Project on House Price Prediction

By Sandeep kumar Pradhan

Date: 07/06/2024

Abstract

An accurate prediction of house prices is a fundamental requirement for various sectors, including real estate and mortgage lending. It is widely recognized that a property's value is not solely determined by its physical attributes but is significantly influenced by its surrounding neighborhood. Meeting the diverse housing needs of individuals while balancing budget constraints is a primary concern for real estate developers. To this end, we addressed the house price prediction problem as a regression task and thus employed various machine learning (ML) techniques capable of expressing the significance of independent variables. We made use of the housing dataset to compare Linear Regression, Forward Feature selection, Ridge Model, Lasso Model for house price prediction. Afterwards, we identified the key factors that influence housing costs. My results show that Lasso Model is the best performing model for house price prediction. Our findings present valuable insights and tools for stakeholders, facilitating more accurate property price estimates and, in turn, enabling more informed decision making to meet the housing needs of diverse populations while considering budget constraints.

1. Introduction

This report presents the analysis and prediction of house prices using a machine learning model. The primary objective is to develop a predictive model that accurately estimates the price of a house based on various features such as the number of bedrooms, bathrooms, square footage, location, and other relevant factors. The dataset used for this analysis is assumed to be comprehensive and representative of the housing market.

2. Challenging process :

1. High Demand and Low Supply

- **Market Saturation**: In many urban areas, the demand for rental properties often exceeds the supply, leading to increased competition among renters.
- **Bidding Wars**: Potential tenants may have to participate in bidding wars, driving up rental prices and making it difficult to secure an affordable place.

2. Affordability

- **High Rent Prices**: The cost of rent in desirable locations can be prohibitively high, making it difficult for many people to find affordable housing.
- **Additional Costs**: Besides rent, there are often additional costs such as security deposits, utility bills, and maintenance fees, which can further strain finances.

3. Location Constraints

- **Proximity to Work/School**: Finding a rental property close to work or educational institutions can be challenging, leading to longer commutes.
- **Safety and Amenities**: Ensuring that the location is safe and has access to necessary amenities (grocery stores, public transportation, parks) can limit options.

4. Quality of Housing

- **Condition of the Property**: Many rental properties may be in poor condition, with issues like outdated appliances, poor maintenance, or structural problems.
- **Misleading Listings**: Online listings can sometimes be misleading, with photos and descriptions not accurately representing the actual condition of the property.

5. Lease Terms and Restrictions

- **Lease Flexibility**: Some landlords require long-term leases, which might not be suitable for individuals looking for short-term accommodation.
- **Restrictions**: Many rental properties have restrictions on pets, modifications, and other personal preferences, limiting options for some renters.

6. Landlord and Tenant Issues

- **Unresponsive Landlords**: Dealing with unresponsive or uncooperative landlords can be frustrating, especially when maintenance issues arise.
- **Background Checks**: Stringent background checks and credit score requirements can disqualify potential renters, especially those with past financial difficulties or no rental history.

7. Navigating the Rental Market

- **Lack of Information**: Inadequate information about the rental market, including current rates and availability, can make it hard to make informed decisions.
- **Scams and Fraud**: There are risks of encountering rental scams, where fake landlords deceive potential tenants out of money without providing a legitimate rental property.

8. Legal and Regulatory Issues

- **Complex Rental Laws**: Understanding local rental laws and tenant rights can be complicated, leading to potential legal issues or disputes with landlords.

- **Zoning Regulations**: Some areas have zoning laws that restrict the types of properties that can be rented, further limiting availability.

9. Accessibility

- **Disability-Friendly Housing**: Finding rental properties that are accessible for individuals with disabilities can be particularly challenging due to limited availability and inadequate facilities.

10. Cultural and Language Barriers

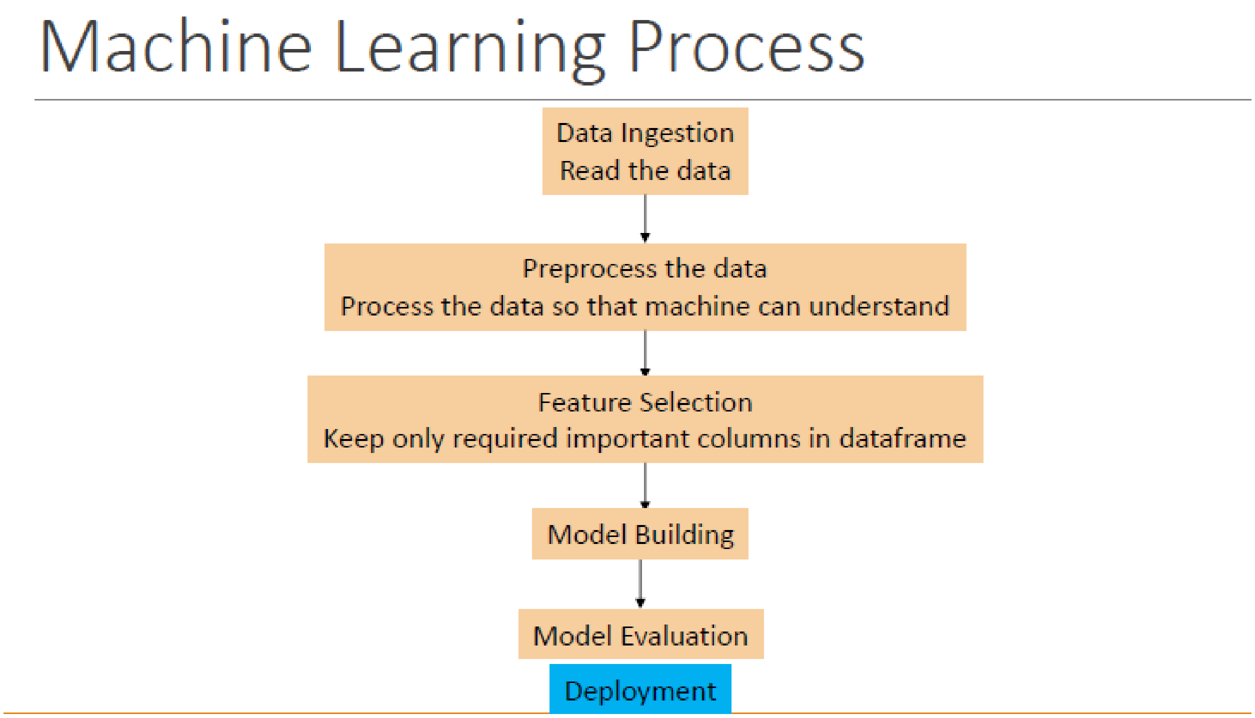
- **Non-Local Renters**: For people moving from other countries or regions, language barriers and cultural differences can make the search for rental housing more difficult.

