice'].skew())
```

```
Skewness: 1.882876
```

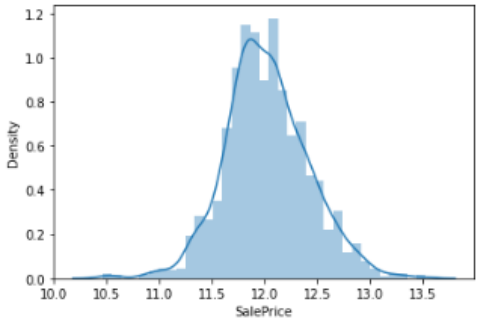
In [18]:
```
housing['SalePrice']=np.log(housing.SalePrice) #tranforming to form normal disribution
```

In [19]:
```
housing['SalePrice'].describe()
```

Out[19]:
```
count    1460.000000
mean       12.024051
std         0.399452
min        10.460242
25%        11.775097
50%        12.001505
75%        12.273731
max        13.534473
Name: SalePrice, dtype: float64
```

In [20]:
```
sns.distplot(housing['SalePrice']) # Normally distributed now
```

Out[20]:
```
<AxesSubplot:xlabel='SalePrice', ylabel='Density'>
```



In [21]:
```
#skewness
print("Skewness: %f" % housing['SalePrice'].skew())
```

```
Skewness: 0.121335
```

Now it can be said that the dependent variable SalePrice is normally distributed.

EDA

Now let's check the Categorical columns and their effect on price

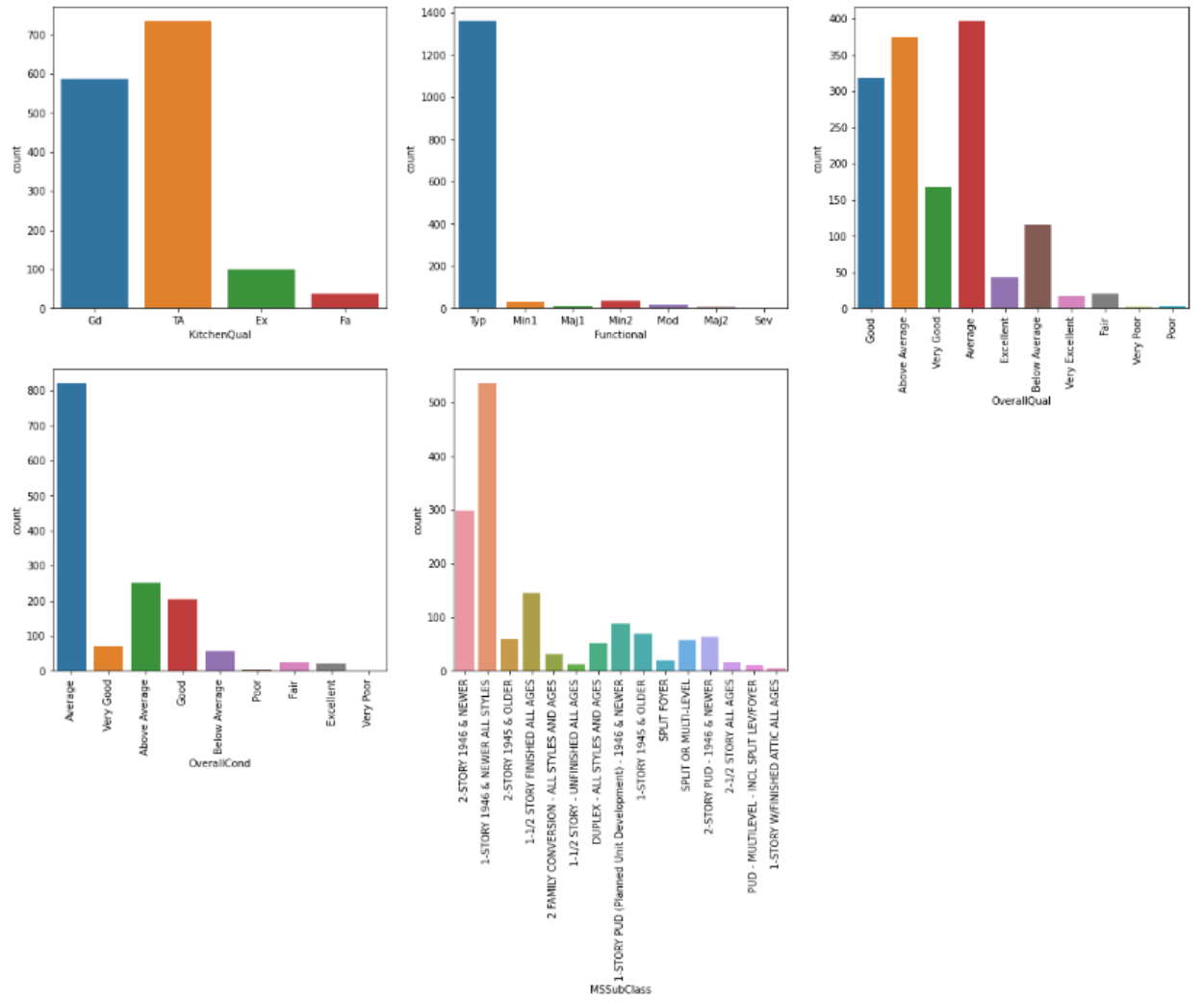
In [22]:
```
Cat = housing.select_dtypes(include=['object'])
Cat.columns
```

Out[22]:
```
Index(['MSSubClass', 'MSZoning', 'Street', 'LotShape', 'LandContour',
       'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1',
       'Condition2', 'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC',
       'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'GarageType',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'SaleType',
       'SaleCondition'],
      dtype='object')
```

In [23]:
```
# as we have many columns so let's make groups randomly and visualize them
# we would be making two graphs to understand firstly the countplot and secondaly effect of that variable on dependent variable
```

In [24]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'KitchenQual',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'Functional', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'OverallQual', data = housing)
plt.xticks(rotation=90)
plt.subplot(2,3,4)
sns.countplot(x = 'OverallCond', data = housing)
plt.xticks(rotation=90)

plt.subplot(2,3,5)
sns.countplot(x = 'MSSubClass',  data = housing)
plt.xticks(rotation=90)


plt.show()
```

```
plt.show()
```



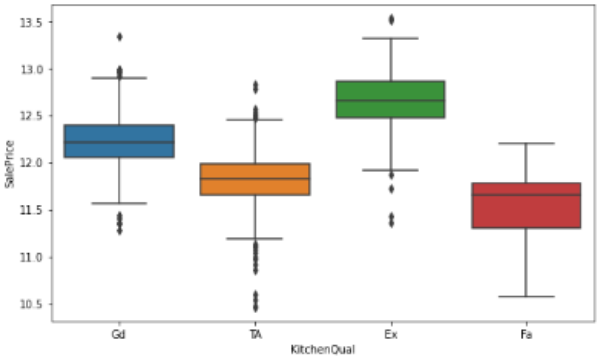
In [25]:
```
housing['MSSubClass'].value_counts() #Checking other columns for skewness
```

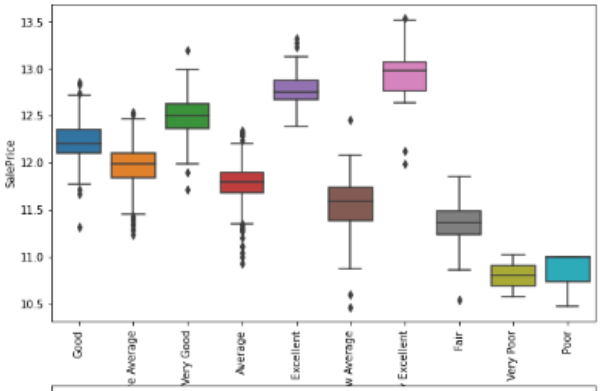
Out[25]:
```
1-STORY 1946 & NEWER ALL STYLES                        536
2-STORY 1946 & NEWER                                   299
1-1/2 STORY FINISHED ALL AGES                          144
1-STORY PUD (Planned Unit Development) - 1946 & NEWER   87
1-STORY 1945 & OLDER                                    69
2-STORY PUD - 1946 & NEWER                              63
2-STORY 1945 & OLDER                                    60
SPLIT OR MULTI-LEVEL                                    58
DUPLEX - ALL STYLES AND AGES                            52
2 FAMILY CONVERSION - ALL STYLES AND AGES               30
SPLIT FOYER                                             20
2-1/2 STORY ALL AGES                                    16
1-1/2 STORY - UNFINISHED ALL AGES                       12
PUD - MULTILEVEL - INCL SPLIT LEV/FOYER                 10
1-STORY W/FINISHED ATTIC ALL AGES                        4
Name: MSSubClass, dtype: int64
```

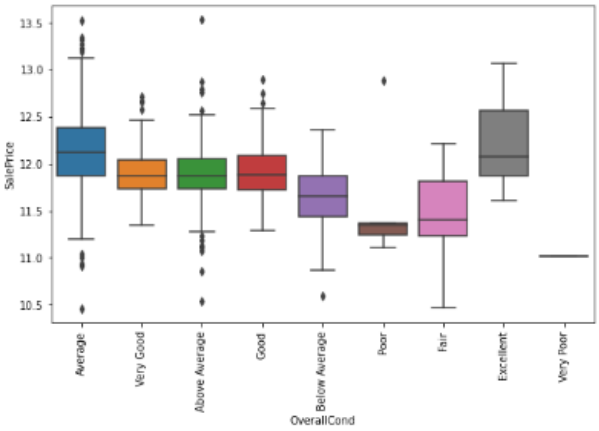
In [26]:
```
# Dropping the highly skewed column
housing.drop(['Functional'],axis=1,inplace=True)
```

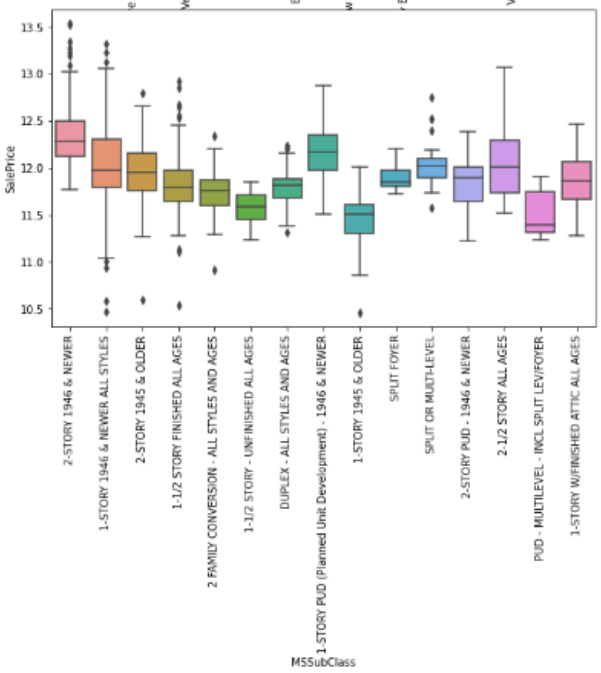
..

