```python
import sys
import os
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import
train_test_split,GridSearchCV,KFold,cross_val_score
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.feature_selection import RFE
from sklearn.metrics import r2_score

warnings.filterwarnings('ignore')
%matplotlib inline

# Perform basic checks in the dataset
house_price = pd.read_csv('train.csv')
```

```python
house_price.shape
```

```
(1460, 81)
```

```python
house_price.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
```

```
house_price.head()
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave   NaN      Reg

1   2          20       RL         80.0     9600   Pave   NaN      Reg

2   3          60       RL         68.0    11250   Pave   NaN      IR1

3   4          70       RL         60.0     9550   Pave   NaN      IR1

4   5          60       RL         84.0    14260   Pave   NaN      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000
```

```
[5 rows x 81 columns]
```

```
house_price.describe([0.25,0.50,0.75,0.99])
```

```
                Id    MSSubClass  LotFrontage        LotArea  
OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  
1460.000000   
mean    730.500000    56.897260    70.049958   10516.828082  
6.099315   
std     421.610009    42.300571    24.284752    9981.264932  
1.382997   
min       1.000000    20.000000    21.000000    1300.000000  
1.000000   
25%     365.750000    20.000000    59.000000    7553.500000  
5.000000   
50%     730.500000    50.000000    69.000000    9478.500000  
6.000000   
75%    1095.250000    70.000000    80.000000   11601.500000  
7.000000   
99%    1445.410000   190.000000   141.000000   37567.640000  
10.000000   
max    1460.000000   190.000000   313.000000  215245.000000  
10.000000   

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea  
BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000  
1460.000000  ...   
mean      5.575342  1971.267808   1984.865753   103.685262  
443.639726  ...   
std       1.112799    30.202904     20.645407   181.066207  
456.098091  ...   
min       1.000000  1872.000000   1950.000000     0.000000  
0.000000  ...   
25%       5.000000  1954.000000   1967.000000     0.000000  
0.000000  ...   
50%       5.000000  1973.000000   1994.000000     0.000000  
383.500000  ...   
75%       6.000000  2000.000000   2004.000000   166.000000  
712.250000  ...   
99%       9.000000  2009.000000   2009.000000   791.920000  
1572.410000  ...   
max       9.000000  2010.000000   2010.000000  1600.000000  
5644.000000  ...   

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  
ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000  
```

```
1460.000000
mean     94.244521    46.660274      21.954110     3.409589
15.060959
std     125.338794    66.256028      61.119149    29.317331
55.757415
min       0.000000     0.000000       0.000000     0.000000
0.000000
25%       0.000000     0.000000       0.000000     0.000000
0.000000
50%       0.000000    25.000000       0.000000     0.000000
0.000000
75%     168.000000    68.000000       0.000000     0.000000
0.000000
99%     505.460000   285.820000     261.050000   168.000000
268.050000
max     857.000000   547.000000     552.000000   508.000000
480.000000

          PoolArea       MiscVal       MoSold       YrSold
SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000
1460.000000
mean      2.758904     43.489041     6.321918  2007.815753
180921.195890
std      40.177307    496.123024     2.703626     1.328095
79442.502883
min       0.000000      0.000000     1.000000  2006.000000
34900.000000
25%       0.000000      0.000000     5.000000  2007.000000
129975.000000
50%       0.000000      0.000000     6.000000  2008.000000
163000.000000
75%       0.000000      0.000000     8.000000  2009.000000
214000.000000
99%       0.000000    700.000000    12.000000  2010.000000
442567.010000
max     738.000000  15500.000000    12.000000  2010.000000
755000.000000

[9 rows x 38 columns]
```

```
#Check the dataset for the amount of null present
round(house_price.isnull().sum()/len(house_price.index),2).sort_values
(ascending=False).head(18)
```

```
PoolQC          1.00
MiscFeature     0.96
Alley           0.94
Fence           0.81
FireplaceQu     0.47
```

```
LotFrontage     0.18
GarageYrBlt     0.06
GarageFinish    0.06
GarageType      0.06
GarageQual      0.06
GarageCond      0.06
BsmtExposure    0.03
BsmtQual        0.03
BsmtCond        0.03
BsmtFinType2    0.03
BsmtFinType1    0.03
MasVnrType      0.01
MasVnrArea      0.01
dtype: float64
```

```
#Considering 10% as my threshold and dropping the column having more
than the threshold
round(house_price.isnull().sum()/len(house_price.index),2)
[round(house_price.isnull().sum()/

len(house_price.index),2).values>0.10]
```

```
LotFrontage    0.18
Alley          0.94
FireplaceQu    0.47
PoolQC         1.00