3. Dataset Overview

The dataset used for this house price prediction model includes the following features:

1. Number of Bedrooms: Integer value representing the number of bedrooms in the house.
2. Number of Bathrooms: Integer or float value representing the number of bathrooms in the house.
3. Square Footage: Integer value representing the total square footage of the house.
4. Location: Categorical variable representing the geographical location of the house.
5. Year Built: Integer value representing the year the house was constructed.
6. Lot Size: Integer value representing the size of the lot in square feet.
7. Number of Floors: Integer value representing the number of floors in the house.
8. Condition: Categorical variable representing the overall condition of the house (e.g., Excellent, Good, Fair, Poor).
9. Nearby Amenities: Categorical variable representing the proximity to amenities (e.g., schools, parks, public transportation).
10. Previous Sale Price: Float value representing the previous sale price of the house.

4.Steps are following to building an Machine Learning Model



5. Methodology

This section presents the methods employed for the house price prediction. Here I compared several regression models, including linear regression (LR), Ridge Model, Lasso Model to ascertain the interpretable best performing model. I have used the housing data from the Kaggle repository. The dataset consists of 1461 records (houses) with 81 variables. Furthermore, we discussed the regression techniques used to forecast house prices in the subsections below. This encompassed the selection of suitable ML algorithms evidenced in the literature.

Linear Regression:

A simple and popular approach to house price prediction is linear regression (LR). LR is a statistical tool that establishes a relationship between a dependent variable (Y) and one or more independent variables (X). This relationship is represented by an equation in the form of

Where β_0 is the intercept, β_i are the slopes, X are the independent variables, and ε is the error term.

Simple linear Regression Objective

- fit a line $yactual = \beta_0 + \beta_1 \cdot x + \varepsilon$
- $ypred = \beta_0 + \beta_1 \cdot x$
- Minimise the Squared error for given relationships
- Least Squares error method
- Formula for slope : $\boldsymbol{\beta_1} = \frac{\boldsymbol{cov(x,y)}}{\boldsymbol{var(x)}} = \frac{\boldsymbol{\Sigma(x-\bar{x})(y-\bar{y})/n}}{\boldsymbol{\Sigma(x-\bar{x})^2/n}}$
- Formula for Intercept : $\boldsymbol{\beta_0} = \boldsymbol{\bar{y}} - \boldsymbol{\beta_1} \cdot \boldsymbol{\bar{x}}$
- $\boldsymbol{\bar{x}}$: ***Mean of all x values***
- $\boldsymbol{\bar{y}}$: **Mean of all y values**

Ridge Regression:

Is also known as L2 regularization. It is one of several types of regularization for linear regression models. Regularization is a statistical method to reduce errors caused by over fitting on training data. Ridge regression specifically corrects for multicollinearity in regression analysis.

Ridge (L2 Regularisation)

Penalty applied on Square of coefficients

$$RSS_{ridge}(w,b) = \underbrace{\sum_{i=1}^{n}(y_i - (w_i x_i + b))^2}_{\text{Fit training data well}} + \alpha \overbrace{\underbrace{\sum_{j=1}^{p} w_j^2}_{\text{Keep parameters small}}}^{\text{L2 penalty / Penalty Term / Regularisation Term}}$$

Independent

A trade-off between fitting the training data well and keeping parameters small

Dependent

Purchase(Ycap) = B0 + B1*Age + B2*Income

Loss Function = (Y-Ycap)^2 + alpha*(B0^2 + B1^2 + B2^2)

Lasso Regression:

In statistics and machine learning, lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model.

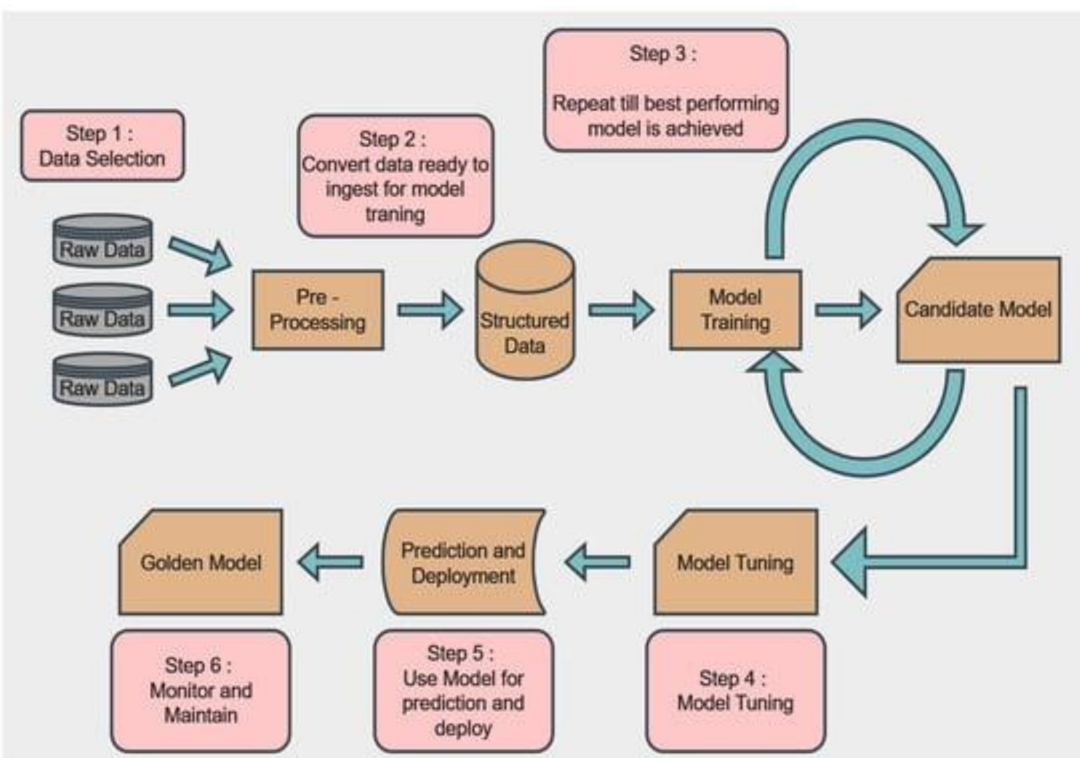
Lasso (L1 Regularisation)