In [27]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,2,1)
sns.boxplot(x ='KitchenQual', y ='SalePrice', data = housing)
plt.subplot(2,2,2)
sns.boxplot(x = 'OverallQual', y = 'SalePrice', data = housing)
plt.xticks(rotation=90)

plt.subplot(2,2,3)
sns.boxplot(x = 'OverallCond', y = 'SalePrice', data = housing)
plt.xticks(rotation=90)

plt.subplot(2,2,4)
sns.boxplot(x = 'MSSubClass', y = 'SalePrice', data = housing)
plt.xticks(rotation=90)

plt.show()
```

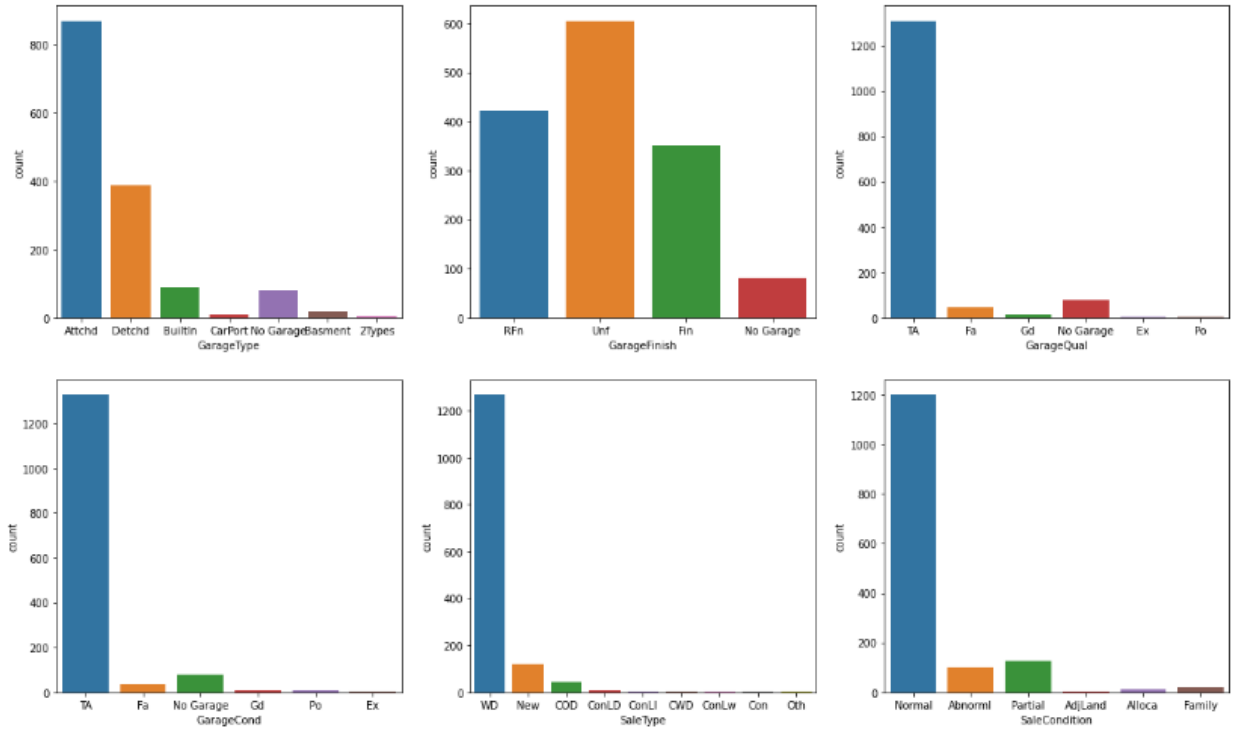

SalePrice
KitchenQual
Gd
TA
Ex
Fa


SalePrice
Good
Very Good
Average
Excellent
Fair
Very Poor
Poor


SalePrice
OverallCond
Average
Very Good
Above Average
Good
Below Average
Poor
Fair
Excellent
Very Poor


SalePrice
MSSubClass
2-STORY 1946 & NEWER
1-STORY 1946 & NEWER ALL STYLES
2-STORY 1945 & OLDER
1-1/2 STORY FINISHED ALL AGES
2 FAMILY CONVERSION - ALL STYLES AND AGES
1-1/2 STORY - UNFINISHED ALL AGES
DUPLEX - ALL STYLES AND AGES
1-STORY PUD (Planned Unit Development) - 1946 & NEWER
1-STORY 1945 & OLDER
SPLIT FOYER
SPLIT OR MULTI-LEVEL
2-STORY PUD - 1946 & NEWER
2-1/2 STORY ALL AGES
PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
1-STORY W/FINISHED ATTIC ALL AGES

In [28]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'GarageType',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'GarageFinish', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'GarageQual', data = housing)
plt.subplot(2,3,4)
sns.countplot(x = 'GarageCond', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'SaleType',  data = housing)
plt.subplot(2,3,6)
sns.countplot(x = 'SaleCondition', data = housing)
plt.show()
```



In [29]:
```
housing['SaleCondition'].value_counts() #Checking other columns for skewness
```

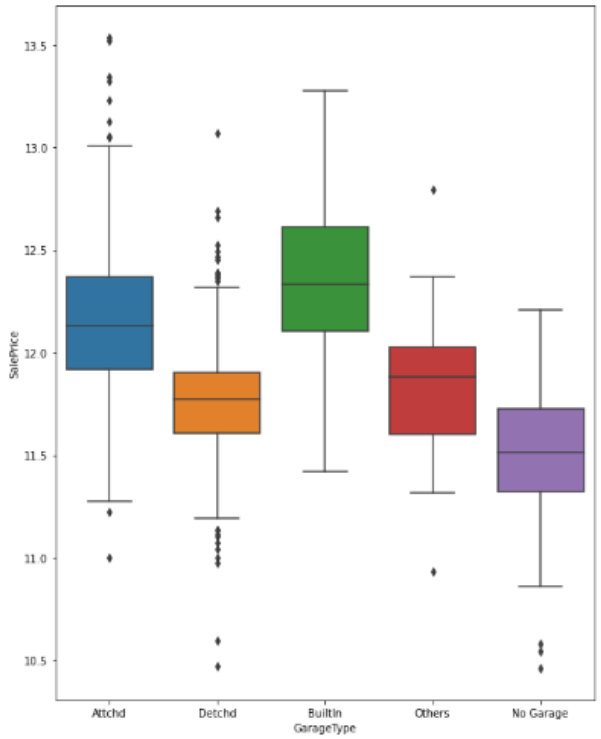
Out[29]:
```
Normal     1198
Partial     125
Abnorml     101
Family       20
Alloca       12
AdjLand       4
Name: SaleCondition, dtype: int64
```

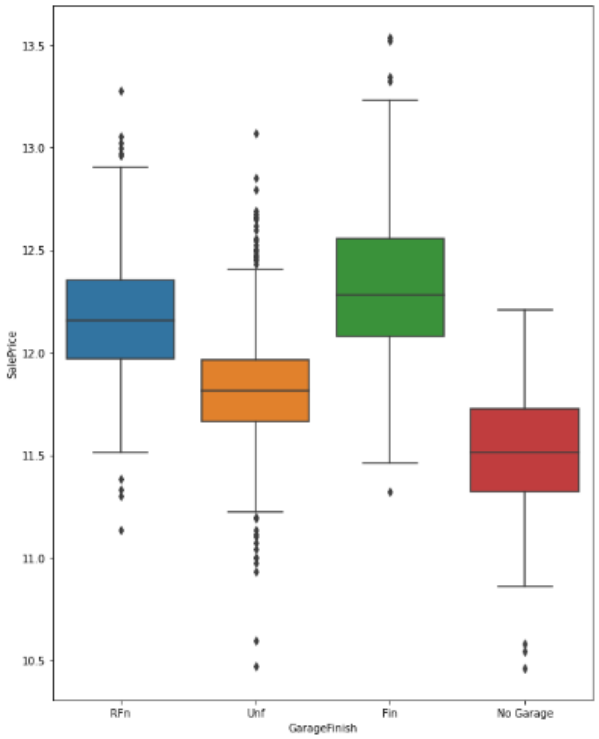
#- We can see that in this group except GarageType and Garage Finsih almost all are skewed so we can drop these columns.