Fence          0.81
MiscFeature    0.96
dtype: float64
```

```
house_price =
house_price.drop(['LotFrontage','Alley','FireplaceQu','PoolQC','Fence'
,'MiscFeature','MoSold'],axis='columns')
```

```
# Check the columns where the missing values are between 0-10%
round(house_price.isnull().sum()/len(house_price.index),2)
[round(house_price.isnull().sum()/

len(house_price.index),2).values>0.00]
```

```
MasVnrType      0.01
MasVnrArea      0.01
BsmtQual        0.03
BsmtCond        0.03
BsmtExposure    0.03
BsmtFinType1    0.03
BsmtFinType2    0.03
GarageType      0.06
GarageYrBlt     0.06
GarageFinish    0.06
GarageQual      0.06
```

```
GarageCond      0.06
dtype: float64
```

```
house_price['YearBuilt_Old'] = house_price.YearBuilt.max()-
house_price.YearBuilt
house_price['YearRemodAdd_Old'] = house_price.YearRemodAdd.max()-
house_price.YearRemodAdd
house_price['GarageYrBlt_Old'] = house_price.GarageYrBlt.max()-
house_price.GarageYrBlt
house_price['YrSold_Old'] = house_price.YrSold.max()-
house_price.YrSold
house_price[['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold','YearBu
ilt_Old','YearRemodAdd_Old',
             'GarageYrBlt_Old','YrSold_Old']].sample(8)
```

```
      YearBuilt  YearRemodAdd  GarageYrBlt  YrSold  YearBuilt_Old  \
581        2008          2009       2009.0    2009              2
1287       1964          1964       1964.0    2006             46
1196       2006          2006       2006.0    2006              4
586        1918          2000       1961.0    2008             92
436        1920          1950       1990.0    2006             90
481        2003          2004       2003.0    2006              7
932        2006          2006       2006.0    2007              4
1168       1935          1986       1935.0    2008             75

      YearRemodAdd_Old  GarageYrBlt_Old  YrSold_Old
581                  1              1.0           1
1287                46             46.0           4
1196                 4              4.0           4
586                 10             49.0           2
436                 60             20.0           4
481                  6              7.0           4
932                  4              4.0           3
1168                24             75.0           2
```

```
# Drop the actual year columns
house_price =
house_price.drop(['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold'],a
xis='columns')

#Imputing GarageYrBlt_Old with -1 as these house do not have garage
house_price.MasVnrType.fillna('None',inplace=True)
house_price.MasVnrArea.fillna(house_price.MasVnrArea.mean(),inplace=Tr
ue)
house_price.BsmtQual.fillna('TA',inplace=True)
house_price.BsmtCond.fillna('TA',inplace=True)
house_price.BsmtExposure.fillna('No',inplace=True)
house_price.BsmtFinType1.fillna('Unf',inplace=True)
house_price.BsmtFinType2.fillna('Unf',inplace=True)
house_price.GarageType.fillna('Attchd',inplace=True)
house_price.GarageYrBlt_Old.fillna(-1,inplace=True)
```

```
house_price.GarageFinish.fillna('Unf',inplace=True)
house_price.GarageQual.fillna('TA',inplace=True)
house_price.GarageCond.fillna('TA',inplace=True)
```

```
house_price.Street.value_counts()
house_price.Utilities.value_counts()
```

```
AllPub    1459
NoSeWa       1
Name: Utilities, dtype: int64
```

```
house_price = house_price.drop(['Street','Utilities'],axis='columns')
```

```
house_price = house_price.drop('Id',axis='columns')
```

```
house_price[list(house_price.dtypes[house_price.dtypes!
='object'].index)].describe()
```

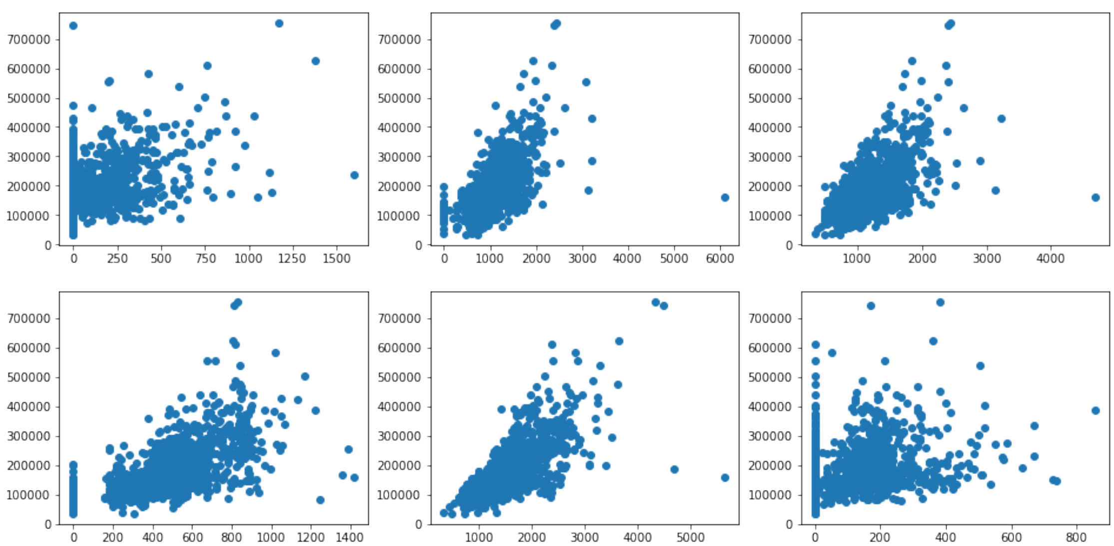
```
        MSSubClass        LotArea  OverallQual  OverallCond
MasVnrArea  \
count  1460.000000    1460.000000  1460.000000  1460.000000
1460.000000
mean     56.897260   10516.828082     6.099315     5.575342
103.685262
std      42.300571    9981.264932     1.382997     1.112799
180.569112
min      20.000000    1300.000000     1.000000     1.000000
0.000000
25%      20.000000    7553.500000     5.000000     5.000000
0.000000
50%      50.000000    9478.500000     6.000000     5.000000
0.000000
75%      70.000000   11601.500000     7.000000     6.000000
164.250000
max     190.000000  215245.000000    10.000000     9.000000
1600.000000

        BsmtFinSF1   BsmtFinSF2    BsmtUnfSF  TotalBsmtSF     1stFlrSF
...  \
count  1460.000000  1460.000000  1460.000000  1460.000000  1460.000000
...
mean    443.639726    46.549315   567.240411  1057.429452  1162.626712
...
std     456.098091   161.319273   441.866955   438.705324   386.587738
...
min       0.000000     0.000000     0.000000     0.000000   334.000000
...
25%       0.000000     0.000000   223.000000   795.750000   882.000000
...
50%     383.500000     0.000000   477.500000   991.500000  1087.000000
...
```

```
75%      712.250000     0.000000   808.000000  1298.250000  1391.250000
...
max     5644.000000  1474.000000  2336.000000  6110.000000  4692.000000
...

