Coincent
Learning Drives Excellence

DATA SCIENCE WITH PYTHON

PROJECT TITLE

Surprise Housing Case Study

Submitted by: -

Diksha Sharma

Institute of Technology, Nirma University

B Tech. CSE 3rd year

Abstract

This project centres around Surprise Housing, a prominent US-based housing company, as it endeavours to expand its operations into the Australian real estate market. The core of the initiative involves leveraging data analytics to construct a predictive model for house prices in Australia. The primary aim is to unravel the complex interplay between various independent variables and housing prices, offering valuable insights for strategic decision-making. The project adopts a methodology incorporating data analysis, machine learning, and regression techniques, with a focus on optimizing Surprise Housing's approach for optimal returns in the Australian market.

This report presents the findings of a study conducted to predict house prices in Australia using machine learning techniques. The study employed two regression models, Ridge and Lasso, to identify the most significant variables influencing house prices and assess the predictive performance of the models. The results indicate that both Ridge and Lasso regression achieved satisfactory prediction accuracy, with Ridge yielding a slightly higher training accuracy of 90.9% compared to Lasso's 89.8%. However, Lasso regression emerged as the preferred model due to its superior feature selection capabilities. The top five most significant variables identified by both Ridge and Lasso regression were SaleCondition, GarageFinish, and OverallQual. The optimal values of lambda for Ridge and Lasso regression were found to be 10 and 0.001, respectively.

Objective

The primary objective of this project is to develop a robust predictive model for house prices in the Australian real estate market. By thoroughly analyzing and understanding the relationships between diverse independent variables and housing prices, Surprise Housing seeks to gain actionable insights. These insights are crucial for informing strategic decisions, guiding investment, and maximizing returns in the Australian housing market. The objective is not merely predictive accuracy but a comprehensive understanding of the factors influencing house prices to facilitate informed decision-making.

The objective of this study is summarized as:

- Identify the most significant variables influencing house prices in Australia
- Assess the predictive performance of Ridge and Lasso regression models for house price prediction
- Determine the optimal value of lambda for both Ridge and Lasso regression models

Introduction

Predicting house prices accurately is a crucial aspect of the real estate industry, enabling informed decision-making for both buyers and sellers. Machine learning techniques have emerged as powerful tools for house price prediction, offering a data-driven approach to identify patterns and relationships within real estate data. This study aims to utilize Ridge and Lasso regression models to predict house prices in Australia and evaluate their effectiveness.

Surprise Housing, a leading US-based housing company, is strategically expanding its operations into the Australian real estate market. Recognizing the significance of data-driven decision-making, the company has undertaken a comprehensive data analytics initiative. The project revolves around a dataset comprising information on house sales in Australia, encompassing various attributes such as location, size, amenities, and more. This understanding is not a mere statistical exercise but a strategic imperative. In a market where each locality possesses its unique set of dynamics, informed decision-making hinges on a granular comprehension of these intricacies.

The objective is to create a robust predictive model that captures the nuanced relationships between these variables and housing prices. This model will serve as a pivotal tool for Surprise Housing, guiding its market strategy, pricing decisions, and overall approach in the Australian housing market. However, navigating this landscape necessitates more than a superficial understanding. It requires a predictive tool that can assimilate multifaceted data and distil it into actionable insights.

Choice of Regression Techniques: Lasso and Ridge Regression:

The choice of regression techniques in constructing the predictive model is a critical aspect of this project. In this context, both Lasso (Least Absolute Shrinkage and Selection Operator) and Ridge Regression techniques are employed, and their selection is not arbitrary but based on their specific attributes.

- **Handling Multicollinearity**: Multicollinearity, the phenomenon where independent variables are correlated, poses a common challenge in regression analysis. Ridge Regression, by introducing a regularization term to the cost function, mitigates multicollinearity. It prevents the model from assigning disproportionate weights to correlated variables, thereby enhancing its stability.
- **Feature Selection:** Lasso Regression is particularly effective in feature selection. By adding a penalty term based on the absolute values of the coefficients, Lasso encourages sparsity in the model. This means that it tends to drive some coefficients to exactly zero, effectively selecting a subset of the most relevant features. In a real estate market influenced by myriad factors, feature selection is crucial for focusing on the most impactful variables.
- **Preventing Overfitting:** Both Ridge and Lasso Regression are regularization techniques that help prevent overfitting. Overfitting occurs when a model captures noise in the training data rather than underlying patterns. Ridge and Lasso introduce regularization terms that penalize overly complex models, promoting generalizability to new, unseen data.

In summary, the choice of Lasso and Ridge Regression is deliberate, addressing specific challenges inherent in real-world datasets and enhancing the model's robustness in predicting house prices in the Australian real estate market. These techniques align with the project's overarching goal of providing Surprise Housing with a reliable and interpretable predictive tool for strategic decision-making.

Methodology

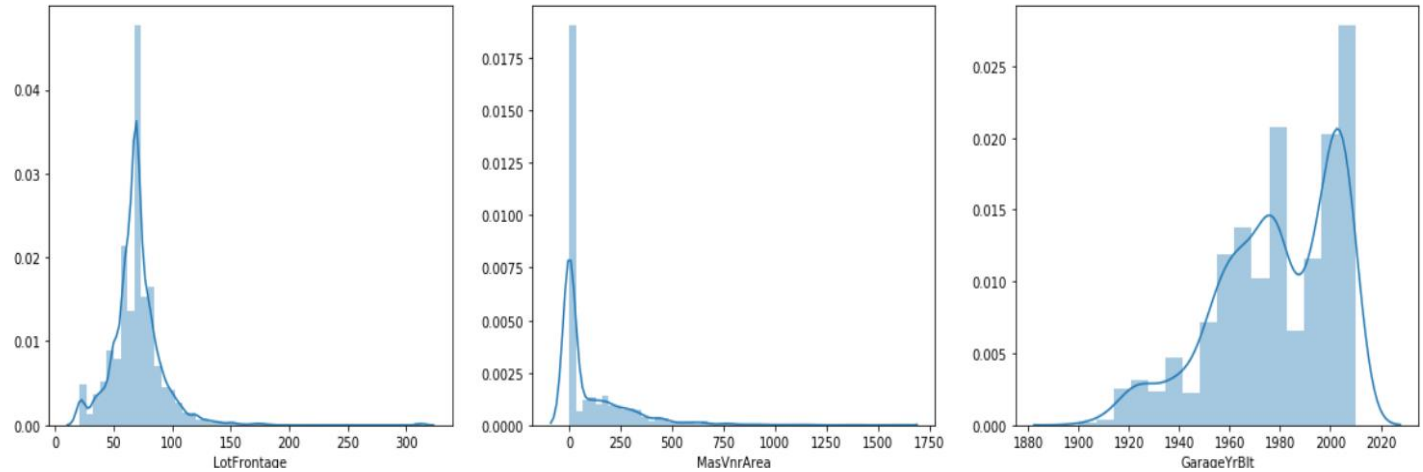
The solution is divided into the following sections:

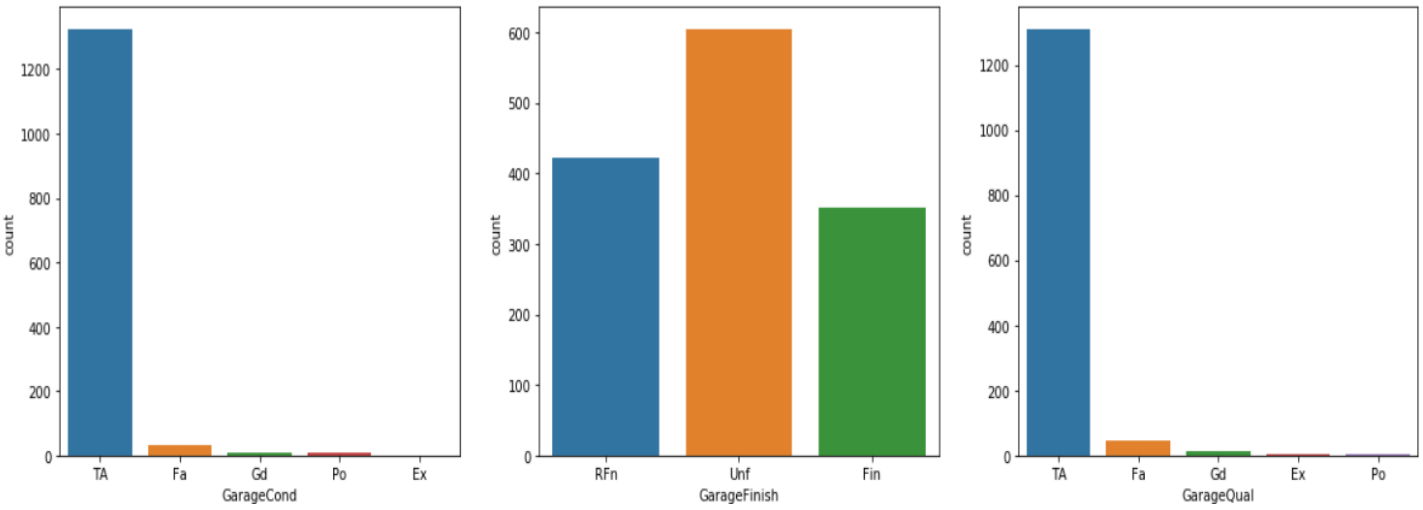
- Data understanding and exploration
- Data cleaning and preparation
- Model building and evaluation

