Advanced Regression

Surprise Housing Case Study

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them on at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia.

Problem Statement

The company is looking at prospective properties to buy to enter the market. It is required to build a regression model using regularisation in order to predict the actual value of the prospective properties and decide whether to invest in them or not.

The company wants to know:

- Which variables are significant in predicting the price of a house, and

- How well those variables describe the price of a house.

Also, determine the optimal value of lambda for ridge and lasso regression.

Business Goal

To model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for management to understand the pricing dynamics of a new market.

The solution is divided into the following sections:

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

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
import datetime
from sklearn import linear_model, metrics
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.metrics import mean_squared_error, r2_score

import os

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

pd.set_option('display.max_rows', 500)
```

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

Out[2]:

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

5 rows × 81 columns

In [3]:
```
# check the shape

df.shape
```

Out[3]:
```
(1460, 81)
```

The dataset has 1460 rows and 81 columns.

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

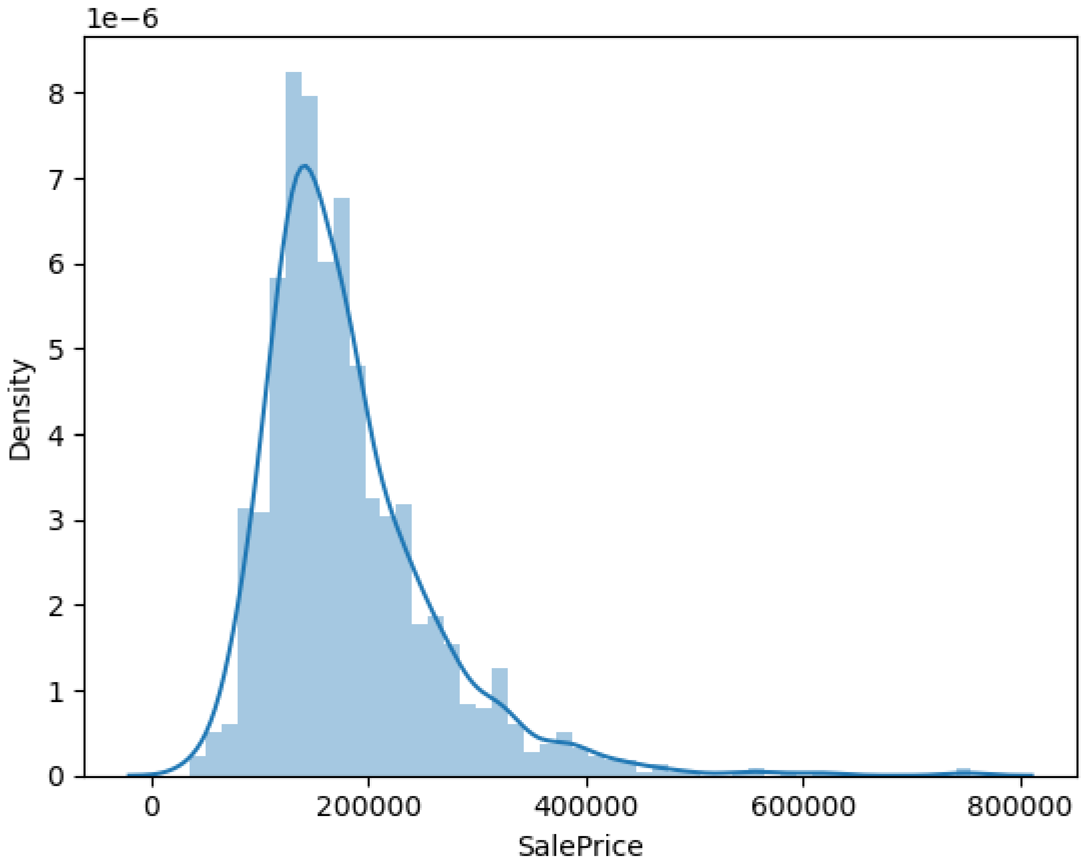
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       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 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
```

```
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

- This shows that there are null values present.
- `Alley` , `PoolQC` , `MiscFeature` and `Fence` have very less non-null values.
- Here target variable is `SalePrice`

In [5]:

```
# target variable: SalePrice

sns.distplot(df['SalePrice'])
plt.show()
```



2. Data Cleaning

Let's now conduct some data cleaning steps.

In [6]:
```
# duplicacy check

df["Id"].is_unique
```

Out[6]:
```
True
```

This means that no two Ids are same, hence we have all rows unique.

Let us check the percentage of null values.

In [7]:
```
print(round(df.isnull().sum()/len(df.index)*100,2))
```

```
Id                0.00
MSSubClass        0.00
MSZoning          0.00
LotFrontage      17.74
LotArea           0.00
Street            0.00
Alley            93.77
LotShape          0.00
LandContour       0.00
Utilities         0.00
LotConfig         0.00
LandSlope         0.00
Neighborhood      0.00
Condition1        0.00
Condition2        0.00
BldgType          0.00
HouseStyle        0.00
OverallQual       0.00
OverallCond       0.00
YearBuilt         0.00
YearRemodAdd      0.00
RoofStyle         0.00
RoofMatl          0.00
Exterior1st       0.00
Exterior2nd       0.00
MasVnrType        0.55
MasVnrArea        0.55
ExterQual         0.00
ExterCond         0.00
Foundation        0.00
BsmtQual          2.53
BsmtCond          2.53
BsmtExposure      2.60
BsmtFinType1      2.53
BsmtFinSF1        0.00
BsmtFinType2      2.60
BsmtFinSF2        0.00
BsmtUnfSF         0.00
TotalBsmtSF       0.00
Heating           0.00
HeatingQC         0.00
CentralAir        0.00
Electrical        0.07
1stFlrSF          0.00
2ndFlrSF          0.00
LowQualFinSF      0.00
GrLivArea         0.00
BsmtFullBath      0.00
BsmtHalfBath      0.00
FullBath          0.00
HalfBath          0.00
BedroomAbvGr      0.00
KitchenAbvGr      0.00
KitchenQual       0.00
TotRmsAbvGrd      0.00
Functional        0.00
Fireplaces        0.00
FireplaceQu      47.26
GarageType        5.55
GarageYrBlt       5.55
```

```
GarageFinish      5.55
GarageCars        0.00
GarageArea        0.00
GarageQual        5.55
GarageCond        5.55
PavedDrive        0.00
WoodDeckSF        0.00
OpenPorchSF       0.00
EnclosedPorch     0.00
3SsnPorch         0.00
ScreenPorch       0.00
PoolArea          0.00
PoolQC           99.52
Fence            80.75
MiscFeature      96.30
MiscVal           0.00
MoSold            0.00
YrSold            0.00
SaleType          0.00
SaleCondition     0.00
SalePrice         0.00
dtype: float64
```

We have seen in the data description that there are few 'NA' values present. These are not null values rather they are 'not available' values. For e.g. NA in garage variables mean that there is no garage present in that house.

So we need to impute these wherever present so that it is not counted as null values.

We have seen that Alley, PoolQC, MiscFeature and Fence have more than 80% missing values. Even if we impute the NA with some other categorical value, that will become the dominating value. Since the percentage is so high, we can drop these columns.

In [8]:
```
df = df.drop( columns = ['Alley','PoolQC','MiscFeature','Fence'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 77 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   LotShape       1460 non-null   object 
 7   LandContour    1460 non-null   object 
 8   Utilities      1460 non-null   object 
 9   LotConfig      1460 non-null   object 
 10  LandSlope      1460 non-null   object 
 11  Neighborhood   1460 non-null   object 
 12  Condition1     1460 non-null   object 
 13  Condition2     1460 non-null   object 
 14  BldgType       1460 non-null   object 
 15  HouseStyle     1460 non-null   object 
 16  OverallQual    1460 non-null   int64  
 17  OverallCond    1460 non-null   int64  
 18  YearBuilt      1460 non-null   int64  
 19  YearRemodAdd   1460 non-null   int64  
 20  RoofStyle      1460 non-null   object 
 21  RoofMatl       1460 non-null   object 
 22  Exterior1st    1460 non-null   object 
 23  Exterior2nd    1460 non-null   object 
 24  MasVnrType     1452 non-null   object 
 25  MasVnrArea     1452 non-null   float64
 26  ExterQual      1460 non-null   object 
 27  ExterCond      1460 non-null   object 
 28  Foundation     1460 non-null   object 
 29  BsmtQual       1423 non-null   object 
 30  BsmtCond       1423 non-null   object 
 31  BsmtExposure   1422 non-null   object 
 32  BsmtFinType1   1423 non-null   object 
 33  BsmtFinSF1     1460 non-null   int64  
 34  BsmtFinType2   1422 non-null   object 
 35  BsmtFinSF2     1460 non-null   int64  
 36  BsmtUnfSF      1460 non-null   int64  
 37  TotalBsmtSF    1460 non-null   int64  
 38  Heating        1460 non-null   object 
 39  HeatingQC      1460 non-null   object 
 40  CentralAir     1460 non-null   object 
 41  Electrical     1459 non-null   object 
 42  1stFlrSF       1460 non-null   int64  
 43  2ndFlrSF       1460 non-null   int64  
 44  LowQualFinSF   1460 non-null   int64  
 45  GrLivArea      1460 non-null   int64  
 46  BsmtFullBath   1460 non-null   int64  
 47  BsmtHalfBath   1460 non-null   int64  
 48  FullBath       1460 non-null   int64  
 49  HalfBath       1460 non-null   int64  
 50  BedroomAbvGr   1460 non-null   int64  
 51  KitchenAbvGr   1460 non-null   int64  
 52  KitchenQual    1460 non-null   object 
 53  TotRmsAbvGrd   1460 non-null   int64  
 54  Functional     1460 non-null   object 
```

```
 55  Fireplaces     1460 non-null   int64
 56  FireplaceQu    770 non-null    object
 57  GarageType     1379 non-null   object
 58  GarageYrBlt    1379 non-null   float64
 59  GarageFinish   1379 non-null   object
 60  GarageCars     1460 non-null   int64
 61  GarageArea     1460 non-null   int64
 62  GarageQual     1379 non-null   object
 63  GarageCond     1379 non-null   object
 64  PavedDrive     1460 non-null   object
 65  WoodDeckSF     1460 non-null   int64
 66  OpenPorchSF    1460 non-null   int64
 67  EnclosedPorch  1460 non-null   int64
 68  3SsnPorch      1460 non-null   int64
 69  ScreenPorch    1460 non-null   int64
 70  PoolArea       1460 non-null   int64
 71  MiscVal        1460 non-null   int64
 72  MoSold         1460 non-null   int64
 73  YrSold         1460 non-null   int64
 74  SaleType       1460 non-null   object
 75  SaleCondition  1460 non-null   object
 76  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(39)
memory usage: 878.4+ KB
```

In [9]:
```
df.shape
```

Out[9]:
```
(1460, 77)
```

Now we have 77 columns. Lets start imputing the NA values now one by one.

Garage

Lets talk about Garage related variables i.e. `GarageType`, `GarageFinish`, `GarageQual` and `GarageCond`.

In [10]:
```
print('#NA in Garage Type :',df.GarageType.isnull().sum())
print('#NA in Garage Finish :',df.GarageFinish.isnull().sum())
print('#NA in Garage Qual :',df.GarageQual.isnull().sum())
print('#NA in Garage Cond :',df.GarageCond.isnull().sum())
```

```
#NA in Garage Type : 81
#NA in Garage Finish : 81
#NA in Garage Qual : 81
#NA in Garage Cond : 81
```

This means that 81 houses don't have garage. We can impute these with some other value like NoGarage.

In [11]:
```
columns = ['GarageType', 'GarageFinish', 'GarageQual','GarageCond']
for col in columns:
    df[col].fillna('NoGarage',inplace=True)
```

In [12]:
```
print('#NA in Garage Type :',df.GarageType.isnull().sum())
print('#NA in Garage Finish :',df.GarageFinish.isnull().sum())
print('#NA in Garage Qual :',df.GarageQual.isnull().sum())
print('#NA in Garage Cond :',df.GarageCond.isnull().sum())
```

```
#NA in Garage Type : 0
#NA in Garage Finish : 0
#NA in Garage Qual : 0
#NA in Garage Cond : 0
```

Basement

Similarily we have to do the same thing with Basement.
Basement related variables are : `BsmtQual` , `BsmtCond` , `BsmtExposure` , `BsmtFinType1` , `BsmtFinType2`

In [13]:
```
print('#NA in BsmtQual :',df.BsmtQual.isnull().sum())
print('#NA in BsmtCond :',df.BsmtCond.isnull().sum())
print('#NA in BsmtExposure :',df.BsmtExposure.isnull().sum())
print('#NA in BsmtFinType1 :',df.BsmtFinType1.isnull().sum())
print('#NA in BsmtFinType2 :',df.BsmtFinType2.isnull().sum())
```