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

Ridge (L2)
Penalty on Square of coefficients

Lasso (L1)
Penalty on Absolute value of Coefficients

Implementation of Machine Learning Algorithm



6. Steps required (E2E Processing):

Step 1 : Read Training CSV file from Local System or Directly from Link.

Step 2 : Check Data Quality and missing values

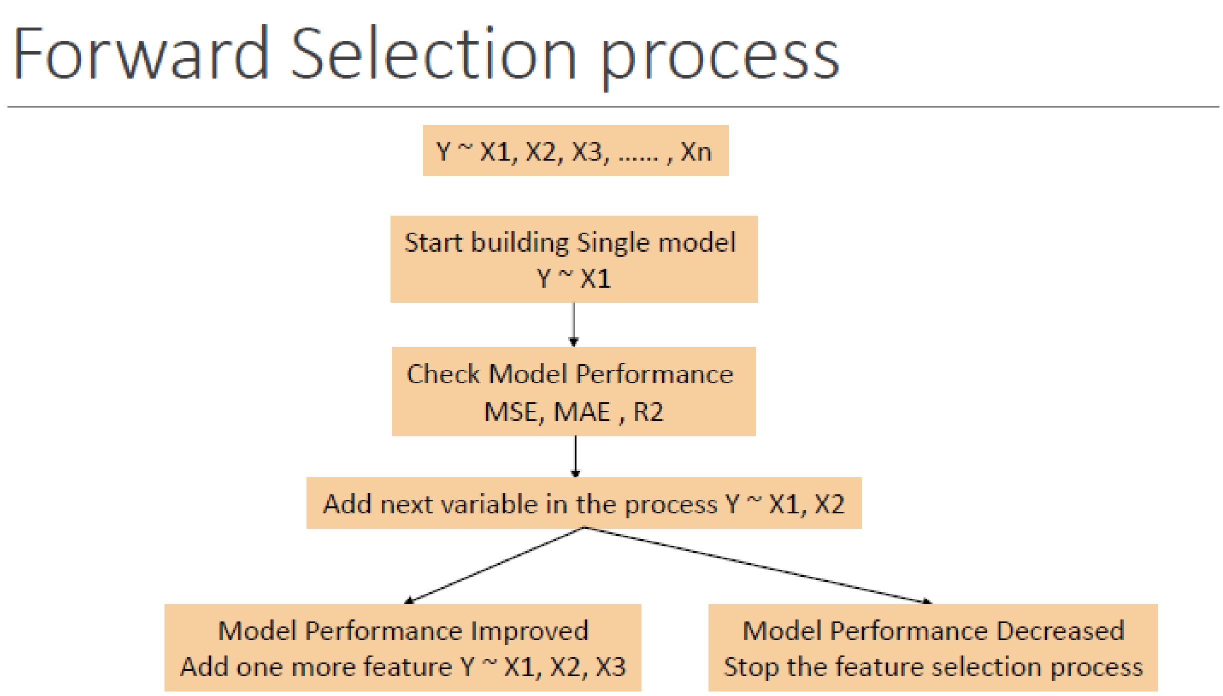
Step 3 : Separate X and Y (SalePrice) features

Step 4 : Create a preprocessing pipeline for X

Before training the model, the dataset underwent several preprocessing steps:

1. **Handling Missing Values**: Missing values were imputed using appropriate strategies, such as mean imputation for numerical features and mode imputation for categorical features.
2. **Encoding Categorical Variables**: Categorical variables were encoded using techniques like one-hot encoding to convert them into numerical formats suitable for machine learning algorithms.
3. **Feature Scaling**: Numerical features were scaled using standardization or normalization to ensure all features contribute equally to the model.
4. **Train-Test Split**: The dataset was split into training and testing sets to evaluate the model's performance on unseen data.

Step 5 : Forward feature selection



Step 6 : Create a final pipeline

Step 7: Train Test split

Step 8 : Building Model and Evaluate Model

Each model was trained using the training dataset and evaluated on the testing dataset. The evaluation metrics included:

- **Mean Absolute Error (MAE)**: The average absolute difference between predicted and actual house prices.
- **Mean Squared Error (MSE)**: The average squared difference between predicted and actual house prices.
- **Root Mean Squared Error (RMSE)**: The square root of the MSE, providing a measure of the average error magnitude.
- **R-squared (R^2)**: The proportion of variance in the dependent variable that is predictable from the independent variables.