       EnclosedPorch    3SsnPorch  ScreenPorch     PoolArea
MiscVal  \
count    1460.000000  1460.000000  1460.000000  1460.000000
1460.000000
mean       21.954110     3.409589    15.060959     2.758904
43.489041
std        61.119149    29.317331    55.757415    40.177307
496.123024
min         0.000000     0.000000     0.000000     0.000000
0.000000
25%         0.000000     0.000000     0.000000     0.000000
0.000000
50%         0.000000     0.000000     0.000000     0.000000
0.000000
75%         0.000000     0.000000     0.000000     0.000000
0.000000
max       552.000000   508.000000   480.000000   738.000000
15500.000000

           SalePrice  YearBuilt_Old  YearRemodAdd_Old  GarageYrBlt_Old
\
count    1460.000000    1460.000000       1460.000000      1460.000000

mean   180921.195890      38.732192         25.134247        29.691096

std     79442.502883      30.202904         20.645407        25.121824

min     34900.000000       0.000000          0.000000        -1.000000

25%    129975.000000      10.000000          6.000000         7.000000

50%    163000.000000      37.000000         16.000000        25.500000

75%    214000.000000      56.000000         43.000000        48.000000

max    755000.000000     138.000000         60.000000       110.000000


        YrSold_Old
count  1460.000000
mean      2.184247
std       1.328095
min       0.000000
25%       1.000000
```

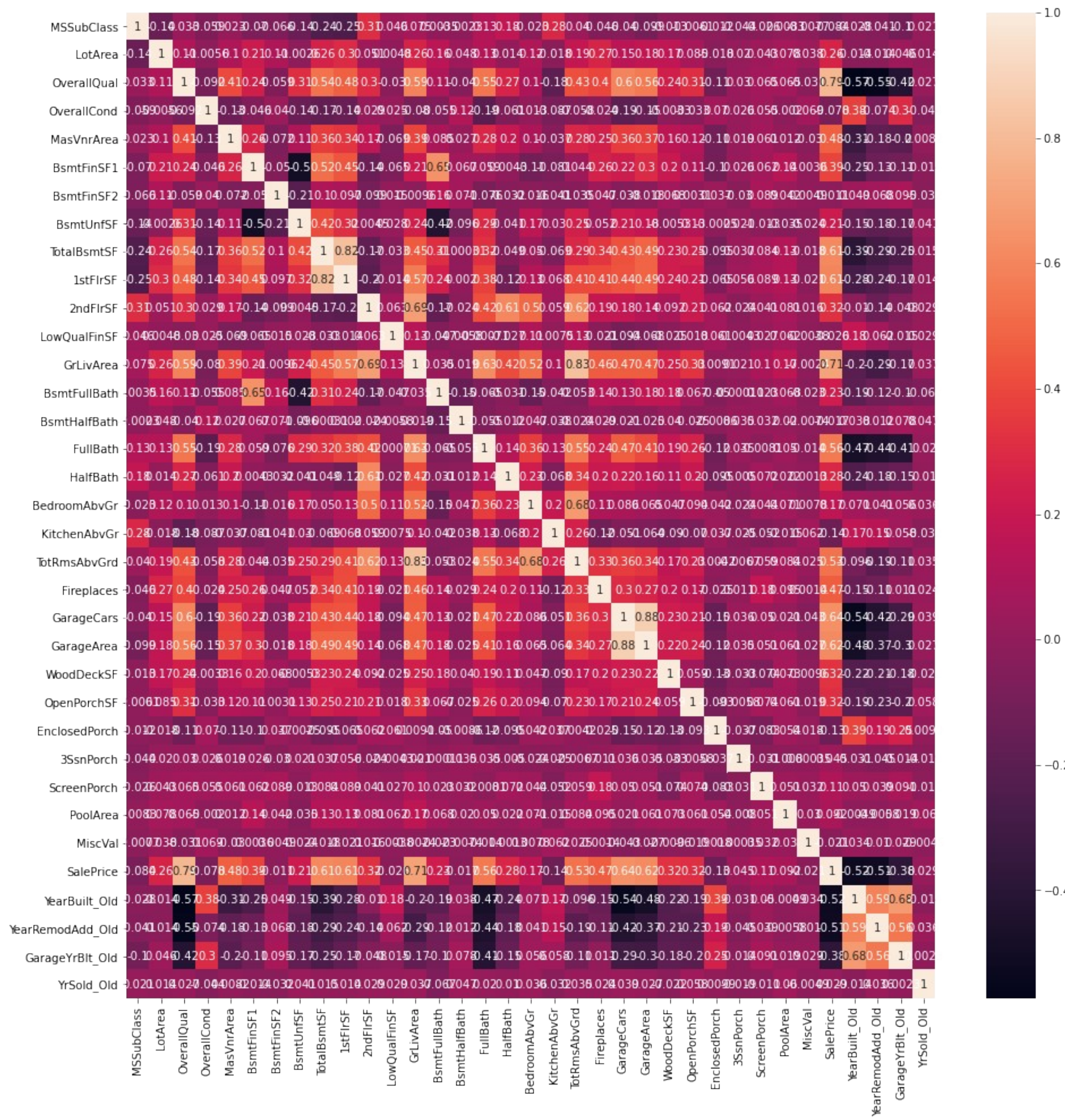
```
50%        2.000000
75%        3.000000
max        4.000000

[8 rows x 35 columns]
```

```python
# Plot some graph for the EDA purpose
plt.figure(figsize=(16,8))
plt.subplot(2,3,1)
plt.scatter(house_price.MasVnrArea,house_price.SalePrice)
plt.subplot(2,3,2)
plt.scatter(house_price.TotalBsmtSF,house_price.SalePrice)
plt.subplot(2,3,3)
plt.scatter(house_price['1stFlrSF'],house_price.SalePrice)
plt.subplot(2,3,4)
plt.scatter(house_price['GarageArea'],house_price.SalePrice)
plt.subplot(2,3,5)
plt.scatter(house_price['GrLivArea'],house_price.SalePrice)
plt.subplot(2,3,6)
plt.scatter(house_price['WoodDeckSF'],house_price.SalePrice)
```

```
<matplotlib.collections.PathCollection at 0x1cd7496e700>
```




```python
#Plotting the heatmap to check the correlation between variables
plt.figure(figsize=(16,16))
sns.heatmap(house_price[list(house_price.dtypes[house_price.dtypes!