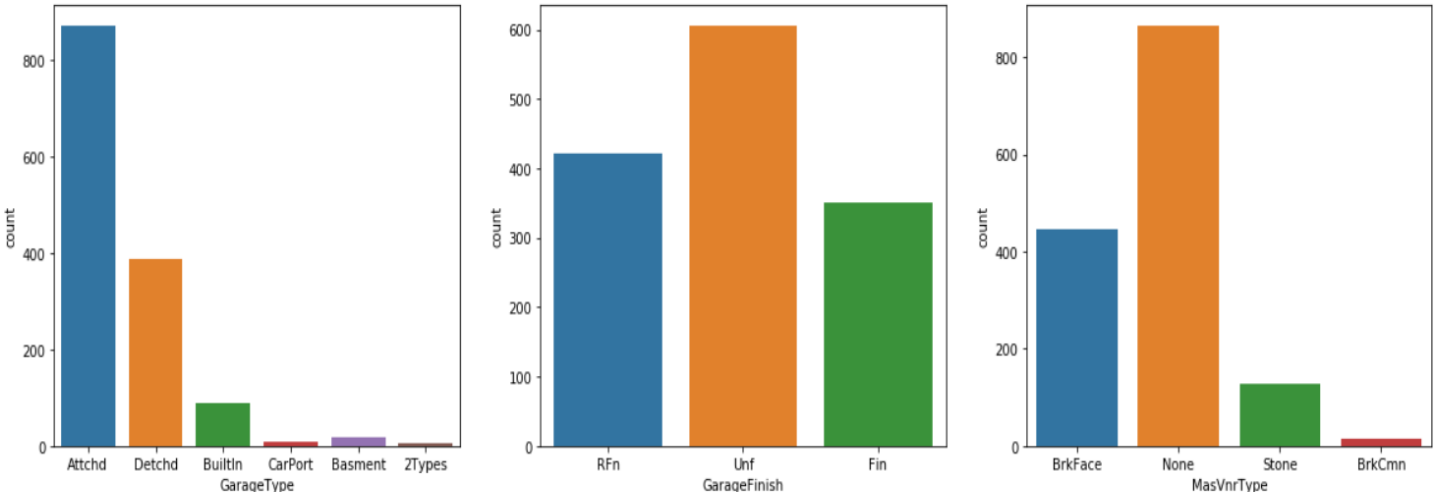
1. **Data Understanding and Exploration**:

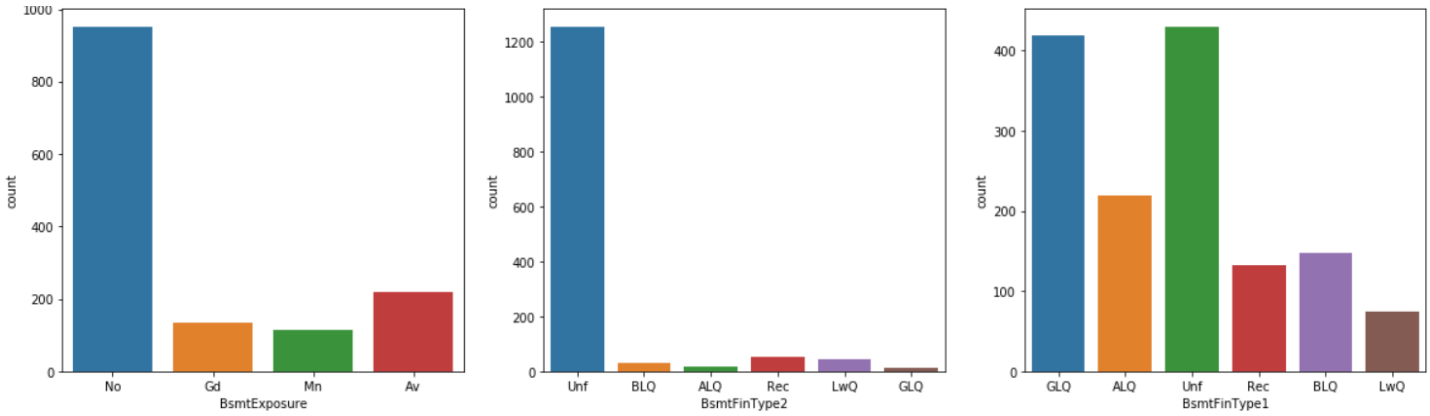
- *Import Libraries:* Import necessary libraries such as NumPy, Pandas, Matplotlib, Seaborn, and various modules from scikit-learn.
- *Read Dataset:* Load the housing dataset using Pandas.
- *Check Missing Values:* Calculate the percentage of missing values for each column and drop columns with more than 40% missing values.
- *Impute Missing Values*: Impute missing values in selected columns using median and mean.

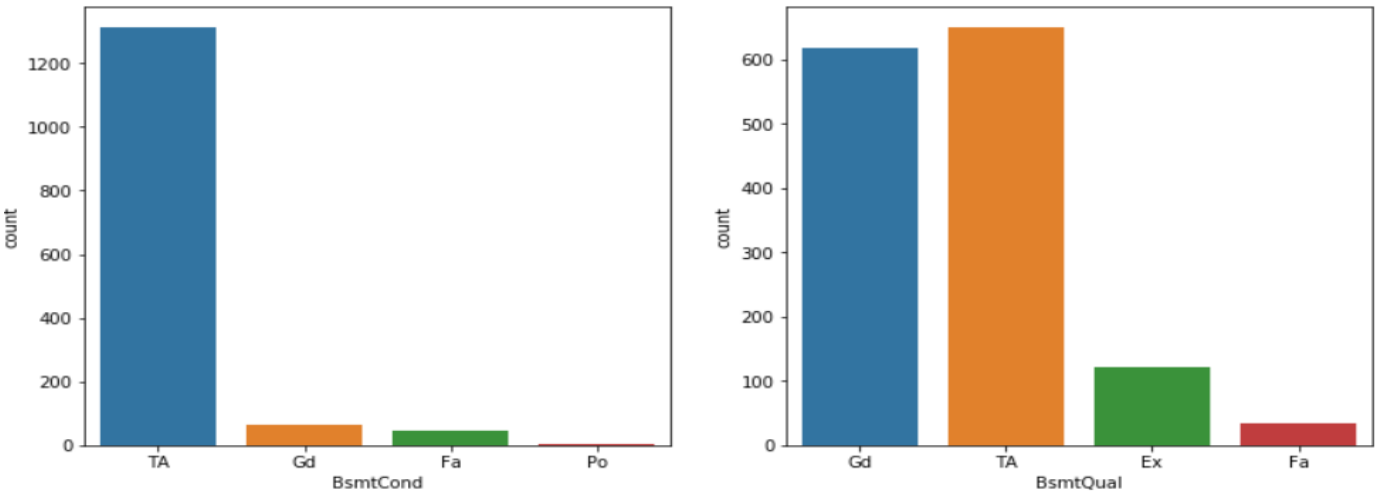
- *Visualize Distributions:* Plot histograms and count plots to visualize the distribution of selected numerical and categorical variables.



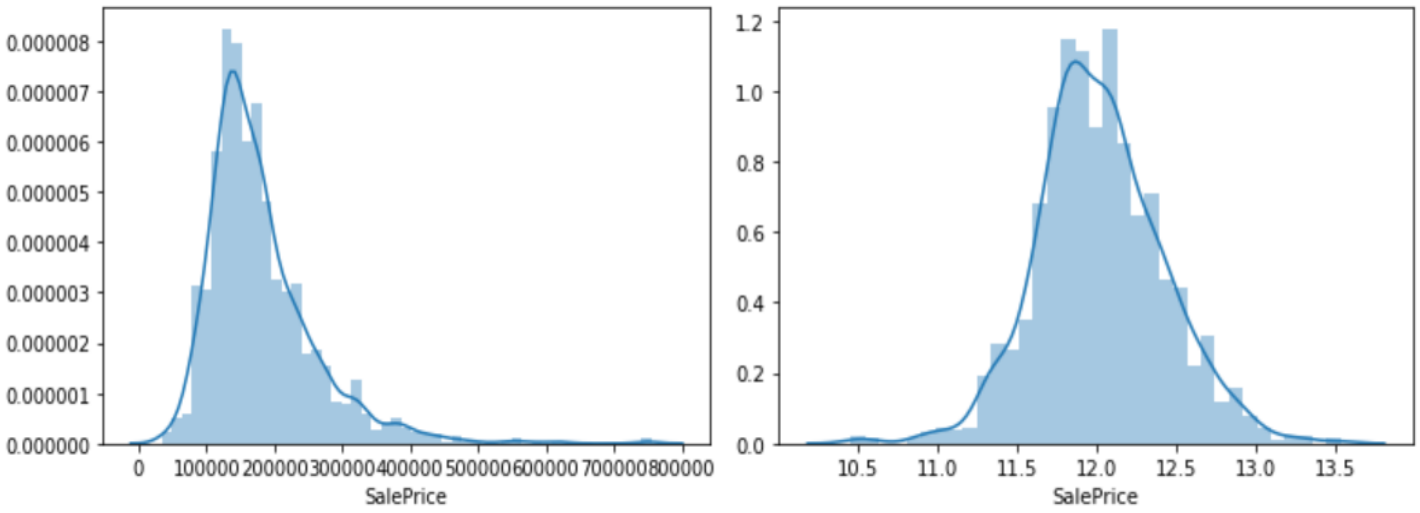




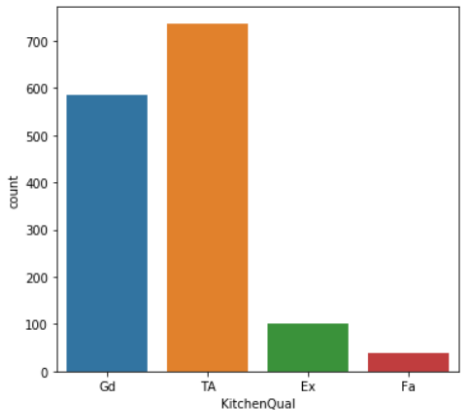


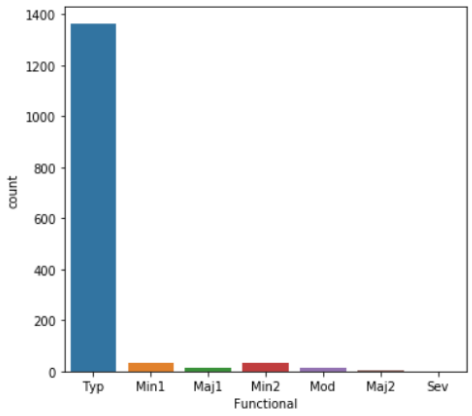


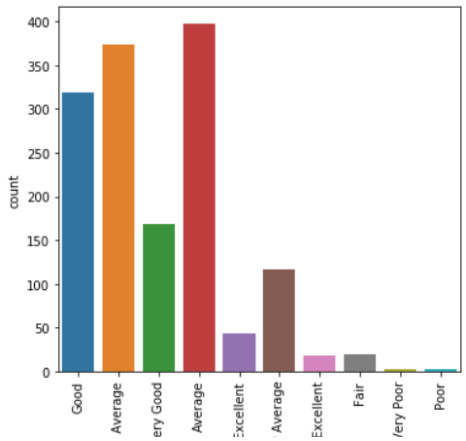
- *Handle Categorical Variables*: Replace missing values in categorical columns with appropriate values and convert numerical values to categorical where needed.
- *Transform Target Variable:* Log-transform the target variable 'SalePrice' to make its distribution more normal.