Step 9 : Ridge Model : Tuning the ridge model with GridSearchCV

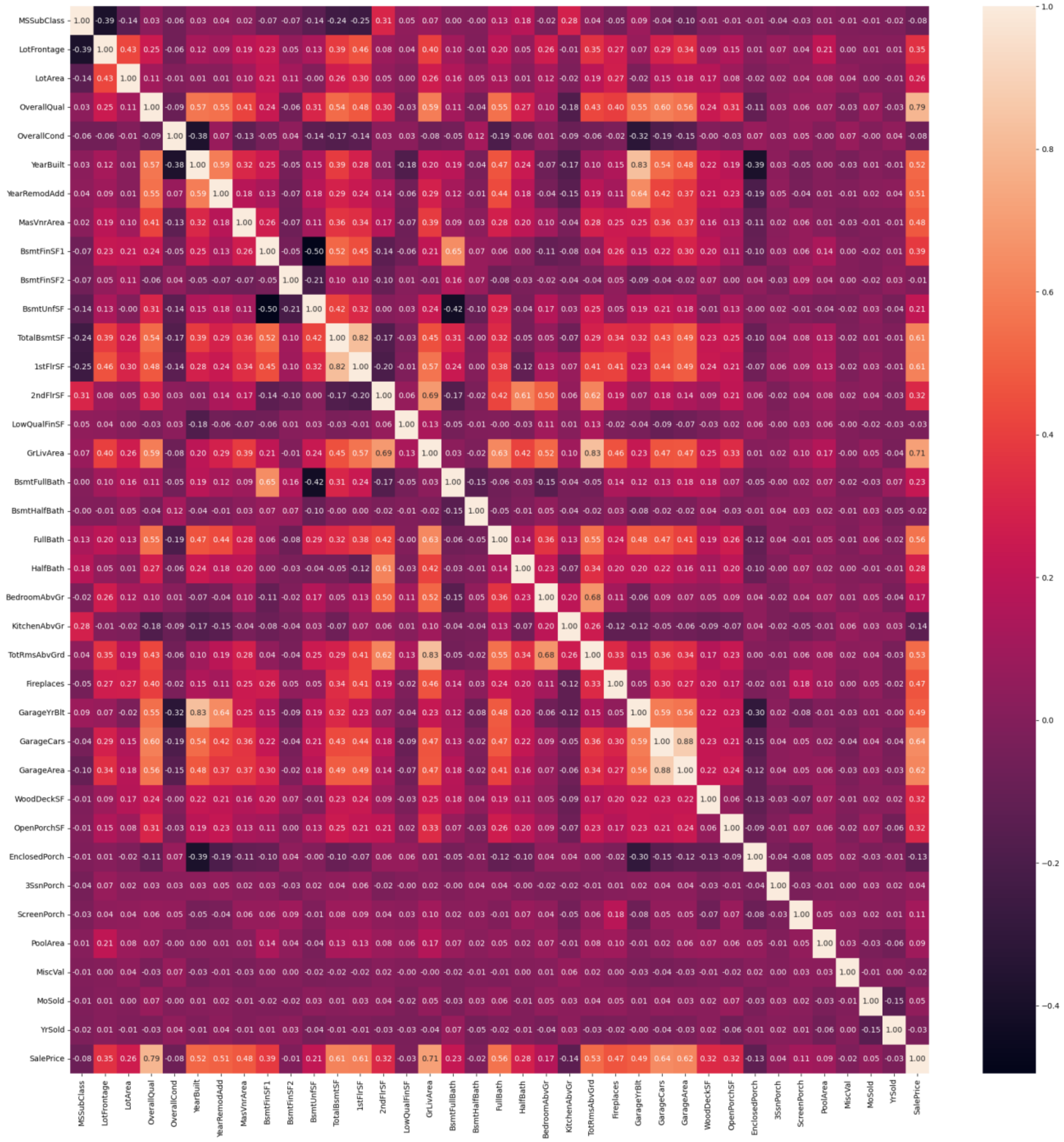
Step 10 : Lasso Model

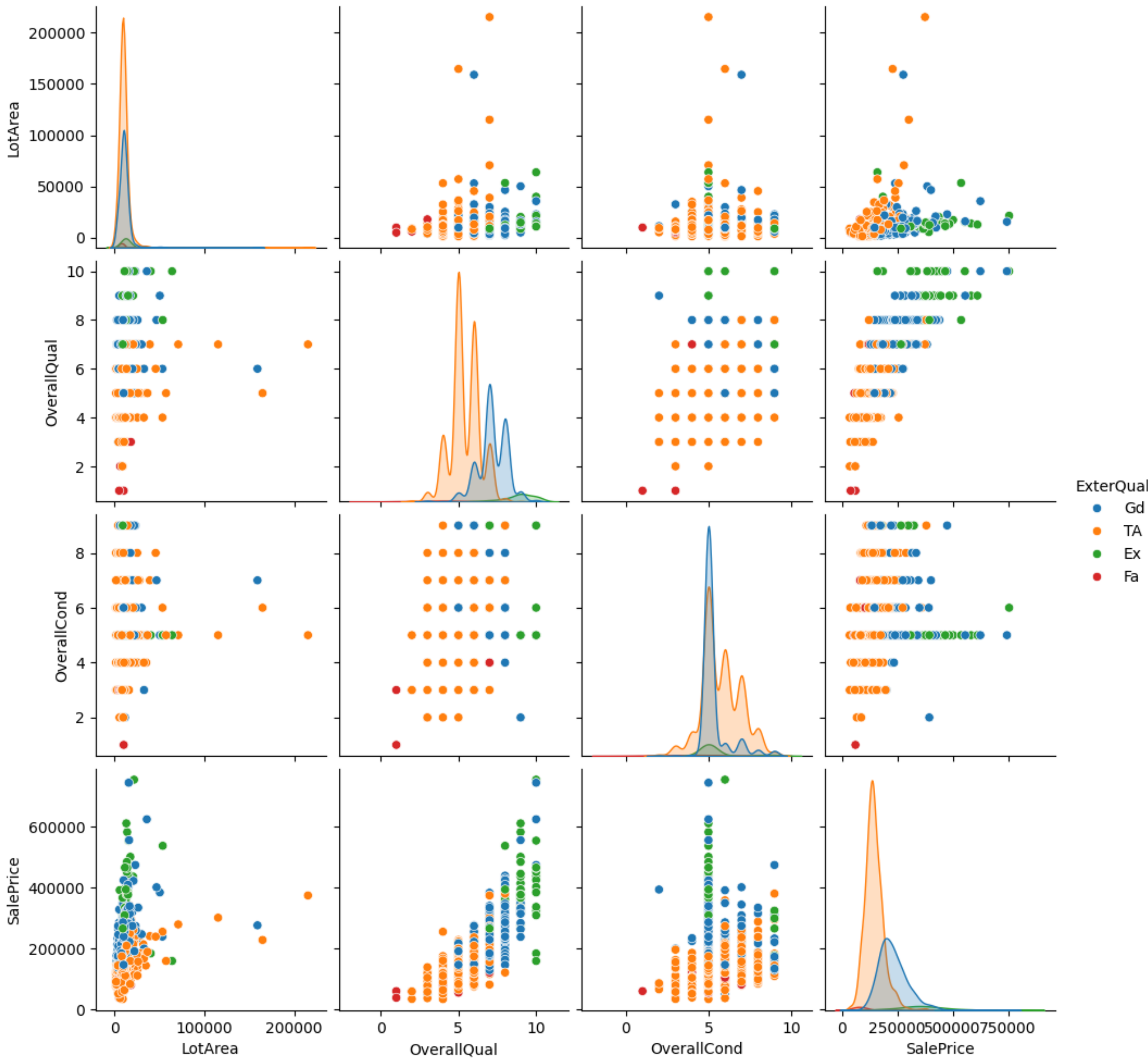
Step 11 : Predicting out of sample data

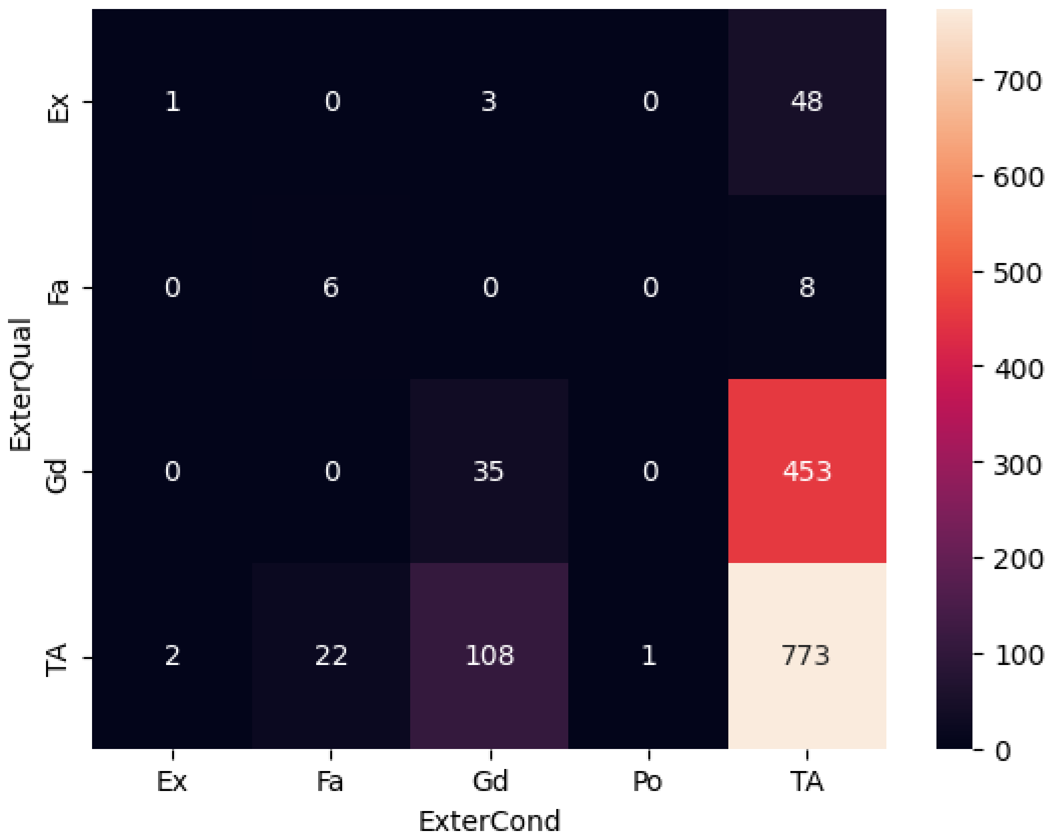
7. Results:

Here as compare to Ridge my Lasso Regression is slidely higher in R2 score. So I can use my Lasso Model to predict out of sample data.

8. EDA: Correlation Heatmap




LotArea
OverallQual
OverallCond
SalePrice
ExterQual
Gd
TA
Ex
Fa



9. Conclusion

These challenges can make the process of finding a suitable rental house daunting. Prospective renters need to be well-prepared, conduct thorough research, and possibly seek assistance from real estate agents or rental services to navigate these obstacles effectively. Implementing technology, such as rental apps and online platforms with detailed reviews and ratings, can also help alleviate some of these issues by providing more transparent and accessible information.