='object'].index)].corr(),annot=True)
plt.show()
```



```
house_price.shape
```

```
(1460, 71)
```

```
num_col = list(house_price.dtypes[house_price.dtypes !
='object'].index)
num_col =
['LotArea','MasVnrArea','BsmtFinSF1','BsmtFinSF2','TotalBsmtSF','1stFl
rSF','GrLivArea','OpenPorchSF',
           'EnclosedPorch','3SsnPorch',
           'ScreenPorch' ,'PoolArea','MiscVal','SalePrice']
def drop_outliers(x):
    list = []
    for col in num_col:
        Q1 = x[col].quantile(.25)
        Q3 = x[col].quantile(.99)
```

```
        IQR = Q3-Q1
        x =  x[(x[col] >= (Q1-(1.5*IQR))) & (x[col] <=
(Q3+(1.5*IQR)))]
    return x

house_price = drop_outliers(house_price)

house_price.shape

(1441, 71)

house_price[list(house_price.dtypes[house_price.dtypes=='object'].inde
x)].head()

  MSZoning LotShape LandContour LotConfig LandSlope Neighborhood
Condition1  \
0       RL      Reg         Lvl    Inside       Gtl      CollgCr
Norm
1       RL      Reg         Lvl       FR2       Gtl      Veenker
Feedr
2       RL      IR1         Lvl    Inside       Gtl      CollgCr
Norm
3       RL      IR1         Lvl    Corner       Gtl      Crawfor
Norm
4       RL      IR1         Lvl       FR2       Gtl      NoRidge
Norm

  Condition2 BldgType HouseStyle  ... Electrical KitchenQual
Functional  \
0       Norm     1Fam     2Story  ...      SBrkr          Gd
Typ
1       Norm     1Fam     1Story  ...      SBrkr          TA
Typ
2       Norm     1Fam     2Story  ...      SBrkr          Gd
Typ
3       Norm     1Fam     2Story  ...      SBrkr          Gd
Typ
4       Norm     1Fam     2Story  ...      SBrkr          Gd
Typ

  GarageType GarageFinish GarageQual GarageCond PavedDrive SaleType  \
0     Attchd          RFn         TA         TA          Y       WD
1     Attchd          RFn         TA         TA          Y       WD
2     Attchd          RFn         TA         TA          Y       WD
3     Detchd          Unf         TA         TA          Y       WD
4     Attchd          RFn         TA         TA          Y       WD

  SaleCondition
0        Normal
1        Normal
2        Normal
```

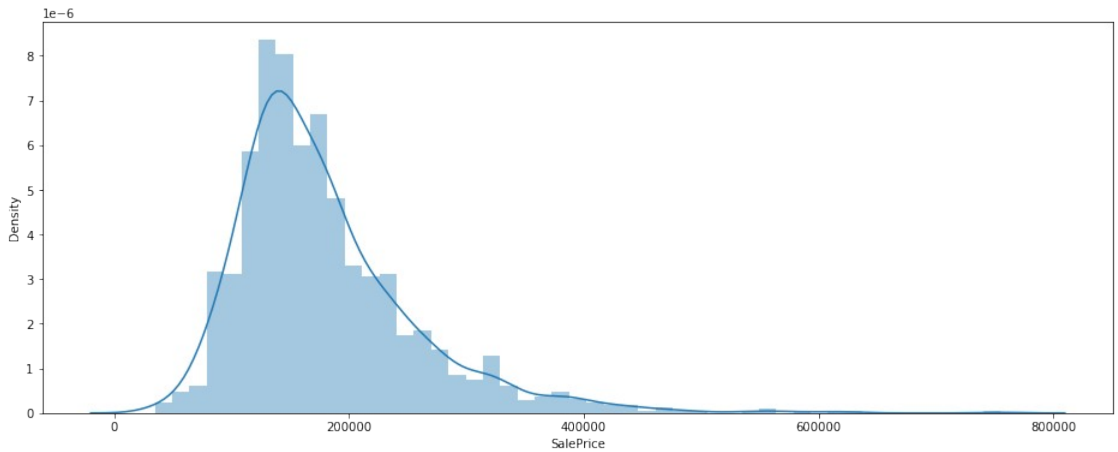
```
3       Abnorml
4        Normal

[5 rows x 36 columns]

house_price[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposu
re','BsmtFinType1','BsmtFinType2',
            'HeatingQC','CentralAir',
'KitchenQual','GarageFinish','GarageQual','GarageCond',
             'ExterCond','LotShape']].head()

  LandSlope ExterQual BsmtQual BsmtCond BsmtExposure BsmtFinType1  \
0       Gtl        Gd       Gd       TA           No          GLQ
1       Gtl        TA       Gd       TA           Gd          ALQ
2       Gtl        Gd       Gd       TA           Mn          GLQ
3       Gtl        TA       TA       Gd           No          ALQ
4       Gtl        Gd       Gd       TA           Av          GLQ

  BsmtFinType2 HeatingQC CentralAir KitchenQual GarageFinish
GarageQual  \
0          Unf        Ex          Y          Gd          RFn
TA
1          Unf        Ex          Y          TA          RFn
TA
2          Unf        Ex          Y          Gd          RFn
TA
3          Unf        Gd          Y          Gd          Unf
TA
4          Unf        Ex          Y          Gd          RFn
TA

  GarageCond ExterCond LotShape
0         TA        TA      Reg
1         TA        TA      Reg
2         TA        TA      IR1
3         TA        TA      IR1
4         TA        TA      IR1

#As the house prices are ordinal in nature we are mapping it to
different numbers
house_price['LandSlope'] =
house_price.LandSlope.map({'Gtl':0,'Mod':1,'Sev':2})
house_price['ExterQual'] =
house_price.ExterQual.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
house_price['BsmtQual'] =
house_price.BsmtQual.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5})
house_price['BsmtCond'] =
house_price.BsmtCond.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5})
house_price['BsmtExposure'] =
house_price.BsmtExposure.map({'NA':0,'No':1,'Mn':2,'Av':3,'Gd':4})