```
#NA in BsmtQual : 37
#NA in BsmtCond : 37
#NA in BsmtExposure : 38
#NA in BsmtFinType1 : 37
#NA in BsmtFinType2 : 38
```

In [14]:
```
bsmt_columns = ['BsmtQual', 'BsmtCond', 'BsmtExposure','BsmtFinType1','BsmtFinType2']
for col in bsmt_columns:
    df[col].fillna('NoBasement',inplace=True)
```

GarageYrBlt

Now, lets check `GarageYrBlt` as it has around 5.5% NA values.

In [15]:
```
df.GarageYrBlt.describe()
```

Out[15]:
```
count    1379.000000
mean     1978.506164
std        24.689725
min      1900.000000
25%      1961.000000
50%      1980.000000
75%      2002.000000
max      2010.000000
Name: GarageYrBlt, dtype: float64
```

In [16]:
```
df.GarageYrBlt.isnull().sum()
```

Out[16]:
```
81
```

In [17]:
```
df['GarageYrBlt'].fillna(df.GarageYrBlt.median(),inplace=True)
```

In [18]:
```
df.GarageYrBlt.isnull().sum()
```

Out[18]:
```
0
```

FireplaceQu

we can see `FireplaceQu` has 47% null values. So lets check the value_counts for that.

In [19]:
```
df.FireplaceQu.value_counts()
```

Out[19]:
```
Gd    380
TA    313
Fa     33
Ex     24
Po     20
Name: FireplaceQu, dtype: int64
```

In [20]:
```
df.FireplaceQu.isnull().sum()
```

Out[20]:
```
690
```

690 out of 1460 values are NA in this and NA means No fireplace according to the data description. So lets impute this.

In [21]:
```
df['FireplaceQu'].fillna('NoFireplace',inplace=True)
```

In [22]:
```
df.FireplaceQu.value_counts()
```

Out[22]:
```
NoFireplace    690
Gd             380
TA             313
Fa              33
Ex              24
Po              20
Name: FireplaceQu, dtype: int64
```

LotFrontage

Now, lets check `LotFrontage` as it has 17% NA values.

In [23]:
```
df.LotFrontage.describe()
```

Out[23]:
```
count    1201.000000
mean       70.049958
std        24.284752
min        21.000000
25%        59.000000
50%        69.000000
75%        80.000000
max       313.000000
Name: LotFrontage, dtype: float64
```

In [24]:
```
df['LotFrontage'].isnull().sum()
```

Out[24]:
```
259
```

In [25]:
```
df['LotFrontage'].fillna(df.LotFrontage.median(),inplace=True)
```

In [26]:
```
df['LotFrontage'].isnull().sum()
```

Out[26]:
```
0
```

MasVnrType

In [27]:
```
df.MasVnrType.isnull().sum()
```
Out[27]:
```
8
```

In [28]:
```
df.MasVnrType.value_counts()
```
Out[28]:
```
None       864
BrkFace    445
Stone      128
BrkCmn      15
Name: MasVnrType, dtype: int64
```

In [29]:
```
df.MasVnrType.fillna(df['MasVnrType'].mode()[0],inplace=True)
```

In [30]:
```
df.MasVnrType.isnull().sum()
```
Out[30]:
```
0
```

MasVnrArea

In [31]:
```
print("#Null in MasVnrArea :",df.MasVnrArea.isnull().sum())
```
```
#Null in MasVnrArea : 8
```

In [32]:
```
df['MasVnrArea'].fillna(df.MasVnrArea.median(),inplace=True)
```

In [33]:
```
df.MasVnrArea.isnull().sum()
```
Out[33]:
```
0
```

Electrical

In [34]:
```
print("#Null in Electrical :",df.Electrical.isnull().sum())
```
```
#Null in Electrical : 1
```

In [35]:
```
df.Electrical.value_counts()
```
Out[35]:
```
SBrkr    1334
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64
```

In [36]:
```
df.Electrical.fillna(df['Electrical'].mode()[0],inplace=True)
```

In [37]:
```
df.Electrical.value_counts()
```

Out[37]:
```
SBrkr    1335
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64
```

In [38]:
```
#Let us check the percentage of null values again

print(round(df.isnull().sum()/len(df.index)*100,2))
```

```
Id               0.0
MSSubClass       0.0
MSZoning         0.0
LotFrontage      0.0
LotArea          0.0
Street           0.0
LotShape         0.0
LandContour      0.0
Utilities        0.0
LotConfig        0.0
LandSlope        0.0
Neighborhood     0.0
Condition1       0.0
Condition2       0.0
BldgType         0.0
HouseStyle       0.0
OverallQual      0.0
OverallCond      0.0
YearBuilt        0.0
YearRemodAdd     0.0
RoofStyle        0.0
RoofMatl         0.0
Exterior1st      0.0
Exterior2nd      0.0
MasVnrType       0.0
MasVnrArea       0.0
ExterQual        0.0
ExterCond        0.0
Foundation       0.0
BsmtQual         0.0
BsmtCond         0.0
BsmtExposure     0.0
BsmtFinType1     0.0
BsmtFinSF1       0.0
BsmtFinType2     0.0
BsmtFinSF2       0.0
BsmtUnfSF        0.0
TotalBsmtSF      0.0
Heating          0.0
HeatingQC        0.0
CentralAir       0.0
Electrical       0.0
1stFlrSF         0.0
2ndFlrSF         0.0
LowQualFinSF     0.0
GrLivArea        0.0
BsmtFullBath     0.0
BsmtHalfBath     0.0
FullBath         0.0
HalfBath         0.0
BedroomAbvGr     0.0
KitchenAbvGr     0.0
KitchenQual      0.0
TotRmsAbvGrd     0.0
Functional       0.0
Fireplaces       0.0
FireplaceQu      0.0
GarageType       0.0
GarageYrBlt      0.0
GarageFinish     0.0
```

```
GarageCars       0.0
GarageArea       0.0
GarageQual       0.0
GarageCond       0.0
PavedDrive       0.0
WoodDeckSF       0.0
OpenPorchSF      0.0
EnclosedPorch    0.0
3SsnPorch        0.0
ScreenPorch      0.0
PoolArea         0.0
MiscVal          0.0
MoSold           0.0
YrSold           0.0
SaleType         0.0
SaleCondition    0.0
SalePrice        0.0
dtype: float64
```

Alright, so all the NA values are now replaced and the unimportant columns are deleted. We have a clean dataset now.

In [39]:
```
# all numeric (float and int) variables in the dataset

df_numeric = df.select_dtypes(include=['float64', 'int64'])
df_numeric.head()
```

Out[39]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasV
0	1	60	65.0	8450	7	5	2003	2003	
1	2	20	80.0	9600	6	8	1976	1976	
2	3	60	68.0	11250	7	5	2001	2002	
3	4	70	60.0	9550	7	5	1915	1970	
4	5	60	84.0	14260	8	5	2000	2000	

5 rows × 38 columns

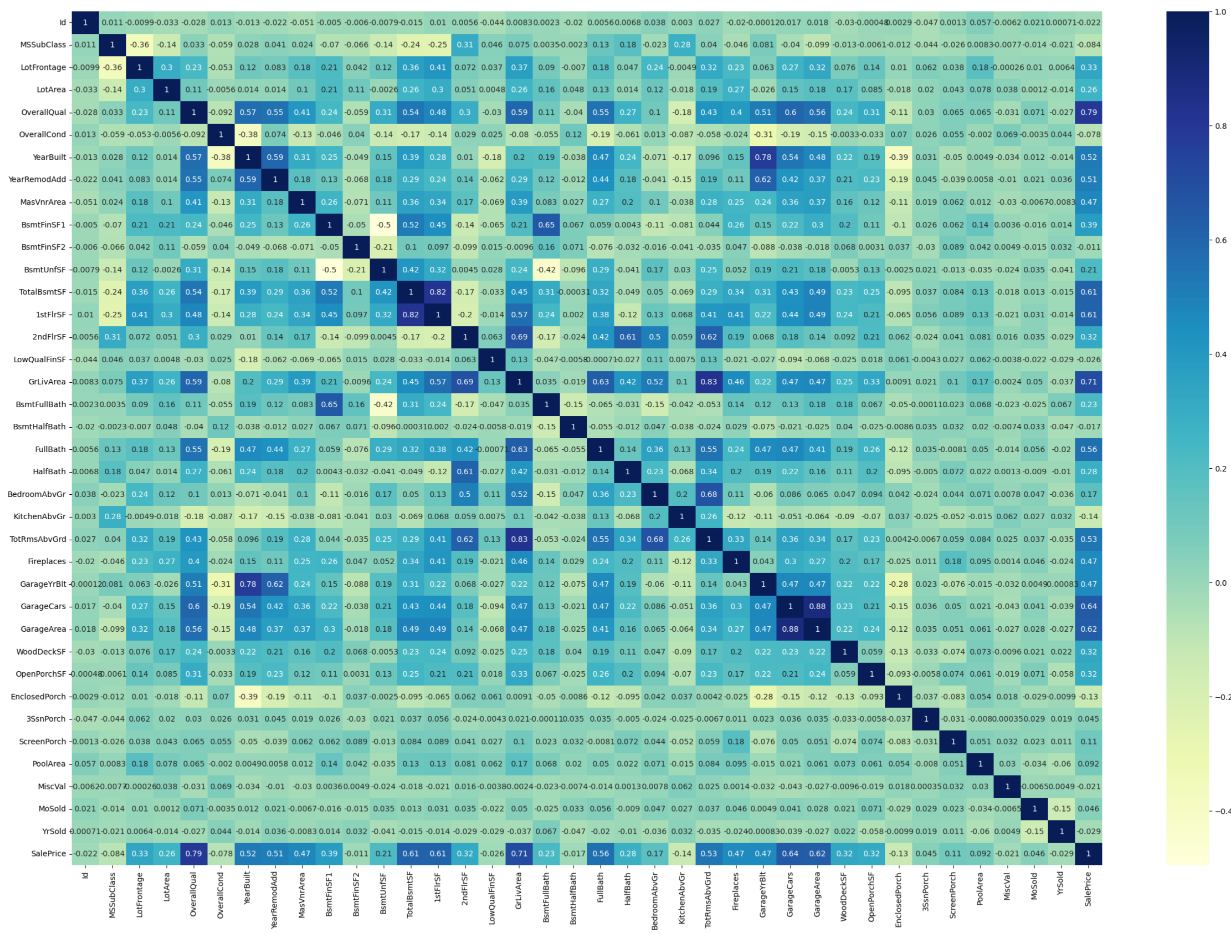
In [40]:
```
# correlation matrix
cor = df_numeric.corr()
cor
```

Out[40]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt
Id	1.000000	0.011156	-0.009921	-0.033226	-0.028365	0.012609	-0.012713
MSSubClass	0.011156	1.000000	-0.356718	-0.139781	0.032628	-0.059316	0.027850
LotFrontage	-0.009921	-0.356718	1.000000	0.304522	0.234812	-0.053281	0.116685
LotArea	-0.033226	-0.139781	0.304522	1.000000	0.105806	-0.005636	0.014228
OverallQual	-0.028365	0.032628	0.234812	0.105806	1.000000	-0.091932	0.572323
OverallCond	0.012609	-0.059316	-0.053281	-0.005636	-0.091932	1.000000	-0.375983
YearBuilt	-0.012713	0.027850	0.116685	0.014228	0.572323	-0.375983	1.000000
YearRemodAdd	-0.021998	0.040581	0.083348	0.013788	0.550684	0.073741	0.592855
MasVnrArea	-0.051071	0.023573	0.178469	0.103321	0.407252	-0.125694	0.311600
BsmtFinSF1	-0.005024	-0.069836	0.214367	0.214103	0.239666	-0.046231	0.249503
BsmtFinSF2	-0.005968	-0.065649	0.042463	0.111170	-0.059119	0.040229	-0.049107
BsmtUnfSF	-0.007940	-0.140759	0.124098	-0.002618	0.308159	-0.136841	0.149040
TotalBsmtSF	-0.015415	-0.238518	0.363472	0.260833	0.537808	-0.171098	0.391452
1stFlrSF	0.010496	-0.251758	0.413773	0.299475	0.476224	-0.144203	0.281986
2ndFlrSF	0.005590	0.307886	0.072388	0.050986	0.295493	0.028942	0.010308
LowQualFinSF	-0.044230	0.046474	0.037469	0.004779	-0.030429	0.025494	-0.183784
GrLivArea	0.008273	0.074853	0.368007	0.263116	0.593007	-0.079686	0.199010
BsmtFullBath	0.002289	0.003491	0.090343	0.158155	0.111098	-0.054942	0.187599
BsmtHalfBath	-0.020155	-0.002333	-0.006979	0.048046	-0.040150	0.117821	-0.038162
FullBath	0.005587	0.131608	0.180534	0.126031	0.550600	-0.194149	0.468271
HalfBath	0.006784	0.177354	0.047222	0.014259	0.273458	-0.060769	0.242656
BedroomAbvGr	0.037719	-0.023438	0.236840	0.119690	0.101676	0.012980	-0.070651
KitchenAbvGr	0.002951	0.281721	-0.004905	-0.017784	-0.183882	-0.087001	-0.174800
TotRmsAbvGrd	0.027239	0.040380	0.320518	0.190015	0.427452	-0.057583	0.095589
Fireplaces	-0.019772	-0.045569	0.233221	0.271364	0.396765	-0.023820	0.147716
GarageYrBlt	-0.000122	0.081396	0.062996	-0.025865	0.514231	-0.306276	0.777182
GarageCars	0.016570	-0.040110	0.269539	0.154871	0.600671	-0.185758	0.537850
GarageArea	0.017634	-0.098672	0.323511	0.180403	0.562022	-0.151521	0.478954
WoodDeckSF	-0.029643	-0.012579	0.075542	0.171698	0.238923	-0.003334	0.224880
OpenPorchSF	-0.000477	-0.006100	0.137014	0.084774	0.308819	-0.032589	0.188686
EnclosedPorch	0.002889	-0.012037	0.010287	-0.018340	-0.113937	0.070356	-0.387268
3SsnPorch	-0.046635	-0.043825	0.061945	0.020423	0.030371	0.025504	0.031355
ScreenPorch	0.001330	-0.026030	0.037655	0.043160	0.064886	0.054811	-0.050364

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt
PoolArea	0.057044	0.008283	0.180819	0.077672	0.065166	-0.001985	0.004950
MiscVal	-0.006242	-0.007683	-0.000255	0.038068	-0.031406	0.068777	-0.034383
MoSold	0.021172	-0.013585	0.010451	0.001205	0.070815	-0.003511	0.012398
YrSold	0.000712	-0.021407	0.006380	-0.014261	-0.027347	0.043950	-0.013618
SalePrice	-0.021917	-0.084284	0.334771	0.263843	0.790982	-0.077856	0.522897

In [41]:
```python
# plotting correlations on a heatmap