In [30]:
```
housing['GarageType'] = housing['GarageType'].replace(['Basment','CarPort','2Types'],'Others')
housing['SaleCondition'] = housing['SaleCondition'].replace(['Family','Alloca','AdjLand'],'Others')

#Highly skewed column being dropped
housing.drop(['GarageQual','GarageCond','SaleType'],axis=1,inplace=True)
```

In [31]:
```
#Let's see effect of Garage type and GarageFinish on SalePrice

plt.figure(figsize=(20, 12))
plt.subplot(1,2,1)
sns.boxplot(x = 'GarageType', y ='SalePrice', data = housing)
plt.subplot(1,2,2)
sns.boxplot(x = 'GarageFinish', y = 'SalePrice', data = housing)
```

Out[31]:
```
<AxesSubplot:xlabel='GarageFinish', ylabel='SalePrice'>
```



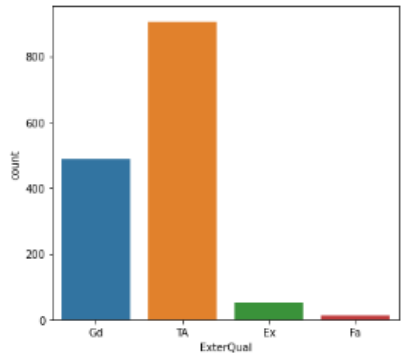


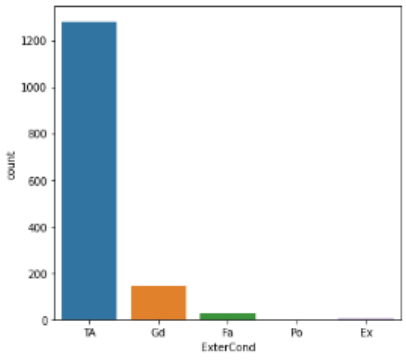
- Price of Builtin Garagetype and Finished garage is the highest

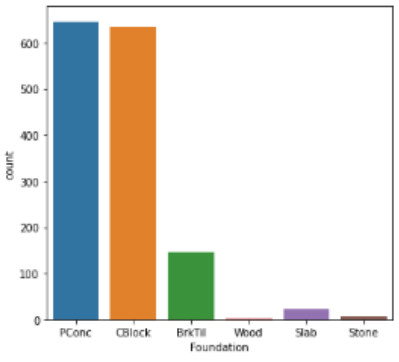
g7= 'Electrical', 'KitchenQual', 'Functional'

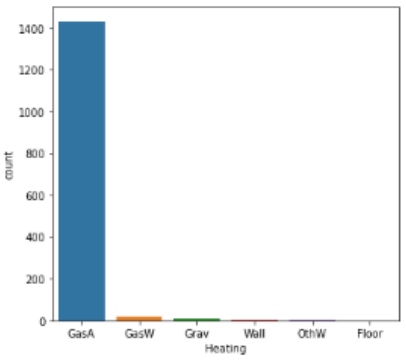
g7= 'Electrical', 'KitchenQual', 'Functional'

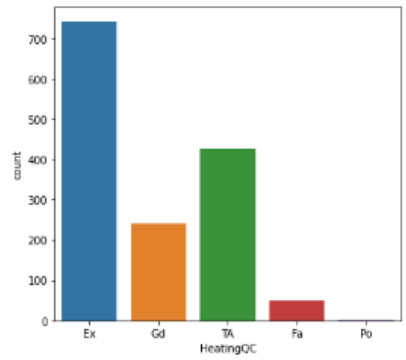
In [32]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'ExterQual',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'ExterCond', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'Foundation', data = housing)
plt.subplot(2,3,4)
sns.countplot(x = 'Heating', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'HeatingQC',  data = housing)
plt.subplot(2,3,6)
sns.countplot(x = 'CentralAir', data = housing)
plt.show()
```

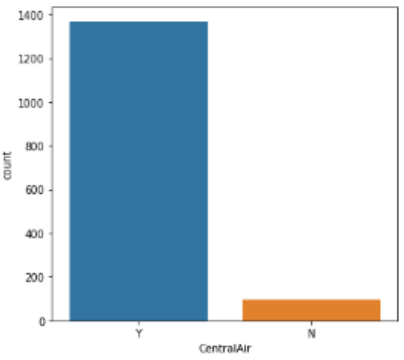












- Majority of ExterQual, ExerCond is TA
- Poured Contrete foundation are the highest in number
- Meanwhile variables like Heating , Central Airand Exter Cond are skewed so would be dropping these variables

In [33]:
```
housing['HeatingQC'].value_counts() #Checked for all variables to check the skewness
```

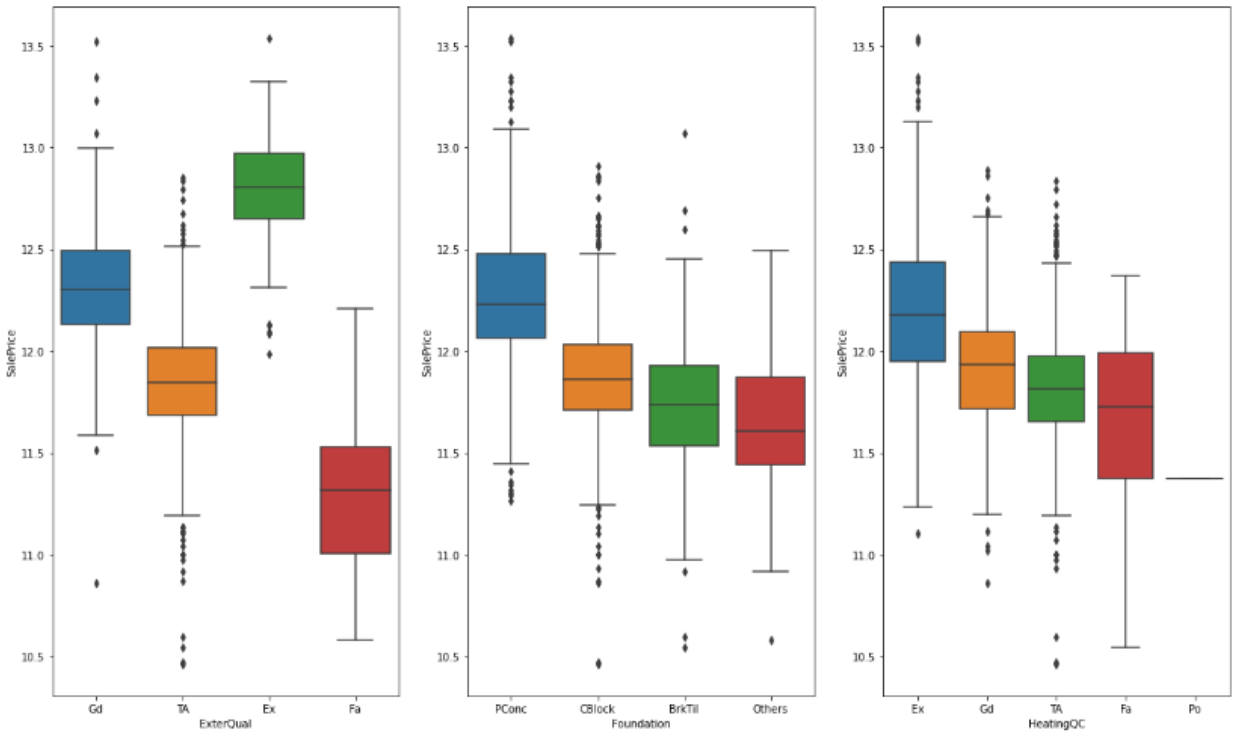
Out[33]:
```
Ex    741
TA    428
Gd    241
Fa     49
Po      1
Name: HeatingQC, dtype: int64
```

In [34]:
```
housing['Foundation'] = housing['Foundation'].replace(['Slab','Stone','Wood'],'Others')

# Dropping highly skewed column
housing.drop(['CentralAir','Heating','ExterCond'],axis=1,inplace=True)
```

In [35]:
```
#Let's see effect of Garage type and GarageFinish on SalePrice

plt.figure(figsize=(20, 12))
plt.subplot(1,3,1)
sns.boxplot(x = 'ExterQual', y ='SalePrice', data = housing)
plt.subplot(1,3,2)
sns.boxplot(x = 'Foundation', y = 'SalePrice', data = housing)
plt.subplot(1,3,3)
sns.boxplot(x = 'HeatingQC', y = 'SalePrice', data = housing)
```

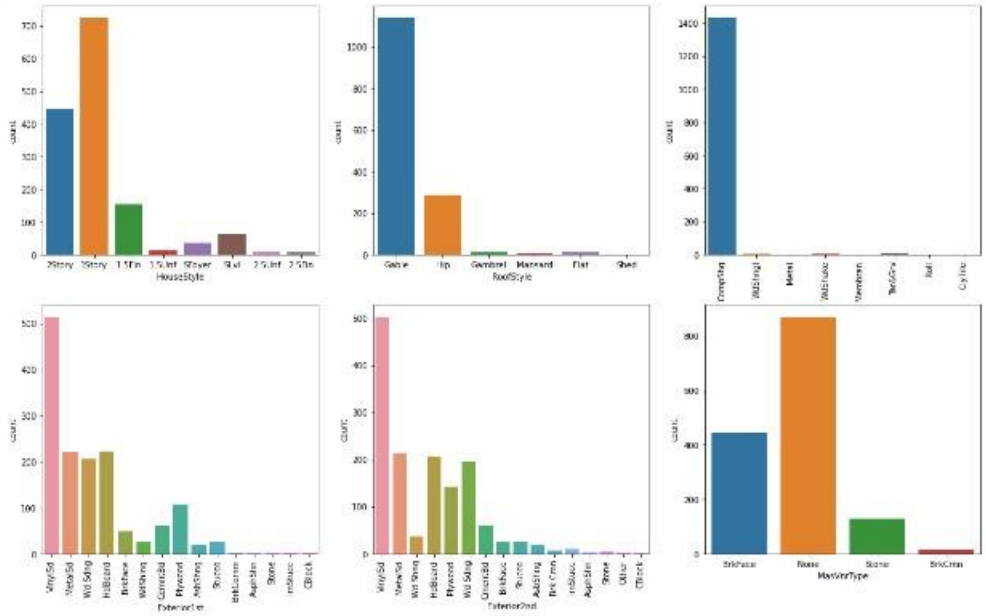
Out[35]: <AxesSubplot:xlabel='HeatingQC', ylabel='SalePrice'>

Out[35]: <AxesSubplot:xlabel='HeatingQC', ylabel='SalePrice'>



- Price of Excellent ExterQual and HeatingQc is highest
- Price of Poured Contrete Foundation is highest.

In [36]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'HouseStyle',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'RoofStyle', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'RoofMatl', data = housing)
plt.xticks(rotation=90)
plt.subplot(2,3,4)
sns.countplot(x = 'Exterior1st', data = housing)
plt.xticks(rotation=90)
plt.subplot(2,3,5)
sns.countplot(x = 'Exterior2nd',  data = housing)
plt.xticks(rotation=90)
plt.subplot(2,3,6)
sns.countplot(x = 'MasVnrType', data = housing)
plt.show()
```