```

```
house_price['BsmtFinType1'] =
house_price.BsmtFinType1.map({'NA':0,'Unf':1,'LwQ':2,'Rec':3,'BLQ':4,'
ALQ':5,'GLQ':6})
house_price['BsmtFinType2'] =
house_price.BsmtFinType2.map({'NA':0,'Unf':1,'LwQ':2,'Rec':3,'BLQ':4,'
ALQ':5,'GLQ':6})
house_price['HeatingQC'] =
house_price.HeatingQC.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
house_price['CentralAir'] = house_price.CentralAir.map({'N':0,'Y':1})
house_price['KitchenQual'] =
house_price.KitchenQual.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
house_price['GarageFinish'] =
house_price.GarageFinish.map({'NA':0,'Unf':1,'RFn':2,'Fin':3})
house_price['GarageQual'] =
house_price.GarageQual.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5}
)
house_price['GarageCond'] =
house_price.GarageCond.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5}
)
house_price['ExterCond'] =
house_price.ExterCond.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
house_price['LotShape'] =
house_price.LotShape.map({'IR1':0,'IR2':1,'IR3':2,'Reg':3})

house_price[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposu
re','BsmtFinType1','BsmtFinType2',
             'HeatingQC','CentralAir',
'KitchenQual','GarageFinish','GarageQual','GarageCond',
              'ExterCond','LotShape']].head()
```

```
   LandSlope  ExterQual  BsmtQual  BsmtCond  BsmtExposure
BsmtFinType1  \
0          0          3         4         3             1
6
1          0          2         4         3             4
5
2          0          3         4         3             2
6
3          0          2         3         4             1
5
4          0          3         4         3             3
6

   BsmtFinType2  HeatingQC  CentralAir  KitchenQual  GarageFinish
GarageQual  \
0             1          4           1            3             2
3
1             1          4           1            2             2
3
2             1          4           1            3             2
```

```
3
3              1          3           1            3             1
3
4              1          4           1            3             2
3

   GarageCond  ExterCond  LotShape
0           3          2         3
1           3          2         3
2           3          2         0
3           3          2         0
4           3          2         0
```

```
# Creating and joining dummy column with actual dataset
dummy_col = 
pd.get_dummies(house_price[['MSZoning','LandContour','LotConfig','Neig
hborhood','Condition1','Condition2','BldgType',
             'HouseStyle','RoofStyle','RoofMatl','Exterior1st',
'Exterior2nd','MasVnrType','Foundation',

'Heating','Electrical','Functional','GarageType','PavedDrive','SaleTyp
e','SaleCondition']],
                           drop_first=True)

house_price = pd.concat([house_price,dummy_col],axis='columns')

house_price = 
house_price.drop(['MSZoning','LandContour','LotConfig','Neighborhood',
'Condition1','Condition2','BldgType',
             'HouseStyle','RoofStyle','RoofMatl','Exterior1st',
'Exterior2nd','MasVnrType','Foundation',

'Heating','Electrical','Functional','GarageType','PavedDrive','SaleTyp
e','SaleCondition'],axis='columns')

# Checking the distribution of our target variable before Scaling and 
Splitting
plt.figure(figsize=(16,6))
sns.distplot(house_price.SalePrice)
plt.show()
```



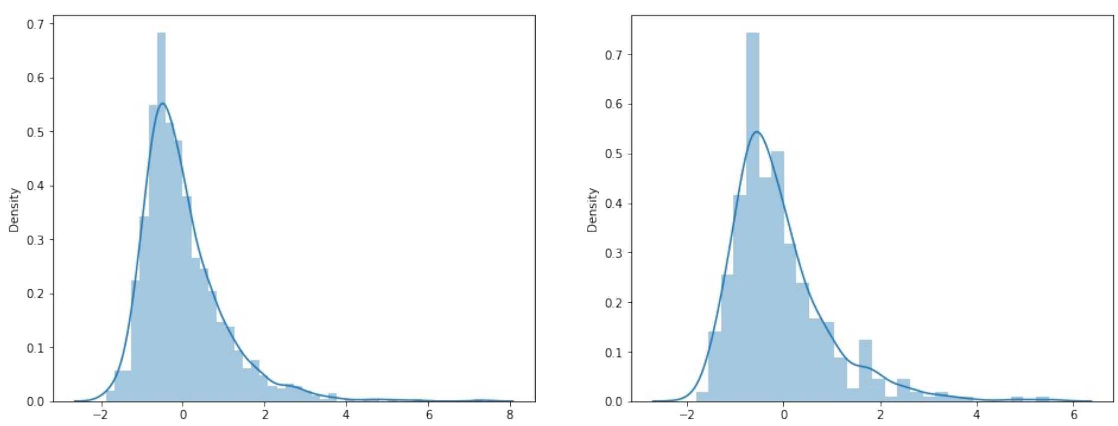

```
#Creating train and test dataset for validation purpose
df_train,df_test =
train_test_split(house_price,train_size=0.7,test_size=0.3,random_state
=42)

house_price[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposu
re','BsmtFinType1','BsmtFinType2',
            'HeatingQC','CentralAir',
'KitchenQual','GarageFinish','GarageQual','GarageCond',
             'ExterCond','LotShape']].head()
```

```
   LandSlope  ExterQual  BsmtQual  BsmtCond  BsmtExposure
BsmtFinType1  \
0          0          3         4         3             1
6
1          0          2         4         3             4
5