CODING PART

Step 1 : Read Training CSV file

```
]:  import pandas as pd
    df = pd.read_csv('training_set.csv')
    pd.set_option('display.max_columns',None)
    df.head()
```

]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neigh
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	

step 2 : Check Data Quality and missing values

```
In [3]:  df.shape

Out[3]:  (1460, 81)

In [4]:  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
```

In [6]:
```
mis = df.isna().sum()
mis[mis>=1]
```
Out[6]:
```
LotFrontage      259
Alley           1369
MasVnrType       872
MasVnrArea         8
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Electrical         1
FireplaceQu      690
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
GarageCond        81
PoolQC          1453
Fence           1179
MiscFeature     1406
dtype: int64
```
In [7]:
```
df.duplicated().sum()
```
Out[7]:
```
0
```
Step 3 : separate X and Y (SalePrice) features

In [9]:
```
X = df.drop(columns=['Id','SalePrice'])
Y = df[['SalePrice']]
```
In [10]:
```
X.head()
```
Out[10]:

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

step 4 : create a preprocessing pipeline for X

[12]:
```
cat= X.columns[X.dtypes=='object']
con = X.columns[X.dtypes!='object']
```

[13]:
```
cat
```

[13]:
```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       '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',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

[14]:
```
con
```

[14]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       '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'],
      dtype='object')
```

[15]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler,OrdinalEncoder
from sklearn.compose import ColumnTransformer
```

[16]:
```
num_pipe = Pipeline(steps=[('impute',SimpleImputer(strategy='mean')),
                           ('scaler', StandardScaler())])
```

[17]:
```
cat_pipe = Pipeline(steps=[('impute',SimpleImputer(strategy='constant', fill_value='Not_Avail')),
                           ('ohe',OrdinalEncoder())])
```

[18]:
```
pre = ColumnTransformer([('num', num_pipe, con),
                         ('cat',cat_pipe,cat)]).set_output(transform='pandas')
```

[19]:
```
pre
```

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:

```python
X_pre = pre.fit_transform(X)
X_pre.head()
```

Out[20]:

	num__MSSubClass	num__LotFrontage	num__LotArea	num__OverallQual	num__OverallCond	num__YearBuilt	num__YearRemodAdd	nu
0	0.073375	-0.229372	-0.207142	0.651479	-0.517200	1.050994	0.878668	
1	-0.872563	0.451936	-0.091886	-0.071836	2.179628	0.156734	-0.429577	
2	0.073375	-0.093110	0.073480	0.651479	-0.517200	0.984752	0.830215	
3	0.309859	-0.456474	-0.096897	0.651479	-0.517200	-1.863632	-0.720298	
4	0.073375	0.633618	0.375148	1.374795	-0.517200	0.951632	0.733308	

step 5 : Forward feature selection

In [21]:

```python
# feature selection
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import SequentialFeatureSelector
lr = LinearRegression()
sel = SequentialFeatureSelector(lr,n_features_to_select='auto',direction='forward')
sel.fit(X_pre,Y)
sel_cols = sel.get_feature_names_out()
sel_cols
```

Out[21]:

```
array(['num__MSSubClass', 'num__LotArea', 'num__OverallQual',
       'num__OverallCond', 'num__YearBuilt', 'num__MasVnrArea',
       'num__BsmtFinSF1', 'num__GrLivArea', 'num__BsmtFullBath',
       'num__Fireplaces', 'num__GarageCars', 'num__WoodDeckSF',
       'num__EnclosedPorch', 'num__ScreenPorch', 'num__PoolArea',
       'num__YrSold', 'cat__Street', 'cat__LandContour', 'cat__Utilities',
       'cat__Neighborhood', 'cat__BldgType', 'cat__HouseStyle',
       'cat__RoofStyle', 'cat__RoofMatl', 'cat__Exterior1st',
       'cat__MasVnrType', 'cat__ExterQual', 'cat__Foundation',
       'cat__BsmtQual', 'cat__BsmtCond', 'cat__BsmtExposure',
       'cat__HeatingQC', 'cat__KitchenQual', 'cat__Functional',
       'cat__GarageFinish', 'cat__GarageCond', 'cat__PavedDrive',
```

In [23]:

```
sel_cols[0]
```

Out[23]: 'num__MSSubClass'

In [24]:

```
sel_cols[0].split('__')
```

Out[24]: ['num', 'MSSubClass']

In [25]:

```
sel_cols[0].split('__')[1]
```

Out[25]: 'MSSubClass'

In [26]:

```
# Extracting original column name
imp_cols = []
for i in sel_cols:
    s = i.split('__')[1]
    imp_cols.append(s)
```

In [27]:

```
imp_cols
```

Out[27]:

```
['MSSubClass',
 'LotArea',
 'OverallQual',
 'OverallCond',
```

In [28]:

```
X_sel = X[imp_cols]
X_sel
```

Out[28]:

	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	MasVnrArea	BsmtFinSF1	GrLivArea	BsmtFullBath	Fireplaces	Garage
0	60	8450	7	5	2003	196.0	706	1710	1	0	
1	20	9600	6	8	1976	0.0	978	1262	0	1	
2	60	11250	7	5	2001	162.0	486	1786	1	1	
3	70	9550	7	5	1915	0.0	216	1717	1	1	
4	60	14260	8	5	2000	350.0	655	2198	1	1	
...	...	...	...	...	...	...	...	...	...	...	
1455	60	7917	6	5	1999	0.0	0	1647	0	1	
1456	20	13175	6	6	1978	119.0	790	2073	1	2	
1457	70	9042	7	9	1941	0.0	275	2340	0	2	
1458	20	9717	5	6	1950	0.0	49	1078	1	0	
1459	20	9937	5	6	1965	0.0	830	1256	1	0	

1460 rows × 39 columns

Step 6 : Create a final pipeline

categorical - OneHotEncoder

In [29]:
```
cat_sel = list(X_sel.columns[X_sel.dtypes=='object'])
con_sel = list(X_sel.columns[X_sel.dtypes!='object'])
```

In [30]:
```
cat_sel
```

Out[30]:
```
['Street',
 'LandContour',
 'Utilities',
 'Neighborhood',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'MasVnrType',
 'ExterQual',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'HeatingQC',
 'KitchenQual',
```

In [32]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [33]:
```
num_pipe1 = Pipeline(steps=[('impute',SimpleImputer(strategy='mean')),
                            ('scaler',StandardScaler())])
```

In [34]:
```
cat_pipe1 = Pipeline([('impute',SimpleImputer(strategy='constant', fill_value='Not_Avail')),
                      ('ohe',OneHotEncoder(handle_unknown='ignore',sparse_output=False))])
```

In [35]:
```
pre1 = ColumnTransformer([('num',num_pipe1,con_sel),
                          ('cat',cat_pipe1,cat_sel)]).set_output(transform='pandas')
```

In [36]:
```
X_sel_pre = pre1.fit_transform(X_sel)
X_sel_pre.head()
```

Out[36]:

	num__MSSubClass	num__LotArea	num__OverallQual	num__OverallCond	num__YearBuilt	num__MasVnrArea	num__BsmtFinSF1	num__
0	0.073375	-0.207142	0.651479	-0.517200	1.050994	0.511418	0.575425	
1	-0.872563	-0.091886	-0.071836	2.179628	0.156734	-0.574410	1.171992	
2	0.073375	0.073480	0.651479	-0.517200	0.984752	0.323060	0.092907	
3	0.309859	-0.096897	0.651479	-0.517200	-1.863632	-0.574410	-0.499274	
4	0.073375	0.375148	1.374795	-0.517200	0.951632	1.364570	0.463568	

Step 7: Train Test split

20% Unseen to model

In [37]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(X_sel_pre,Y,test_size=0.20,random_state=42)
```