# figure size
plt.figure(figsize=(30,20))

# heatmap
sns.heatmap(cor, cmap="YlGnBu", annot=True)
plt.show()
```



If we concentarte on the last row, we can see the following variables are highly correlated with target variable `SalePrice`:

It is positively correlated with following variables:

- OverallQual

- YearBuilt
- YearRemodAdd
- MasVnrArea
- TotalBsmtSF
- 1stFlrSF
- GrLivArea
- FullBath
- TotRmsAbvGrd
- GarageCars
- GarageArea

It is negatively correlated with following variables:

- Id
- MSSubClass
- OverallCond
- LowQualFinSF
- BsmtHalfBath
- KitchenAbvGr
- EnclosedPorch
- MiscVal
- YrSold

There is some multicollinearity present as well:

- 1stFlrSF and TotalBsmtSF
- GarageCars and GarageArea
- GrLivArea and TotRmsAbvGrd

and so on..

We will keep these points in mind while building the model.

since finding the multicollinearity here will be little difficult. Lets try to find top variables which has a correlation greater than 80%.

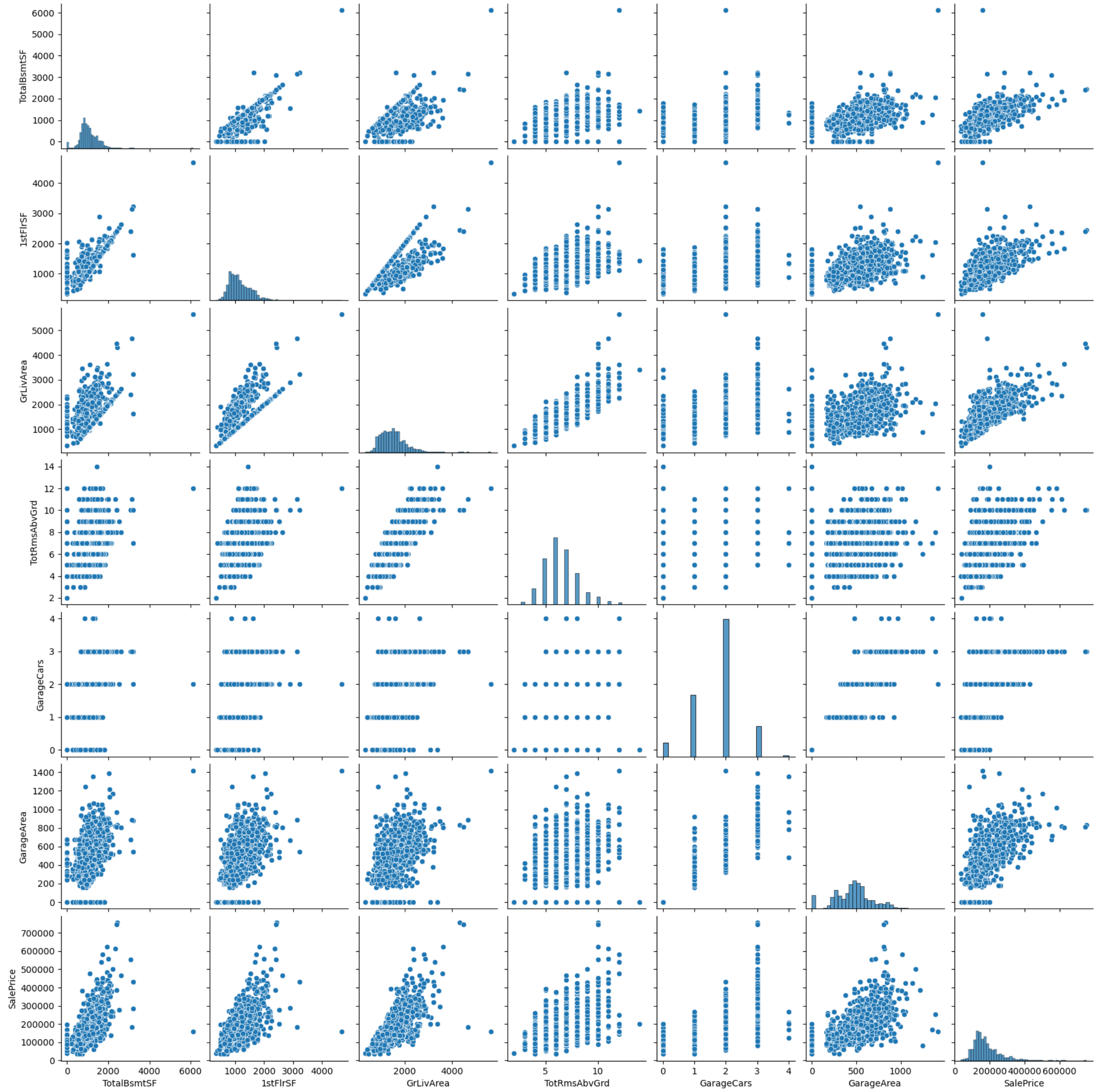
In [42]:

```
cor = cor.abs()
top_cor_variables = np.where(cor>0.8)
top_cor_variables = [(cor.columns[x],cor.columns[y]) for x,y in zip(*top_cor_variables
print(top_cor_variables)
```

```
[('TotalBsmtSF', '1stFlrSF'), ('GrLivArea', 'TotRmsAbvGrd'), ('GarageCars', 'GarageAr
ea')]
```

In [43]:

```
# lets do a pairplot to check the patterns
cols = ['TotalBsmtSF','1stFlrSF','GrLivArea', 'TotRmsAbvGrd', 'GarageCars', 'GarageAre
sns.pairplot(df[cols])
plt.show()
```



From pairplot, we can say `TotRmsAbvGrd` and `GarageCars` are not related to SalePrice and hence we can delete these.

In [44]:
```
df = df.drop( columns = ['TotRmsAbvGrd','GarageArea'])
df.shape
```

Out[44]:
```
(1460, 75)
```

Derived Variables

Lets try to find the derived variables now.
One can be `AgeOfHouse` (age of the house) which can be formulated from YrSold minus YearBuilt.

In [45]:
```
# create a new column AgeOfHouse
df['AgeOfHouse'] = df.YrSold - df.YearBuilt
```

```
# drop current year and year built columns
df.drop(['YrSold','YearBuilt'], axis=1, inplace=True)
```

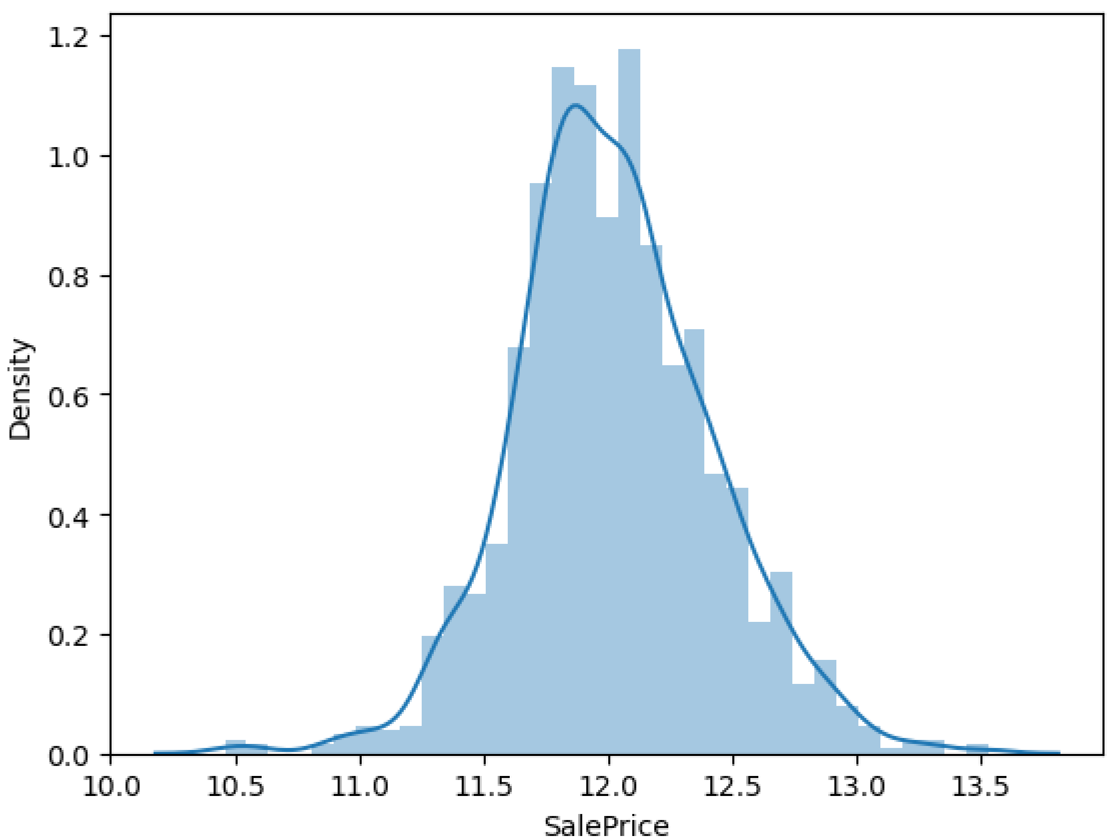
We can drop the Id column as that is also not going to add any value to the model building.

In [46]:
```
df = df.drop(['Id'],axis=1)
df.shape
```

Out[46]:
```
(1460, 73)
```

In [47]:
```
df['SalePrice'] = np.log1p(df['SalePrice'])
sns.distplot(df['SalePrice'])
```

Out[47]:
```
<Axes: xlabel='SalePrice', ylabel='Density'>
```



3. Data Preparation

Data Preparation

Let's now prepare the data and build the model.

In [48]:
```
# split into X and y

 # predictors in variable X
X = df.drop('SalePrice',axis=1)

# response variable in Y
y = df['SalePrice']
```

In [49]:
```python
# creating dummy variables for categorical variables

# subset all categorical variables
df_categorical = X.select_dtypes(include=['object'])
df_categorical.head()
```

Out[49]:

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Conditi
0	RL	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	N
1	RL	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	F
2	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	N
3	RL	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	N
4	RL	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	N

5 rows × 39 columns

In [50]:
```python
# convert into dummies - one hot encoding
df_dummies = pd.get_dummies(df_categorical, drop_first=True)
df_dummies.head()
```

Out[50]:

	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	Street_Pave	LotShape_IR2	LotShape_I
0	0	0	1	0	1	0	
1	0	0	1	0	1	0	
2	0	0	1	0	1	0	
3	0	0	1	0	1	0	
4	0	0	1	0	1	0	

5 rows × 210 columns

In [51]:
```python
# drop categorical variables
X = X.drop(list(df_categorical.columns), axis=1)
```

In [52]:
```python
# concat dummy variables with X
X = pd.concat([X, df_dummies], axis=1)
```

In [53]:
```python
# scaling the features - necessary before using Ridge or Lasso
from sklearn.preprocessing import scale

# storing column names in cols, since column names are lost after
# scaling (the df is converted to a numpy array)
cols = X.columns
X = pd.DataFrame(scale(X))
X.columns = cols
X.columns
```

Out[53]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       ...
       'SaleType_ConLI', 'SaleType_ConLw', 'SaleType_New', 'SaleType_Oth',
       'SaleType_WD', 'SaleCondition_AdjLand', 'SaleCondition_Alloca',
       'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=243)
```

In [54]:
```
# split into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,test_size = (
```

Scaling

In [55]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

Lets check the number of numerical and categorical values.

In [56]:
```
# Checking number of numerical and categorical values

numerical_var = df.dtypes[df.dtypes !='object'].index
print("Number of Numerical Variables : ", len(numerical_var))
print("Numerical variables : ",numerical_var)

categorical_var = df.dtypes[df.dtypes =='object'].index
print("Number of Categorical Variables : " ,len(categorical_var))
print("Categorical variables : ",categorical_var)
```

```
Number of Numerical Variables :  34
Numerical variables :  Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
'OverallCond',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
       'KitchenAbvGr', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
       'MiscVal', 'MoSold', 'SalePrice', 'AgeOfHouse'],
      dtype='object')
Number of Categorical Variables :  39
Categorical variables :  Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Uti
lities',
       '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', 'SaleType', 'SaleCondition'],
      dtype='object')
```

So we have 34 numerical values and 39 categorical values.