2          0          3         4         3             2
6
3          0          2         3         4             1
5
4          0          3         4         3             3
6

   BsmtFinType2  HeatingQC  CentralAir  KitchenQual  GarageFinish
GarageQual  \
0             1          4           1            3             2
3
1             1          4           1            2             2
3
2             1          4           1            3             2
3
3             1          3           1            3             1
3
4             1          4           1            3             2
3
```

```
   GarageCond  ExterCond  LotShape
0           3          2         3
1           3          2         3
2           3          2         0
3           3          2         0
4           3          2         0
```

```
# Scaling the train dataset
num_col = ['MSSubClass','LotArea','OverallQual','OverallCond',
           'MasVnrArea','BsmtFinSF1',

'BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1stFlrSF','2ndFlrSF',

'LowQualFinSF','GrLivArea','BsmtFullBath','BsmtHalfBath','FullBath','H
alfBath','BedroomAbvGr',
           'KitchenAbvGr','TotRmsAbvGrd','Fireplaces','GarageCars',

'GarageArea','WoodDeckSF','OpenPorchSF','EnclosedPorch','3SsnPorch',
           'ScreenPorch','PoolArea','MiscVal','SalePrice']

scaler = StandardScaler()
df_train[num_col] = scaler.fit_transform(df_train[num_col])
df_test[num_col] = scaler.transform(df_test[num_col])

#Checking the distribution again after scaling
plt.figure(figsize=(16,6))
plt.subplot(121)
sns.distplot(df_train.SalePrice)
plt.subplot(122)
sns.distplot(df_test.SalePrice)
```

```
<AxesSubplot:xlabel='SalePrice', ylabel='Density'>
```




```
y_train = df_train.pop('SalePrice')
X_train = df_train
```

```
y_test = df_test.pop('SalePrice')
X_test = df_test
```

```
len(X_train.columns)
```

```
192
```

```
lm  = LinearRegression()
lm.fit(X_train,y_train)
rfe = RFE(lm,70)
rfe.fit(X_train,y_train)
```

```
RFE(estimator=LinearRegression(), n_features_to_select=70)
```

```
rfe_scores =
pd.DataFrame(list(zip(X_train.columns,rfe.support_,rfe.ranking_)))
rfe_scores.columns = ['Column_Names','Status','Rank']
```

```
rfe_sel_columns =
list(rfe_scores[rfe_scores.Status==True].Column_Names)
```

```
# Filtering the train and test set for RFE Selected columns
X_train = X_train[rfe_sel_columns]
X_test = X_test[rfe_sel_columns]
```

```
# Using Lasso Regression Model
lm = Lasso(alpha=0.001)
lm.fit(X_train,y_train)

y_train_pred = lm.predict(X_train)
print(r2_score(y_true=y_train,y_pred=y_train_pred))

y_test_pred  = lm.predict(X_test)
print(r2_score(y_true=y_test,y_pred=y_test_pred))
```

```
0.8982771915853062
0.8536507408487869
```

```
lm = Lasso(alpha=0.001)
lm.fit(X_train,y_train)

y_train_pred = lm.predict(X_train)
print(r2_score(y_true=y_train,y_pred=y_train_pred))

y_test_pred  = lm.predict(X_test)
print(r2_score(y_true=y_test,y_pred=y_test_pred))
```

```
0.8982771915853062
0.8536507408487869
```

```
#Improving the model with the optimal value of alpha using Grid
SearchCV
```

```
folds = KFold(n_splits=10,shuffle=True,random_state=42)

hyper_param = {'alpha':[0.001, 0.01, 0.1,1.0, 5.0, 10.0,20.0]}

model = Lasso()

model_cv = GridSearchCV(estimator = model,
                        param_grid=hyper_param,
                        scoring='r2',
                        cv=folds,
                        verbose=1,
                        return_train_score=True
                       )

model_cv.fit(X_train,y_train)
```

```
Fitting 10 folds for each of 7 candidates, totalling 70 fits
```

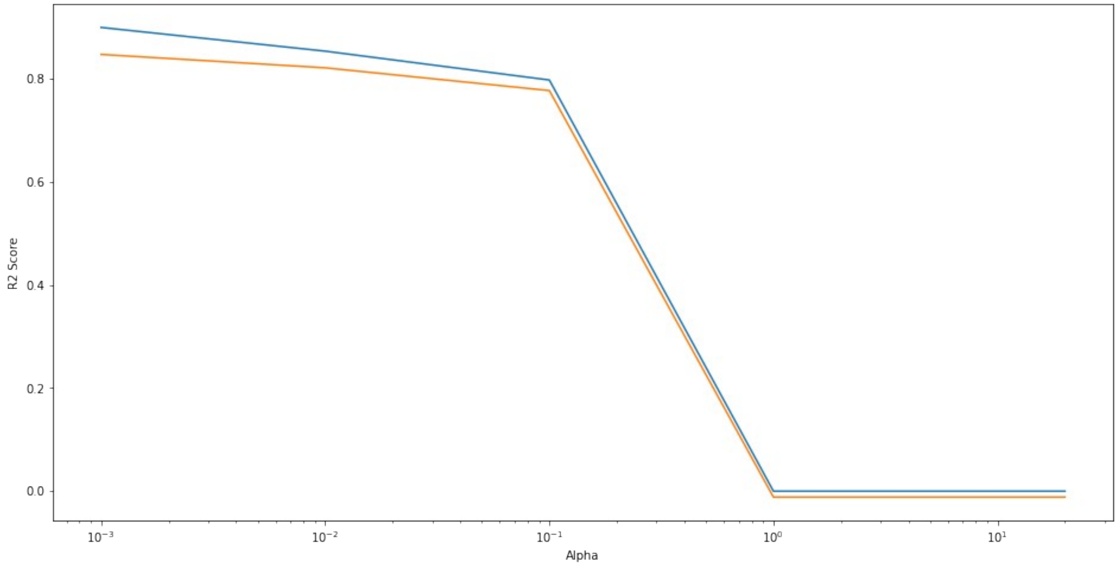
```
GridSearchCV(cv=KFold(n_splits=10, random_state=42, shuffle=True),
             estimator=Lasso(),
             param_grid={'alpha': [0.001, 0.01, 0.1, 1.0, 5.0, 10.0,
20.0]},
             return_train_score=True, scoring='r2', verbose=1)
```

```
cv_result_l = pd.DataFrame(model_cv.cv_results_)
cv_result_l['param_alpha'] =