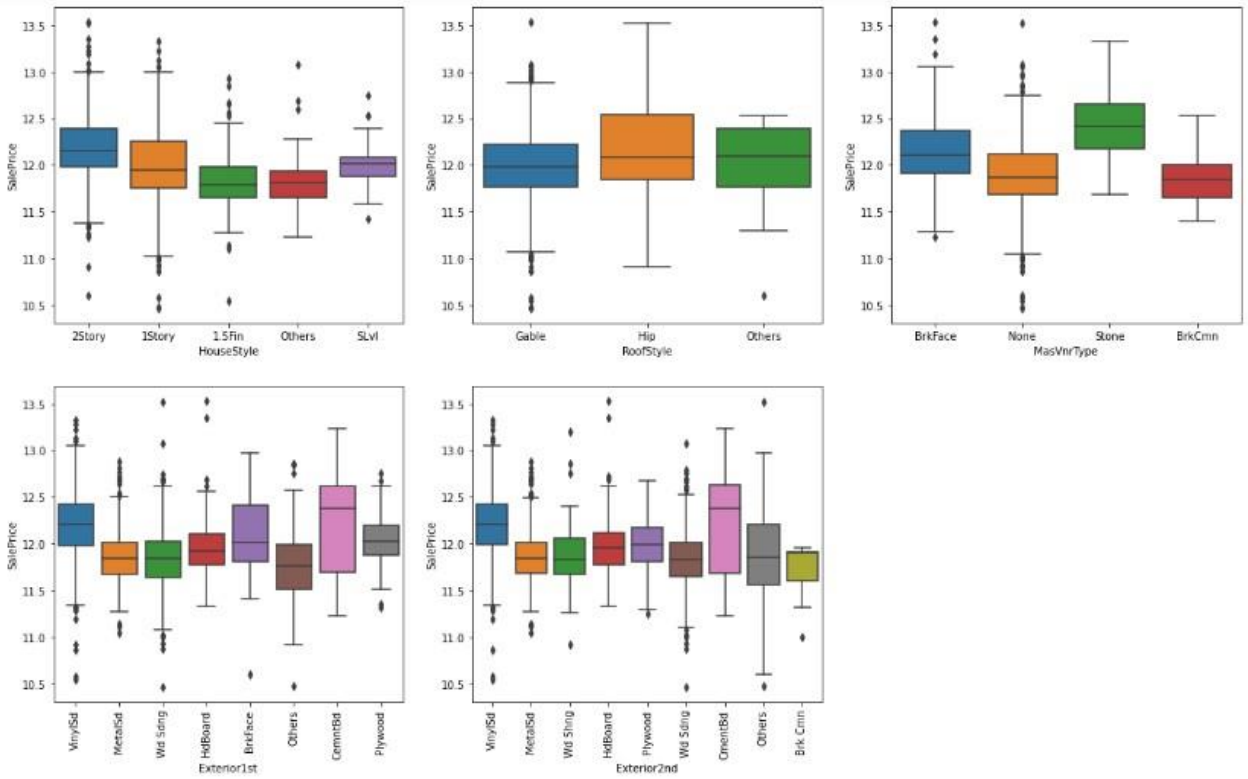


In [37]:
```
housing['Exterior2nd'].value_counts() #Checking the skewness for other columns
```

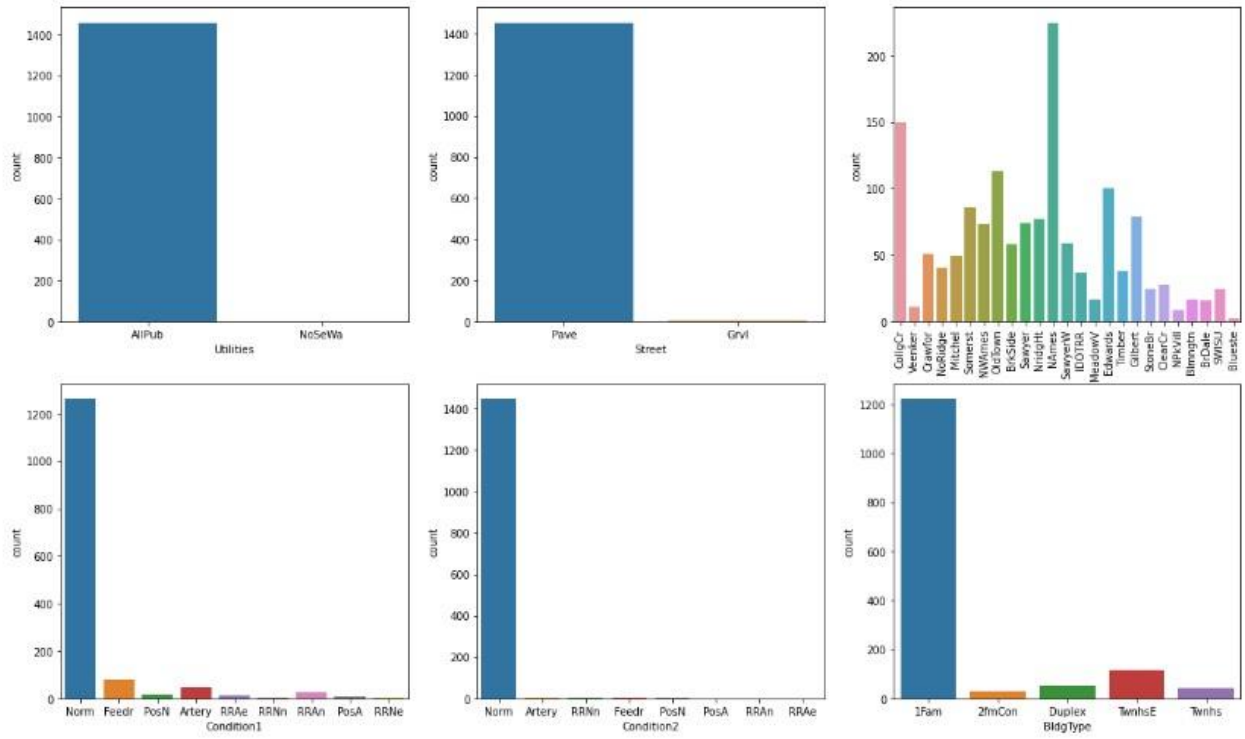
Out[37]:
```
VinylSd    504
MetalSd    214
HdBoard    207
Wd Sdng    197
Plywood    142
CmentBd     60
Wd Shng     38
Stucco      26
BrkFace     25
AsbShng     20
ImStucc     10
Brk Cmn      7
Stone        5
AsphShn      3
CBlock       1
Other        1
Name: Exterior2nd, dtype: int64
```

In [38]:
```
housing['HouseStyle'] = housing['HouseStyle'].replace(['SFoyer','1.5Unf','2.5Unf','2.5Fin'],'Others')
housing['RoofStyle'] = housing['RoofStyle'].replace(['Shed','Mansard','Gambrel','Flat'],'Others')
housing['Exterior1st'] = housing['Exterior1st'].replace(['AsphShn','ImStucc','CBlock','Stone','BrkComm','AsbShng','Stucco','WdSh
housing['Exterior2nd'] = housing['Exterior2nd'].replace(['Other','AsphShn','ImStucc','CBlock','Stone','BrkComm','AsbShng','Stucc

# Dropping highly skewed column
housing.drop(['RoofMatl'],axis=1,inplace=True)
```

In [42]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.boxplot(x = 'HouseStyle', y ='SalePrice', data = housing)
plt.subplot(2,3,2)
sns.boxplot(x = 'RoofStyle', y = 'SalePrice', data = housing)
plt.subplot(2,3,3)
sns.boxplot(x = 'MasVnrType', y = 'SalePrice', data = housing)
plt.subplot(2,3,4)
sns.boxplot(x = 'Exterior1st', y = 'SalePrice', data = housing)
plt.xticks(rotation=90)
plt.subplot(2,3,5)
sns.boxplot(x = 'Exterior2nd', y = 'SalePrice', data = housing)
plt.xticks(rotation=90)
plt.show()
```



In [43]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'Utilities',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'Street', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'Neighborhood', data = housing)
plt.xticks(rotation=90)
plt.subplot(2,3,4)
sns.countplot(x = 'Condition1', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'Condition2',  data = housing)
plt.subplot(2,3,6)
sns.countplot(x = 'BldgType', data = housing)
plt.show()
```