In [38]:
```
xtrain.head()
```

Out[38]:

	num__MSSubClass	num__LotArea	num__OverallQual	num__OverallCond	num__YearBuilt	num__MasVnrArea	num__BsmtFinSF1	nu
254	-0.872563	-0.212153	-0.795151	0.381743	-0.472560	-0.574410	1.049169	
1066	0.073375	-0.268578	-0.071836	1.280685	0.719786	-0.574410	-0.973018	
638	-0.636078	-0.174369	-0.795151	1.280685	-2.029235	-0.574410	-0.973018	
799	-0.163109	-0.332419	-0.795151	1.280685	-1.134975	0.821655	0.274948	
380	-0.163109	-0.552908	-0.795151	0.381743	-1.565545	-0.574410	-0.494887	

In [39]:
```
ytrain.head()
```

Step 8 : Building Model and Evaluate Model

In [42]:
```
model_linear = LinearRegression()
model_linear.fit(xtrain,ytrain)
```

Out[42]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [43]:
```
model_linear.score(xtrain,ytrain)
```

Out[43]:
```
0.9079802219811148
```

In [44]:
```
model_linear.score(xtest,ytest)
```

Out[44]:
```
-7.239616171417617e+17
```

In [45]:
```
def adj_r2(model,xtrain,ytrain):
    r2 = model.score(xtrain,ytrain)
    N = xtrain.shape[0]
    p = xtrain.shape[1]
    num = (1 - r2)*(N - 1)
    den = N-p-1
    r2a = 1 - (num/den)
    return r2a
```

In [46]:

```
adj_r2(model_linear,xtrain,ytrain)
```

Out[46]:

```
0.8934651974721836
```

In [47]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
def evaluate_model(model_linear,x,y):
    ypred = model_linear.predict(x)
    mse = mean_squared_error(y, ypred)
    rmse = (mse)**(1/2)
    mae = mean_absolute_error(y, ypred)
    r2 = r2_score(y, ypred)
    print(f'Mean squared error : {mse:.2f}')
    print(f'Root Mean Squared Error : {rmse:.2f}')
    print(f'Mean absolute error : {mae:.2f}')
    print(f'R2 Score : {r2:.4f}')
```

In [48]:

```
evaluate_model(model_linear,xtrain,ytrain)
```

```
Mean squared error : 548856601.93
Root Mean Squared Error : 23427.69
Mean absolute error : 14648.67
R2 Score : 0.9080
```

In [49]:

```
evaluate_model(model_linear,xtest,ytest)
```

```
Mean squared error : 5553021764960026602843930624.00
Root Mean Squared Error : 74518600127485.12
Mean absolute error : 6121459754421.55
R2 Score : -723961617141761664.0000
```

step 9 : Ridge Model

In [50]:
```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=2)
model_ridge.fit(xtrain,ytrain)
```

Out[50]:
```
Ridge(alpha=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [51]:
```
model_ridge.score(xtrain,ytrain)
```

Out[51]:
```
0.8924188065303781
```

In [52]:
```
model_ridge.score(xtest,ytest)
```

Out[52]:
```
0.884697438853286
```

Tuning the ridge model with GridSearchCV

In [53]:
```
import numpy as np
params = {'alpha':np.arange(start=0.1,stop=100,step=0.1)}
params
```

In [54]:
```
from sklearn.model_selection import GridSearchCV
rr = Ridge()
gscv = GridSearchCV(rr, param_grid=params, cv=5,scoring='neg_mean_squared_error')
gscv.fit(xtrain,ytrain)
```

Out[54]:
```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,  1.1,
        1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,  2.2,
        2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  3.3,
        3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  4.4,
        4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,  5.5,
        5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,  6.6,
        6.7,  6.8,  6.9,  7. ,  7.1,  7.2,  7.3,  7....
       93.6, 93.7, 93.8, 93.9, 94. , 94.1, 94.2, 94.3, 94.4, 94.5, 94.6,
       94.7, 94.8, 94.9, 95. , 95.1, 95.2, 95.3, 95.4, 95.5, 95.6, 95.7,
       95.8, 95.9, 96. , 96.1, 96.2, 96.3, 96.4, 96.5, 96.6, 96.7, 96.8,
       96.9, 97. , 97.1, 97.2, 97.3, 97.4, 97.5, 97.6, 97.7, 97.8, 97.9,
       98. , 98.1, 98.2, 98.3, 98.4, 98.5, 98.6, 98.7, 98.8, 98.9, 99. ,
       99.1, 99.2, 99.3, 99.4, 99.5, 99.6, 99.7, 99.8, 99.9])},
             scoring='neg_mean_squared_error')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [55]:
```
gscv.best_params_
```

Out[55]:
```
{'alpha': 0.8}
```

In [55]:
```
gscv.best_params_
```
Out[55]: {'alpha': 0.8}

In [56]:
```
gscv.best_score_
```
Out[56]: -1041181784.8223242

In [57]:
```
best_ridge = gscv.best_estimator_
best_ridge
```
Out[57]: Ridge(alpha=0.8)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [58]:
```
best_ridge.score(xtrain,ytrain)
```
Out[58]: 0.9001439358205136

In [59]:
```
best_ridge.score(xtest,ytest)
```
Out[59]: 0.8888885119672979

In [60]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
def evaluate_model(model, x, y):
    ypred = model.predict(x)
    mse = mean_squared_error(y, ypred)
    rmse = mse**(1/2)
    mae = mean_absolute_error(y, ypred)
    r2 = r2_score(y, ypred)
    print(f'Mean Squared Error : {mse:.2f}')
    print(f'Root Mean Squared Error : {rmse:.2f}')
    print(f'Mean Absolute Error : {mae:.2f}')
    print(f'R2 Score : {r2:.4f}')
```