cv_result_l['param_alpha'].astype('float32')
cv_result_l.head()
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time
param_alpha  \
0       0.029279      0.004708         0.001672        0.001789
0.001
1       0.009638      0.006484         0.001846        0.002442
0.010
2       0.005538      0.001296         0.002029        0.001187
0.100
3       0.004627      0.000595         0.002395        0.000416
1.000
4       0.005161      0.005979         0.001193        0.001461
5.000

             params  split0_test_score  split1_test_score
split2_test_score  \
0  {'alpha': 0.001}           0.828248           0.917334
0.826818
1   {'alpha': 0.01}           0.810095           0.885950
0.837873
2    {'alpha': 0.1}           0.740455           0.821663
0.774658
3    {'alpha': 1.0}          -0.006496          -0.021566          -
```

```
0.018063
4    {'alpha': 5.0}          -0.006496          -0.021566          -
0.018063

   split3_test_score  ...  split2_train_score  split3_train_score  \
0           0.880123  ...            0.905015            0.899477
1           0.841747  ...            0.854028            0.851280
2           0.758796  ...            0.791976            0.799241
3          -0.001154  ...            0.000000            0.000000
4          -0.001154  ...            0.000000            0.000000

   split4_train_score  split5_train_score  split6_train_score  \
0            0.897455            0.898772            0.896782
1            0.849655            0.849813            0.848107
2            0.795167            0.795939            0.788653
3            0.000000            0.000000            0.000000
4            0.000000            0.000000            0.000000

   split7_train_score  split8_train_score  split9_train_score  \
0            0.909337            0.897803            0.894870
1            0.882083            0.850477            0.846332
2            0.827001            0.795369            0.791412
3            0.000000            0.000000            0.000000
4            0.000000            0.000000            0.000000

   mean_train_score  std_train_score
0          0.899807         0.004443
1          0.853793         0.009849
2          0.797808         0.010296
3          0.000000         0.000000
4          0.000000         0.000000

[5 rows x 31 columns]
```

```
plt.figure(figsize=(16,8))
plt.plot(cv_result_l['param_alpha'],cv_result_l['mean_train_score'])
plt.plot(cv_result_l['param_alpha'],cv_result_l['mean_test_score'])
plt.xscale('log')
plt.ylabel('R2 Score')
plt.xlabel('Alpha')
plt.show()
```




```
# Checking the best parameter (alpha value)
model_cv.best_params_
```

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

```
lasso = Lasso(alpha=0.001)
lasso.fit(X_train,y_train)

y_train_pred = lasso.predict(X_train)
y_test_pred = lasso.predict(X_test)

print(r2_score(y_true=y_train,y_pred=y_train_pred))
print(r2_score(y_true=y_test,y_pred=y_test_pred))
```

```
0.8982771915853062
0.8536507408487869
```

```
model_param = list(lasso.coef_)
model_param.insert(0,lasso.intercept_)
cols = df_train.columns
cols.insert(0,'const')
lasso_coef = pd.DataFrame(list(zip(cols,model_param)))
lasso_coef.columns = ['Feature','Coef']
```

```
lasso_coef.sort_values(by='Coef',ascending=False).head(10)
```

	Feature	Coef
44	MiscVal	1.354856
28	BedroomAbvGr	0.429278
66	Neighborhood_Edwards	0.347471
12	BsmtFinType1	0.340484
27	HalfBath	0.256990
1	LotArea	0.218326
46	YearRemodAdd_Old	0.198670

```
14          BsmtFinType2  0.168496
19            CentralAir  0.150738
17           TotalBsmtSF  0.136790
```

```
# Using Ridge Regression
ridge = Ridge(alpha=0.001)
ridge.fit(X_train,y_train)

y_train_pred = ridge.predict(X_train)
print(r2_score(y_train,y_train_pred))
y_test_pred = ridge.predict(X_test)
print(r2_score(y_test,y_test_pred))
```

```
0.9041563594433633
0.8396959423896674
```

```
# By the clear difference in train and test set,the alpha value is not
optimal for ridge as there is a sign of overfitting.