In [57]:
```
# removing target variable

numerical_var = list(numerical_var)
```

```
numerical_var.remove('SalePrice')

# fit transform on train set
X_train[numerical_var] = scaler.fit_transform(X_train[numerical_var])
X_train.head()
```

Out[57]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	MasVnrArea	B
210	-0.657071	-0.115302	-0.473765	-0.779861	0.383154	-1.694350	-0.558025	
318	0.035976	0.926898	-0.056845	0.649651	-0.533005	0.390956	0.809137	
239	-0.195040	-0.794998	-0.169324	-0.065105	-1.449164	-1.694350	-0.558025	
986	-0.195040	-0.477806	-0.502297	-0.065105	2.215472	0.875911	-0.558025	
1416	3.039179	-0.432493	0.082905	-1.494617	0.383154	-1.694350	-0.558025	

5 rows × 243 columns

In [58]:

```
# transform on test set
X_test[numerical_var] = scaler.transform(X_test[numerical_var])
X_test.head()
```

Out[58]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	MasVnrArea	B
1436	-0.888086	-0.432493	-0.144189	-1.494617	0.383154	-0.675945	-0.558025	
57	0.035976	0.881585	0.112505	0.649651	-0.533005	0.924407	-0.558025	
780	-0.888086	-0.296554	-0.253368	0.649651	-0.533005	0.536443	-0.355087	
382	0.035976	0.428455	-0.120412	0.649651	-0.533005	1.021398	-0.558025	
1170	0.498007	0.292515	-0.058786	-0.065105	0.383154	-0.384972	-0.558025	

5 rows × 243 columns

4. Model Building and Evaluation

Linear Regression

Let's now try predicting car prices, a dataset using linear regression.

In [59]:

```
# Instantiate
lm = LinearRegression()

# Fit a line
lm.fit(X_train, y_train)
```

Out[59]:

```
▾ LinearRegression
LinearRegression()
```

In [60]:

```
# Print the coefficients and intercept
print(lm.intercept_)
print(lm.coef_)
```

```
-3179715.3690656256
[-1.52060786e-02  7.81767943e-03  3.47357315e-02  5.58230959e-02
  4.02892434e-02  2.36918099e-02  1.16774065e-05  1.74605452e+09
  6.41614605e+08  1.68495629e+09 -1.72427926e+09 -1.06235997e+08
 -1.16189582e+08 -1.28675882e+07  1.42005328e+08  1.23034865e-02
  6.69278204e-04  3.19680572e-03  2.25646794e-03  7.33169913e-03
 -1.43100321e-02 -5.29515743e-03  4.39584255e-03  2.91034728e-02
  1.01996064e-02  4.07002866e-03  6.85892999e-03  3.81037593e-03
  8.09580274e-03  2.90453248e-03  1.02119148e-03  7.19979405e-04
 -6.23948425e-02  9.27400291e-02  4.63690683e-02  1.79391026e-01
  1.35035396e-01  5.74129447e-03  2.45153904e-03 -1.29853934e-03
  2.47024000e-03  1.68631598e-03 -6.87431544e-04  7.21601397e-03
 -2.04801746e-03  7.16167688e-03 -7.26862694e-03 -3.52799892e-04
 -2.84837186e-03  4.58277017e-03 -2.06102878e-02  2.84323451e-03
 -7.55995512e-04  1.99696720e-02  1.05839297e-02  7.71742314e-03
  3.35295796e-02 -3.74168903e-03  3.47119570e-03  1.11695826e-02
 -1.19463205e-02 -4.80495393e-04  1.24228746e-02  1.24412775e-03
  1.60464644e-03  7.31796026e-03  1.67711377e-02  1.90507174e-02
  1.28956288e-02  3.16837430e-03  6.47991896e-03  1.99429542e-02
  1.72668733e-02  2.36503035e-03  8.44805688e-03  1.46470964e-02
  3.49997878e-02  7.45096058e-03  8.13373178e-03 -3.75039876e-04
  8.26363266e-03  1.94260478e-03  3.38380039e-03  3.80449276e-03
  4.10519028e-03  5.93052804e-03 -6.37972802e-02 -1.48158595e-02
 -1.69990957e-03  5.28885424e-03 -1.72934122e-03 -2.75892019e-03
 -3.91066074e-03  5.25647402e-03  1.63689256e-04 -2.36412138e-02
 -4.80136462e-03  2.09257007e-03 -7.57655501e-03 -3.33866477e-03
 -2.55093724e-03 -2.59336233e-02 -5.29303402e-03 -2.47038603e-02
  2.67957896e-03  1.62563771e-02  3.80282253e-01  8.54303185e-02
  8.12032670e-02  7.75805861e-02  2.53355794e-01  1.61158755e-01
  1.87016731e-01 -6.17242070e+07 -6.00449927e-03  2.97097862e-02
  1.19180469e+08 -7.54803419e-04  2.79184878e-02  4.46081907e-03
  3.10039222e-02  2.01646388e-02  5.56873903e-03  1.10934526e-02
  4.52892110e-02  1.21361613e-02  1.39368773e-02  1.06836162e+08
  5.82873821e-04 -8.31219554e-03 -1.19180469e+08  1.79044306e-02
 -8.57262313e-03 -4.56400216e-03 -4.34249640e-03 -4.23457045e+04
 -6.07657433e-03 -5.11620939e-03 -4.43309546e-04 -5.13602793e-03
  2.16184556e-03 -7.15029240e-03 -4.36335802e-04 -5.33425063e-03
  2.16078758e-03  2.38744915e-03  6.30659610e-03  5.91017678e-03
 -1.28225982e-02 -2.18351483e-02 -3.91369914e+02 -1.70574207e-02
  8.29645433e-03  1.34671312e-02 -6.68102503e-03  4.12029400e-03
 -2.88008247e-03 -4.85509634e-04 -2.49008536e-02  6.88627262e+08
 -2.18780041e-02  5.01921773e-03 -3.44313490e+08  1.15151517e-02
  7.91320205e-03  1.08566061e-02  3.40916216e-04 -4.64257598e-03
 -1.01690292e-02 -2.14829296e-03  5.58049977e-03 -5.98092750e-03
 -3.44313771e+08 -7.01314211e-03 -7.60790706e-03 -1.22363567e-02
 -5.22464514e-04 -7.25694560e-03 -2.61207223e-02 -6.88020885e-03
 -6.99383020e-03  2.97570564e-02  2.57173944e-02  2.83662975e-03
  5.84350526e-03  1.51006877e-02 -1.66263897e-04 -5.12066111e-03
 -3.97010893e-03 -1.10913254e-02  1.03862528e-02  4.69550490e-03
 -9.72416252e-04  5.01170020e-01 -1.56293064e-03 -8.28453153e-03
 -3.37825902e-02 -2.87038162e-02 -7.74049014e-03  8.90510064e-03
  1.39473975e-02 -5.04982471e-03 -1.08713210e-02  2.74178833e-02
  2.42260098e-03  1.93149894e-02  1.68405473e-03  1.53884292e-03
  1.73361301e-02 -6.00593537e-03 -4.98408824e-03 -4.32330370e-03
  2.70904601e-03 -1.19801015e-02  1.34131180e+09  1.20140157e+09
 -5.86435199e-04 -1.51333213e-03  2.03714644e+08  1.11335188e+08
 -1.27135668e+09  5.17338365e+07  3.45842820e+08 -1.74753442e+08
 -8.94209427e+07 -1.27135668e+09 -7.89161901e+07 -3.29843891e+08
  4.55648452e-03  5.74502721e-03  2.77770311e-03  3.64865363e-03
  2.03541517e-02 -2.71950383e-04  8.53111967e-04  7.57951554e+08
```

```
  2.20305473e-03  2.27481127e-04  5.86719252e-03  8.00196826e-03
  4.26387787e-03  1.93721056e-02 -7.66353446e+08]
```

In [61]:
```
y_pred_train = lm.predict(X_train)
y_pred_test = lm.predict(X_test)

metric = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R2 train:",r2_train_lr)
metric.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R2 test:",r2_test_lr)
metric.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print("RSS train:",rss1_lr)
metric.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print("RSS test:",rss2_lr)
metric.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("MSE train:",mse_train_lr)
metric.append(mse_train_lr)

print("RMSE train:",mse_train_lr**0.5)
metric.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("MSE test:",mse_test_lr)
metric.append(mse_test_lr)

print("RMSE test:",mse_test_lr**0.5)
metric.append(mse_test_lr**0.5)
```

```
R2 train: 0.9581906935195057
R2 test: -4.8487018426528915e+17
RSS train: 6.709983268507258
RSS test: 3.4943434309044646e+19
MSE train: 0.006571971859458626
RMSE train: 0.08106769923624714
MSE test: 7.977953038594669e+16
RMSE test: 282452704.68867296
```

This shows that r2 value is very poor.

Ridge and Lasso Regression

Let's now try predicting sale prices, a dataset used in simple linear regression, to perform ridge and lasso regression.

Ridge Regression

In [62]:
```
# list of alphas to tune - if value too high it will lead to underfitting, if it is to
# it will not handle the overfitting
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}

ridge = Ridge()

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

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[62]:

```
▸ GridSearchCV
▸ estimator: Ridge
    ▸ Ridge
```

In [63]:
```
# Printing the best hyperparameter alpha
print(model_cv.best_params_)
```

```
{'alpha': 4.0}
```

In [64]:
```
#Fitting Ridge model for alpha = 4 and printing coefficients which have been penalised
alpha = 4
ridge = Ridge(alpha=alpha)

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

```
[-1.27927128e-02  2.78527674e-03  3.30398298e-02  5.92514955e-02
  3.86146480e-02  2.33039749e-02 -4.30905825e-04  2.60513509e-02
  8.11483711e-03  7.07988597e-03  3.63183486e-02  4.78749433e-02
  4.88623875e-02 -1.15812446e-04  7.57848502e-02  1.53394349e-02
  8.84307927e-04  7.55386699e-03  4.80266940e-03  1.00108484e-02
 -1.43944220e-02 -7.10149128e-03  1.40926074e-04  3.33796478e-02
  1.04726614e-02  3.64937943e-03  7.35756683e-03  4.73971439e-03
  8.70281528e-03 -2.00889059e-03  9.39217954e-04  4.37557636e-04
 -5.19053325e-02  6.86833696e-02  3.52822521e-02  1.34376493e-01
  1.01119572e-01  6.31910579e-03  2.67457495e-03 -3.90556373e-03
  2.32886074e-03  3.42691741e-03  9.49663119e-04  9.56074962e-03
 -2.48220657e-03  7.14227116e-03 -8.25964535e-03 -1.44238795e-03
 -3.43119416e-03  4.25619570e-03 -1.78083531e-02  1.32599189e-03
 -3.41865522e-03  1.23282186e-02  9.70435682e-03  4.37942814e-03
  3.01039053e-02 -1.00679120e-02  8.12322744e-04  1.51314210e-03
 -1.58194298e-02 -3.32651137e-03  6.34618250e-03 -4.70521870e-04
 -1.29333615e-03  8.11196781e-03  1.83695162e-02  7.90036798e-03
  9.21391958e-03 -6.83962949e-04  4.32838551e-03  1.97509864e-02
  1.62143051e-02  1.08563656e-03  7.57003747e-03  1.18934061e-02
  3.36456519e-02  7.43100021e-03  7.69513170e-03 -7.67772984e-04
  8.88044438e-03  1.66769429e-03  4.60007574e-03  1.96298349e-03
  2.32248062e-03  6.85467887e-03 -5.89694705e-02 -1.45274101e-02
 -1.95737502e-03  4.21150475e-03 -2.26981378e-03 -1.55914707e-03
 -7.44564784e-03 -4.68726376e-04  3.79915853e-04 -1.59796739e-02
 -4.86381915e-03  3.86995247e-04 -9.88107060e-03 -2.65949706e-03
 -3.46400189e-03 -2.25839560e-02 -4.19005854e-03 -1.98229642e-02
  3.31766585e-03  1.58543429e-02  2.75774300e-01  6.27153463e-02
  5.96421637e-02  5.67798788e-02  1.83106336e-01  1.13698604e-01
  1.37017522e-01  3.51909574e-04 -7.23862534e-03  2.19742283e-02
 -6.06695487e-05 -4.66755531e-03  1.37685404e-02  2.61921485e-03
  1.64779918e-02  1.12793965e-02  4.69200997e-03  5.12652023e-03
  2.95841788e-02  8.56945906e-04  8.90511151e-03  2.03314487e-04
  2.45785160e-03 -3.52509160e-03 -6.06695487e-05  2.16688879e-02
  2.42474762e-03 -2.07221157e-04  7.38480611e-03  0.00000000e+00
  2.68390787e-03 -3.97172789e-03  1.15693857e-03  7.03543754e-03
  1.07246569e-02 -3.84533640e-03  7.27065795e-04 -3.35405263e-03
  1.74033502e-03  1.11958625e-03  1.03099426e-02  8.97987335e-03
 -7.42280501e-03 -1.05490974e-02  0.00000000e+00 -5.50621464e-03
  1.11232372e-02  1.55370398e-02 -4.97969535e-03  3.50827150e-03
 -1.96434631e-03 -2.08345295e-03 -2.67497341e-02  6.39580988e-03
 -2.56234073e-02  5.56877957e-03  6.39580988e-03  9.91830835e-03
  8.01717380e-03  1.18187369e-02 -9.93609486e-04 -5.97797185e-03
 -8.69854131e-03 -3.19861822e-03  6.04602252e-03 -6.09994043e-03
  6.39580988e-03 -7.79228842e-03 -1.22730537e-02 -1.22291941e-02
 -4.94931726e-04 -5.96763627e-03 -1.64359786e-02 -6.64512014e-03
 -6.60979171e-03  2.04095047e-02  2.01105109e-02 -9.52136612e-04
  2.31748424e-03  1.11712204e-02  3.42553151e-04 -5.39431213e-03
 -3.68425696e-03 -1.17202750e-02  1.27117554e-02  5.25880737e-03
 -4.30016112e-04  0.00000000e+00 -1.31815784e-03 -8.33555538e-03
 -3.36135424e-02 -2.94196746e-02 -8.72815480e-03  6.47554518e-03
  1.04027767e-02 -4.25124545e-03 -1.02631377e-02  2.21061626e-02
  1.22760621e-03  1.26475662e-02 -8.92483192e-03 -7.68371652e-04
  1.11286072e-02  3.13546690e-03 -2.20994037e-03 -3.75484662e-04
  3.85619714e-03 -3.20336901e-03 -4.35771579e-03 -4.35771579e-03
 -1.07179440e-03 -3.72178155e-03 -1.60977826e-02  2.89412198e-03
 -4.35771579e-03 -2.68438550e-03 -7.00599676e-03 -9.90589003e-03
 -3.67482235e-03 -4.35771579e-03  4.10348697e-03 -1.14694281e-02
  5.31747621e-03  5.96697003e-03  3.23026477e-03  4.10810637e-03
  1.95007371e-02  1.96927340e-04  1.08746589e-03  1.38319565e-02
```

```
 2.51181962e-03  2.17392111e-03  4.74201485e-03  6.61277205e-03
 4.57924472e-03  1.97939383e-02  1.36803103e-02]
```