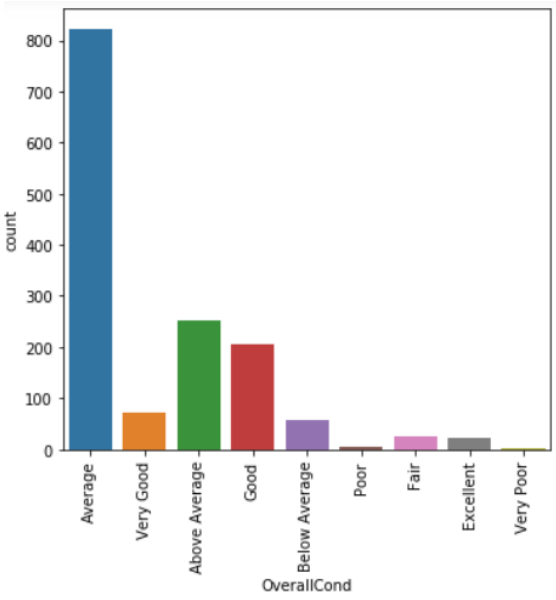


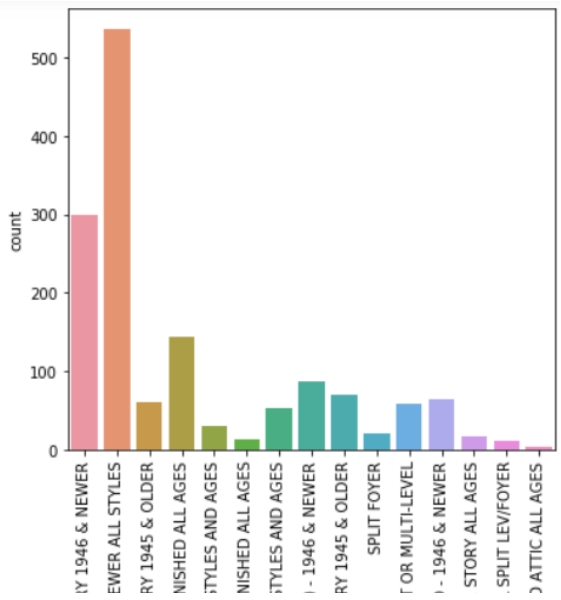
- *Exploratory Data Analysis (EDA):* Explore the distribution of categorical variables and their impact on housing prices using count plots and box plots.

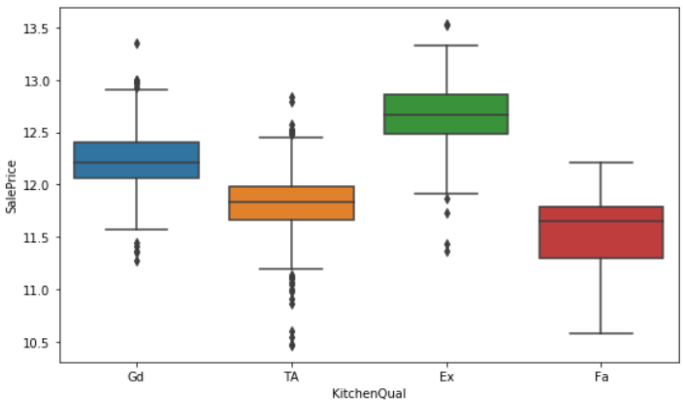


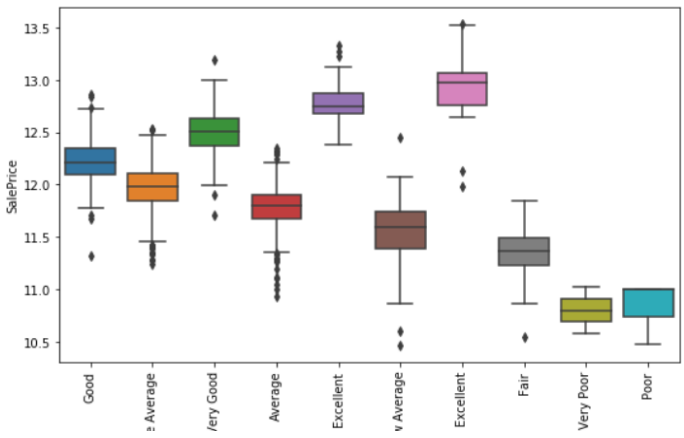


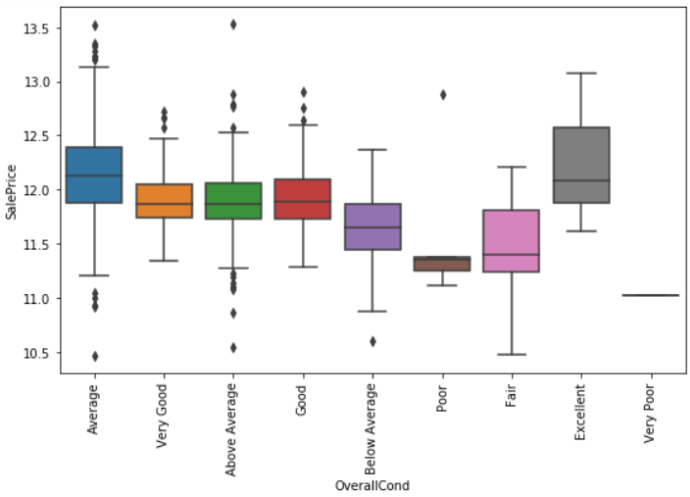


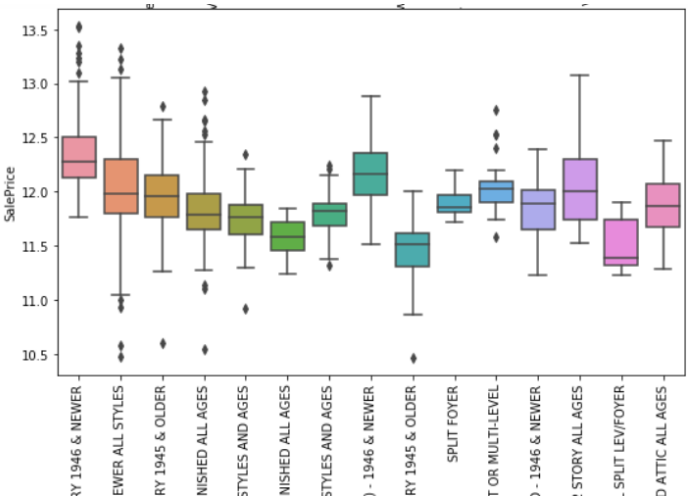


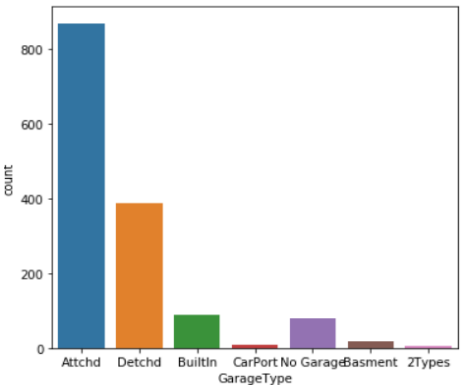


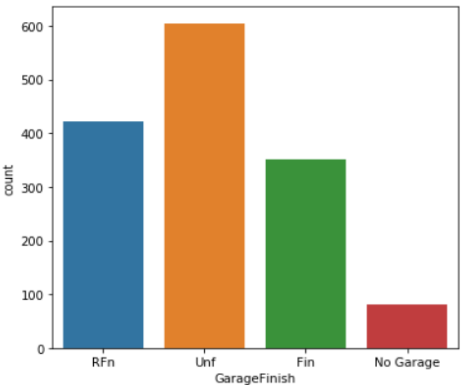


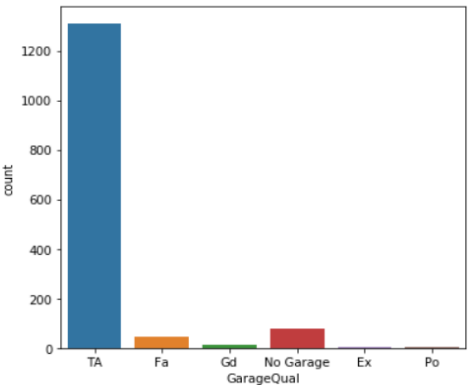


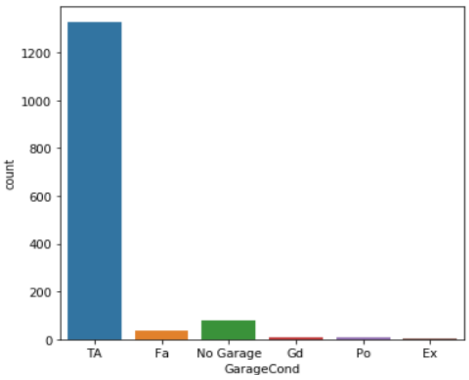

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Average
Very Good
Above Average
Good
Below Average
Poor
Fair
Excellent
Very Poor
OverallCond

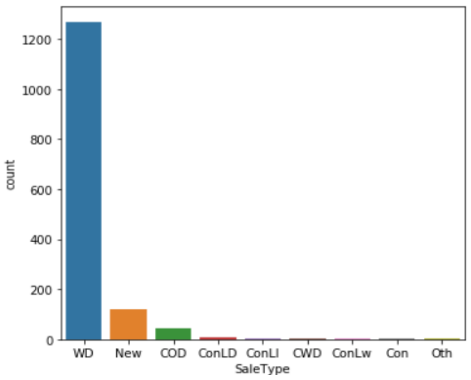

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
SPLIT FOYER

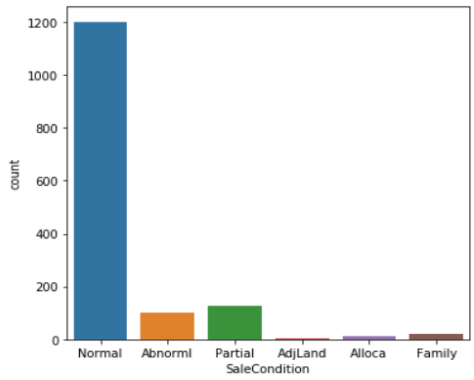

count
800
600
400
200
0
Attchd
Detchd
BuiltIn
CarPort
No Garage
Basment
2Types
GarageType