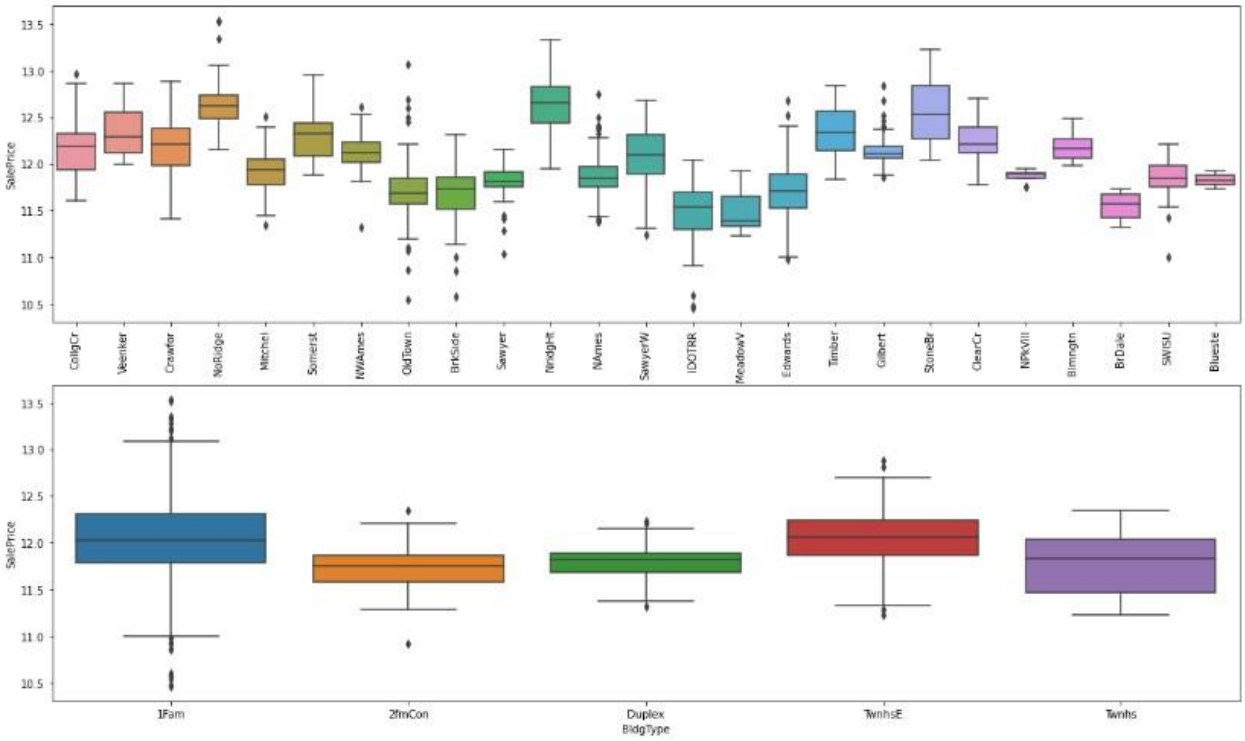
In [44]:
```
housing['BldgType'].value_counts()#similarly checking skewness for other columns
```

Out[44]:
```
1Fam      1220
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: BldgType, dtype: int64
```

In [45]:
```
# Dropping highly skewed column
housing.drop(['Utilities','Street','Condition1','Condition2'],axis=1,inplace=True)
```

In [46]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,1,1)
sns.boxplot(x = 'Neighborhood', y ='SalePrice', data = housing)
plt.xticks(rotation=90)
plt.subplot(2,1,2)
sns.boxplot(x = 'BldgType', y = 'SalePrice', data = housing)
```

Out[46]: <AxesSubplot:xlabel='BldgType', ylabel='SalePrice'>



In [ ]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'LandContour',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'LandSlope', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'LotShape', data = housing)
plt.subplot(2,3,4)
sns.countplot(x = 'Electrical', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'MSZoning',  data = housing)
plt.subplot(2,3,6)
sns.countplot(x = 'LotConfig', data = housing)
plt.show()
```

In [ ]:
```
housing['LotConfig'].value_counts()
```

In [ ]:
```
housing['LotConfig'].value_counts()
```

In [ ]:
```
housing.drop(['LandSlope','LandContour','Electrical'],axis=1,inplace=True)

housing['MSZoning'] = housing['MSZoning'].replace(['RH','C (all)'],'Others')
```

In [ ]:
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'BsmtQual',  data = housing)
plt.subplot(2,3,2)
sns.countplot(x = 'BsmtCond', data = housing)
plt.subplot(2,3,3)
sns.countplot(x = 'BsmtExposure', data = housing)
plt.subplot(2,3,4)
sns.countplot(x = 'BsmtFinType1', data = housing)
plt.subplot(2,3,5)
sns.countplot(x = 'BsmtFinType2',  data = housing)
plt.subplot(2,3,6)
sns.countplot(x = 'PavedDrive', data = housing)
plt.show()
```

In [ ]:
```
housing['BsmtCond'].value_counts() # similarly checking skewness for other columns
```

In [ ]:
```
housing.drop(['BsmtFinType2','PavedDrive','BsmtCond'],axis=1,inplace=True)
```

In [ ]:
```
housing.head()
```

In [ ]:
```
housing.info()
```

In [ ]:
```
#Now the saleprice correlation matrix
corrmat = housing.corr()
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(housing[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=
plt.show()
```

In [53]:
```
housing['YearSinceRemodel'] = 2010 - ((housing['YearRemodAdd'] - housing['YearBuilt']) + housing['YearBuilt']) #feature engi
```

In [54]:
```
Cat1 = housing.select_dtypes(include=['object']) #checking all the categorical columns to form a dummy variables
Cat1.columns
```

Out[54]:
```
Index(['MSSubClass', 'MSZoning', 'LotShape', 'LotConfig', 'Neighborhood',
       'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond', 'RoofStyle',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'Foundation',
       'BsmtQual', 'BsmtExposure', 'BsmtFinType1', 'HeatingQC', 'KitchenQual',
       'GarageType', 'GarageFinish', 'SaleCondition'],
      dtype='object')
```

In [55]:
```
Num = housing.select_dtypes(include=['int64','float64']) #all the numerical variables
Num.columns
```

Out[55]:
```
Index(['Id', 'LotFrontage', 'LotArea', 'YearBuilt', 'YearRemodAdd',
       'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice',
       'YearSinceRemodel'],
      dtype='object')
```

In [56]:
```
Num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 36 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Id               1460 non-null   int64
 1   LotFrontage      1460 non-null   float64
 2   LotArea          1460 non-null   int64
 3   YearBuilt        1460 non-null   int64
 4   YearRemodAdd     1460 non-null   int64
 5   MasVnrArea       1460 non-null   float64
 6   BsmtFinSF1       1460 non-null   int64
 7   BsmtFinSF2       1460 non-null   int64
 8   BsmtUnfSF        1460 non-null   int64
 9   TotalBsmtSF      1460 non-null   int64
 10  1stFlrSF         1460 non-null   int64
 11  2ndFlrSF         1460 non-null   int64
 12  LowQualFinSF     1460 non-null   int64
 13  GrLivArea        1460 non-null   int64
 14  BsmtFullBath     1460 non-null   int64
 15  BsmtHalfBath     1460 non-null   int64
 16  FullBath         1460 non-null   int64
 17  HalfBath         1460 non-null   int64
 18  BedroomAbvGr     1460 non-null   int64
 19  KitchenAbvGr     1460 non-null   int64
 20  TotRmsAbvGrd     1460 non-null   int64
```

```
 18  BedroomAbvGr      1460 non-null   int64
 19  KitchenAbvGr      1460 non-null   int64
 20  TotRmsAbvGrd      1460 non-null   int64
 21  Fireplaces        1460 non-null   int64
 22  GarageYrBlt       1460 non-null   float64
 23  GarageCars        1460 non-null   int64
 24  GarageArea        1460 non-null   int64
 25  WoodDeckSF        1460 non-null   int64
 26  OpenPorchSF       1460 non-null   int64
 27  EnclosedPorch     1460 non-null   int64
 28  3SsnPorch         1460 non-null   int64
 29  ScreenPorch       1460 non-null   int64
 30  PoolArea          1460 non-null   int64
 31  MiscVal           1460 non-null   int64
 32  MoSold            1460 non-null   int64
 33  YrSold            1460 non-null   int64
 34  SalePrice         1460 non-null   float64
 35  YearSinceRemodel  1460 non-null   int64
dtypes: float64(4), int64(32)
memory usage: 410.8 KB
```

In [57]:
```
Cat1 = pd.get_dummies(Cat1,drop_first=True)  # Dummy variables
print(Cat1.shape)
```