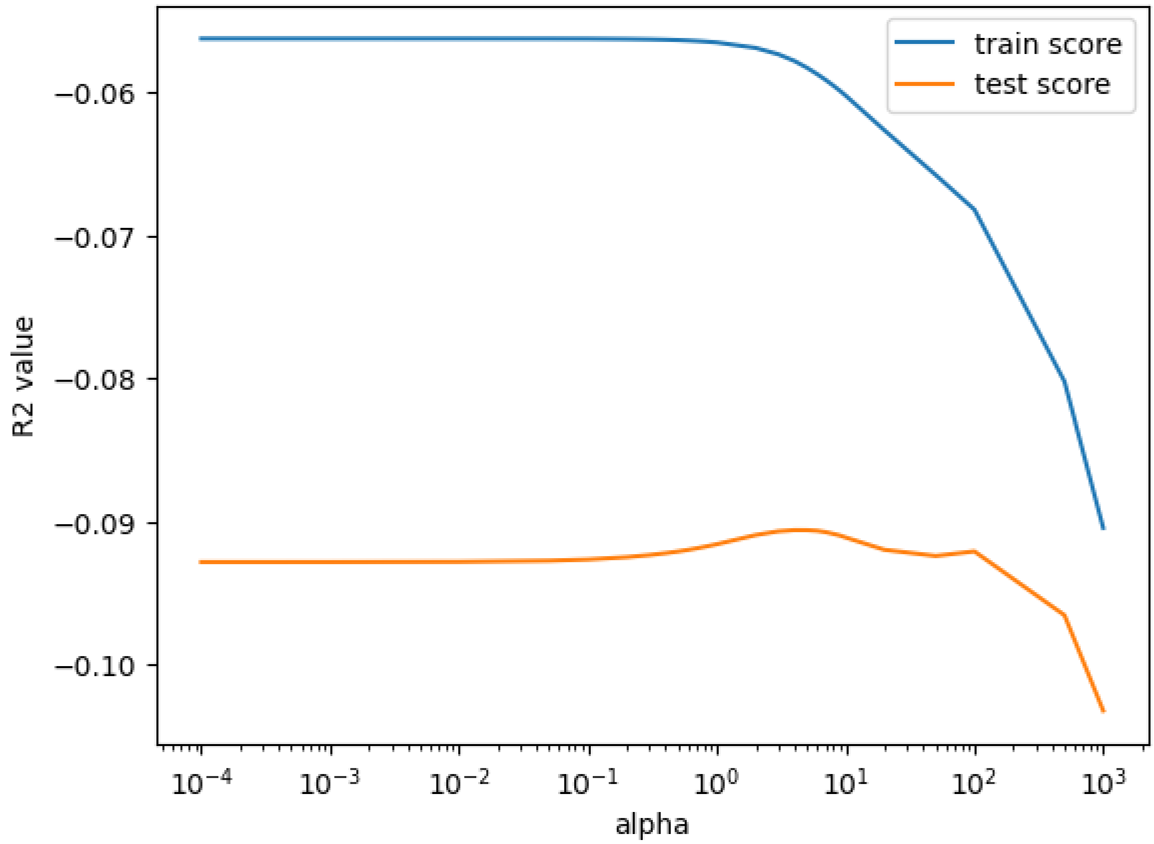
In [65]:

```
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

metric2 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R square train :",r2_train_lr)
metric2.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R square test :",r2_test_lr)
metric2.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print("Rss train :",rss1_lr)
metric2.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print("Rss test :",rss2_lr)
metric2.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("MSE train:",mse_train_lr)
metric2.append(mse_train_lr)

print("RMSE train:",mse_train_lr**0.5)
metric2.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("MSE test :",mse_test_lr)
metric2.append(mse_test_lr)

print("RMSE test :",mse_test_lr**0.5)
metric2.append(mse_test_lr**0.5)
```

```
R square train : 0.9563668026337852
R square test : 0.8692068548139035
Rss train : 7.002699851416305
Rss test : 9.425949099776162
MSE train: 0.0068586678270482915
RMSE train: 0.08281707448979522
MSE test : 0.021520431734648772
RMSE test : 0.14669843807842253
```

In [66]:

```
#plotting
model_cv_results = pd.DataFrame(model_cv.cv_results_)
plt.plot(model_cv_results['param_alpha'], model_cv_results['mean_train_score'])
plt.plot(model_cv_results['param_alpha'], model_cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.xscale('log')
plt.ylabel('R2 value')
plt.legend(['train score', 'test score'])
plt.show()
```



As alpha increases:

- train error decreases
- test error is first constant and then decreases

Lasso

In [67]:
```
lasso = Lasso()

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

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[67]:

▸ **GridSearchCV**

▸ **estimator: Lasso**

▸ Lasso

In [68]:
```
# Printing the best hyperparameter alpha
print(model_cv.best_params_)
```

```
{'alpha': 0.001}
```

In [69]:
```
#Fitting Lasso model for alpha = 0.001 and printing coefficients which have been penal

alpha =0.001

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

Out[69]:
```
▾ Lasso
Lasso(alpha=0.001)
```

In [70]:
```
lasso.coef_
```

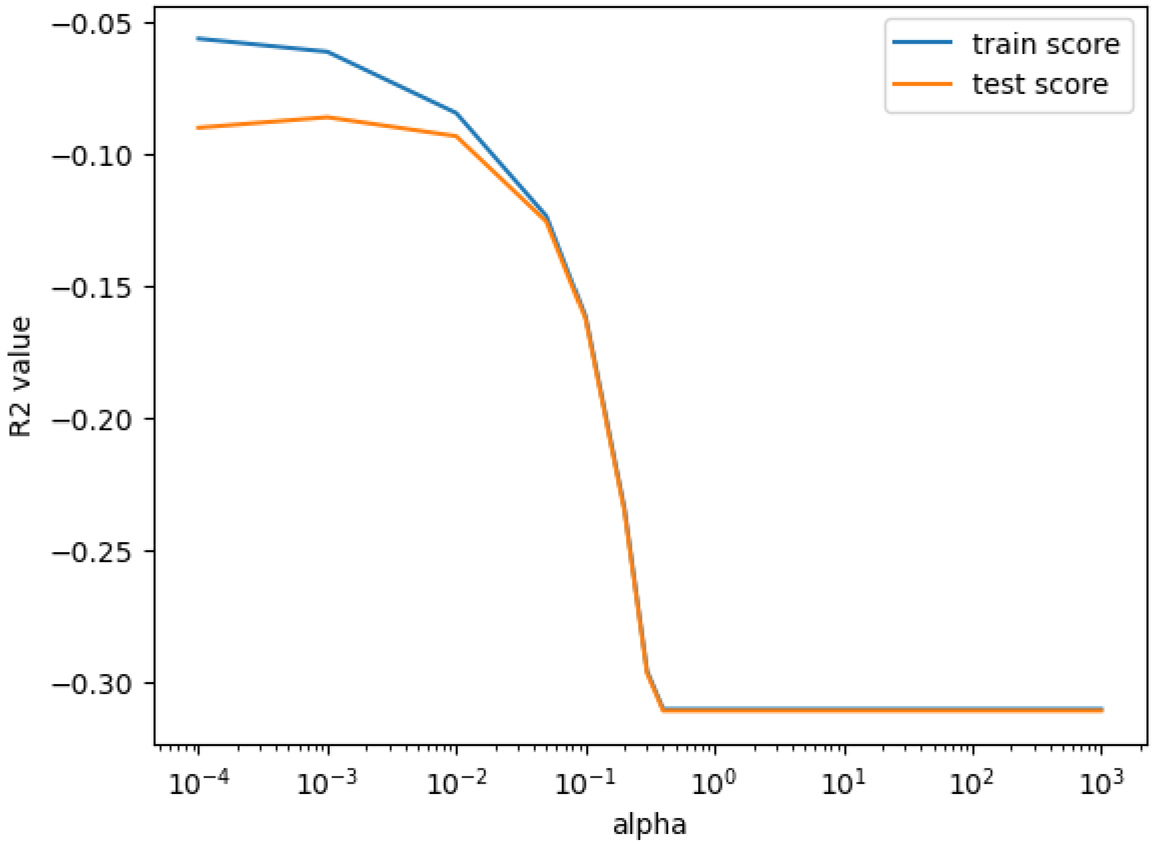
```
Out[70]: array([-1.61262086e-02,  2.08610968e-03,  2.41078698e-02,  6.62689614e-02,
                4.02439898e-02,  2.38003769e-02,  4.56033324e-05,  1.53927672e-02,
                8.55770266e-04, -0.00000000e+00,  3.79414235e-02,  2.40009372e-03,
                0.00000000e+00, -4.05746284e-03,  1.42040081e-01,  2.00867167e-02,
                0.00000000e+00,  6.13439927e-03,  4.09156761e-03,  4.62429597e-03,
               -1.17834084e-02,  0.00000000e+00,  0.00000000e+00,  3.46271101e-02,
                9.62235868e-03,  3.82877612e-03,  3.75925631e-03,  3.80934179e-03,
                7.12251361e-03, -4.79527133e-03, -0.00000000e+00, -0.00000000e+00,
               -5.29846777e-02,  3.52764960e-02,  1.88030062e-02,  7.44493007e-02,
                4.65672455e-02,  4.75788842e-03,  9.19110007e-04, -4.15465631e-03,
                0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  3.70264764e-03,
               -2.15137733e-03,  7.52648211e-03, -4.45221578e-03, -3.11210762e-05,
               -5.96633914e-05,  1.35041959e-03, -8.20107100e-03,  0.00000000e+00,
               -3.55135636e-03,  6.24496247e-03,  9.28037160e-03,  0.00000000e+00,
                2.57092005e-02, -1.24746151e-02, -1.41143816e-03, -4.75963942e-03,
               -1.43563742e-02, -3.73970638e-03,  0.00000000e+00, -0.00000000e+00,
               -1.19258825e-03,  5.29331181e-03,  1.65398550e-02, -0.00000000e+00,
                5.15552926e-03, -1.69266478e-03,  6.07806334e-05,  1.80769194e-02,
                1.21485021e-02,  0.00000000e+00,  5.16149744e-03,  1.21791733e-03,
                2.03958084e-02,  3.02306083e-03,  2.88914232e-03, -3.99748206e-03,
                3.50187352e-03,  0.00000000e+00,  2.18511879e-03, -0.00000000e+00,
                9.91805972e-04,  5.14214865e-03, -5.77445259e-02, -6.11286847e-03,
               -1.45136196e-03,  1.91662622e-03,  0.00000000e+00, -0.00000000e+00,
               -8.91329620e-03, -2.35655659e-04,  9.84559746e-04, -4.06307132e-03,
               -3.81688409e-03,  0.00000000e+00, -0.00000000e+00, -0.00000000e+00,
                0.00000000e+00, -1.72762887e-03,  0.00000000e+00,  0.00000000e+00,
                4.57605452e-03,  7.73313960e-03,  2.42022528e-01,  5.20981205e-02,
                5.06481567e-02,  4.86084044e-02,  1.62587690e-01,  1.01794979e-01,
                1.21258260e-01, -0.00000000e+00, -7.72213839e-03,  1.21181421e-02,
               -1.81081156e-04,  0.00000000e+00, -1.80583138e-04,  3.55284433e-04,
                3.90653504e-03,  0.00000000e+00,  2.53523041e-04, -0.00000000e+00,
                1.06829526e-02, -3.69116002e-03,  2.87349844e-03, -0.00000000e+00,
                0.00000000e+00, -0.00000000e+00, -1.60628598e-05,  6.27686446e-03,
               -0.00000000e+00, -0.00000000e+00,  1.03941374e-03,  0.00000000e+00,
                0.00000000e+00, -0.00000000e+00, -1.04864690e-03,  0.00000000e+00,
                0.00000000e+00, -5.51140228e-03,  0.00000000e+00, -1.85423400e-03,
                2.88313141e-04, -1.79938406e-03,  1.78006929e-03, -3.81036264e-04,
               -2.24569740e-03, -0.00000000e+00,  0.00000000e+00,  4.37783219e-03,
                0.00000000e+00,  8.05034206e-03, -0.00000000e+00,  2.88123779e-03,
               -1.46899754e-03, -0.00000000e+00, -1.68610353e-02, -0.00000000e+00,
               -1.53277648e-02,  2.82542416e-03, -0.00000000e+00,  1.49444147e-03,
                5.42776782e-03,  1.48150976e-02, -0.00000000e+00, -4.85922669e-03,
               -6.83872767e-04, -0.00000000e+00,  7.93590656e-03, -1.04123296e-03,
               -0.00000000e+00, -4.17949632e-03, -8.47910269e-03, -7.05864663e-03,
                2.63415570e-03, -0.00000000e+00, -3.59514846e-03, -0.00000000e+00,
                0.00000000e+00,  0.00000000e+00,  4.35072819e-03, -6.44640908e-03,
               -2.52886597e-03,  1.32830375e-03,  0.00000000e+00, -3.21196360e-03,
               -5.92417825e-04, -9.00288106e-03,  1.57062901e-02,  3.24585104e-03,
               -1.07325705e-03,  0.00000000e+00,  0.00000000e+00, -3.11143648e-03,
               -1.91590424e-02, -1.51467692e-02, -9.75303722e-03,  0.00000000e+00,
                2.11615461e-03, -4.07795769e-03, -9.44256311e-03,  9.88706328e-03,
               -0.00000000e+00,  3.62237616e-03, -1.50910580e-02, -2.10682569e-03,
                0.00000000e+00,  4.67256764e-03, -1.22416573e-03,  0.00000000e+00,
                0.00000000e+00, -3.42428721e-04, -0.00000000e+00, -0.00000000e+00,
               -0.00000000e+00, -2.29807458e-03, -7.90308648e-03,  3.36753764e-03,
               -0.00000000e+00,  0.00000000e+00,  0.00000000e+00, -3.30761854e-03,
               -0.00000000e+00, -0.00000000e+00,  1.91484294e-03,  9.24963532e-04,
                4.33187613e-03,  4.80374297e-03,  2.69014042e-03,  2.28814145e-03,
                1.54591058e-02, -0.00000000e+00,  1.51238024e-04,  2.44847726e-02,
```

```
        7.64416709e-04,  0.00000000e+00,  3.55156311e-03,  1.20613717e-03,
        2.45725324e-03,  1.79753584e-02,  5.20037031e-05])
```