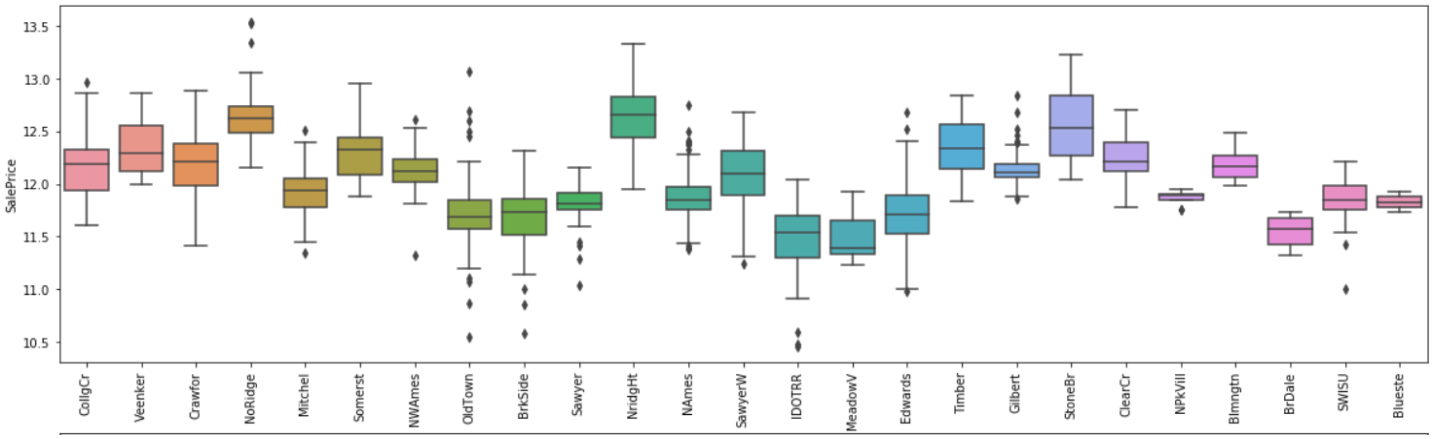

count
600
500
400
300
200
100
0
RFn
Unf
Fin
No Garage
GarageFinish

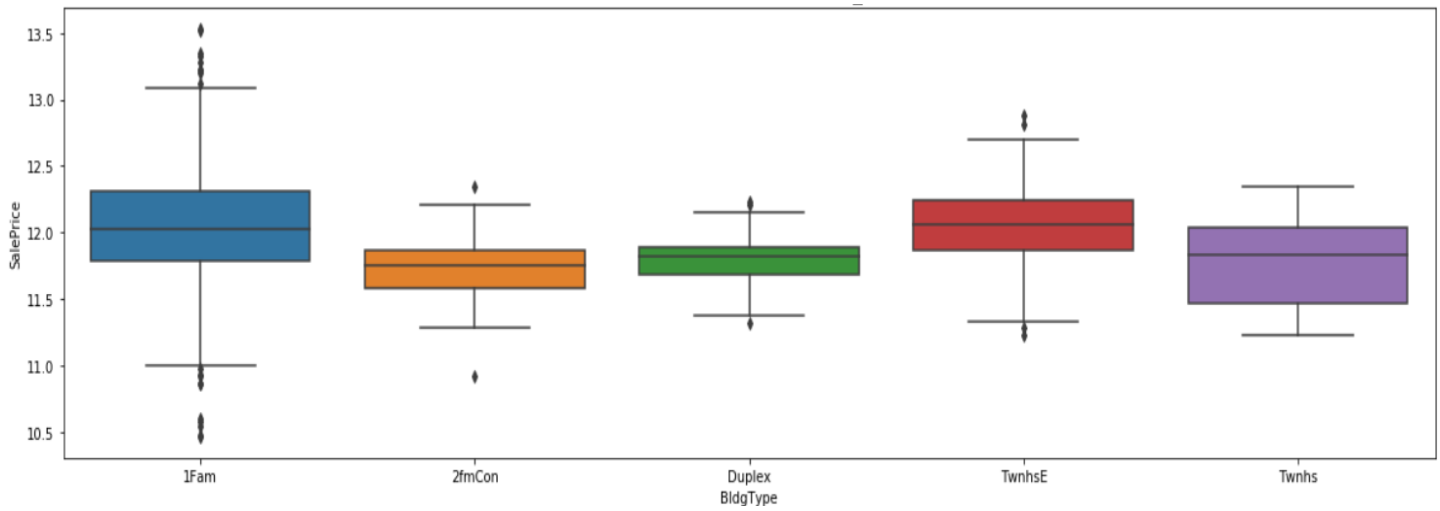

count
1200
1000
800
600
400
200
0
TA
Fa
Gd
No Garage
Ex
Po
GarageQual

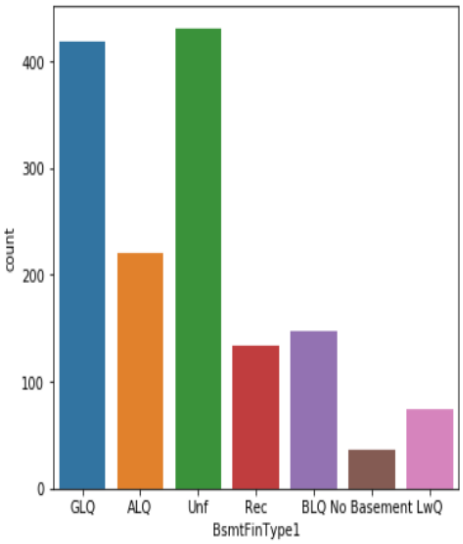

count
1200
1000
800
600
400
200
0
TA
Fa
No Garage
Gd
Po
Ex
GarageCond

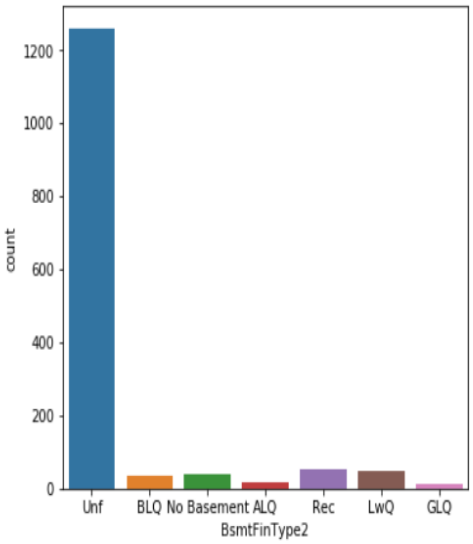

count
1200
1000
800
600
400
200
0
WD
New
COD
ConLD
ConLI
CWD
ConLw
Con
Oth
SaleType

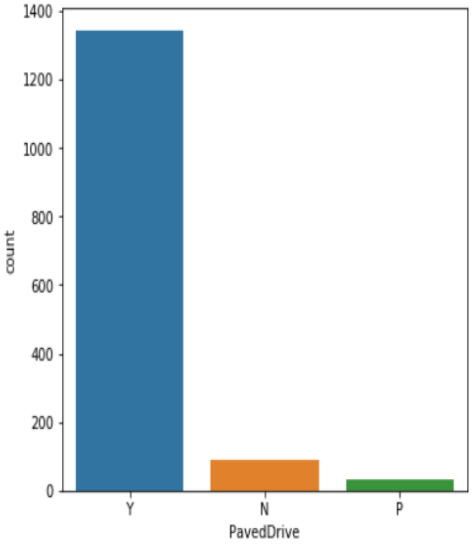

count
1200
1000
800
600
400
200
0
Normal
Abnorml
Partial
AdjLand
Alloca
Family
SaleCondition


SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
CollgCr
Veenker
Crawfor
NoRidge
Mitchel
Somerst
NWAmes
OldTown
BrkSide
Sawyer
NridgHt
NAmes
SawyerW
IDOTRR
MeadowV
Edwards
Timber
Gilbert
StoneBr
ClearCr
NPkVill
Blmngtn
BrDale
SWISU
Blueste