```
(1460, 130)
```

In [58]:
```
Cat1.head()
```

Out[58]:

	MSSubClass_1-1/2 STORY FINISHED ALL AGES	MSSubClass_1-STORY 1945 & OLDER	MSSubClass_1-STORY 1946 & NEWER ALL STYLES	MSSubClass_1-STORY PUD (Planned Unit Development) - 1946 & NEWER	MSSubClass_1-STORY W/FINISHED ATTIC ALL AGES	MSSubClass_2 FAMILY CONVERSION - ALL STYLES AND AGES	MSSubClass_2-1/2 STORY ALL AGES	MSSubClass_2-STORY 1945 & OLDER	MSSubClass_2-STORY 1946 & NEWER	MSSubC STORY 1946 & N
0	0	0	0	0	0	0	0	0	1	
1	0	0	1	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	1	
3	0	0	0	0	0	0	0	1	0	
4	0	0	0	0	0	0	0	0	1	

5 rows × 130 columns

In [59]:
```
# concat the dummy variables with themain dataset
housing = pd.concat([housing, Cat1], axis=1)
```

In [60]:
```
housing.head()
```

Out[60]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt	YearRem
0	1	2-STORY 1946 & NEWER	RL	65.0	8450	Reg	Inside	CollgCr	1Fam	2Story	Good	Average	2003	
1	2	1-STORY 1946 & NEWER ALL STYLES	RL	80.0	9600	Reg	FR2	Veenker	1Fam	1Story	Above Average	Very Good	1976	
2	3	2-STORY 1946 & NEWER	RL	68.0	11250	IR1	Inside	CollgCr	1Fam	2Story	Good	Average	2001	
3	4	2-STORY 1945 & OLDER	RL	60.0	9550	IR1	Corner	Crawfor	1Fam	2Story	Good	Average	1915	
4	5	2-STORY 1946 & NEWER	RL	84.0	14260	IR1	FR2	NoRidge	1Fam	2Story	Very Good	Average	2000	

5 rows × 189 columns

In [61]:
```
housing.drop(['MSZoning', 'LotShape', 'LotConfig', 'Neighborhood', 'BldgType',
       'HouseStyle', 'RoofStyle', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'ExterQual', 'Foundation', 'BsmtQual', 'BsmtExposure', 'BsmtFinType1',
       'HeatingQC', 'KitchenQual', 'GarageType', 'GarageFinish',
       'SaleCondition','Id','OverallCond','MSSubClass','OverallQual'],axis=1,inplace=True) #removing columns as dummy variables
```

In [62]:
```
housing.head()
```

Out[62]:

	LotFrontage	LotArea	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLiv
0	65.0	8450	2003	2003	196.0	706	0	150	856	856	854	0	
1	80.0	9600	1976	1976	0.0	978	0	284	1262	1262	0	0	
2	68.0	11250	2001	2002	162.0	486	0	434	920	920	866	0	
3	60.0	9550	1915	1970	0.0	216	0	540	756	961	756	0	
4	84.0	14260	2000	2000	350.0	655	0	490	1145	1145	1053	0	

5 rows × 165 columns

In [63]:
```
housing.drop(['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold'],axis=1,inplace=True) #no need of these while making a model
```

In [64]:
```
housing.head()
```

Out[64]:

	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtH
0	65.0	8450	196.0	706	0	150	856	856	854	0	1710	1	
1	80.0	9600	0.0	978	0	284	1262	1262	0	0	1262	0	
2	68.0	11250	162.0	486	0	434	920	920	866	0	1786	1	
3	60.0	9550	0.0	216	0	540	756	961	756	0	1717	1	
4	84.0	14260	350.0	655	0	490	1145	1145	1053	0	2198	1	

5 rows × 161 columns

In [65]:
```
# Putting feature variable to X
X = housing.drop(['SalePrice'], axis=1)
X.head()
```

Out[65]:

	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtH
0	65.0	8450	196.0	706	0	150	856	856	854	0	1710	1	
1	80.0	9600	0.0	978	0	284	1262	1262	0	0	1262	0	
2	68.0	11250	162.0	486	0	434	920	920	866	0	1786	1	
3	60.0	9550	0.0	216	0	540	756	961	756	0	1717	1	
4	84.0	14260	350.0	655	0	490	1145	1145	1053	0	2198	1	

5 rows × 160 columns

In [66]:
```
# Putting response variable to y
y = housing['SalePrice']
y.head()
```

Out[66]:
```
0    12.247694
1    12.109011
2    12.317167
3    11.849398
4    12.429216
Name: SalePrice, dtype: float64
```

In [67]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.7,
                                                    test_size = 0.3, random_state=100)
```

In [68]:
```
scaler = StandardScaler()

X_train[['LotFrontage', 'LotArea','MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Fireplaces','GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal']] = scaler.fit_transform(X_train[['LotFrontage', 'LotArea','MasVnrArea', 'BsmtFinSF1
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Fireplaces','GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal']])
X_train.head()
```

Out[68]:

	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath
210	-0.115302	-0.473765	-0.558025	0.043512	-0.293313	-0.374367	-0.430911	-0.765065	-0.787597	-0.124502	-1.228053	1.066863
318	0.926898	-0.056845	0.809137	1.160345	-0.293313	-0.454844	0.621579	0.511914	2.140556	-0.124502	2.123103	1.066863
239	-0.794998	-0.169324	-0.558025	-0.761296	-0.293313	0.171964	-0.712011	-0.930972	0.795996	-0.124502	-0.056465	-0.837450
986	-0.477806	-0.502297	-0.558025	-0.963574	-0.293313	-0.175904	-1.256778	-0.420683	0.669584	-0.124502	0.221858	-0.837450
1416	-0.432493	0.082905	-0.558025	-0.963574	-0.293313	0.475233	-0.620490	0.195183	1.611926	-0.124502	1.453624	-0.837450

5 rows × 160 columns

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.7,
                                                    test_size = 0.3, random_state=100)
```

3. Model Building and Evaluation

- Ridge and Lasso Regression

Ridge

In [70]:
```
# list of alphas to tune
params = {'alpha': [0.00004,0.0001, 0.001, 0.01, 0.1, 10,100,1000]}


ridge = Ridge()

# cross validation
folds = 5
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'r2',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

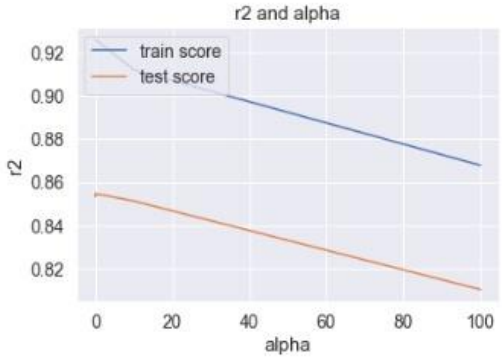
Out[70]:
```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [4e-05, 0.0001, 0.001, 0.01, 0.1, 10, 100,
                                   1000]},
             return_train_score=True, scoring='r2', verbose=1)
```

In [71]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<=100]
cv_results.head()
```

Out[71]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0	0.013030	0.006920	0.003650	0.002264	0.00004	{'alpha': 4e-05}	0.868246	0.777081	0.838201	0.890474
1	0.004950	0.004070	0.004156	0.003734	0.0001	{'alpha': 0.0001}	0.868247	0.777081	0.838201	0.890476
2	0.008817	0.001213	0.002297	0.001167	0.001	{'alpha': 0.001}	0.868258	0.777072	0.838200	0.890516
3	0.010310	0.004400	0.003183	0.000089	0.01	{'alpha': 0.01}	0.868367	0.776981	0.838188	0.890866
4	0.008345	0.000333	0.004250	0.001654	0.1	{'alpha': 0.1}	0.869382	0.776055	0.838034	0.892791

In [72]:
```
# plotting mean test & train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting the graph
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('r2')
plt.title("r2 and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [73]:
```
#optimum alpha
alpha = 10
ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)
ridge.coef_
```

Out[73]:
```
array([-4.74809780e-04,  1.59466646e-06,  6.17112057e-06,  3.73538839e-06,
        1.15081459e-05,  1.82053078e-05,  3.34488345e-05,  7.92009076e-05,
        1.09011749e-04, -7.70408589e-05,  1.11171803e-04,  4.25420135e-02,
        1.51319614e-02,  5.09516505e-02,  3.69272171e-02,  9.07794039e-03,
       -8.12648652e-02,  8.30696546e-03,  3.96097675e-02,  5.34751466e-02,
        3.43279188e-05,  9.90267283e-05, -5.47707192e-05,  1.22380757e-04,
        1.94886076e-04,  2.27453067e-04, -4.04142867e-04, -3.68907754e-06,
        5.21312292e-04, -1.59513446e-03,  1.22746032e-02, -8.43053932e-02,
        4.98263439e-02,  5.96932189e-03,  1.37128949e-04, -1.20780328e-02,
        2.35023365e-02,  2.51877006e-02, -1.48351396e-05, -4.06689958e-02,
        1.48370219e-02, -2.89980792e-02,  2.13765700e-02,  9.23512984e-03,
       -7.75969900e-02,  1.25978119e-02, -2.93034632e-02,  2.18691748e-02,
       -4.94699235e-02,  7.41424037e-03,  3.66405387e-02, -3.24003086e-02,
       -1.39715714e-02, -3.40987853e-03, -2.21542138e-03, -1.30144969e-02,
        2.02737767e-02,  4.27356892e-02, -8.85281265e-04,  9.35120161e-02,
       -9.14209485e-02, -6.78467558e-03, -8.40306219e-02, -6.36592396e-02,
       -1.60183781e-02, -1.82066419e-02, -1.00614266e-02, -9.67857532e-03,
        7.22098712e-02,  9.86574851e-02, -6.61731473e-02, -6.75645642e-03,
       -4.77875309e-02,  2.73991571e-03,  9.19726093e-02,  5.78194024e-02,
       -5.18807921e-03,  3.61514663e-02, -1.20780328e-02,  1.48370219e-02,
       -5.19113158e-02, -1.48893178e-02,  1.40048572e-02, -2.32453361e-02,
        7.33801706e-03,  1.09928742e-02, -3.15493896e-02, -6.00417951e-02,
        1.42911781e-01, -6.63346983e-02,  4.16254214e-02, -8.21094119e-02,
        3.56470931e-02,  1.05491212e-01, -2.83123663e-02, -3.18523601e-02,
       -5.44717727e-02,  4.34477119e-02, -1.42546915e-01,  3.13188597e-02,
       -1.02620212e-02,  2.27712519e-02, -2.83123663e-02,  1.11607023e-02,
        9.25979871e-03, -1.97435860e-02, -1.68695983e-02, -1.89318631e-02,
       -4.79460403e-02, -1.22587718e-02, -3.70670579e-03, -3.20612400e-02,
       -2.35633836e-03,  4.50933034e-03,  2.00561209e-02,  4.14534962e-03,
        7.07504083e-03,  1.45635671e-02,  2.01380947e-03, -1.67513222e-02,
        1.92531202e-02,  1.31141743e-02,  3.31184072e-03, -6.22358692e-02,
        1.38032485e-02, -1.13311044e-02,  1.40480971e-02,  1.33391233e-02,
        3.69727444e-02, -4.98114586e-02, -5.23316950e-02, -5.09581250e-02,
       -5.24397250e-02,  5.55000342e-02, -1.17934223e-02, -2.23271383e-02,
       -5.05201903e-02, -3.54599087e-03,  2.34175152e-02, -2.04470628e-02,
       -5.09581250e-02, -1.49471453e-02, -4.61158104e-02, -1.64604276e-02,
       -7.12140612e-03, -1.69451195e-02, -1.92164117e-02, -6.86534590e-02,
       -4.16327070e-02, -5.17445046e-02, -2.15798656e-02, -2.79515197e-02,
       -2.60682423e-02, -5.51267138e-02, -2.60682423e-02, -1.24984092e-02,
       -3.33359237e-02,  4.32971428e-02,  1.22267613e-02,  5.88483272e-02])
```