In [61]:
```
evaluate_model(best_ridge,xtrain,ytrain)
```

```
Mean Squared Error : 595596525.52
Root Mean Squared Error : 24404.85
Mean Absolute Error : 15297.67
R2 Score : 0.9001
```

In [62]:
```
evaluate_model(best_ridge,xtest,ytest)
```

```
Mean Squared Error : 852261358.58
Root Mean Squared Error : 29193.52
Mean Absolute Error : 18053.34
R2 Score : 0.8889
```

In [63]:
```
from sklearn.model_selection import cross_val_score
ridge_score = cross_val_score(best_ridge,xtrain,ytrain,cv=5,scoring='r2')
ridge_score
```

Out[63]:
```
array([0.85674025, 0.78938492, 0.71723611, 0.8957829 , 0.90033995])
```

In [64]:
```
ridge_score.mean()
```

Out[64]:
```
0.8318968267317238
```

step 10 : Lasso Model

In [65]:
```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=1)
model_lasso.fit(xtrain,ytrain)
```

Out[65]: Lasso(alpha=1)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [66]:
```
model_lasso.score(xtrain,ytrain)
```

Out[66]: 0.9079759047430674

In [67]:
```
model_lasso.score(xtest,ytest)
```

Out[67]: 0.8969579903434817

In [68]:
```
params
```

Out[68]:
```
{'alpha': array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,  1.1,
         1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,  2.2,
         2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  3.3,
         3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  4.4,
         4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,  5.5,
         5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,  6.6,
```

In [69]:
```
ls = Lasso()
gscv2 = GridSearchCV(ls,param_grid=params, cv=5, scoring='neg_mean_squared_error')
gscv2.fit(xtrain,ytrain)
```

Out[69]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,  1.1,
        1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,  2.2,
        2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  3.3,
        3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  4.4,
        4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,  5.5,
        5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,  6.6,
        6.7,  6.8,  6.9,  7. ,  7.1,  7.2,  7.3,  7....
       93.6, 93.7, 93.8, 93.9, 94. , 94.1, 94.2, 94.3, 94.4, 94.5, 94.6,
       94.7, 94.8, 94.9, 95. , 95.1, 95.2, 95.3, 95.4, 95.5, 95.6, 95.7,
       95.8, 95.9, 96. , 96.1, 96.2, 96.3, 96.4, 96.5, 96.6, 96.7, 96.8,
       96.9, 97. , 97.1, 97.2, 97.3, 97.4, 97.5, 97.6, 97.7, 97.8, 97.9,
       98. , 98.1, 98.2, 98.3, 98.4, 98.5, 98.6, 98.7, 98.8, 98.9, 99. ,
       99.1, 99.2, 99.3, 99.4, 99.5, 99.6, 99.7, 99.8, 99.9])},
             scoring='neg_mean_squared_error')
```

In [70]:
```python
gscv.best_params_
```
Out[70]: {'alpha': 0.8}

In [71]:
```python
gscv.best_score_
```
Out[71]: -1041181784.8223242

In [72]:
```python
best_lasso = gscv.best_estimator_
best_lasso
```
Out[72]: Ridge(alpha=0.8)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [73]:
```python
best_lasso.score(xtrain,ytrain)
```
Out[73]: 0.9001439358205136

In [74]:
```python
best_lasso.score(xtest,ytest)
```
Out[74]: 0.8888885119672979

In [75]:
```python
evaluate_model(best_lasso,xtrain,ytrain)
```
```
Mean Squared Error : 595596525.52
Root Mean Squared Error : 24404.85
Mean Absolute Error : 15297.67
R2 Score : 0.9001
```

In [76]:
```python
evaluate_model(best_lasso,xtest,ytest)
```
```
Mean Squared Error : 852261358.58
Root Mean Squared Error : 29193.52
Mean Absolute Error : 18053.34
R2 Score : 0.8889
```

In [77]:
```python
scores_lasso = cross_val_score(best_lasso, xtrain, ytrain, cv=5, scoring='r2')
scores_lasso
```
Out[77]: array([0.85674025, 0.78938492, 0.71723611, 0.8957829 , 0.90033995])

In [78]:
```python
scores_lasso.mean()
```
Out[78]: 0.8318968267317238

In [79]:

```
ypred_train = best_lasso.predict(xtrain)
ypred_test = best_lasso.predict(xtest)
```

In [80]:

```
ypred_train[0:5]
```

Out[80]:

```
array([[142718.90813869],
       [176791.87842717],
       [ 91170.74788301],
       [174816.70167224],
       [159630.75432494]])
```

In [81]:

```
ytrain.head()
```

Out[81]:

	SalePrice
254	145000
1066	178000
638	85000
799	175000
380	127000

In [82]:

```
ypred_test[0:5]
```

Out[82]:

```
array([[146727.29825345],
       [348428.01775191],
       [101697.68495602],
       [178141.03160579],
       [334871.27420215]])
```

In [83]:

```
ytest.head()
```

Out[83]:

	SalePrice
892	154500
1105	325000
413	115000
522	159000
1036	315500