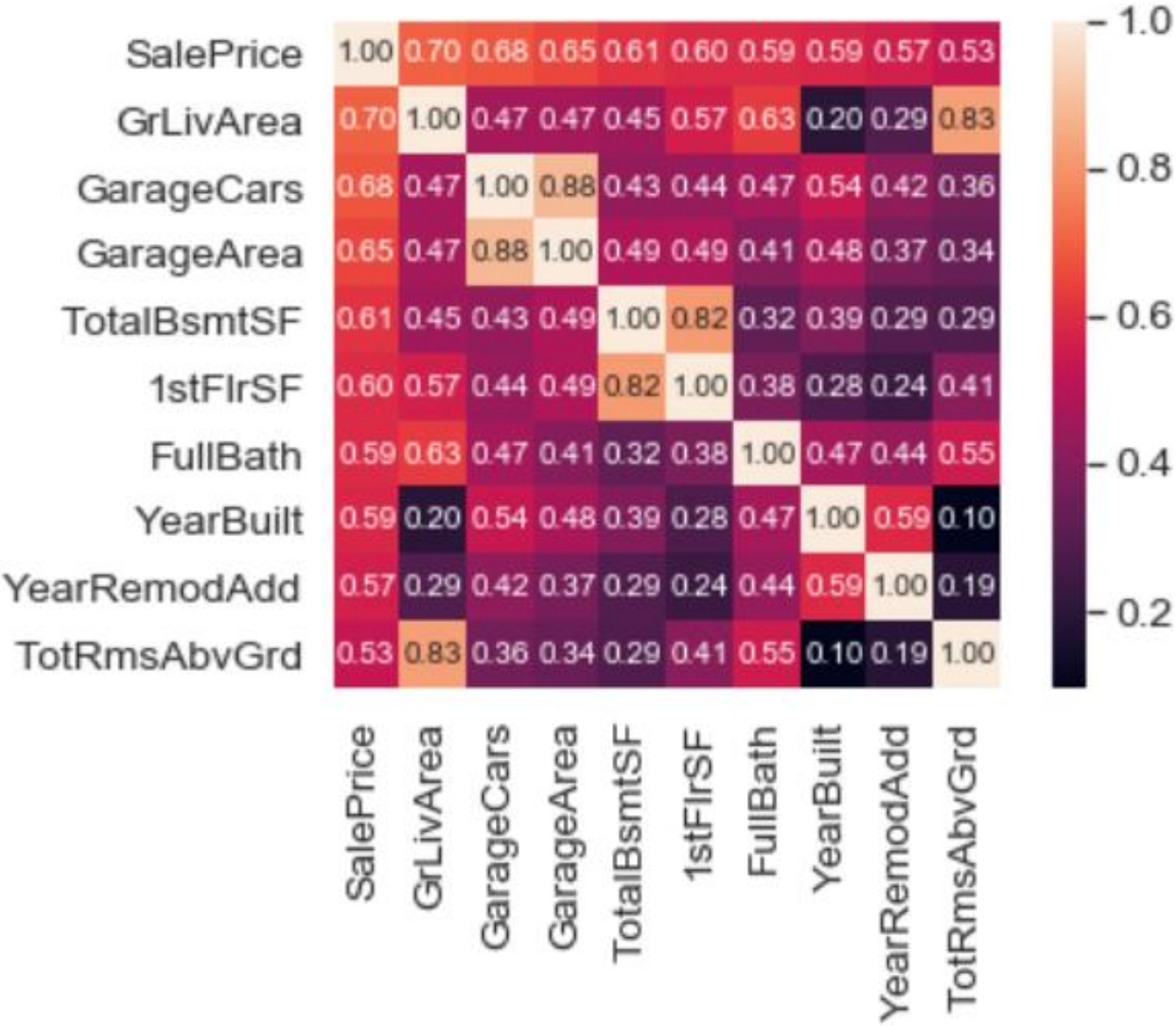


2. Data Cleaning and Preparation:

- *Drop Skewed Columns:* Drop highly skewed columns that are not contributing much information.
- *Group and Replace:* Group certain categories in categorical columns to reduce skewness.
- *Encode Categorical Variables:* Use one-hot encoding to convert categorical variables into dummy variables.
- *Drop Unnecessary Columns:* Drop columns that are no longer needed for model building.

3. Model Building:

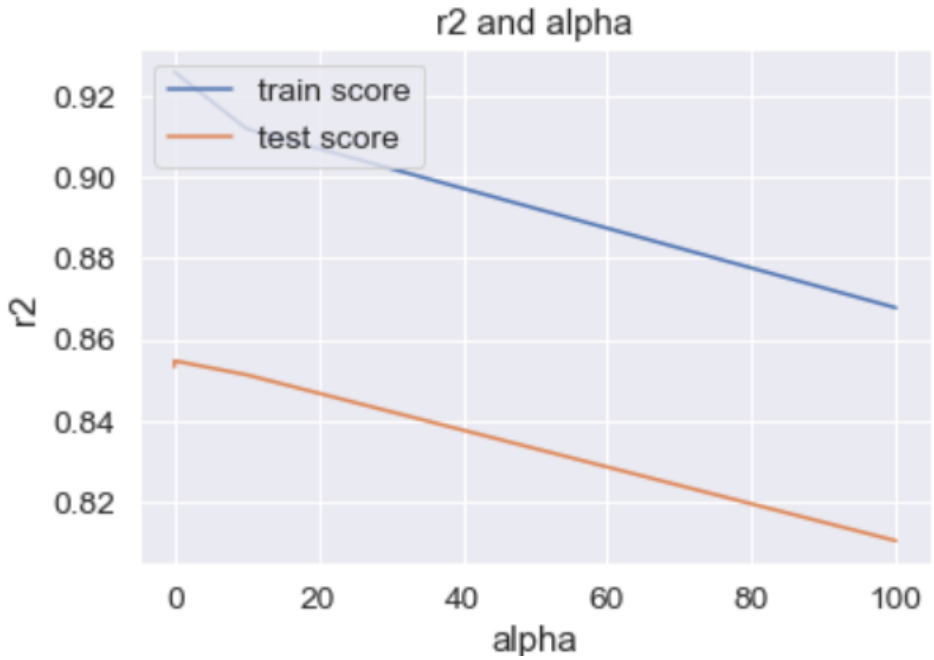
- *Correlation Analysis*: Use a heatmap to visualize the correlation matrix of numerical features.



- *Feature Engineering:* Create a new feature 'YearSinceRemodel' based on the difference between 'YearRemodAdd' and 'YearBuilt'.
- *Split Data:* Split the data into training and testing sets.
- *Standardize Numerical Features:* Standardize numerical features to have zero mean and unit variance.

4. Ridge Regression:

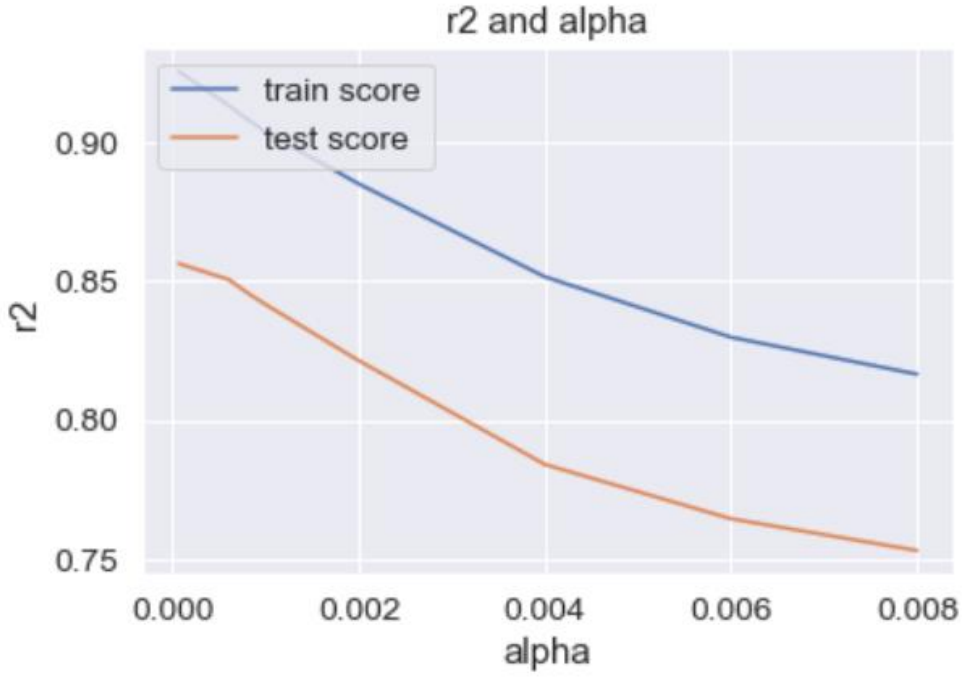
- *Hyperparameter Tuning:* Use GridSearchCV to find the optimal alpha (regularization strength) for Ridge regression.
- *Plot Results:* Plot mean test and train scores against different alpha values.
- *Train Ridge Model:* Train the Ridge regression model with the selected alpha.
- *Model Evaluation:* Evaluate the Ridge model on the training and testing sets, providing the R-squared scores.



5. Lasso Regression:

- *Hyperparameter Tuning*: Use GridSearchCV to find the optimal alpha for Lasso regression.
- *Plot Results:* Plot mean test and train scores against different alpha values for Lasso.
- *Train Lasso Model:* Train the Lasso regression model with the selected alpha.

- *Model Evaluation:* Evaluate the Lasso model on the training and testing sets, providing the R-squared scores.



The Ridge and Lasso regression models are built and evaluated on the dataset. Ridge and Lasso are regularization techniques that help prevent overfitting in linear regression models. The choice between Ridge and Lasso depends on the specific characteristics of the data, and hyperparameter tuning is performed to find the best regularization strength.

In 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 as well we can choose Lasso regression in this case.

Code

1. Data Understanding and Exploration

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

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
surp = pd.read_csv("C://Users//mohit//Downloads//Mohit Data Science//Advanced Regression Assignment//train.csv")
surp.head()
```

In [3]:

```
#check missing percentage
percent_missing = (surp.isnull().sum() / surp.isnull().count()*100).sort_values(ascending = False)
percent_missing.head(15)
```

In [4]:

```
# Dropping Columns with high missing values ,above 40%
surp.drop(['PoolQC','MiscFeature','Alley','Fence','FireplaceQu'],axis=1,inplace=True)
```

In [5]:

```
surp[['LotFrontage','MasVnrArea','GarageYrBlt']].describe()
```

In [6]:

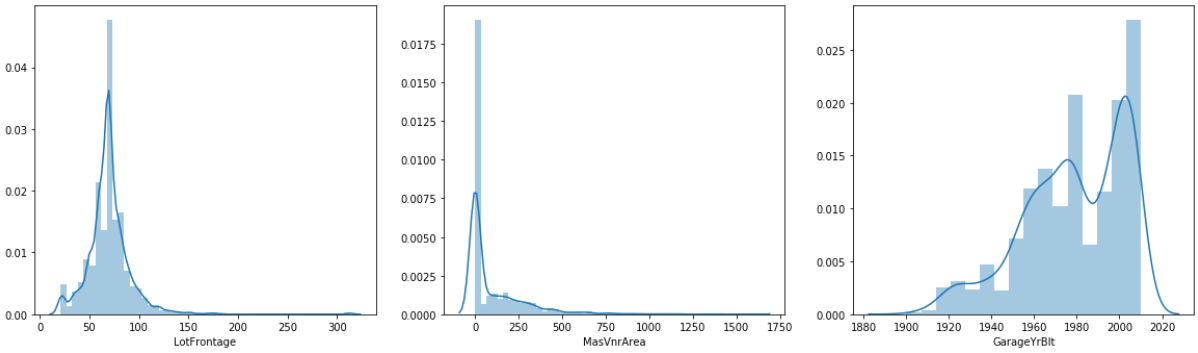
```
#Treaing missing values by imputing for columns with missing values less than or equal to 40%
surp['LotFrontage']= surp.LotFrontage.fillna(surp.LotFrontage.median()) #Can see a presence of outlier so imputing the missing values through median
surp['MasVnrArea']= surp.MasVnrArea.fillna(surp.MasVnrArea.median())  # Can see presence of outlier
surp['GarageYrBlt']= surp.GarageYrBlt.fillna(surp.GarageYrBlt.mean())  # it looks stable with no outlier presence so we can use mean for imputation
```

In [7]:

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

Out[7]:

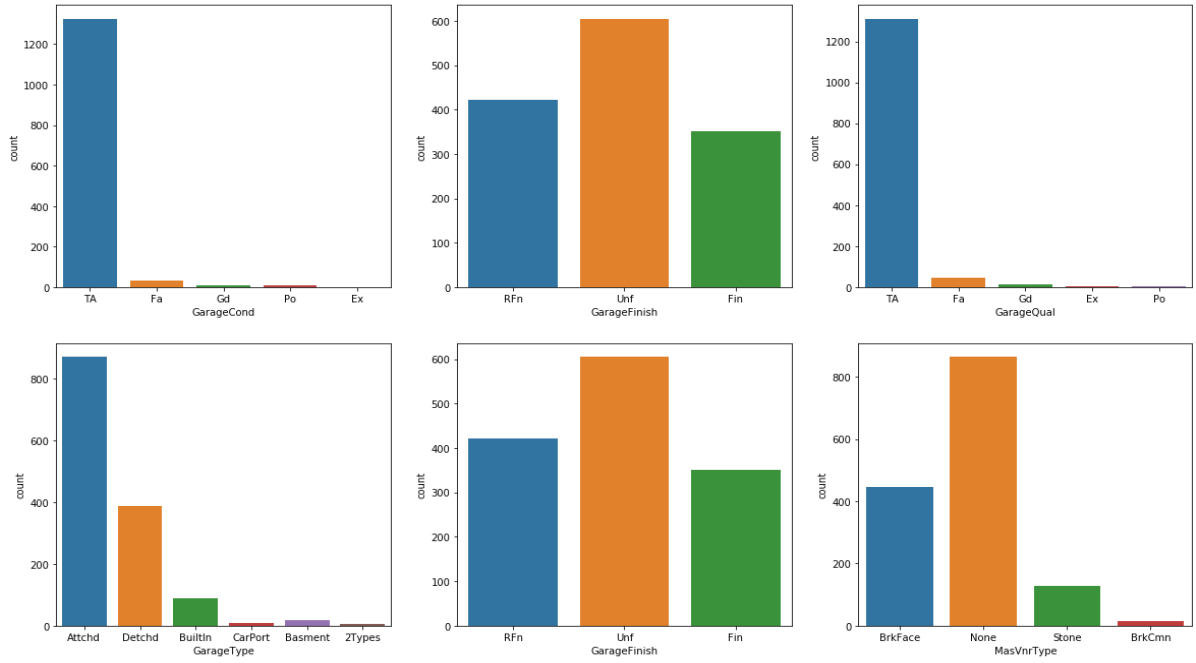
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbb52adbe0>
```



In [8]:

```
#Visualising the variables wih missing values
plt.figure(figsize=(20, 12))
```

```
plt.subplot(2,3,1)
sns.countplot(x = 'GarageCond', data = surp)
plt.subplot(2,3,2)
sns.countplot(x = 'GarageFinish',  data = surp)

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



In [9]:

```
# Cases when the house doesn't have the garrage so replacing null with No Garrage
surp['GarageType'] = surp['GarageType'].replace(np.nan, 'No Garage')
surp['GarageFinish'] = surp['GarageFinish'].replace(np.nan, 'No Garage')
surp['GarageCond'] = surp['GarageCond'].replace(np.nan, 'No Garage')
surp['GarageQual'] = surp['GarageQual'].replace(np.nan, 'No Garage')
surp['MasVnrType'] = surp['MasVnrType'].replace(np.nan, 'None') # replacing nan with the top option of this field
surp['Electrical'] = surp['Electrical'].replace(np.nan, 'SBrkr') # replacing nan with the top option of this field
```

In [10]:

```
#changing num to categorical so as to form these as dummy variables
surp['MSSubClass']=surp['MSSubClass'].replace({20:'1-STORY 1946 & NEWER ALL STYLES',30:'1-STORY 1945 & OLDER',40:'1-STORY W/FINISHED ATTIC ALL AGES',
```

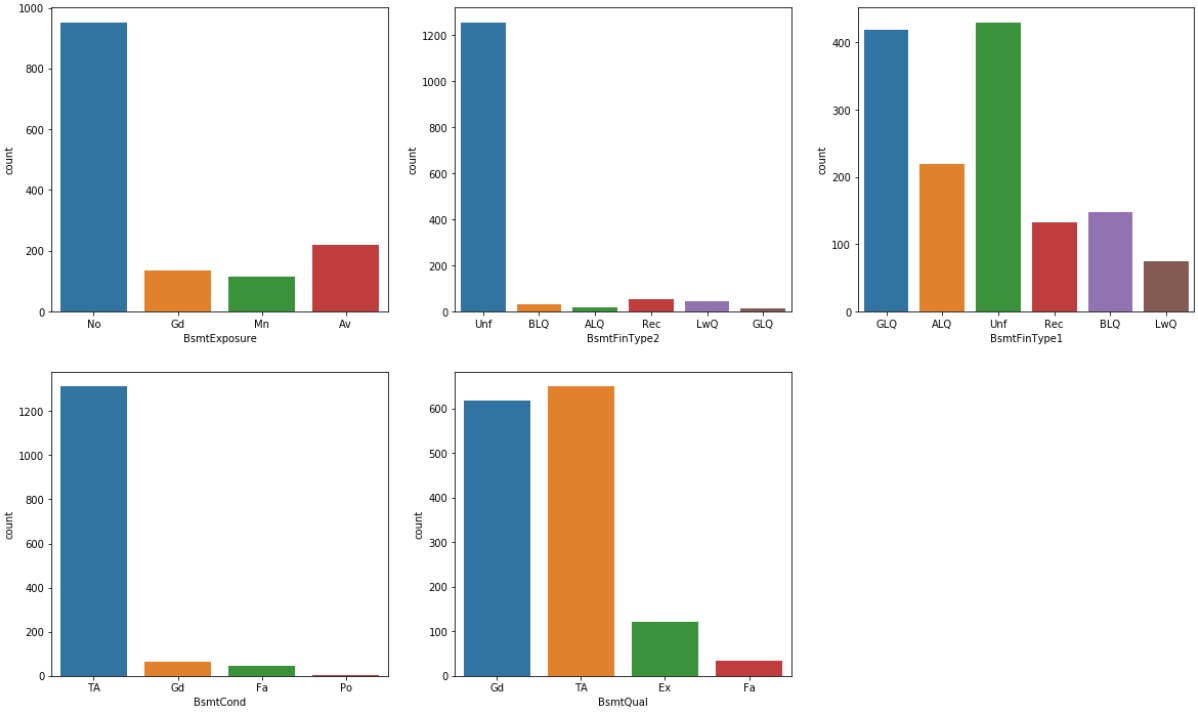
```
    45:'1-1/2 STORY - UNFINISHED ALL AGES',      50:'1-1/2 STORY FINISHED ALL AGES',      60:'2-
STORY 1946 & NEWER',
    70:'2-STORY 1945 & OLDER',
    75:'2-1/2 STORY ALL AGES',
    80:'SPLIT OR MULTI-LEVEL',
    85:'SPLIT FOYER',
    90:'DUPLEX - ALL STYLES AND AGES',
   120:'1-STORY PUD (Planned Unit Development) - 1946 & NEWER',
   150:'1-1/2 STORY PUD - ALL AGES',
   160:'2-STORY PUD - 1946 & NEWER',
   180:'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER',
   190:'2 FAMILY CONVERSION - ALL STYLES AND AGES'})

surp['OverallQual']=surp['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'})
surp['OverallCond']=surp['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]:

```
surp.head()
```

In [12]:

```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'BsmtExposure', data = surp)
plt.subplot(2,3,2)
sns.countplot(x = 'BsmtFinType2',  data = surp)
plt.subplot(2,3,3)
sns.countplot(x = 'BsmtFinType1', data = surp)
plt.subplot(2,3,4)
sns.countplot(x = 'BsmtCond', data = surp)
plt.subplot(2,3,5)
sns.countplot(x = 'BsmtQual',  data = surp)
plt.show()
```



In [13]:

```
# Cases when the house doesn't have the basement so replacing null with No Basement
surp['BsmtExposure'] = surp['BsmtExposure'].replace(np.nan, 'No Basement')
surp['BsmtFinType2'] = surp['BsmtFinType2'].replace(np.nan, 'No Basement')
surp['BsmtFinType1'] = surp['BsmtFinType1'].replace(np.nan, 'No Basement')
surp['BsmtCond'] = surp['BsmtCond'].replace(np.nan, 'No Basement')
surp['BsmtQual'] = surp['BsmtQual'].replace(np.nan, 'No Basement')
```

In [14]:

```
#check missing percentage
percent_missing = (surp.isnull().sum() / surp.isnull().count()*100).sort_values(ascending = False)
percent_missing.head(10)
```

Out[14]:

```
SalePrice      0.0
ExterCond      0.0
RoofStyle      0.0
RoofMatl       0.0
Exterior1st    0.0
Exterior2nd    0.0
MasVnrType     0.0
MasVnrArea     0.0
ExterQual      0.0
Foundation     0.0
dtype: float64
```

All the missing values has been treated

In [15]:

```
#Let's check the dependent variable i.e SalePrice
#descriptive statistics summary
surp['SalePrice'].describe()
```

Out[15]:

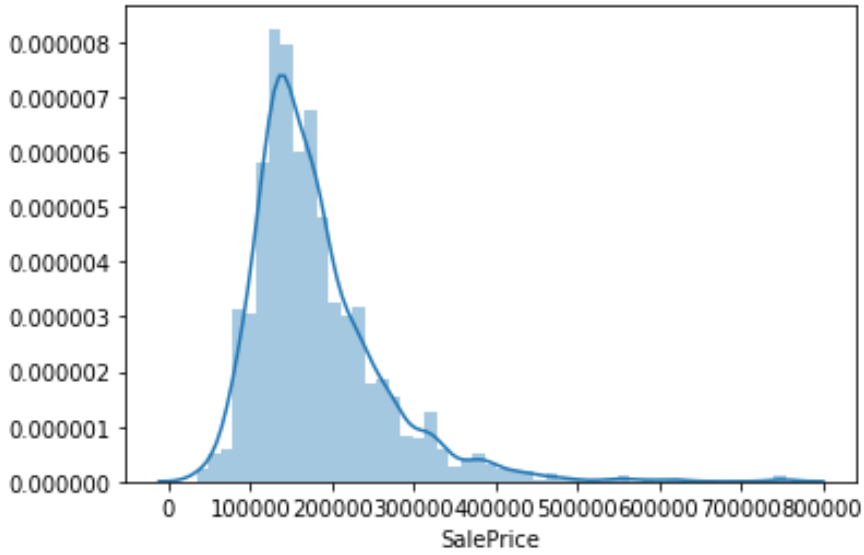
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [16]:

```
sns.distplot(surp['SalePrice']) #it's skewed
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbb5756860>
```



In [17]:

```
#skewness and kurtosis
print("Skewness: %f" % surp['SalePrice'].skew())
Skewness: 1.882876
```

In [18]:

```
surp['SalePrice']=np.log(surp.SalePrice) #tranforming to form normal disribution
```

In [19]:

```
surp['SalePrice'].describe()
```

Out[19]:

```
count    1460.000000
mean       12.024051
```

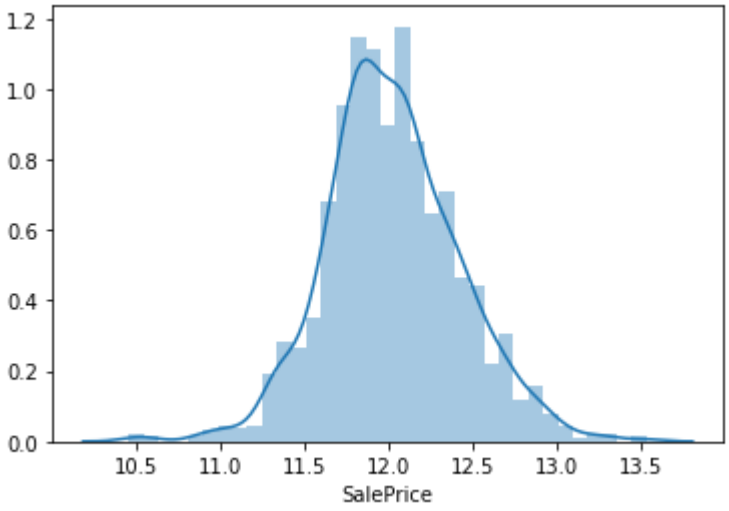
```
std          0.399452
min         10.460242
25%         11.775097
50%         12.001505
75%         12.273731
max         13.534473
Name: SalePrice, dtype: float64
```

In [20]:

```
sns.distplot(surp['SalePrice']) # Normally distributed now
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbb586ec50>
```



In [21]:

```
#skewness
print("Skewness: %f" % surp['SalePrice'].skew())
Skewness: 0.121335
```

We can say that now Dependent variable SalePrice is normally distributed.

EDA

Let's check all the Categorical columns and their effect on price

In [22]:

```
Cat = surp.select_dtypes(include=['object'])
Cat.columns
```

In [24]:

```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'KitchenQual', data = surp)
plt.subplot(2,3,2)
```

```
sns.countplot(x = 'Functional', data = surp)plt.subplot(2,3,3)sns.countplot(x = 'OverallQual', data = surp)
plt.xticks(rotation=90)plt.subplot(2,3,4)
sns.countplot(x = 'OverallCond', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,3,5)
sns.countplot(x = 'MSSubClass',  data = surp)
plt.xticks(rotation=90)
plt.show()
```

In [25]:

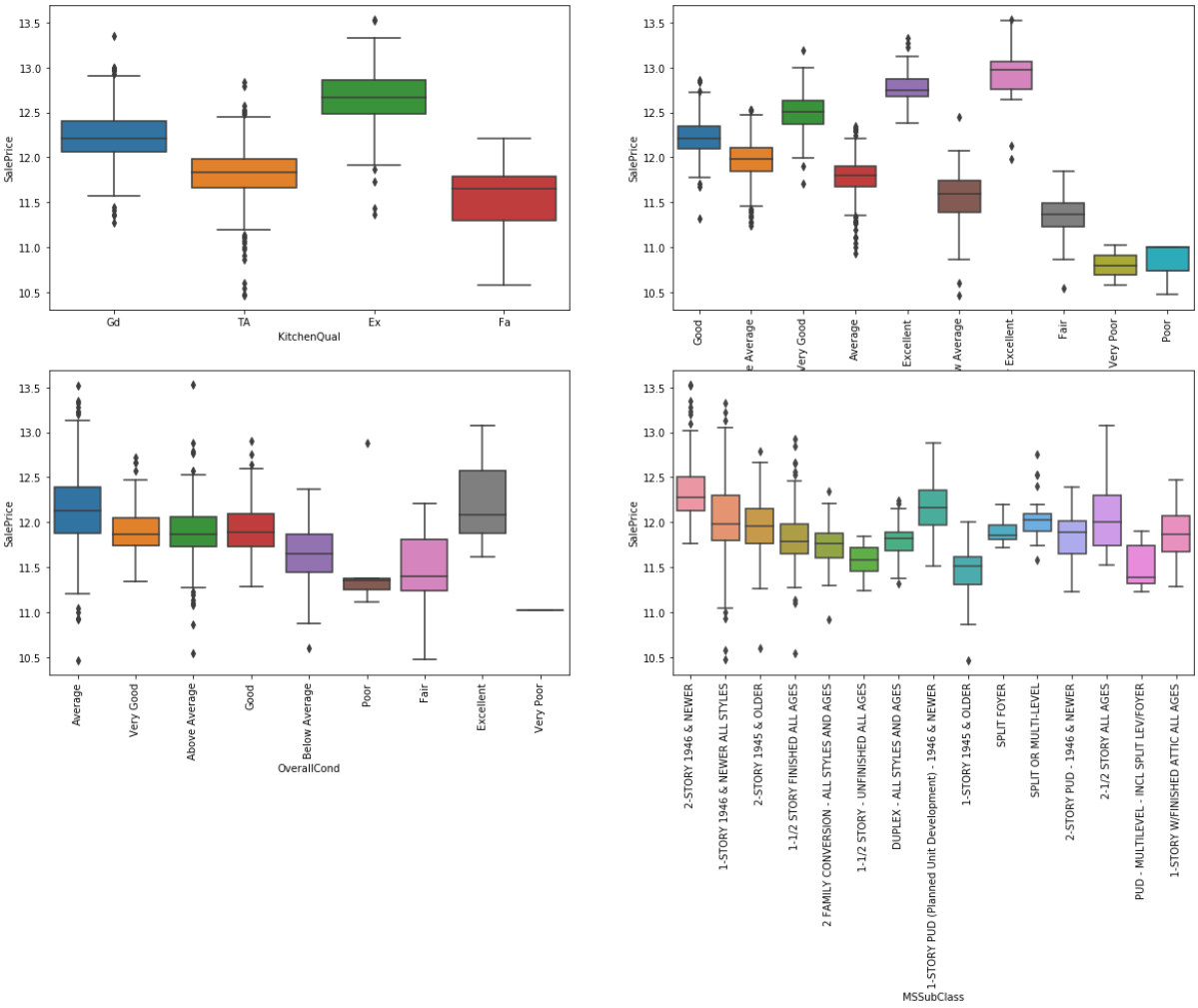
```
surp['MSSubClass'].value_counts() # similarly checking other columns for skewness
```

In [26]:

```
# Dropping highly skewed column
surp.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 = surp)
plt.subplot(2,2,2)
sns.boxplot(x = 'OverallQual', y = 'SalePrice', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,2,3)
sns.boxplot(x = 'OverallCond', y = 'SalePrice', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,2,4)
sns.boxplot(x = 'MSSubClass', y = 'SalePrice', data = surp)
plt.xticks(rotation=90)
plt.show()
```



In [28]:

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

In [29]:

```
surp['SaleCondition'].value_counts() #similarly checking other columns for skewness
```

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

In [30]:

```
surp['GarageType'] = surp['GarageType'].replace(['Basment','CarPort','2Types'],'Others')
surp['SaleCondition'] = surp['SaleCondition'].replace(['Family','Alloca','AdjLand'],'Others')
# Dropping highly skewed column
surp.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 = surp)
plt.subplot(1,2,2)
sns.boxplot(x = 'GarageFinish', y = 'SalePrice', data = surp)
```

- Price of Builtin Garagetype and Finished garage is the highest

.....

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

In [32]:

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

```
surp['HeatingQC'].value_counts() # similarly checked for all variables to check the skewness
```

In [34]:

```
surp['Foundation'] = surp['Foundation'].replace(['Slab','Stone','Wood'],'Others')
# Dropping highly skewed column
surp.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 = surp)
plt.subplot(1,3,2)
sns.boxplot(x = 'Foundation', y = 'SalePrice', data = surp)
plt.subplot(1,3,3)
sns.boxplot(x = 'HeatingQC', y = 'SalePrice', data = surp)
```

- 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 = surp)
plt.subplot(2,3,2)
sns.countplot(x = 'RoofStyle', data = surp)
plt.subplot(2,3,3)
sns.countplot(x = 'RoofMatl', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,3,4)
sns.countplot(x = 'Exterior1st', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,3,5)
sns.countplot(x = 'Exterior2nd',  data = surp)
plt.xticks(rotation=90)
plt.subplot(2,3,6)
sns.countplot(x = 'MasVnrType', data = surp)
plt.show()
```

In [37]:

```
surp['Exterior2nd'].value_counts() #similarly checking the skewness for other columns
```

In [38]:

```
surp['HouseStyle'] = surp['HouseStyle'].replace(['SFoyer','1.5Unf','2.5Unf','2.5Fin'],'Others')
surp['RoofStyle'] = surp['RoofStyle'].replace(['Shed','Mansard','Gambrel','Flat'],'Others')
surp['Exterior1st'] =
surp['Exterior1st'].replace(['AsphShn','ImStucc','CBlock','Stone','BrkComm','AsbShng','Stucco','WdShing'],
'Others')
surp['Exterior2nd'] =
surp['Exterior2nd'].replace(['Other','AsphShn','ImStucc','CBlock','Stone','BrkComm','AsbShng','Stucco','Br
kFace'],'Others')
# Dropping highly skewed column
surp.drop(['RoofMatl'],axis=1,inplace=True)
```

In [39]:

```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.boxplot(x = 'HouseStyle', y ='SalePrice', data = surp)
```

```
plt.subplot(2,3,2)
sns.boxplot(x = 'RoofStyle', y = 'SalePrice', data = surp)
plt.subplot(2,3,3)
sns.boxplot(x = 'MasVnrType', y = 'SalePrice', data = surp)
plt.subplot(2,3,4)
sns.boxplot(x = 'Exterior1st', y = 'SalePrice', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,3,5)
sns.boxplot(x = 'Exterior2nd', y = 'SalePrice', data = surp)
plt.xticks(rotation=90)
plt.show()
```

In [40]:

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

In [41]:

```
surp['BldgType'].value_counts()#similarly checking skewness for other columns
```

In [42]:

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

In [43]:

```
plt.figure(figsize=(20, 12))
plt.subplot(2,1,1)
sns.boxplot(x = 'Neighborhood', y ='SalePrice', data = surp)
plt.xticks(rotation=90)
plt.subplot(2,1,2)
sns.boxplot(x = 'BldgType', y = 'SalePrice', data = surp)
```

In [44]:

```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.countplot(x = 'LandContour',  data = surp)
plt.subplot(2,3,2)
```

```
sns.countplot(x = 'LandSlope', data = surp)
plt.subplot(2,3,3)
sns.countplot(x = 'LotShape', data = surp)
plt.subplot(2,3,4)
sns.countplot(x = 'Electrical', data = surp)
plt.subplot(2,3,5)
sns.countplot(x = 'MSZoning',  data = surp)
plt.subplot(2,3,6)
sns.countplot(x = 'LotConfig', data = surp)
plt.show()
```

In [45]:

```
surp['LotConfig'].value_counts()
```

In [46]:

```
surp.drop(['LandSlope','LandContour','Electrical'],axis=1,inplace=True)
surp['MSZoning'] = surp['MSZoning'].replace(['RH','C (all)'],'Others')
```

In [47]:

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

In [48]:

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

In [49]:

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

In [50]:

```
surp.head()
```

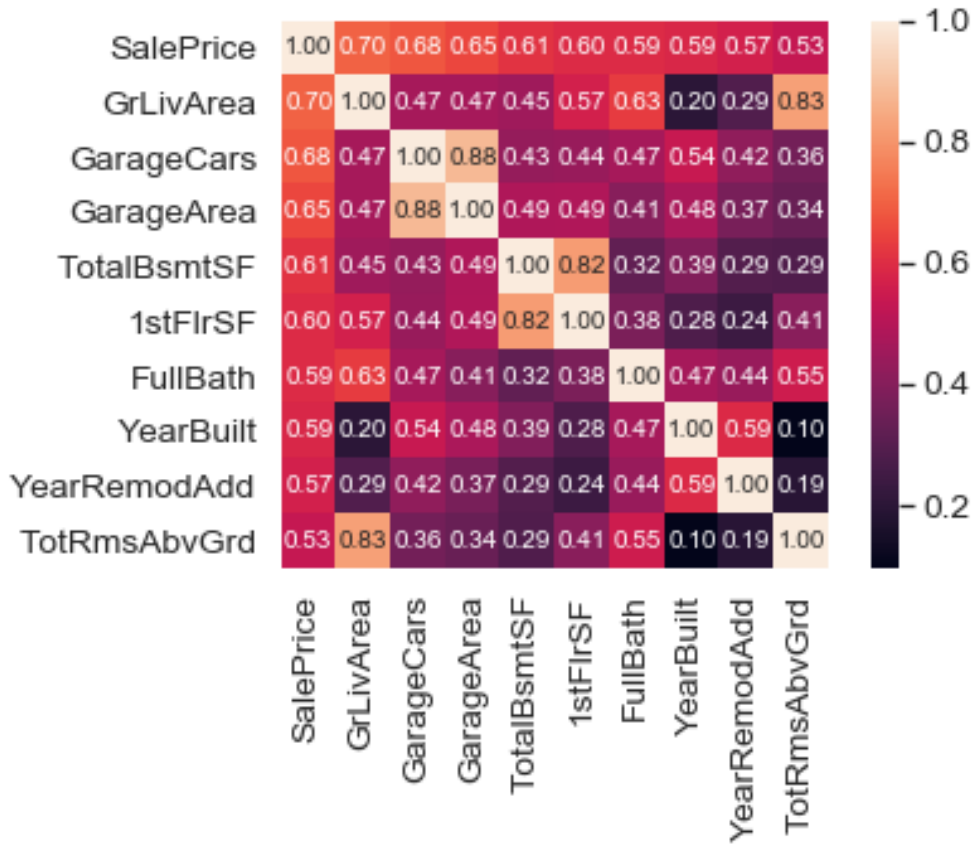
In [51]:

```
surp.info()
```

In [52]:

```
#saleprice correlation matrix
corrmat = surp.corr()
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
```

```
cm = np.corrcoef(surp[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=cols.values)
plt.show()
```



In [53]:

```
surp['YearSinceRemodel'] = 2010 - ((surp['YearRemodAdd'] - surp['YearBuilt']) + surp['YearBuilt'])
#feature engineering
```

In [54]:

```
Cat1 = surp.select_dtypes(include=['object']) #checking all categorical columns to form dummy variables
Cat1.columns
```

In [55]:

```
Num = surp.select_dtypes(include=['int64','float64']) #all numerical variables
Num.columns
```

In [56]:

```
Num.info()
```

In [57]:

```
Cat1 = pd.get_dummies(Cat1,drop_first=True)  # Dummy variables
print(Cat1.shape)
```

```
(1460, 130)
```

In [59]:

```
# concat dummy variables with main dataset
surp = pd.concat([surp, Cat1], axis=1)
```

In [61]:

```
surp.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 already formed
```

In [63]:

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

In [65]:

```
# Putting feature variable to X
X = surp.drop(['SalePrice'], axis=1)
```

In [66]:

```
# Putting response variable to y
y = surp['SalePrice']
```

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', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
    '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
    'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
    'TotRmsAbvGrd', 'Fireplaces','GarageCars', 'GarageArea',
    'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
    'ScreenPorch', 'PoolArea', 'MiscVal']])
```

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]:

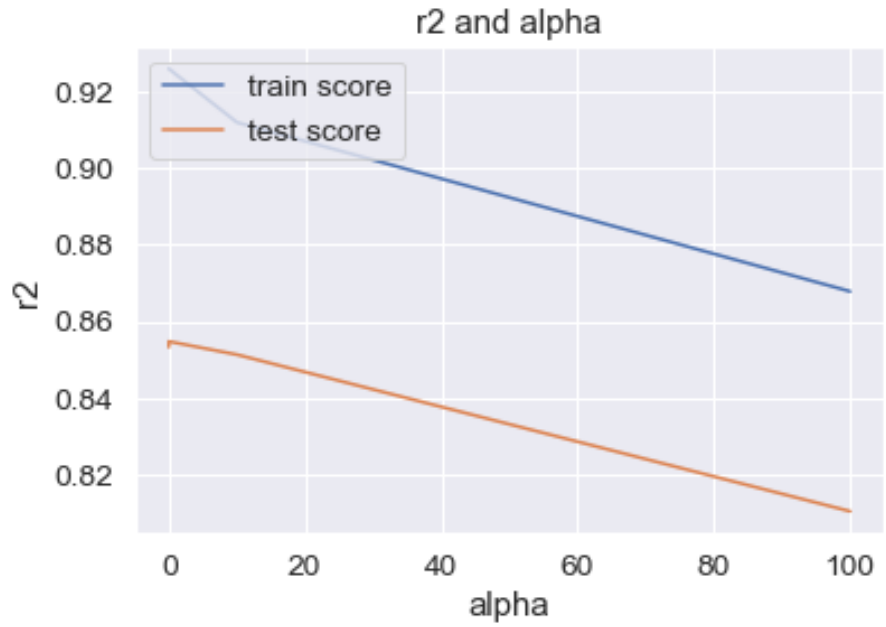
```
# list of alphas to tune
params = {'alpha': [0.00004,0.0001, 0.001, 0.01, 0.1, 10,100,1000]}
ridge = Ridge()
# cross validation
folds = 5
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'r2',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(X_train, y_train)
```

In [71]:

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

In [72]:

```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')
# 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 [73]:

```
#optimum alpha
alpha = 10
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)
ridge.coef_
```

In [74]:

```
ridge.score(X_train,y_train)
```

In [75]:

```
ridge.score(X_test,y_test)
```

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))
```

Lasso

In [77]:

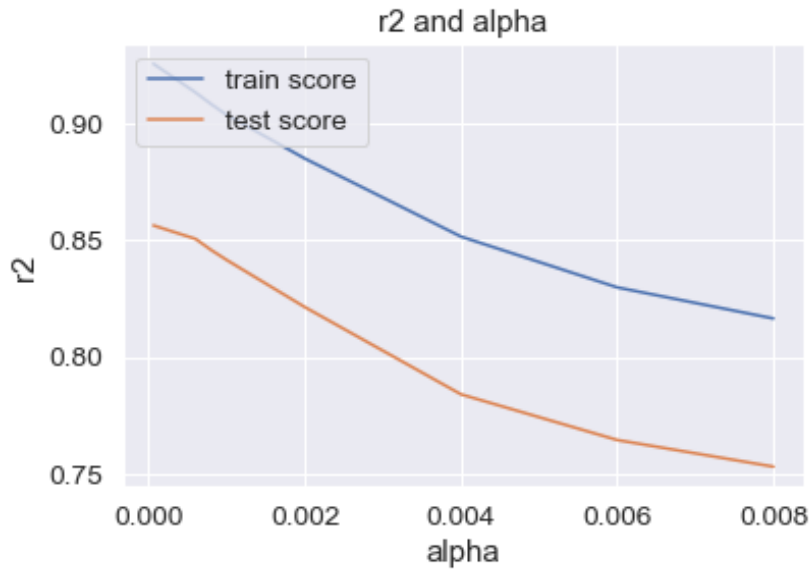
```
params = {'alpha': [0.00006,0.0006, 0.0008, 0.001, 0.002, 0.004, 0.006, 0.008 ]}
lasso = Lasso()
# cross validation
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'r2',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(X_train, y_train)
```

In [78]:

```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

In [79]:

```
# plotting mean test and train 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)
```

In [81]:

```
lasso.coef_
```

In [82]:

```
lasso.score(X_train,y_train)
```

Out[82]:

```
0.898288939025357
```

In [83]:

```
lasso.score(X_test,y_test)
```

Out[83]:

```
0.8646575331441892
```

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)))
```

Conclusion

The predictive models developed through Ridge and Lasso regression provide valuable insights for Surprise Housing's entry into the Australian housing market. Both models achieved satisfactory prediction accuracy, with Ridge slightly outperforming Lasso in terms of training accuracy. However, Lasso regression emerged as the preferred model due to its superior feature selection capabilities. The top five most significant variables identified by both models were SaleCondition, GarageFinish, and OverallQual. The optimal values of lambda for Ridge and Lasso regression were found to be 10 and 0.001, respectively. Continuous monitoring and adaptation of the model will be crucial for sustained success in the dynamic real estate landscape. These findings provide valuable insights for Surprise Housing in formulating strategies to purchase houses at competitive prices and maximize profit margins.