In [74]:
```
ridge.score(X_train,y_train)
```

Out[74]:
```
0.9092068605070026
```

In [75]:
```
ridge.score(X_test,y_test)
```

Out[75]:
```
0.8744204967072815
```

In [76]:
```
# Ridge model parameters
model_parameters = list(sorted(ridge.coef_))
model_parameters.insert(0, ridge.intercept_)
model_parameters = [round(x, 3) for x in model_parameters]
cols = X.columns
cols = cols.insert(0, "constant")
list(zip(cols, model_parameters))
```

Out[76]:
```
[('constant', 11.541),
 ('LotFrontage', -0.143),
 ('LotArea', -0.091),
 ('MasVnrArea', -0.084),
 ('BsmtFinSF1', -0.084),
 ('BsmtFinSF2', -0.082),
 ('BsmtUnfSF', -0.081),
 ('TotalBsmtSF', -0.078),
 ('1stFlrSF', -0.069),
 ('2ndFlrSF', -0.066),
 ('LowQualFinSF', -0.066),
 ('GrLivArea', -0.064),
 ('BsmtFullBath', -0.062),
 ('BsmtHalfBath', -0.06),
 ('FullBath', -0.055),
 ('HalfBath', -0.054),
 ('BedroomAbvGr', -0.052),
 ('KitchenAbvGr', -0.052),
 ('TotRmsAbvGrd', -0.052),
```

Lasso

In [77]:
```
params = {'alpha': [0.00006,0.0006, 0.0008, 0.001, 0.002, 0.004, 0.006, 0.008 ]}
lasso = Lasso()

# cross validation system
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'r2',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

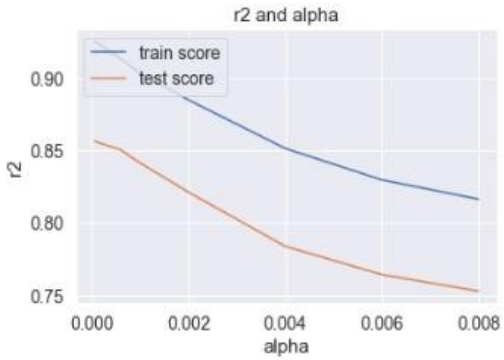
Out[77]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [6e-05, 0.0006, 0.0008, 0.001, 0.002, 0.004,
                                   0.006, 0.008]},
             return_train_score=True, scoring='r2', verbose=1)
```

In [78]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

Out[78]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0	0.141056	0.052591	0.002445	0.001267	0.00006	{'alpha': 6e-05}	0.872325	0.773018	0.839580	0.895919
1	0.029880	0.011531	0.003661	0.001175	0.0006	{'alpha': 0.0006}	0.884401	0.724748	0.835348	0.898012
2	0.026247	0.004623	0.002433	0.001233	0.0008	{'alpha': 0.0008}	0.883902	0.709901	0.832624	0.895453
3	0.021222	0.008519	0.001970	0.001623	0.001	{'alpha': 0.001}	0.883078	0.699448	0.827873	0.892849
4	0.019566	0.008161	0.003429	0.000811	0.002	{'alpha': 0.002}	0.877497	0.652922	0.806379	0.884122

In [79]:
```
# plotting the mean test and training scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('r2')

plt.title("r2 and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [80]:
```
#optimum alpha
alpha =0.001
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)
```

Out[80]:
```
Lasso(alpha=0.001)
```

In [81]:
```
lasso.coef_
```

Out[81]:
```
array([-4.09356126e-04,  1.41043514e-06,  1.80666838e-05,  4.75861642e-05,
        4.91108947e-05,  5.55828023e-05,  3.22848128e-05,  1.72776225e-04,
        1.95176491e-04, -2.93129965e-05,  2.33960625e-05,  4.05631982e-02,
        8.97338843e-04,  4.46795488e-02,  3.11477938e-02,  1.36517476e-03,
       -9.96565920e-02,  9.16320797e-03,  4.00955714e-02,  6.11563721e-02,
        3.68096081e-05,  1.11663349e-04, -4.88469932e-05,  1.16324360e-04,
        1.94148996e-04,  2.48687005e-04, -4.72011474e-04, -3.65282465e-06,
        4.33777426e-04, -2.04610388e-03,  0.00000000e+00, -1.01268932e-01,
        3.91866781e-02,  0.00000000e+00,  0.00000000e+00, -0.00000000e+00,
        0.00000000e+00,  0.00000000e+00, -0.00000000e+00, -4.15042585e-02,
        0.00000000e+00, -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
       -7.80791267e-02,  7.18055960e-03, -3.22277887e-02,  0.00000000e+00,
       -3.77577996e-03,  0.00000000e+00,  3.38538223e-02, -0.00000000e+00,
       -0.00000000e+00, -0.00000000e+00, -0.00000000e+00, -0.00000000e+00,
        0.00000000e+00,  1.71369941e-02,  0.00000000e+00,  9.80444492e-02,
       -8.59137343e-02, -0.00000000e+00, -8.30818267e-02, -5.23364891e-02,
       -0.00000000e+00, -0.00000000e+00, -0.00000000e+00, -0.00000000e+00,
        5.01165152e-02,  8.35409554e-02, -6.64082200e-02, -0.00000000e+00,
       -2.44227454e-02,  0.00000000e+00,  7.89978723e-02,  0.00000000e+00,
       -0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
       -5.28335449e-02, -0.00000000e+00,  0.00000000e+00, -1.65696232e-02,
        0.00000000e+00,  0.00000000e+00, -2.16748081e-02, -5.48248917e-02,
        1.97772557e-01, -3.46471618e-02,  4.58670213e-02, -0.00000000e+00,
        0.00000000e+00,  1.19965493e-01, -0.00000000e+00, -3.67128819e-02,
       -5.74182905e-02,  0.00000000e+00, -2.10164729e-01,  1.81661568e-02,
       -0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  4.52511947e-04,
        0.00000000e+00, -0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
       -3.00864512e-02,  0.00000000e+00,  0.00000000e+00, -1.55475193e-02,
       -0.00000000e+00,  0.00000000e+00,  1.91566681e-03,  0.00000000e+00,
        0.00000000e+00,  1.48848904e-02, -0.00000000e+00, -0.00000000e+00,
        2.00200834e-03,  0.00000000e+00, -0.00000000e+00, -2.85976366e-02,
        1.39387050e-03, -1.77039340e-02,  0.00000000e+00, -0.00000000e+00,
        2.94541370e-02, -0.00000000e+00, -1.68470979e-02, -0.00000000e+00,
       -9.02099626e-03,  5.82368003e-02, -0.00000000e+00, -1.19803050e-02,
       -2.45862057e-02,  0.00000000e+00,  3.15878285e-02, -0.00000000e+00,
       -0.00000000e+00, -7.25667018e-05, -3.40816817e-02, -0.00000000e+00,
       -0.00000000e+00, -0.00000000e+00, -1.23665915e-02, -3.22899864e-02,
       -7.41505487e-03, -1.68877399e-02, -0.00000000e+00, -1.82728113e-02,
       -9.53528110e-03, -3.85783311e-02, -3.24128882e-04, -0.00000000e+00,
       -1.72777053e-02,  2.96218009e-02,  0.00000000e+00,  4.22153542e-02])
```

In [82]:
```
lasso.score(X_train,y_train)
```

Out[82]:
```
0.898288939025357
```

In [83]:
```
lasso.score(X_test,y_test)
```

Out[83]:
```
0.864657533144189
```

In [84]:
```
# lasso model parameters
model_parameters = list(sorted(lasso.coef_))
model_parameters.insert(0, lasso.intercept_)
model_parameters = [round(x, 3) for x in model_parameters]
cols = X.columns
cols = cols.insert(0, "constant")
list((zip(cols, model_parameters)))
```

Out[84]:
```
[('constant', 11.482),
 ('LotFrontage', -0.21),
 ('LotArea', -0.101),
 ('MasVnrArea', -0.1),
 ('BsmtFinSF1', -0.086),
 ('BsmtFinSF2', -0.083),
 ('BsmtUnfSF', -0.078),
 ('TotalBsmtSF', -0.066),
 ('1stFlrSF', -0.057),
 ('2ndFlrSF', -0.055),
 ('LowQualFinSF', -0.053),
 ('GrLivArea', -0.052),
 ('BsmtFullBath', -0.042),
 ('BsmtHalfBath', -0.039),
 ('FullBath', -0.037),
 ('HalfBath', -0.035),
 ('BedroomAbvGr', -0.034),
 ('KitchenAbvGr', -0.032),
 ('TotRmsAbvGrd', -0.032),
 ('Fireplaces', -0.03),
 ('GarageCars', -0.029),
 ('GarageArea', -0.025),
 ('WoodDeckSF', -0.024),
 ('OpenPorchSF', -0.022),
 ('EnclosedPorch', -0.018),
 ('3SsnPorch', -0.018),
 ('ScreenPorch', -0.017),
 ('PoolArea', -0.017),
 ('MiscVal', -0.017),
 ('MoSold', -0.017),
 ('YearSinceRemodel', -0.016),
 ('MSSubClass_1-1/2 STORY FINISHED ALL AGES', -0.012),
 ('MSSubClass_1-STORY 1945 & OLDER', -0.012),
 ('MSSubClass_1-STORY 1946 & NEWER ALL STYLES', -0.01),
 ('MSSubClass_1-STORY PUD (Planned Unit Development) - 1946 & NEWER', -0.009),
 ('MSSubClass_1-STORY W/FINISHED ATTIC ALL AGES', -0.007),
 ('MSSubClass_2 FAMILY CONVERSION - ALL STYLES AND AGES', -0.004),
 ('MSSubClass_2-1/2 STORY ALL AGES', -0.002),
 ('MSSubClass_2-STORY 1945 & OLDER', -0.0),
 ('MSSubClass_2-STORY 1946 & NEWER', -0.0),
 ('MSSubClass_2-STORY PUD - 1946 & NEWER', -0.0),
 ('MSSubClass_DUPLEX - ALL STYLES AND AGES', -0.0),
 ('MSSubClass_PUD - MULTILEVEL - INCL SPLIT LEV/FOYER', -0.0),
 ('MSSubClass_SPLIT FOYER', -0.0),
 ('MSSubClass_SPLIT OR MULTI-LEVEL', -0.0),
 ('MSZoning_Others', 0.0),
```

```
('MSSubClass_PUD - MULTILEVEL - INCL SPLIT LEV/FOYER', -0.0),
('MSSubClass_SPLIT FOYER', -0.0),
('MSSubClass_SPLIT OR MULTI-LEVEL', -0.0),
('MSZoning_Others', 0.0),
('MSZoning_RL', 0.0),
('MSZoning_RM', 0.0),
('LotShape_IR2', -0.0),
('LotShape_IR3', 0.0),
('LotShape_Reg', 0.0),
('LotConfig_CulDSac', -0.0),
('LotConfig_FR2', 0.0),
('LotConfig_FR3', -0.0),
('LotConfig_Inside', 0.0),
('Neighborhood_Blueste', 0.0),
('Neighborhood_BrDale', 0.0),
('Neighborhood_BrkSide', 0.0),
('Neighborhood_ClearCr', -0.0),
('Neighborhood_CollgCr', -0.0),
('Neighborhood_Crawfor', -0.0),
('Neighborhood_Edwards', -0.0),
('Neighborhood_Gilbert', -0.0),
('Neighborhood_IDOTRR', 0.0),
('Neighborhood_MeadowV', 0.0),
('Neighborhood_Mitchel', -0.0),
('Neighborhood_NAmes', -0.0),
('Neighborhood_NPkVill', -0.0),
('Neighborhood_NWAmes', -0.0),
('Neighborhood_NoRidge', -0.0),
('Neighborhood_NridgHt', -0.0),
('Neighborhood_OldTown', 0.0),
('Neighborhood_SWISU', 0.0),
('Neighborhood_Sawyer', -0.0),
('Neighborhood_SawyerW', 0.0),
('Neighborhood_Somerst', -0.0),
('Neighborhood_StoneBr', 0.0),
('Neighborhood_Timber', -0.0),
('Neighborhood_Veenker', 0.0),
('BldgType_2fmCon', 0.0),
('BldgType_Duplex', 0.0),
('BldgType_Twnhs', -0.0),
('BldgType_TwnhsE', 0.0),
('HouseStyle_1Story', -0.0),
('HouseStyle_2Story', 0.0),
('HouseStyle_Others', -0.0),
('HouseStyle_SLvl', 0.0),
('OverallQual_Average', -0.0),
('OverallQual_Below Average', 0.0),
('OverallQual_Excellent', -0.0),
('OverallQual_Fair', -0.0),
('OverallQual_Good', 0.0),
('OverallQual_Poor', 0.0),
('OverallQual_Very Excellent', 0.0),
('OverallQual_Very Good', -0.0),
('OverallQual_Very Poor', 0.0),
('OverallCond_Average', 0.0),
('OverallCond_Below Average', 0.0),
('OverallCond_Excellent', -0.0),
('OverallCond_Fair', -0.0),
```

```
('OverallQual_Poor', 0.0),
('OverallQual_Very Excellent', 0.0),
('OverallQual_Very Good', -0.0),
('OverallQual_Very Poor', 0.0),
('OverallCond_Average', 0.0),
('OverallCond_Below Average', 0.0),
('OverallCond_Excellent', -0.0),
('OverallCond_Fair', -0.0),
('OverallCond_Good', 0.0),
('OverallCond_Poor', -0.0),
('OverallCond_Very Good', 0.0),
('OverallCond_Very Poor', -0.0),
('RoofStyle_Hip', -0.0),
('RoofStyle_Others', -0.0),
('Exterior1st_CemntBd', -0.0),
('Exterior1st_HdBoard', 0.0),
('Exterior1st_MetalSd', -0.0),
('Exterior1st_Others', -0.0),
('Exterior1st_Plywood', -0.0),
('Exterior1st_VinylSd', -0.0),
('Exterior1st_Wd Sdng', -0.0),
('Exterior2nd_CmentBd', -0.0),
('Exterior2nd_HdBoard', -0.0),
('Exterior2nd_MetalSd', 0.0),
('Exterior2nd_Others', 0.0),
('Exterior2nd_Plywood', 0.0),
('Exterior2nd_VinylSd', 0.0),
('Exterior2nd_Wd Sdng', 0.0),
('Exterior2nd_Wd Shng', 0.0),
('MasVnrType_BrkFace', 0.0),
('MasVnrType_None', 0.0),
('MasVnrType_Stone', 0.0),
('ExterQual_Fa', 0.0),
('ExterQual_Gd', 0.0),
('ExterQual_TA', 0.0),
('Foundation_CBlock', 0.0),
('Foundation_Others', 0.0),
('Foundation_PConc', 0.0),
('BsmtQual_Fa', 0.0),
('BsmtQual_Gd', 0.0),
('BsmtQual_No Basement', 0.001),
('BsmtQual_TA', 0.001),
('BsmtExposure_Gd', 0.001),
('BsmtExposure_Mn', 0.002),
('BsmtExposure_No', 0.002),
('BsmtExposure_No Basement', 0.007),
('BsmtFinType1_BLQ', 0.009),
('BsmtFinType1_GLQ', 0.015),
('BsmtFinType1_LwQ', 0.017),
('BsmtFinType1_No Basement', 0.018),
('BsmtFinType1_Rec', 0.029),
```


```
('Exterior2nd_VinylSd', 0.0),
('Exterior2nd_Wd Sdng', 0.0),
('Exterior2nd_Wd Shng', 0.0),
('MasVnrType_BrkFace', 0.0),
('MasVnrType_None', 0.0),
('MasVnrType_Stone', 0.0),
('ExterQual_Fa', 0.0),
('ExterQual_Gd', 0.0),
('ExterQual_TA', 0.0),
('Foundation_CBlock', 0.0),
('Foundation_Others', 0.0),
('Foundation_PConc', 0.0),
('BsmtQual_Fa', 0.0),
('BsmtQual_Gd', 0.0),
('BsmtQual_No Basement', 0.001),
('BsmtQual_TA', 0.001),
('BsmtExposure_Gd', 0.001),
('BsmtExposure_Mn', 0.002),
('BsmtExposure_No', 0.002),
('BsmtExposure_No Basement', 0.007),
('BsmtFinType1_BLQ', 0.009),
('BsmtFinType1_GLQ', 0.015),
('BsmtFinType1_LwQ', 0.017),
('BsmtFinType1_No Basement', 0.018),
('BsmtFinType1_Rec', 0.029),
('BsmtFinType1_Unf', 0.03),
('HeatingQC_Fa', 0.031),
('HeatingQC_Gd', 0.032),
('HeatingQC_Po', 0.034),
('HeatingQC_TA', 0.039),
('KitchenQual_Fa', 0.04),
('KitchenQual_Gd', 0.041),
('KitchenQual_TA', 0.042),
('GarageType_BuiltIn', 0.045),
('GarageType_Detchd', 0.046),
('GarageType_No Garage', 0.05),
('GarageType_Others', 0.058),
('GarageFinish_No Garage', 0.061),
('GarageFinish_RFn', 0.079),
('GarageFinish_Unf', 0.084),
('SaleCondition_Normal', 0.098),
('SaleCondition_Others', 0.12),
('SaleCondition_Partial', 0.198)]
```

CONCLUSION

- We got a decent score for both Ridge and Lasso regression.
- Ridge : Train :90.9 Test :87.4
- Lasso : Train :89.8 Test :86.4

Top 5 most significant variables in Ridge are:

- ('SaleCondition_Partial', 0.143)
- ('SaleCondition_Others', 0.105)
- ('SaleCondition_Normal', 0.099)
- ('GarageFinish_Unf', 0.094)
- ('GarageFinish_RFn', 0.092)

Top 5 most significant variables in Lasso are:

- ('SaleCondition_Partial', 0.198)
- ('SaleCondition_Others', 0.12)
- ('SaleCondition_Normal', 0.098)
- ('GarageFinish_Unf', 0.084)
- ('GarageFinish_RFn', 0.079)

These Varaiables are directly proportional to each other.

- Optimal Value of lamda for ridge : 10
- Optimal Value of lamda for Lasso : 0.001

Because of Feature selection we can choose Lasso regression in this case.