In [71]:
```python
# Lets calculate some metrics such as R2 score, RSS and RMSE

y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

metric3 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R square train :",r2_train_lr)
metric3.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R square test :",r2_test_lr)
metric3.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print("Rss train :",rss1_lr)
metric3.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print("Rss test :",rss2_lr)
metric3.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("MSE train:",mse_train_lr)
metric3.append(mse_train_lr)

print("RMSE train:",mse_train_lr**0.5)
metric3.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("MSE test :",mse_test_lr)
metric3.append(mse_test_lr)

print("RMSE test :",mse_test_lr**0.5)
metric3.append(mse_test_lr**0.5)
```

```
R square train : 0.9505553591166847
R square test : 0.8724446319321947
Rss train : 7.9353794877985155
Rss test : 9.192610248034278
MSE train: 0.007772164042897664
RMSE train: 0.0881598777386724
MSE test : 0.020987694630215246
RMSE test : 0.14487130368094037
```

In [72]:
```python
model_cv_results2 = pd.DataFrame(model_cv.cv_results_)

#plotting
plt.plot(model_cv_results2['param_alpha'], model_cv_results2['mean_train_score'])
plt.plot(model_cv_results2['param_alpha'], model_cv_results2['mean_test_score'])
plt.xlabel('alpha')
plt.xscale('log')
plt.ylabel('R2 value')
plt.legend(['train score', 'test score'])
plt.show()
```



With increase in alpha value, both train and test error decreases.

In [73]:
```
# Creating a table which contain all the metrics

lr_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS (Test)'
        'Linear Regression': metric }

lr_metric = pd.DataFrame(lr_table ,columns = ['Metric', 'Linear Regression'] )

rg_metric = pd.Series(metric2, name = 'Ridge Regression')
ls_metric = pd.Series(metric3, name = 'Lasso Regression')

final_metric = pd.concat([lr_metric, rg_metric, ls_metric], axis = 1)

final_metric
```

Out[73]:

	Metric	Linear Regression	Ridge Regression	Lasso Regression
0	R2 Score (Train)	9.581907e-01	0.956367	0.950555
1	R2 Score (Test)	-4.848702e+17	0.869207	0.872445
2	RSS (Train)	6.709983e+00	7.002700	7.935379
3	RSS (Test)	3.494343e+19	9.425949	9.192610
4	MSE (Train)	6.571972e-03	0.006859	0.007772
5	MSE (Test)	8.106770e-02	0.082817	0.088160
6	RMSE (Train)	7.977953e+16	0.021520	0.020988
7	RMSE (Test)	2.824527e+08	0.146698	0.144871

Lets observe the changes in the coefficients after regularization

In [74]:
```
betas = pd.DataFrame(index=X.columns)
```

In [75]:
```
betas.rows = X.columns
```

In [76]:
```
betas['Linear'] = lm.coef_
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
```

In [77]:
```
pd.set_option('display.max_rows', None)
betas.head(80)
```

Out[77]:

	Linear	Ridge	Lasso
MSSubClass	-1.520608e-02	-0.012793	-0.016126
LotFrontage	7.817679e-03	0.002785	0.002086
LotArea	3.473573e-02	0.033040	0.024108
OverallQual	5.582310e-02	0.059251	0.066269
OverallCond	4.028924e-02	0.038615	0.040244
YearRemodAdd	2.369181e-02	0.023304	0.023800
MasVnrArea	1.167741e-05	-0.000431	0.000046
BsmtFinSF1	1.746055e+09	0.026051	0.015393
BsmtFinSF2	6.416146e+08	0.008115	0.000856
BsmtUnfSF	1.684956e+09	0.007080	-0.000000
TotalBsmtSF	-1.724279e+09	0.036318	0.037941
1stFlrSF	-1.062360e+08	0.047875	0.002400
2ndFlrSF	-1.161896e+08	0.048862	0.000000
LowQualFinSF	-1.286759e+07	-0.000116	-0.004057
GrLivArea	1.420053e+08	0.075785	0.142040
BsmtFullBath	1.230349e-02	0.015339	0.020087
BsmtHalfBath	6.692782e-04	0.000884	0.000000
FullBath	3.196806e-03	0.007554	0.006134
HalfBath	2.256468e-03	0.004803	0.004092
BedroomAbvGr	7.331699e-03	0.010011	0.004624
KitchenAbvGr	-1.431003e-02	-0.014394	-0.011783
Fireplaces	-5.295157e-03	-0.007101	0.000000
GarageYrBlt	4.395843e-03	0.000141	0.000000
GarageCars	2.910347e-02	0.033380	0.034627
WoodDeckSF	1.019961e-02	0.010473	0.009622
OpenPorchSF	4.070029e-03	0.003649	0.003829
EnclosedPorch	6.858930e-03	0.007358	0.003759
3SsnPorch	3.810376e-03	0.004740	0.003809
ScreenPorch	8.095803e-03	0.008703	0.007123
PoolArea	2.904532e-03	-0.002009	-0.004795
MiscVal	1.021191e-03	0.000939	-0.000000
MoSold	7.199794e-04	0.000438	-0.000000
AgeOfHouse	-6.239484e-02	-0.051905	-0.052985

	Linear	Ridge	Lasso
MSZoning_FV	9.274003e-02	0.068683	0.035276
MSZoning_RH	4.636907e-02	0.035282	0.018803
MSZoning_RL	1.793910e-01	0.134376	0.074449
MSZoning_RM	1.350354e-01	0.101120	0.046567
Street_Pave	5.741294e-03	0.006319	0.004758
LotShape_IR2	2.451539e-03	0.002675	0.000919
LotShape_IR3	-1.298539e-03	-0.003906	-0.004155
LotShape_Reg	2.470240e-03	0.002329	0.000000
LandContour_HLS	1.686316e-03	0.003427	0.000000
LandContour_Low	-6.874315e-04	0.000950	0.000000
LandContour_Lvl	7.216014e-03	0.009561	0.003703
Utilities_NoSeWa	-2.048017e-03	-0.002482	-0.002151
LotConfig_CulDSac	7.161677e-03	0.007142	0.007526
LotConfig_FR2	-7.268627e-03	-0.008260	-0.004452
LotConfig_FR3	-3.527999e-04	-0.001442	-0.000031
LotConfig_Inside	-2.848372e-03	-0.003431	-0.000060
LandSlope_Mod	4.582770e-03	0.004256	0.001350
LandSlope_Sev	-2.061029e-02	-0.017808	-0.008201
Neighborhood_Blueste	2.843235e-03	0.001326	0.000000
Neighborhood_BrDale	-7.559955e-04	-0.003419	-0.003551
Neighborhood_BrkSide	1.996967e-02	0.012328	0.006245
Neighborhood_ClearCr	1.058393e-02	0.009704	0.009280
Neighborhood_CollgCr	7.717423e-03	0.004379	0.000000
Neighborhood_Crawfor	3.352958e-02	0.030104	0.025709
Neighborhood_Edwards	-3.741689e-03	-0.010068	-0.012475
Neighborhood_Gilbert	3.471196e-03	0.000812	-0.001411
Neighborhood_IDOTRR	1.116958e-02	0.001513	-0.004760
Neighborhood_MeadowV	-1.194632e-02	-0.015819	-0.014356
Neighborhood_Mitchel	-4.804954e-04	-0.003327	-0.003740
Neighborhood_NAmes	1.242287e-02	0.006346	0.000000
Neighborhood_NPkVill	1.244128e-03	-0.000471	-0.000000
Neighborhood_NWAmes	1.604646e-03	-0.001293	-0.001193
Neighborhood_NoRidge	7.317960e-03	0.008112	0.005293

	Linear	Ridge	Lasso
Neighborhood_NridgHt	1.677114e-02	0.018370	0.016540
Neighborhood_OldTown	1.905072e-02	0.007900	-0.000000
Neighborhood_SWISU	1.289563e-02	0.009214	0.005156
Neighborhood_Sawyer	3.168374e-03	-0.000684	-0.001693
Neighborhood_SawyerW	6.479919e-03	0.004328	0.000061
Neighborhood_Somerst	1.994295e-02	0.019751	0.018077
Neighborhood_StoneBr	1.726687e-02	0.016214	0.012149
Neighborhood_Timber	2.365030e-03	0.001086	0.000000
Neighborhood_Veenker	8.448057e-03	0.007570	0.005161
Condition1_Feedr	1.464710e-02	0.011893	0.001218
Condition1_Norm	3.499979e-02	0.033646	0.020396
Condition1_PosA	7.450961e-03	0.007431	0.003023
Condition1_PosN	8.133732e-03	0.007695	0.002889
Condition1_RRAe	-3.750399e-04	-0.000768	-0.003997

In [78]:
```
betas[betas['Lasso'] == 0].shape
```

Out[78]:
```
(73, 3)
```

73 features have been removed by lasso

In [79]:
```
# selected features :

betas.loc[betas['Lasso'] != 0, 'Lasso']
```

Out[79]:

```
MSSubClass              -0.016126
LotFrontage              0.002086
LotArea                  0.024108
OverallQual              0.066269
OverallCond              0.040244
YearRemodAdd             0.023800
MasVnrArea               0.000046
BsmtFinSF1               0.015393
BsmtFinSF2               0.000856
TotalBsmtSF              0.037941
1stFlrSF                 0.002400
LowQualFinSF            -0.004057
GrLivArea                0.142040
BsmtFullBath             0.020087
FullBath                 0.006134
HalfBath                 0.004092
BedroomAbvGr             0.004624
KitchenAbvGr            -0.011783
GarageCars               0.034627
WoodDeckSF               0.009622
OpenPorchSF              0.003829
EnclosedPorch            0.003759
3SsnPorch                0.003809
ScreenPorch              0.007123
PoolArea                -0.004795
AgeOfHouse              -0.052985
MSZoning_FV              0.035276
MSZoning_RH              0.018803
MSZoning_RL              0.074449
MSZoning_RM              0.046567
Street_Pave              0.004758
LotShape_IR2             0.000919
LotShape_IR3            -0.004155
LandContour_Lvl          0.003703
Utilities_NoSeWa        -0.002151
LotConfig_CulDSac        0.007526
LotConfig_FR2           -0.004452
LotConfig_FR3           -0.000031
LotConfig_Inside        -0.000060
LandSlope_Mod            0.001350
LandSlope_Sev           -0.008201
Neighborhood_BrDale     -0.003551
Neighborhood_BrkSide     0.006245
Neighborhood_ClearCr     0.009280
Neighborhood_Crawfor     0.025709
Neighborhood_Edwards    -0.012475
Neighborhood_Gilbert    -0.001411
Neighborhood_IDOTRR     -0.004760
Neighborhood_MeadowV    -0.014356
Neighborhood_Mitchel    -0.003740
Neighborhood_NWAmes     -0.001193
Neighborhood_NoRidge     0.005293
Neighborhood_NridgHt     0.016540
Neighborhood_SWISU       0.005156
Neighborhood_Sawyer     -0.001693
Neighborhood_SawyerW     0.000061
Neighborhood_Somerst     0.018077
Neighborhood_StoneBr     0.012149
Neighborhood_Veenker     0.005161
Condition1_Feedr         0.001218
```