# Hence,we can improve our model with the optimal value of alpha using
Grid Search CV
folds  = KFold(n_splits=10,shuffle=True,random_state=42)

hyper_param = {'alpha':[0.001,0.01,0.1,0.2,0.5,0.9,1.0, 5.0,
10.0,20.0]}

model = Ridge()

model_cv = GridSearchCV(estimator=model,
                        param_grid=hyper_param,
                        scoring='r2',
                        cv=folds,
                        verbose=1,
                        return_train_score=True)

model_cv.fit(X_train,y_train)
```

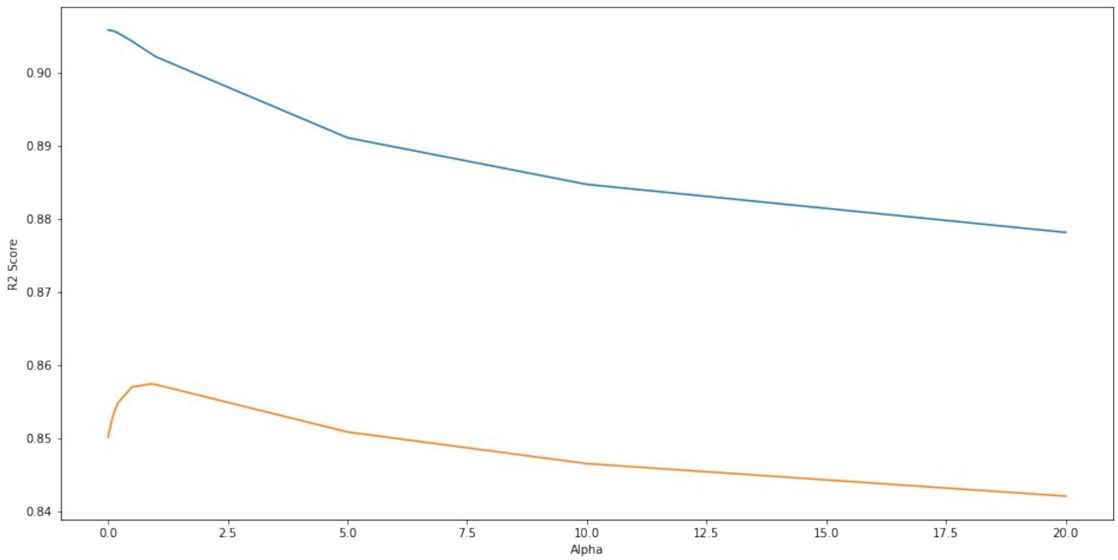
```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

GridSearchCV(cv=KFold(n_splits=10, random_state=42, shuffle=True),
             estimator=Ridge(),
             param_grid={'alpha': [0.001, 0.01, 0.1, 0.2, 0.5, 0.9,
1.0, 5.0,
                                   10.0, 20.0]},
             return_train_score=True, scoring='r2', verbose=1)
```

```
cv_result_r = pd.DataFrame(model_cv.cv_results_)
cv_result_r['param_alpha'] =
cv_result_r['param_alpha'].astype('float32')
cv_result_r.head()
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time
param_alpha  \
0       0.005480      0.001354         0.002933        0.000719
0.001
1       0.004369      0.006283         0.004022        0.006237
0.010
2       0.004379      0.002087         0.001650        0.001379
0.100
3       0.004815      0.007362         0.004609        0.006629
0.200
4       0.005601      0.006989         0.003338        0.004900
0.500

             params  split0_test_score  split1_test_score
split2_test_score  \
0  {'alpha': 0.001}           0.813634           0.922156
0.784142
1   {'alpha': 0.01}           0.814441           0.922122
0.786942
2    {'alpha': 0.1}           0.819975           0.921731
0.810162
3    {'alpha': 0.2}           0.823192           0.921219
0.828412
4    {'alpha': 0.5}           0.826973           0.919488
0.857991

   split3_test_score  ...  split2_train_score  split3_train_score  \
0           0.886505  ...            0.912392            0.905498
1           0.886465  ...            0.912388            0.905497
2           0.886080  ...            0.912065            0.905410
3           0.885688  ...            0.911335            0.905198
4           0.884675  ...            0.908453            0.904222

   split4_train_score  split5_train_score  split6_train_score  \
0            0.903497            0.905206            0.902209
1            0.903495            0.905204            0.902208
2            0.903405            0.905110            0.902118
3            0.903186            0.904883            0.901903
4            0.902182            0.903852            0.900924

   split7_train_score  split8_train_score  split9_train_score  \
0            0.915071            0.903728            0.900867
1            0.915070            0.903727            0.900866
2            0.915029            0.903626            0.900768
3            0.914932            0.903388            0.900535
4            0.914511            0.902309            0.899480

   mean_train_score  std_train_score
0          0.905847         0.004586
1          0.905846         0.004586
```

```
2           0.905733         0.004563
3           0.905469         0.004520
4           0.904321         0.004443

[5 rows x 31 columns]
```

```
plt.figure(figsize=(16,8))
plt.plot(cv_result_r['param_alpha'],cv_result_r['mean_train_score'])
plt.plot(cv_result_r['param_alpha'],cv_result_r['mean_test_score'])
plt.xlabel('Alpha')
# plt.xscale('log')
plt.ylabel('R2 Score')
plt.show()
```




```
# On the basis of above graph,lets create a model.....Checking the
best parameter(Alpha value)
model_cv.best_params_
```

```
{'alpha': 0.9}
```

```
ridge = Ridge(alpha = 0.9)
ridge.fit(X_train,y_train)

y_pred_train = ridge.predict(X_train)
print(r2_score(y_train,y_pred_train))

y_pred_test = ridge.predict(X_test)
print(r2_score(y_test,y_pred_test))
```

```
0.9015234159348484
0.8489408227967137
```

```
model_parameter = list(ridge.coef_)
model_parameter.insert(0,ridge.intercept_)
```

```
cols = df_train.columns
cols.insert(0,'constant')
ridge_coef = pd.DataFrame(list(zip(cols,model_parameter)))
ridge_coef.columns = ['Feaure','Coef']

ridge_coef.sort_values(by='Coef',ascending=False).head(10)
```

```
                  Feaure      Coef
44               MiscVal  1.353223
66  Neighborhood_Edwards  0.451164
28          BedroomAbvGr  0.431459
40         EnclosedPorch  0.323817
67  Neighborhood_Gilbert  0.304232
57         LotConfig_FR2  0.261708
27              HalfBath  0.251428
39           OpenPorchSF  0.234494
54       LandContour_Low  0.225297
19            CentralAir  0.222835
```

```
# After creating a model in both Ridge and Lasso,we can see that the r2_scores are almost same for both of them.
# But as lasso will penalize more on the dataset and also help in feature elimination,I am going to consider this as my final model.
# Final Model
lasso = Lasso(alpha=0.001)
lasso.fit(X_train,y_train)

y_train_pred = lasso.predict(X_train)
y_test_pred = lasso.predict(X_test)

print(r2_score(y_true=y_train,y_pred=y_train_pred))
print(r2_score(y_true=y_test,y_pred=y_test_pred))
```

```
0.8982771915853062
0.8536507408487869
```

```
# After comparing both the models we can see that the below Features are best explaining the DataSet
#MiscVal      : $Value of miscellaneous feature
#BsmtHalfBath : Basement half bathrooms
#LowQualFinSF : Low quality finished square feet (all floors)
#BsmtFullBath : Basement full bathrooms
#HalfBath     : Half baths above grade

# Best alpha value for Lasso:- {'alpha' : 0.001}
# Best alpha value for Ridge:- {'alpha' :0.9}
```