Condition1_Norm	0.020396
Condition1_PosA	0.003023
Condition1_PosN	0.002889
Condition1_RRAe	-0.003997
Condition1_RRAn	0.003502
Condition1_RRNn	0.002185
Condition2_Norm	0.000992
Condition2_PosA	0.005142
Condition2_PosN	-0.057745
Condition2_RRAe	-0.006113
Condition2_RRAn	-0.001451
Condition2_RRNn	0.001917
BldgType_Twnhs	-0.008913
BldgType_TwnhsE	-0.000236
HouseStyle_1.5Unf	0.000985
HouseStyle_1Story	-0.004063
HouseStyle_2.5Fin	-0.003817
RoofStyle_Gable	-0.001728
RoofStyle_Mansard	0.004576
RoofStyle_Shed	0.007733
RoofMatl_CompShg	0.242023
RoofMatl_Membran	0.052098
RoofMatl_Metal	0.050648
RoofMatl_Roll	0.048608
RoofMatl_Tar&Grv	0.162588
RoofMatl_WdShake	0.101795
RoofMatl_WdShngl	0.121258
Exterior1st_BrkComm	-0.007722
Exterior1st_BrkFace	0.012118
Exterior1st_CBlock	-0.000181
Exterior1st_HdBoard	-0.000181
Exterior1st_ImStucc	0.000355
Exterior1st_MetalSd	0.003907
Exterior1st_Stone	0.000254
Exterior1st_VinylSd	0.010683
Exterior1st_Wd Sdng	-0.003691
Exterior1st_WdShing	0.002873
Exterior2nd_CBlock	-0.000016
Exterior2nd_CmentBd	0.006277
Exterior2nd_MetalSd	0.001039
Exterior2nd_Stucco	-0.001049
Exterior2nd_Wd Shng	-0.005511
MasVnrType_None	-0.001854
MasVnrType_Stone	0.000288
ExterQual_Fa	-0.001799
ExterQual_Gd	0.001780
ExterQual_TA	-0.000381
ExterCond_Fa	-0.002246
ExterCond_TA	0.004378
Foundation_PConc	0.008050
Foundation_Stone	0.002881
Foundation_Wood	-0.001469
BsmtQual_Gd	-0.016861
BsmtQual_TA	-0.015328
BsmtCond_Gd	0.002825
BsmtCond_Po	0.001494
BsmtCond_TA	0.005428
BsmtExposure_Gd	0.014815
BsmtExposure_No	-0.004859
BsmtExposure_NoBasement	-0.000684

```
BsmtFinType1_GLQ          0.007936
BsmtFinType1_LwQ         -0.001041
BsmtFinType1_Rec         -0.004179
BsmtFinType1_Unf         -0.008479
BsmtFinType2_BLQ         -0.007059
BsmtFinType2_GLQ          0.002634
BsmtFinType2_NoBasement  -0.003595
Heating_GasW              0.004351
Heating_Grav             -0.006446
Heating_OthW             -0.002529
Heating_Wall              0.001328
HeatingQC_Gd             -0.003212
HeatingQC_Po             -0.000592
HeatingQC_TA             -0.009003
CentralAir_Y              0.015706
Electrical_FuseF          0.003246
Electrical_FuseP         -0.001073
KitchenQual_Fa           -0.003111
KitchenQual_Gd           -0.019159
KitchenQual_TA           -0.015147
Functional_Maj2          -0.009753
Functional_Min2           0.002116
Functional_Mod           -0.004078
Functional_Sev           -0.009443
Functional_Typ            0.009887
FireplaceQu_Gd            0.003622
FireplaceQu_NoFireplace  -0.015091
FireplaceQu_Po           -0.002107
GarageType_Attchd         0.004673
GarageType_Basment       -0.001224
GarageType_Detchd        -0.000342
GarageFinish_Unf         -0.002298
GarageQual_Fa            -0.007903
GarageQual_Gd             0.003368
GarageCond_Fa            -0.003308
GarageCond_Po             0.001915
GarageCond_TA             0.000925
PavedDrive_P              0.004332
PavedDrive_Y              0.004804
SaleType_CWD              0.002690
SaleType_Con              0.002288
SaleType_ConLD            0.015459
SaleType_ConLw            0.000151
SaleType_New              0.024485
SaleType_Oth              0.000764
SaleCondition_AdjLand     0.003552
SaleCondition_Alloca      0.001206
SaleCondition_Family      0.002457
SaleCondition_Normal      0.017975
SaleCondition_Partial     0.000052
Name: Lasso, dtype: float64
```

Top 10 features in Ridge Regression

In [80]:
```
betas['Ridge'].sort_values(ascending=False)[:10]
```

Out[80]:

```
RoofMatl_CompShg    0.275774
RoofMatl_Tar&Grv    0.183106
RoofMatl_WdShngl    0.137018
MSZoning_RL         0.134376
RoofMatl_WdShake    0.113699
MSZoning_RM         0.101120
GrLivArea           0.075785
MSZoning_FV         0.068683
RoofMatl_Membran    0.062715
RoofMatl_Metal      0.059642
Name: Ridge, dtype: float64
```

Top 10 features in Lasso Regression

In [81]:

```
betas['Lasso'].sort_values(ascending=False)[:10]
```

Out[81]:

```
RoofMatl_CompShg    0.242023
RoofMatl_Tar&Grv    0.162588
GrLivArea           0.142040
RoofMatl_WdShngl    0.121258
RoofMatl_WdShake    0.101795
MSZoning_RL         0.074449
OverallQual         0.066269
RoofMatl_Membran    0.052098
RoofMatl_Metal      0.050648
RoofMatl_Roll       0.048608
Name: Lasso, dtype: float64
```

Optimal value of alpha in Ridge : 4

Optimal value of aplha in Lasso : 0.001

Coding Questions:

1. Double the alpha

In [82]:

```
#Fitting Ridge model for alpha = 4*2 i.e. 8 and printing coefficients
alpha = 8
ridge = Ridge(alpha=alpha)

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

```
[-1.22268357e-02  2.75892347e-04  3.17078121e-02  6.09779876e-02
  3.76089785e-02  2.32102549e-02 -6.00220501e-04  2.02992812e-02
  7.38280273e-03  6.46270672e-03  2.96181397e-02  4.71718833e-02
  4.73560498e-02  5.74302843e-04  7.40889216e-02  1.70575483e-02
  9.28368717e-04  1.01561609e-02  6.27359029e-03  1.14705211e-02
 -1.42448830e-02 -7.38936885e-03 -1.94686713e-03  3.55508558e-02
  1.06230273e-02  3.34489675e-03  7.50200865e-03  5.15476054e-03
  8.99706167e-03 -4.72046655e-03  8.69967735e-04  2.62549776e-04
 -4.58324933e-02  5.46098945e-02  2.87962705e-02  1.08055384e-01
  8.08319712e-02  6.54462437e-03  2.74975166e-03 -5.48671075e-03
  2.17502100e-03  4.35745253e-03  1.84176784e-03  1.07569590e-02
 -2.70578682e-03  7.20684161e-03 -8.75369298e-03 -2.06419942e-03
 -3.71482461e-03  4.06216312e-03 -1.59765756e-02  5.47157372e-04
 -4.65239327e-03  8.57637531e-03  9.51160856e-03  2.90901461e-03
  2.85130651e-02 -1.31338441e-02 -4.32405024e-04 -3.46975569e-03
 -1.76720117e-02 -4.59208371e-03  3.55864167e-03 -1.24213417e-03
 -2.41688148e-03  8.87150527e-03  1.95830877e-02  2.23792618e-03
  7.51456616e-03 -2.43641600e-03  3.40569816e-03  1.99680594e-02
  1.58328564e-02  6.71212342e-04  7.23436472e-03  9.90426690e-03
  3.22838332e-02  7.25347556e-03  7.33596173e-03 -1.12086966e-03
  9.09207071e-03  1.44113995e-03  5.11751478e-03  1.40829199e-03
  1.96275371e-03  7.17069469e-03 -5.59116128e-02 -1.39218689e-02
 -1.94299276e-03  3.87890926e-03 -2.34990785e-03 -7.44363484e-04
 -9.04897202e-03 -3.06154749e-03  4.86175751e-04 -1.23045219e-02
 -4.66370512e-03 -4.60620656e-04 -1.06223935e-02 -2.30608397e-03
 -3.86088408e-03 -1.94280137e-02 -3.39863221e-03 -1.58235278e-02
  3.79188308e-03  1.54673610e-02  2.17453292e-01  4.99417873e-02
  4.75741119e-02  4.51055077e-02  1.44053106e-01  8.74171827e-02
  1.09204831e-01  2.15242003e-04 -7.48431770e-03  1.98460204e-02
  1.37695250e-04 -3.75473542e-03  1.01772033e-02  1.91376101e-03
  1.30622748e-02  9.33106958e-03  4.65420662e-03  3.40314473e-03
  2.57074052e-02 -1.07152971e-03  7.65243904e-03  1.24355291e-04
  2.47368499e-03 -2.70342060e-03  1.37695250e-04  2.01664723e-02
  3.53937911e-03  1.02249102e-03  8.33940152e-03  0.00000000e+00
  3.39768023e-03 -4.09886431e-03  1.24374708e-04  7.85831594e-03
  1.04108279e-02 -4.26910391e-03  1.26858036e-03 -2.33245071e-03
  1.38894247e-03 -2.85351856e-05  1.13063650e-02  9.16966829e-03
 -5.79858503e-03 -7.12201253e-03  0.00000000e+00 -2.16201446e-03
  1.22080474e-02  1.63405432e-02 -4.12279773e-03  3.20809781e-03
 -1.44800838e-03 -2.69734855e-03 -2.72772289e-02  3.35716867e-03
 -2.70870159e-02  5.93017368e-03  3.35716867e-03  8.92721971e-03
  8.12261028e-03  1.23818860e-02 -1.69056236e-03 -6.71731878e-03
 -7.99086981e-03 -3.69432845e-03  6.44714524e-03 -5.99299427e-03
  3.35716867e-03 -8.10174005e-03 -1.46847504e-02 -1.20187865e-02
 -2.68972937e-04 -4.96335213e-03 -1.18795981e-02 -6.26884586e-03
 -5.80431916e-03  1.55612180e-02  1.74308587e-02 -2.80085074e-03
  5.40853493e-04  9.06895649e-03  6.46951796e-04 -5.56440866e-03
 -3.45943425e-03 -1.20246546e-02  1.41288859e-02  5.53042990e-03
  2.57010677e-05  0.00000000e+00 -1.06212860e-03 -8.07780134e-03
 -3.26582852e-02 -2.89757146e-02 -9.36677784e-03  5.01111100e-03
  8.34009769e-03 -3.71541441e-03 -9.84773670e-03  1.89489977e-02
  6.83712518e-04  9.45510643e-03 -1.37968270e-02 -1.88813031e-03
  8.07657857e-03  4.16378539e-03 -1.64151795e-03 -3.58525400e-05
  3.75358304e-03 -2.23541856e-03 -3.25878248e-03 -3.25878248e-03
 -1.38805945e-03 -4.89592282e-03 -1.49599279e-02  3.51011748e-03
 -3.25878248e-03 -2.62979174e-03 -5.17363567e-03 -8.78926870e-03
 -3.52453972e-03 -3.25878248e-03  4.47104295e-03 -9.34922825e-03
  5.68878070e-03  6.17906912e-03  3.48249055e-03  4.30756583e-03
  1.90268754e-02  4.54455503e-04  1.22264470e-03  1.41150158e-02
```

```
 2.68348661e-03  3.26133828e-03  4.27890349e-03  5.80804059e-03
 4.67909410e-03  1.99649841e-02  1.39602663e-02]
```

In [83]:

```python
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

metric2 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R square train :",r2_train_lr)
metric2.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R square test :",r2_test_lr)
metric2.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print("Rss train :",rss1_lr)
metric2.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print("Rss test :",rss2_lr)
metric2.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("MSE train:",mse_train_lr)
metric2.append(mse_train_lr)

print("RMSE train:",mse_train_lr**0.5)
metric2.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("MSE test :",mse_test_lr)
metric2.append(mse_test_lr)

print("RMSE test :",mse_test_lr**0.5)
metric2.append(mse_test_lr**0.5)
```

```
R square train : 0.9537251879652039
R square test : 0.8721002342302973
Rss train : 7.426653074278145
Rss test : 9.217430166567063
MSE train: 0.007273901150125509
RMSE train: 0.08528716873085605
MSE test : 0.021044361110883706
RMSE test : 0.1450667470886547
```

In [84]:

```python
lasso = Lasso(alpha=0.002)

lasso.fit(X_train, y_train)
print(lasso.coef_)
```

```
[-1.81729275e-02  0.00000000e+00  1.79277302e-02  7.74425269e-02
  3.87982651e-02  2.25123786e-02  0.00000000e+00  4.67153891e-03
  0.00000000e+00 -0.00000000e+00  2.69621334e-02  2.57148749e-03
  0.00000000e+00 -2.47926646e-03  1.37906627e-01  2.51074610e-02
  0.00000000e+00  7.70660788e-03  3.60084624e-03  4.50337940e-03
 -9.19304934e-03  0.00000000e+00 -0.00000000e+00  3.89809306e-02
  8.96031751e-03  1.63425190e-03  2.42212987e-03  2.68895257e-03
  6.68041484e-03 -9.57214431e-03 -0.00000000e+00 -0.00000000e+00
 -4.78551138e-02  5.01167552e-03  4.60324406e-03  2.06749814e-02
  0.00000000e+00  4.03643376e-03  4.70546249e-04 -6.91927177e-03
  0.00000000e+00  0.00000000e+00  0.00000000e+00  2.63176396e-03
 -2.22199809e-03  7.76956318e-03 -2.96609919e-03 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -2.69510428e-03  3.35017403e-03  1.08748521e-02  0.00000000e+00
  2.50871647e-02 -1.22856070e-02 -3.52464177e-04 -9.53433420e-03
 -1.15827327e-02 -1.72562306e-03  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  6.89846544e-03  2.03410448e-02 -1.83794774e-03
  3.05061755e-03 -1.00883910e-03  0.00000000e+00  2.06573452e-02
  1.03781719e-02  0.00000000e+00  4.61417406e-03 -0.00000000e+00
  1.87315016e-02  1.81775880e-03  2.00872944e-03 -3.37937654e-03
  2.87473979e-03  0.00000000e+00  1.40907286e-03 -0.00000000e+00
  2.63014080e-04  3.69358812e-03 -5.15556893e-02 -1.44177848e-06
 -5.32293402e-04  0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -1.04596812e-02 -6.13790994e-04  3.32144207e-05 -0.00000000e+00
 -1.61177628e-03 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -2.27953513e-03  0.00000000e+00  0.00000000e+00
  3.42816278e-03  2.22002840e-04  1.27549406e-01  2.48032041e-02
  2.46098825e-02  2.45774608e-02  8.32408493e-02  5.22931979e-02
  6.65765980e-02 -0.00000000e+00 -7.88907980e-03  1.09862051e-02
 -0.00000000e+00  0.00000000e+00 -1.57675981e-03  0.00000000e+00
  1.12666659e-03 -0.00000000e+00  0.00000000e+00 -1.44017786e-03
  5.75479865e-03 -5.30971117e-03  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00  3.86881622e-03
 -0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -3.69354277e-03  0.00000000e+00
 -0.00000000e+00 -5.13181190e-03  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -5.01913992e-03  0.00000000e+00 -3.83620388e-03
 -1.62091502e-03 -0.00000000e+00  0.00000000e+00  2.34822021e-03
  0.00000000e+00  8.88572800e-03 -0.00000000e+00  1.70728706e-03
 -0.00000000e+00  0.00000000e+00 -1.15615385e-02 -0.00000000e+00
 -8.73746264e-03  1.45194190e-03 -0.00000000e+00  0.00000000e+00
  5.10344149e-03  1.66491439e-02 -0.00000000e+00 -4.70605819e-03
 -2.59199501e-03 -0.00000000e+00  8.69301957e-03 -0.00000000e+00
 -0.00000000e+00 -3.37195017e-03 -1.22854754e-02 -6.18678611e-03
  1.68266989e-03 -0.00000000e+00 -2.72305489e-03 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  5.50156850e-03 -3.83825505e-03
 -2.43876551e-03  1.49085518e-04  0.00000000e+00 -1.34770746e-03
 -0.00000000e+00 -8.31609179e-03  1.63422699e-02  2.02219763e-03
 -0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.87276204e-04
 -8.00903310e-03 -5.20464973e-03 -8.94829883e-03  0.00000000e+00
  0.00000000e+00 -8.17352786e-04 -7.85293092e-03  7.00626913e-03
  0.00000000e+00  2.63590329e-03 -1.82933178e-02 -1.67084493e-03
  0.00000000e+00  4.32253678e-03 -4.44889518e-04  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -4.03065630e-03 -6.79922091e-03  2.69536145e-03
 -0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.64119341e-03
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  3.63965538e-03  3.37453105e-03  2.10136440e-03  1.61987324e-03
  1.10780478e-02 -0.00000000e+00  0.00000000e+00  2.25643001e-02
```

```
  0.00000000e+00  0.00000000e+00  9.70378752e-04 -0.00000000e+00
  8.52051642e-04  1.55748089e-02  0.00000000e+00]
```

In [85]:

```python
# Lets calculate some metrics such as R2 score, RSS and RMSE

y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

metric3 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R square train :",r2_train_lr)
metric3.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R square test :",r2_test_lr)
metric3.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print("Rss train :",rss1_lr)
metric3.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print("Rss test :",rss2_lr)
metric3.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("MSE train:",mse_train_lr)
metric3.append(mse_train_lr)

print("RMSE train:",mse_train_lr**0.5)
metric3.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("MSE test :",mse_test_lr)
metric3.append(mse_test_lr)

print("RMSE test :",mse_test_lr**0.5)
metric3.append(mse_test_lr**0.5)
```

```
R square train : 0.938111442795243
R square test : 0.8714626375913678
Rss train : 9.932505901520138
Rss test : 9.26338023112271
MSE train: 0.009728213419706306
RMSE train: 0.09863170595557144
MSE test : 0.021149269934070115
RMSE test : 0.14542788568245815
```

In [86]:

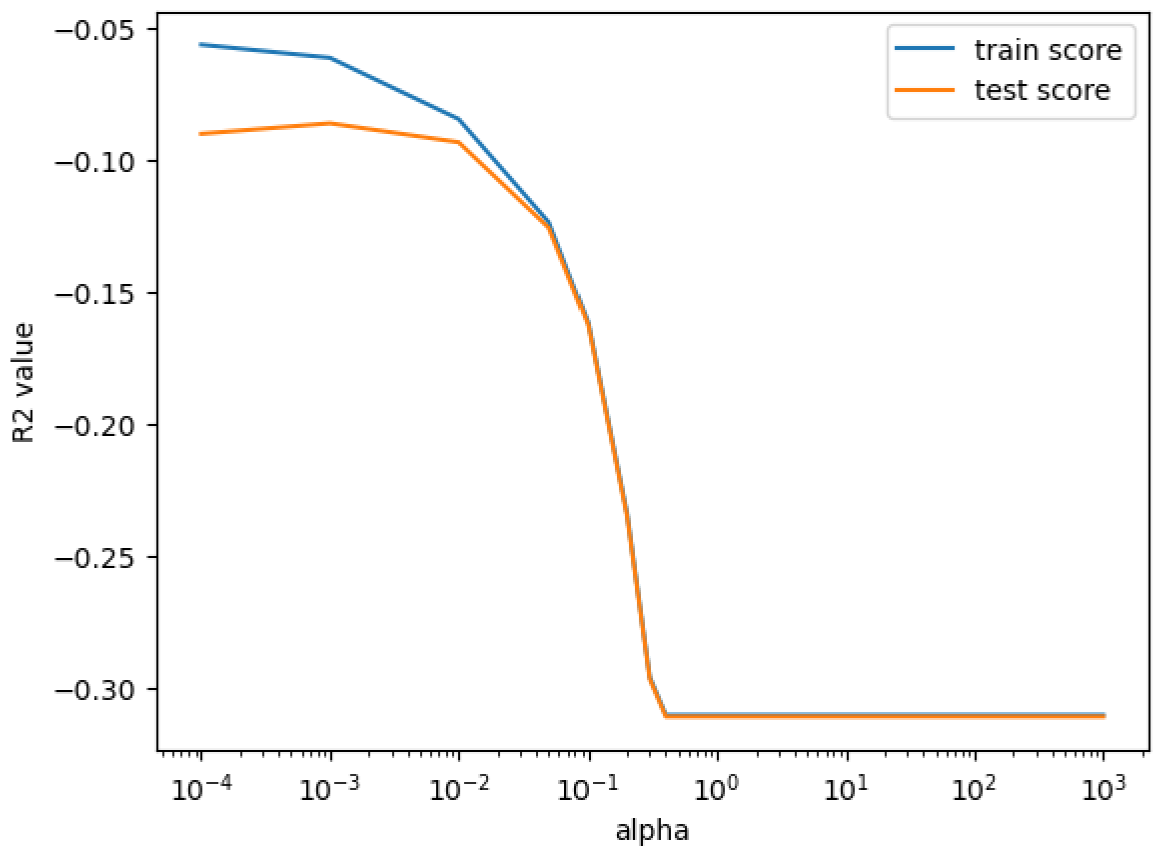
```python
# Creating a table which contain all the metrics

lr_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS (Test)'

lr_metric = pd.DataFrame(lr_table ,columns = ['Metric'] )

rg_metric = pd.Series(metric2, name = 'Ridge Regression')
ls_metric = pd.Series(metric3, name = 'Lasso Regression')

final_metric = pd.concat([lr_metric, rg_metric, ls_metric], axis = 1)

final_metric
```

Out[86]:

	Metric	Ridge Regression	Lasso Regression
0	R2 Score (Train)	0.953725	0.938111
1	R2 Score (Test)	0.872100	0.871463
2	RSS (Train)	7.426653	9.932506
3	RSS (Test)	9.217430	9.263380
4	MSE (Train)	0.007274	0.009728
5	MSE (Test)	0.085287	0.098632
6	RMSE (Train)	0.021044	0.021149
7	RMSE (Test)	0.145067	0.145428

In [87]:

```
#plotting
model_cv_results = pd.DataFrame(model_cv.cv_results_)
plt.plot(model_cv_results['param_alpha'], model_cv_results['mean_train_score'])
plt.plot(model_cv_results['param_alpha'], model_cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.xscale('log')
plt.ylabel('R2 value')
plt.legend(['train score', 'test score'])
plt.show()
```



In [88]:

```
betas = pd.DataFrame(index=X.columns)
betas.rows = X.columns
betas['Linear'] = lm.coef_
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
```

In [89]:
```
betas['Ridge'].sort_values(ascending=False)[:10]
```

Out[89]:
```
RoofMatl_CompShg    0.217453
RoofMatl_Tar&Grv    0.144053
RoofMatl_WdShngl    0.109205
MSZoning_RL         0.108055
RoofMatl_WdShake    0.087417
MSZoning_RM         0.080832
GrLivArea           0.074089
OverallQual         0.060978
MSZoning_FV         0.054610
RoofMatl_Membran    0.049942
Name: Ridge, dtype: float64
```

In [90]:
```
betas['Lasso'].sort_values(ascending=False)[:10]
```

Out[90]:
```
GrLivArea           0.137907
RoofMatl_CompShg    0.127549
RoofMatl_Tar&Grv    0.083241
OverallQual         0.077443
RoofMatl_WdShngl    0.066577
RoofMatl_WdShake    0.052293
GarageCars          0.038981
OverallCond         0.038798
TotalBsmtSF         0.026962
BsmtFullBath        0.025107
Name: Lasso, dtype: float64
```

3. Removing the top 5 variables

In [91]:
```
# top 5 variables are : RoofMatl_CompShg, RoofMatl_Tar&Grv, GrLivArea, RoofMatl_WdShng

# storing them in a variable
top5 = ['RoofMatl_CompShg', 'RoofMatl_Tar&Grv', 'GrLivArea', 'RoofMatl_WdShngl', 'Roof
top5
```

Out[91]:
```
['RoofMatl_CompShg',
 'RoofMatl_Tar&Grv',
 'GrLivArea',
 'RoofMatl_WdShngl',
 'RoofMatl_WdShake']
```

In [92]:
```
# printing X train and test set
X_train.head()
```

Out[92]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	MasVnrArea	B
210	-0.657071	-0.115302	-0.473765	-0.779861	0.383154	-1.694350	-0.558025	
318	0.035976	0.926898	-0.056845	0.649651	-0.533005	0.390956	0.809137	
239	-0.195040	-0.794998	-0.169324	-0.065105	-1.449164	-1.694350	-0.558025	
986	-0.195040	-0.477806	-0.502297	-0.065105	2.215472	0.875911	-0.558025	
1416	3.039179	-0.432493	0.082905	-1.494617	0.383154	-1.694350	-0.558025	

5 rows × 243 columns

In [93]:
```
X_test.head()
```

Out[93]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	MasVnrArea	B
1436	-0.888086	-0.432493	-0.144189	-1.494617	0.383154	-0.675945	-0.558025	
57	0.035976	0.881585	0.112505	0.649651	-0.533005	0.924407	-0.558025	
780	-0.888086	-0.296554	-0.253368	0.649651	-0.533005	0.536443	-0.355087	
382	0.035976	0.428455	-0.120412	0.649651	-0.533005	1.021398	-0.558025	
1170	0.498007	0.292515	-0.058786	-0.065105	0.383154	-0.384972	-0.558025	

5 rows × 243 columns

In [94]:
```
X_train1 = X_train.drop(top5, axis = 1)
X_test1 = X_test.drop(top5, axis = 1)
```

In [95]:
```
lasso = Lasso()

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

model_cv.fit(X_train1, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[95]:

```
▸ GridSearchCV
▸ estimator: Lasso
    ▸ Lasso
```

In [96]:
```
model_cv.best_params_
```

Out[96]: {'alpha': 0.001}

In [97]:
```
#Fitting Lasso model for alpha = 0.001 and printing coefficients which have been penal

alpha =0.001

lasso = Lasso(alpha=alpha)

lasso.fit(X_train1, y_train)
```

Out[97]:

▾ Lasso

Lasso(alpha=0.001)

In [98]:
```
# Lets calculate some metrics such as R2 score, RSS and RMSE

y_pred_train = lasso.predict(X_train1)
y_pred_test = lasso.predict(X_test1)

metric3 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R square train :",r2_train_lr)
metric3.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R square test :",r2_test_lr)
metric3.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print("Rss train :",rss1_lr)
metric3.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print("Rss test :",rss2_lr)
metric3.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("MSE train:",mse_train_lr)
metric3.append(mse_train_lr)

print("RMSE train:",mse_train_lr**0.5)
metric3.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("MSE test :",mse_test_lr)
metric3.append(mse_test_lr)

print("RMSE test :",mse_test_lr**0.5)
metric3.append(mse_test_lr**0.5)
```

```
R square train : 0.9332528937199062
R square test : 0.8710398117759115
Rss train : 10.712255334100195
Rss test : 9.293852276189687
MSE train: 0.010491924910969829
RMSE train: 0.10243009768114951
MSE test : 0.021218840813218464
RMSE test : 0.1456668830352955
```

In [99]:

```
betas = pd.DataFrame(index=X_train1.columns)
betas.rows = X_train1.columns

betas['Lasso'] = lasso.coef_
```

In [100…

```
betas['Lasso'].sort_values(ascending=False)[:10]
```

Out[100]:

```
2ndFlrSF                0.100230
1stFlrSF                0.099158
MSZoning_RL             0.084011
OverallQual             0.072898
MSZoning_RM             0.057104
GarageCars              0.041877
MSZoning_FV             0.039181
OverallCond             0.033899
Neighborhood_Crawfor    0.027526
BsmtFullBath            0.025117
Name: Lasso, dtype: float64
```

In [ ]: