In [ ]:

```
#Step 1: Reading and Understanding the Data
#Let us first import NumPy and Pandas and read the dataset
```

In [30]:

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
```

In [81]:

```
# Importing all required packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

In [32]:

```
on of imports to refer version specific documentation
 {np.__version__} | Pandas version: {pd.__version__} | Seaborn version: {sns.__version__}'
```

Out[32]:

```
'NumPy version: 1.19.2 | Pandas version: 1.1.3 | Seaborn version: 0.11.0'
```

In [33]:

```
# Changing default display options for better visibility of data
pd.options.display.max_colwidth = 255
pd.options.display.max_rows=225
```

In [34]:

```
# Importing dataset

housingInfo = pd.read_csv('C:/Users/Admin/Desktop/Assignment - Advanced Regression/train.cs
housingInfo
```

Out[34]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandConto
0	1	60	RL	65.0	8450	Pave	NaN	Reg	L
1	2	20	RL	80.0	9600	Pave	NaN	Reg	L
2	3	60	RL	68.0	11250	Pave	NaN	IR1	L
3	4	70	RL	60.0	9550	Pave	NaN	IR1	L
4	5	60	RL	84.0	14260	Pave	NaN	IR1	L
...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	L
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	L
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	L
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	L
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	L

1460 rows × 81 columns

In [35]:

```
# inspect housingInfo dataframe

print("********************************  Info ****************************************")
print(housingInfo.info())
print("********************************  Shape ****************************************")
print(housingInfo.shape)
print("**************************** Columns having null values ****************************")
print(housingInfo.isnull().any())
print("**************************** Describe ****************************")
housingInfo.describe()
```

Out[35]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearR
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	14(
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	19
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	19
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	19(
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	19
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	20(
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	20

8 rows × 38 columns

In [36]:

```
'''
1. There are 1460 rows and 81 columns in dataframe
2. These 81 columns comprises of both dimensions (categorical value) and measures (numeric
3. The dataset is not clean, i.e. consists of missing values as well
'''
```

Out[36]:

```
'\n1. There are 1460 rows and 81 columns in dataframe\n2. These 81 columns c
omprises of both dimensions (categorical value) and measures (numeric value)
\n3. The dataset is not clean, i.e. consists of missing values as well\n'
```

In [37]:

```
#Step 2: Data Cleaning¶
#Removing/Imputing NaN values in Categorical attributes
```

In [38]:

```
# check for null values in all categorical columns

housingInfo.select_dtypes(include='object').isnull().sum()[housingInfo.select_dtypes(includ
```

Out[38]:

```
Alley          1369
MasVnrType        8
BsmtQual         37
BsmtCond         37
BsmtExposure     38
BsmtFinType1     37
BsmtFinType2     38
Electrical        1
FireplaceQu     690
GarageType       81
GarageFinish     81
GarageQual       81
GarageCond       81
PoolQC         1453
Fence          1179
MiscFeature    1406
dtype: int64
```

In [39]:

```
# Replace NA with None in the following columns below :

for col in ('Alley', 'MasVnrType', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
            'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'M

    housingInfo[col]=housingInfo[col].fillna('None')
```

In [40]:

```
'''
Remove categorical attributes that have more than 85% data associated to one
value.
- We will remove any column that has one value repeating 1241 times
(1241/1450)*100 = 85%) as this column would be skewed
  to one value
'''
```

Out[40]:

```
'\nRemove categorical attributes that have more than 85% data associated to
one \nvalue.\n- We will remove any column that has one value repeating 1241
times \n(1241/1450)*100 = 85%) as this column would be skewed\n  to one valu
e\n'
```

In [41]:

```
# Drop the following columns that have more than 85% values associated to a specific value

# Method to get the column names that have count of one value more than 85%

def getHighCategoricalValueCounts():
    column = []
    categorical_columns = housingInfo.select_dtypes(include=['object'])
    for col in (categorical_columns):
        if(housingInfo[col].value_counts().max() >= 1241):
            column.append(col)
    return column

columnsToBeRemoved = getHighCategoricalValueCounts()

# Remove the columns with skewed data

housingInfo.drop(columnsToBeRemoved, axis = 1, inplace = True)

housingInfo.head()
```

Out[41]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	Bldg
0	1	60	RL	65.0	8450	Reg	Inside	CollgCr	
1	2	20	RL	80.0	9600	Reg	FR2	Veenker	
2	3	60	RL	68.0	11250	IR1	Inside	CollgCr	
3	4	70	RL	60.0	9550	IR1	Corner	Crawfor	
4	5	60	RL	84.0	14260	IR1	FR2	NoRidge	

5 rows × 60 columns

In [42]:

```
# once again check for null values in all categorical columns

housingInfo.select_dtypes(include='object').isnull().sum()[housingInfo.select_dtypes(includ
```

Out[42]:

```
Series([], dtype: int64)
```

In [43]:

```
#No more null values in the categorical variables
```

In [44]:

```
#Removing null values in Numerical attributes

# check the null values in the numerical data

housingInfo.select_dtypes(include=['int64','float']).isnull().sum()[housingInfo.select_dtyp
                                                                    .sum()>0]
```

Out[44]:

```
LotFrontage    259
MasVnrArea       8
GarageYrBlt     81
dtype: int64
```

In [45]:

```
# Impute the null values with median values for LotFrontage and MasVnrArea columns

housingInfo['LotFrontage'] = housingInfo['LotFrontage'].replace(np.nan, housingInfo['LotFro
housingInfo['MasVnrArea'] = housingInfo['MasVnrArea'].replace(np.nan, housingInfo['MasVnrAr
```

In [46]:

```
# Setting the null values with 0 for GarageYrBlt for now as we would be handling this colum

housingInfo['GarageYrBlt']=housingInfo['GarageYrBlt'].fillna(0)
housingInfo['GarageYrBlt'] = housingInfo['GarageYrBlt'].astype(int)
```

In [47]:

```
# Create a new column named IsRemodelled - This column would determine whether the house ha
# the difference between remodelled and built years

def checkForRemodel(row):
    if(row['YearBuilt'] == row['YearRemodAdd']):
        return 0
    elif(row['YearBuilt'] < row['YearRemodAdd']):
        return 1
    else:
        return 2

housingInfo['IsRemodelled'] = housingInfo.apply(checkForRemodel, axis=1)
housingInfo.head()
```

Out[47]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	Bldg
0	1	60	RL	65.0	8450	Reg	Inside	CollgCr	
1	2	20	RL	80.0	9600	Reg	FR2	Veenker	
2	3	60	RL	68.0	11250	IR1	Inside	CollgCr	
3	4	70	RL	60.0	9550	IR1	Corner	Crawfor	
4	5	60	RL	84.0	14260	IR1	FR2	NoRidge	

5 rows × 61 columns

In [48]:

```
# Create a new column named BuiltOrRemodelledAge and determine the age of the building at t

def getBuiltOrRemodelAge(row):
    if(row['YearBuilt'] == row['YearRemodAdd']):
        return row['YrSold'] - row['YearBuilt']
    else:
        return row['YrSold'] - row['YearRemodAdd']

housingInfo['BuiltOrRemodelAge'] = housingInfo.apply(getBuiltOrRemodelAge, axis=1)
housingInfo.head()
```

Out[48]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	B
0	1	60	RL	65.0	8450	Reg	Inside	CollgCr	
1	2	20	RL	80.0	9600	Reg	FR2	Veenker	
2	3	60	RL	68.0	11250	IR1	Inside	CollgCr	
3	4	70	RL	60.0	9550	IR1	Corner	Crawfor	
4	5	60	RL	84.0	14260	IR1	FR2	NoRidge	

5 rows × 62 columns

In [49]:

```
# Create a new column which would indicate if the Garage is old or new.
# Garage Yr Built less than 2000 will be considered as old (0) else new(1).
# For GarageYrBuilt , where we have imputed the value as 0 will also be treated as old.

def getGarageConstructionPeriod(row):
    if row == 0:
        return 0
    elif row >= 1900 and row < 2000:
        return 0
    else:
        return 1

housingInfo['OldOrNewGarage'] = housingInfo['GarageYrBlt'].apply(getGarageConstructionPerio
housingInfo.head()
```

Out[49]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	Bldg
0	1	60	RL	65.0	8450	Reg	Inside	CollgCr	
1	2	20	RL	80.0	9600	Reg	FR2	Veenker	
2	3	60	RL	68.0	11250	IR1	Inside	CollgCr	
3	4	70	RL	60.0	9550	IR1	Corner	Crawfor	
4	5	60	RL	84.0	14260	IR1	FR2	NoRidge	

5 rows × 63 columns

In [50]:

```
# Since we have created new features from YearBuilt, YearRemodAdd, YrSold and GarageYrBlt,
# would only be using the derived columns for further analysis

housingInfo.drop(['YearBuilt', 'YearRemodAdd', 'YrSold', 'GarageYrBlt'], axis = 1, inplace
```

In [51]:

```
'''
Remove numerical attributes that have more than 85% data associated to one
value.
- We will remove any column that has one value repeating 1241 times
(1241/1450)*100 = 85%) as this column would be skewed
to one value'''
```

Out[51]:

```
'\nRemove numerical attributes that have more than 85% data associated to on
e \nvalue.\n- We will remove any column that has one value repeating 1241 ti
mes \n(1241/1450)*100 = 85%) as this column would be skewed\nto one value'
```

In [52]:

```
# Drop the following columns that have more than 85% values associated to a specific value
# We will also drop MoSold as we will not be using that for further analysis

def getHighNumericalValueCounts():
    column = []
    numerical_columns = housingInfo.select_dtypes(include=['int64', 'float'])
    for col in (numerical_columns):
        if(housingInfo[col].value_counts().max() >= 1241):
            column.append(col)
    return column

columnsToBeRemoved = getHighNumericalValueCounts()
housingInfo.drop(columnsToBeRemoved, axis = 1, inplace = True)

housingInfo.drop(['MoSold'], axis = 1, inplace = True)

housingInfo.head()
```

Out[52]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	Bldg
0	1	60	RL	65.0	8450	Reg	Inside	CollgCr	
1	2	20	RL	80.0	9600	Reg	FR2	Veenker	
2	3	60	RL	68.0	11250	IR1	Inside	CollgCr	
3	4	70	RL	60.0	9550	IR1	Corner	Crawfor	
4	5	60	RL	84.0	14260	IR1	FR2	NoRidge	

5 rows × 49 columns

In [53]:

```
# check for percentage of null values in each column

percent_missing = round(100*(housingInfo.isnull().sum()/len(housingInfo.index)), 2)
print(percent_missing)
```

```
Id                   0.0
MSSubClass           0.0
MSZoning             0.0
LotFrontage          0.0
LotArea              0.0
LotShape             0.0
LotConfig            0.0
Neighborhood         0.0
BldgType             0.0
HouseStyle           0.0
OverallQual          0.0
OverallCond          0.0
RoofStyle            0.0
Exterior1st          0.0
Exterior2nd          0.0
MasVnrType           0.0
MasVnrArea           0.0
ExterQual            0.0
Foundation           0.0
BsmtQual             0.0
BsmtExposure         0.0
BsmtFinType1         0.0
BsmtFinSF1           0.0
BsmtUnfSF            0.0
TotalBsmtSF          0.0
HeatingQC            0.0
1stFlrSF             0.0
2ndFlrSF             0.0
GrLivArea            0.0
BsmtFullBath         0.0
FullBath             0.0
HalfBath             0.0
BedroomAbvGr         0.0
KitchenQual          0.0
TotRmsAbvGrd         0.0
Fireplaces           0.0
FireplaceQu          0.0
GarageType           0.0
GarageFinish         0.0
GarageCars           0.0
GarageArea           0.0
WoodDeckSF           0.0
OpenPorchSF          0.0
Fence                0.0
SaleCondition        0.0
SalePrice            0.0
IsRemodelled         0.0
BuiltOrRemodelAge    0.0
OldOrNewGarage       0.0
dtype: float64
```

In [54]:

```
#Hence there are no null values in the dataset
```

In [55]:

```
#Check for Duplicates

# Check if there are any duplicate values in the dataset

housingInfo[housingInfo.duplicated(keep=False)]
```

Out[55]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	BldgT

0 rows × 49 columns

In [56]:

```
#No duplicate entries found !!!
```

In [57]:

```
#Outlier Treatment

# Checking outliers at 25%,50%,75%,90%,95% and above

housingInfo.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[57]:

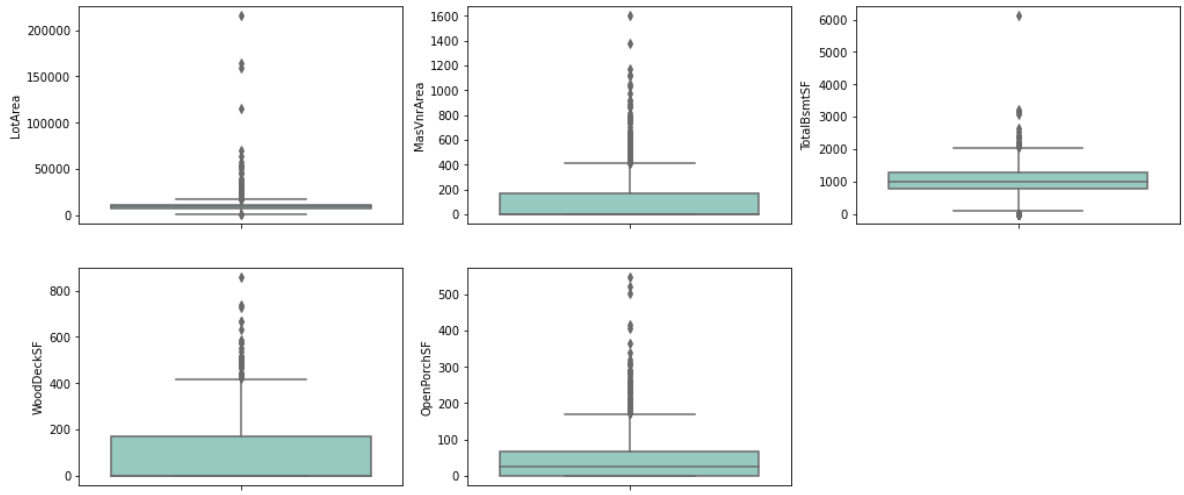
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasV
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.(
mean	730.500000	56.897260	69.863699	10516.828082	6.099315	5.575342	103.
std	421.610009	42.300571	22.027677	9981.264932	1.382997	1.112799	180.7
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	0.(
25%	365.750000	20.000000	60.000000	7553.500000	5.000000	5.000000	0.(
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	0.(
75%	1095.250000	70.000000	79.000000	11601.500000	7.000000	6.000000	164.2
90%	1314.100000	120.000000	92.000000	14381.700000	8.000000	7.000000	335.(
95%	1387.050000	160.000000	104.000000	17401.150000	8.000000	8.000000	456.(
99%	1445.410000	190.000000	137.410000	37567.640000	10.000000	9.000000	791.2
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	1600.(

11 rows × 27 columns

In [58]:

```
# Check the outliers in all the numeric columns

plt.figure(figsize=(17, 20))
plt.subplot(5,3,1)
sns.boxplot(y = 'LotArea', palette='Set3', data = housingInfo)
plt.subplot(5,3,2)
sns.boxplot(y = 'MasVnrArea', palette='Set3', data = housingInfo)
plt.subplot(5,3,3)
sns.boxplot(y = 'TotalBsmtSF', palette='Set3', data = housingInfo)
plt.subplot(5,3,4)
sns.boxplot(y = 'WoodDeckSF', palette='Set3', data = housingInfo)
plt.subplot(5,3,5)
sns.boxplot(y = 'OpenPorchSF', palette='Set3', data = housingInfo)
plt.show()
```



In [59]:

```
# Removing Outliers

# Removing values beyond 98% for LotArea

nn_quartile_LotArea = housingInfo['LotArea'].quantile(0.98)
housingInfo = housingInfo[housingInfo["LotArea"] < nn_quartile_LotArea]

# Removing values beyond 98% for MasVnrArea

nn_quartile_MasVnrArea = housingInfo['MasVnrArea'].quantile(0.98)
housingInfo = housingInfo[housingInfo["MasVnrArea"] < nn_quartile_MasVnrArea]

# Removing values beyond 99% for TotalBsmtSF

nn_quartile_TotalBsmtSF = housingInfo['TotalBsmtSF'].quantile(0.99)
housingInfo = housingInfo[housingInfo["TotalBsmtSF"] < nn_quartile_TotalBsmtSF]

# Removing values beyond 99% for WoodDeckSF

nn_quartile_WoodDeckSF = housingInfo['WoodDeckSF'].quantile(0.99)
housingInfo = housingInfo[housingInfo["WoodDeckSF"] < nn_quartile_WoodDeckSF]

# Removing values beyond 99% for OpenPorchSF

nn_quartile_OpenPorchSF = housingInfo['OpenPorchSF'].quantile(0.99)
housingInfo = housingInfo[housingInfo["OpenPorchSF"] < nn_quartile_OpenPorchSF]
```

In [60]:

```
# Determine the percentage of data retained

num_data = round(100*(len(housingInfo)/1460),2)
print(num_data)
```

```
93.01
```

In [61]:

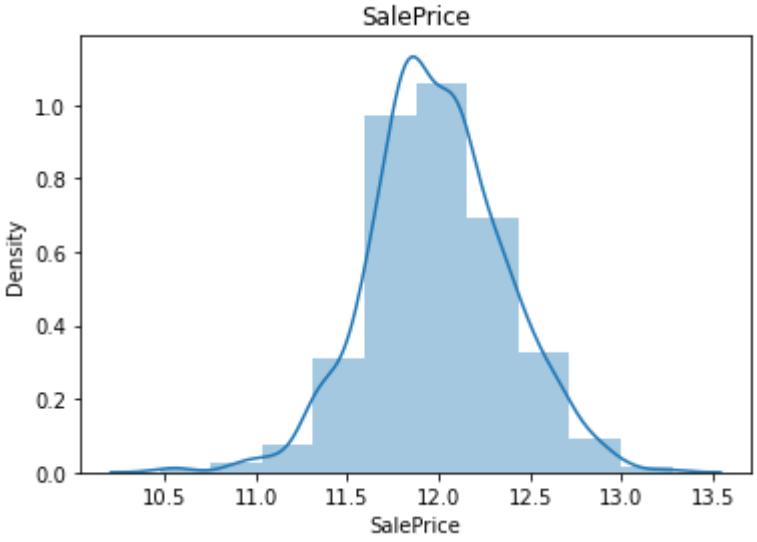
```
#Step 3: Data Visualization
```

In [62]:

```
# Visualise the target variable -> SalePrice after transforming the sales price

housingInfo['SalePrice'] = np.log1p(housingInfo['SalePrice'])

plt.title('SalePrice')
sns.distplot(housingInfo['SalePrice'], bins=10)
plt.show()
```



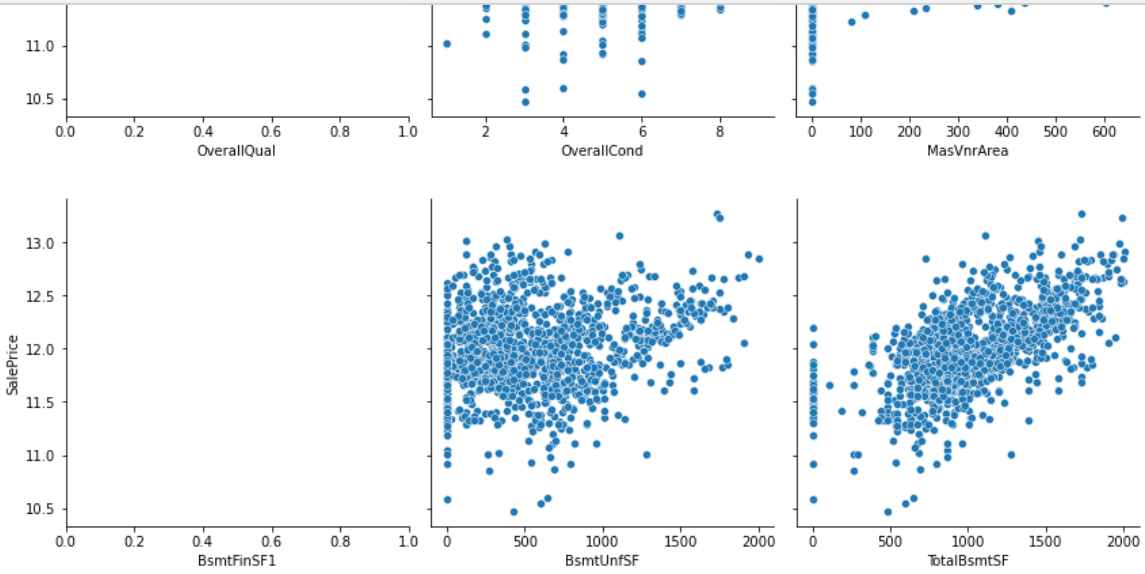
In [63]:

```
#The target value seems to be normalized with some noise.
```

In [64]:

```
# Check the numerical values using pairplots

plt.figure(figsize=(10,5))
sns.pairplot(housingInfo, x_vars=['MSSubClass','LotFrontage','LotArea'], y_vars='SalePrice'
sns.pairplot(housingInfo, x_vars=['OverallQual', 'OverallCond','MasVnrArea'], y_vars='SaleP
sns.pairplot(housingInfo, x_vars=['BsmtFinSF1', 'BsmtUnfSF','TotalBsmtSF'], y_vars='SalePri
sns.pairplot(housingInfo, x_vars=['1stFlrSF','2ndFlrSF', 'GrLivArea'], y_vars='SalePrice',h
sns.pairplot(housingInfo, x_vars=['BsmtFullBath','FullBath', 'HalfBath'], y_vars='SalePrice
sns.pairplot(housingInfo, x_vars=['BedroomAbvGr','TotRmsAbvGrd', 'Fireplaces'], y_vars='Sal
sns.pairplot(housingInfo, x_vars=['GarageCars','GarageArea', 'WoodDeckSF'], y_vars='SalePri
sns.pairplot(housingInfo, x_vars=['OpenPorchSF','SalePrice', 'IsRemodelled'], y_vars='SaleP
sns.pairplot(housingInfo, x_vars=['BuiltOrRemodelAge'], y_vars='SalePrice',height=4, aspect
plt.show()
```



In [ ]:

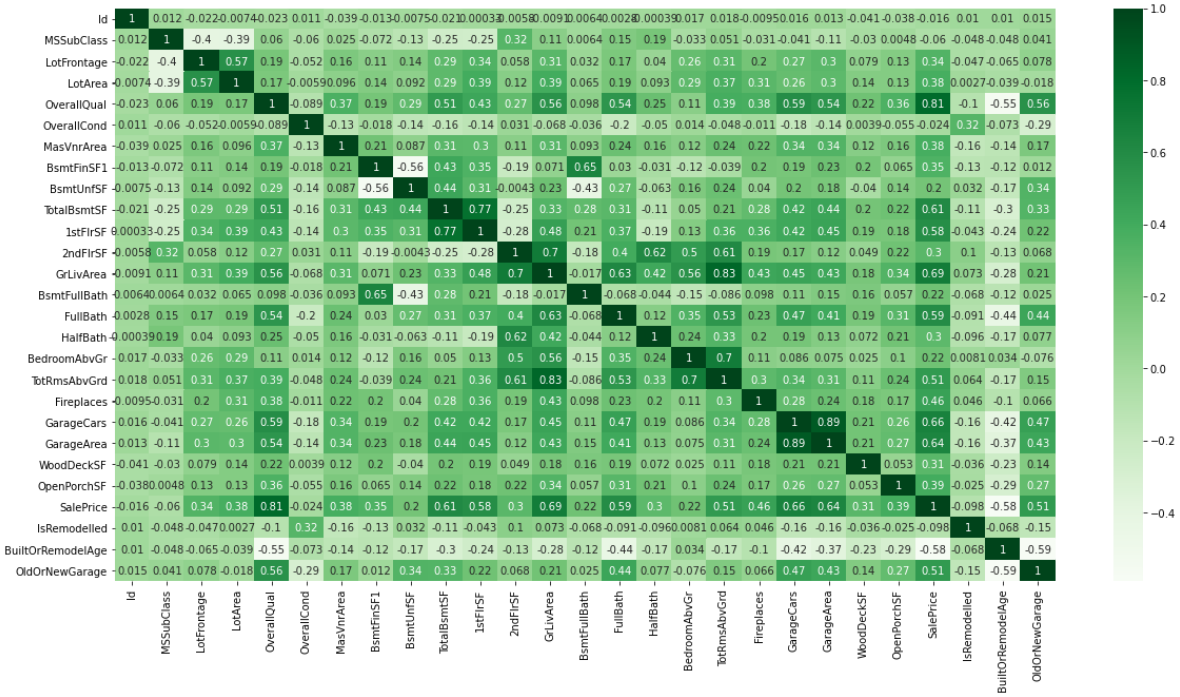
```
'''Observations :

- 1stFlrSF, GrLivArea seems to be showing correlation towards right
- Rest of the variables are too scattered and hence can be understood during further analys
'''
```

In [65]:

```
# Check the correlation of numerical columns

plt.figure(figsize = (20, 10))
sns.heatmap(housingInfo.corr(), annot = True, cmap="Greens")
plt.show()
```



In [ ]:

```
'''
Removing following columns which shows high correlation
- TotRmsAbvGrd and GrLivArea show 82%
- Garage Area and Garage Cars show 88%

Hence dropping TotRmsAbvGrd and Garage Cars'''
```

In [66]:

```
# Removing the highly correlated variables

housingInfo.drop(['TotRmsAbvGrd', 'GarageArea'], axis = 1, inplace = True)
```

In [67]:

```
# Check the shape of the dataframe

housingInfo.shape
```

Out[67]:

```
(1358, 47)
```

In [ ]:

```
#Step 4: Data Preparation¶

#Converting categorical data into numercal data
#Creating Dummies
```

In [68]:

```
# Since the values of the following fields are ordered list, we shall assign values to them

# For values which can be ordered, we have given an ordered sequence value
# For values which cannot be ordered, we have categorised them into 0 and 1

housingInfo['d_LotShape'] = housingInfo['LotShape'].map({'Reg': 3, 'IR1': 2, 'IR2': 1, 'IR3
housingInfo['d_ExterQual'] = housingInfo['ExterQual'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa':
housingInfo['d_BsmtQual'] = housingInfo['BsmtQual'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2
housingInfo['d_BsmtExposure'] = housingInfo['BsmtExposure'].map({'Gd': 4, 'Av': 3, 'Mn': 2,
housingInfo['d_BsmtFinType1'] = housingInfo['BsmtFinType1'].map({'GLQ': 6, 'ALQ': 5, 'BLQ':
                                                                 'None': 0})
housingInfo['d_HeatingQC'] = housingInfo['HeatingQC'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa':
housingInfo['d_KitchenQual'] = housingInfo['KitchenQual'].map({'Ex': 5, 'Gd': 4, 'TA': 3, '
housingInfo['d_FireplaceQu'] = housingInfo['FireplaceQu'].map({'Ex': 5, 'Gd': 4, 'TA': 3, '
housingInfo['d_GarageFinish'] = housingInfo['GarageFinish'].map({'Fin': 3, 'RFn': 2, 'Unf':
housingInfo['d_BldgType'] = housingInfo['BldgType'].map({'Twnhs': 5, 'TwnhsE': 4, 'Duplex':
                                                         'None': 0 })
housingInfo['d_HouseStyle'] = housingInfo['HouseStyle'].map({'SLvl': 8, 'SFoyer': 7, '2.5Fi
                                                             '1.5Fin': 3, '1.5Unf': 2,
housingInfo['d_Fence'] = housingInfo['Fence'].map({'GdPrv': 4, 'GdWo': 3, 'MnPrv': 2, 'MnWw
housingInfo['d_LotConfig'] = housingInfo['LotConfig'].map({'Inside': 5, 'Corner': 4, 'CulDS
                                                           'None': 0  })
housingInfo['d_MasVnrType'] = housingInfo['MasVnrType'].map({'BrkCmn': 1, 'BrkFace': 1, 'CB
housingInfo['d_SaleCondition'] = housingInfo['SaleCondition'].map({'Normal': 1, 'Partial':
                                                                   'Alloca': 0, 'AdjLand':
housingInfo.head()
```

Out[68]:

:	d_KitchenQual	d_FireplaceQu	d_GarageFinish	d_BldgType	d_HouseStyle	d_Fence	d_LotCon
5	4	0	2	1	4	0	
5	3	3	2	1	1	0	
5	4	3	2	1	4	0	
4	4	4	1	1	4	0	
5	4	3	2	1	4	0	

In [69]:

```
# drop the old columns from which the new columns were derived
# We can also drop the id column as it will not be used any more

housingInfo = housingInfo.drop(['Id', 'LotShape', 'ExterQual', 'BsmtQual', 'BsmtExposure',
                                'KitchenQual', 'FireplaceQu', 'GarageFinish', 'BldgType', '
                                'LotConfig', 'MasVnrType', 'SaleCondition'], axis=1)

housingInfo.head()
```

Out[69]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Neighborhood	OverallQual	OverallCond	Roo
0	60	RL	65.0	8450	CollgCr	7	5	
1	20	RL	80.0	9600	Veenker	6	8	
2	60	RL	68.0	11250	CollgCr	7	5	
3	70	RL	60.0	9550	Crawfor	7	5	
4	60	RL	84.0	14260	NoRidge	8	5	

5 rows × 46 columns

In [70]:

```
# For the following columns create dummies

# Creating dummies for MSZoning

d_MSZoning = pd.get_dummies(housingInfo['MSZoning'], prefix='MSZoning', drop_first = True)
housingInfo = pd.concat([housingInfo, d_MSZoning], axis = 1)

# Creating dummies for Neighborhood

d_Neighborhood = pd.get_dummies(housingInfo['Neighborhood'], prefix='Neighborhood', drop_fi
housingInfo = pd.concat([housingInfo, d_Neighborhood], axis = 1)

# Creating dummies for RoofStyle

d_RoofStyle = pd.get_dummies(housingInfo['RoofStyle'], prefix='RoofStyle', drop_first = Tru
housingInfo = pd.concat([housingInfo, d_RoofStyle], axis = 1)

# Creating dummies for Exterior1st

d_Exterior1st = pd.get_dummies(housingInfo['Exterior1st'], prefix='Exterior1st', drop_first
housingInfo = pd.concat([housingInfo, d_Exterior1st], axis = 1)

# Creating dummies for Exterior2nd

d_Exterior2nd = pd.get_dummies(housingInfo['Exterior2nd'], prefix='Exterior2nd', drop_first
housingInfo = pd.concat([housingInfo, d_Exterior2nd], axis = 1)

# Creating dummies for Foundation

d_Foundation = pd.get_dummies(housingInfo['Foundation'], prefix='Foundation', drop_first =
housingInfo = pd.concat([housingInfo, d_Foundation], axis = 1)

# Creating dummies for GarageType

d_GarageType = pd.get_dummies(housingInfo['GarageType'], prefix='GarageType', drop_first =
housingInfo = pd.concat([housingInfo, d_GarageType], axis = 1)

housingInfo.head()
```

Out[70]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Neighborhood	OverallQual	OverallCond	Roc
0	60	RL	65.0	8450	CollgCr	7	5	
1	20	RL	80.0	9600	Veenker	6	8	
2	60	RL	68.0	11250	CollgCr	7	5	
3	70	RL	60.0	9550	Crawfor	7	5	
4	60	RL	84.0	14260	NoRidge	8	5	

5 rows × 119 columns

In [71]:

```
# drop the below columns as we now have new columns derived from these columns

housingInfo = housingInfo.drop(['MSZoning', 'Neighborhood', 'RoofStyle', 'Exterior1st', 'Ex
                                'GarageType'], axis=1)

housingInfo.head()
```

Out[71]:

rallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtUnfSF	TotalBsmtSF	1stFlrSF	...	Foundati
7	5	196.0	706	150	856	856	...	
6	8	0.0	978	284	1262	1262	...	
7	5	162.0	486	434	920	920	...	
7	5	0.0	216	540	756	961	...	
8	5	350.0	655	490	1145	1145	...	

In [72]:

```
housingInfo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1358 entries, 0 to 1458
Columns: 112 entries, MSSubClass to GarageType_None
dtypes: float64(3), int64(36), uint8(73)
memory usage: 521.2 KB
```

In [ ]:

```
#All columns in the data set are now numeric !!!
```

In [73]:

```
#Step 5: Train Test Split

# Putting all feature variable to X

X = housingInfo.drop(['SalePrice'], axis=1)
X.head()
```

Out[73]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	Bsı
0	60	65.0	8450	7	5	196.0	706	
1	20	80.0	9600	6	8	0.0	978	
2	60	68.0	11250	7	5	162.0	486	
3	70	60.0	9550	7	5	0.0	216	
4	60	84.0	14260	8	5	350.0	655	

5 rows × 111 columns

In [74]:

```
# Putting response variable to y

y = housingInfo['SalePrice']
y.head()
```

Out[74]:

```
0    12.247699
1    12.109016
2    12.317171
3    11.849405
4    12.429220
Name: SalePrice, dtype: float64
```

In [82]:

```
# scaling the features

from sklearn.preprocessing import scale

# storing column names in cols
# scaling (the dataframe is converted to a numpy array)

cols = X.columns
X = pd.DataFrame(scale(X))
X.columns = cols
X.columns
```

Out[82]:

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'MasVnrArea', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF',
       ...
       'Foundation_PConc', 'Foundation_Slab', 'Foundation_Stone',
       'Foundation_Wood', 'GarageType_Attchd', 'GarageType_Basment',
       'GarageType_BuiltIn', 'GarageType_CarPort', 'GarageType_Detchd',
       'GarageType_None'],
      dtype='object', length=111)
```

In [85]:

```
# split into train and test

from sklearn.model_selection  import train_test_split

np.random.seed(0)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size = 0.3,
```

In [ ]:

```
#Step 5: Recursive feature elimination (RFE)

''' Since there are around 111 features, we will use RFE to get the best 50
features out of the 111 features and use the new
features for further analysis
'''
```

In [86]:

```
# Running RFE with the output number of the variable equal to 50

lm = LinearRegression()
lm.fit(X_train, y_train)

# running RFE
rfe = RFE(lm, 50)
rfe = rfe.fit(X_train, y_train)
```

In [87]:

```
# Assign the columns selected by RFE to cols

col = X_train.columns[rfe.support_]

# assign the 50 features selected using RFE to a dataframe and view them

temp_df = pd.DataFrame(list(zip(X_train.columns,rfe.support_,rfe.ranking_)), columns=['Vari
temp_df = temp_df.loc[temp_df['rfe_support'] == True]
temp_df.reset_index(drop=True, inplace=True)

temp_df
```

Out[87]:

	Variable	rfe_support	rfe_ranking
0	LotArea	True	1
1	OverallQual	True	1
2	OverallCond	True	1
3	BsmtFinSF1	True	1
4	TotalBsmtSF	True	1
5	1stFlrSF	True	1
6	2ndFlrSF	True	1
7	GrLivArea	True	1
8	BsmtFullBath	True	1
9	FullBath	True	1
10	HalfBath	True	1
11	Fireplaces	True	1
12	GarageCars	True	1
13	WoodDeckSF	True	1
14	IsRemodelled	True	1
15	BuiltOrRemodelAge	True	1
16	OldOrNewGarage	True	1
17	d_BsmtQual	True	1
18	d_BsmtExposure	True	1
19	d_HeatingQC	True	1
20	d_KitchenQual	True	1
21	d_GarageFinish	True	1
22	d_BldgType	True	1
23	d_SaleCondition	True	1
24	MSZoning_FV	True	1
25	MSZoning_RH	True	1
26	MSZoning_RL	True	1

	Variable	rfe_support	rfe_ranking
27	MSZoning_RM	True	1
28	Neighborhood_Crawfor	True	1
29	Neighborhood_Edwards	True	1
30	Neighborhood_MeadowV	True	1
31	Neighborhood_NWAmes	True	1
32	Neighborhood_NridgHt	True	1
33	Neighborhood_OldTown	True	1
34	Neighborhood_SWISU	True	1
35	Neighborhood_StoneBr	True	1
36	Exterior1st_BrkComm	True	1
37	Exterior1st_CemntBd	True	1
38	Exterior1st_Stucco	True	1
39	Exterior1st_VinylSd	True	1
40	Exterior1st_Wd Sdng	True	1
41	Exterior2nd_CmentBd	True	1
42	Exterior2nd_Stucco	True	1
43	Exterior2nd_VinylSd	True	1
44	Exterior2nd_Wd Sdng	True	1
45	Foundation_CBlock	True	1
46	Foundation_PConc	True	1
47	Foundation_Slab	True	1
48	Foundation_Stone	True	1
49	GarageType_CarPort	True	1

In [88]:

```
# Assign the 50 columns to X_train_rfe

X_train_rfe = X_train[col]
```

In [89]:

```
# Associate the new 50 columns to X_train and X_test for further analysis

X_train = X_train_rfe[X_train_rfe.columns]
X_test =  X_test[X_train.columns]
```

In [90]:

```
#Step 6: Model Building and Evaluation

#Ridge


# list pf alphas

params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
                    9.0, 10.0, 20, 50, 100, 500, 1000 ]}

ridge = Ridge()

# cross validation

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

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done 135 out of 135 | elapsed:    2.1s finished
```

Out[90]:

```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 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, 50
0,
                                   1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [91]:

```
# display the mean scores

ridge_cv_results = pd.DataFrame(ridge_model_cv.cv_results_)
ridge_cv_results = ridge_cv_results[ridge_cv_results['param_alpha']<=500]
ridge_cv_results[['param_alpha', 'mean_train_score', 'mean_test_score', 'rank_test_score']]
```

Out[91]:

	param_alpha	mean_train_score	mean_test_score	rank_test_score
16	5	-0.077858	-0.083995	1
17	6	-0.077911	-0.084008	2
18	7	-0.077960	-0.084023	3
15	4	-0.077810	-0.084028	4
19	8	-0.078009	-0.084038	5
20	9	-0.078055	-0.084054	6
21	10	-0.078096	-0.084068	7
14	3	-0.077763	-0.084105	8
22	20	-0.078414	-0.084171	9
13	2	-0.077718	-0.084206	10
12	1	-0.077680	-0.084345	11
11	0.9	-0.077677	-0.084364	12
10	0.8	-0.077676	-0.084384	13
9	0.7	-0.077676	-0.084405	14
8	0.6	-0.077676	-0.084427	15
23	50	-0.078953	-0.084434	16
7	0.5	-0.077676	-0.084450	17
6	0.4	-0.077677	-0.084474	18
5	0.3	-0.077679	-0.084499	19
4	0.2	-0.077681	-0.084525	20
3	0.1	-0.077684	-0.084552	21
2	0.01	-0.077687	-0.084577	22
1	0.001	-0.077688	-0.084580	23
0	0.0001	-0.077688	-0.084580	24
24	100	-0.079621	-0.084937	25
25	500	-0.085785	-0.089789	26

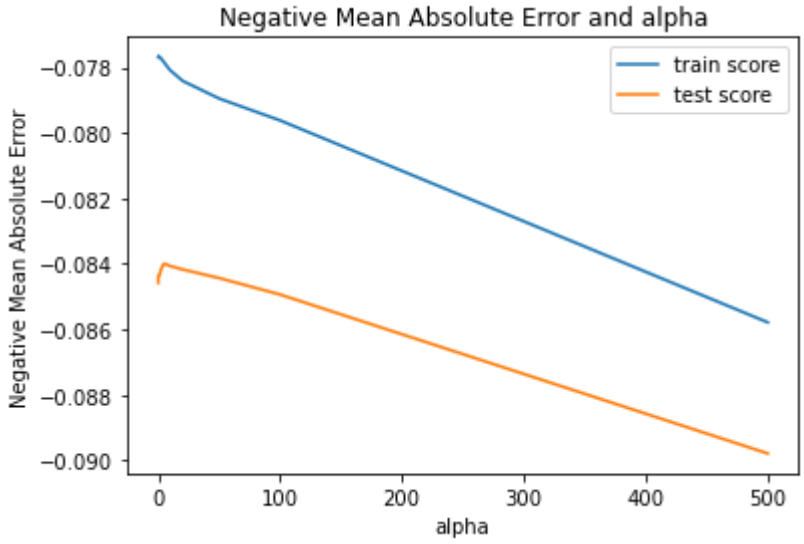
In [92]:

```
# plotting mean test and train scoes with alpha

ridge_cv_results['param_alpha'] = ridge_cv_results['param_alpha'].astype('int32')

# plotting

plt.plot(ridge_cv_results['param_alpha'], ridge_cv_results['mean_train_score'])
plt.plot(ridge_cv_results['param_alpha'], ridge_cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')
plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper right')
plt.show()
```



In [93]:

```
# get the best estimator for lambda

ridge_model_cv.best_estimator_
```

Out[93]:

```
Ridge(alpha=5.0)
```

In [94]:

```
# check the coefficient values with lambda = 10

alpha = 10
ridge = Ridge(alpha=alpha)

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

Out[94]:

```
array([ 0.02234555,  0.06899577,  0.04632114,  0.03236319,  0.04446942,
        0.02397996,  0.0195701 ,  0.07553071,  0.01192122,  0.01262324,
        0.02112442,  0.02159163,  0.0360826 ,  0.01012539, -0.01279944,
       -0.01795906,  0.01351146,  0.01438787,  0.01590523,  0.01837812,
        0.01687347,  0.01482815, -0.0197357 ,  0.01504214,  0.05806608,
        0.02220867,  0.0858179 ,  0.05919111,  0.02555963, -0.01109267,
       -0.00819598, -0.0065469 ,  0.02226521, -0.01495186, -0.00889316,
        0.01698499, -0.00932085, -0.01066399,  0.01215644, -0.03393861,
       -0.0303401 ,  0.00970572, -0.01655244,  0.02914909,  0.0192137 ,
        0.02086807,  0.0414212 ,  0.0169615 ,  0.00614735, -0.01006853])
```

In [95]:

```
# Check the mean squared error

mean_squared_error(y_test, ridge.predict(X_test))
```

Out[95]:

```
0.013677973586354895
```

In [110]:

```
# Put the Features and coefficienst in a dataframe

ridge_df = pd.DataFrame({'Features':X_train.columns, 'Coefficient':ridge.coef_.round(4)})
ridge_df.reset_index(drop=True, inplace=True)
ridge_df
```

Out[110]:

	Features	Coefficient
0	LotArea	0.0223
1	OverallQual	0.0690
2	OverallCond	0.0463
3	BsmtFinSF1	0.0324
4	TotalBsmtSF	0.0445
5	1stFlrSF	0.0240
6	2ndFlrSF	0.0196
7	GrLivArea	0.0755
8	BsmtFullBath	0.0119
9	FullBath	0.0126
10	HalfBath	0.0211
11	Fireplaces	0.0216
12	GarageCars	0.0361
13	WoodDeckSF	0.0101
14	IsRemodelled	-0.0128
15	BuiltOrRemodelAge	-0.0180
16	OldOrNewGarage	0.0135
17	d_BsmtQual	0.0144
18	d_BsmtExposure	0.0159
19	d_HeatingQC	0.0184
20	d_KitchenQual	0.0169
21	d_GarageFinish	0.0148
22	d_BldgType	-0.0197
23	d_SaleCondition	0.0150
24	MSZoning_FV	0.0581
25	MSZoning_RH	0.0222
26	MSZoning_RL	0.0858
27	MSZoning_RM	0.0592
28	Neighborhood_Crawfor	0.0256
29	Neighborhood_Edwards	-0.0111
30	Neighborhood_MeadowV	-0.0082

	Features	Coefficient
31	Neighborhood_NWAmes	-0.0065
32	Neighborhood_NridgHt	0.0223
33	Neighborhood_OldTown	-0.0150
34	Neighborhood_SWISU	-0.0089
35	Neighborhood_StoneBr	0.0170
36	Exterior1st_BrkComm	-0.0093
37	Exterior1st_CemntBd	-0.0107
38	Exterior1st_Stucco	0.0122
39	Exterior1st_VinylSd	-0.0339
40	Exterior1st_Wd Sdng	-0.0303
41	Exterior2nd_CmentBd	0.0097
42	Exterior2nd_Stucco	-0.0166
43	Exterior2nd_VinylSd	0.0291
44	Exterior2nd_Wd Sdng	0.0192
45	Foundation_CBlock	0.0209
46	Foundation_PConc	0.0414
47	Foundation_Slab	0.0170
48	Foundation_Stone	0.0061
49	GarageType_CarPort	-0.0101

In [111]:

```
# Assign the Features and their coefficient values to a dictionary which would be used whil

ridge_coeff_dict = dict(pd.Series(ridge.coef_.round(4), index = X_train.columns))
ridge_coeff_dict
```

Out[111]:

```
{'LotArea': 0.0223,
 'OverallQual': 0.069,
 'OverallCond': 0.0463,
 'BsmtFinSF1': 0.0324,
 'TotalBsmtSF': 0.0445,
 '1stFlrSF': 0.024,
 '2ndFlrSF': 0.0196,
 'GrLivArea': 0.0755,
 'BsmtFullBath': 0.0119,
 'FullBath': 0.0126,
 'HalfBath': 0.0211,
 'Fireplaces': 0.0216,
 'GarageCars': 0.0361,
 'WoodDeckSF': 0.0101,
 'IsRemodelled': -0.0128,
 'BuiltOrRemodelAge': -0.018,
 'OldOrNewGarage': 0.0135,
 'd_BsmtQual': 0.0144,
 'd_BsmtExposure': 0.0159,
 'd_HeatingQC': 0.0184,
 'd_KitchenQual': 0.0169,
 'd_GarageFinish': 0.0148,
 'd_BldgType': -0.0197,
 'd_SaleCondition': 0.015,
 'MSZoning_FV': 0.0581,
 'MSZoning_RH': 0.0222,
 'MSZoning_RL': 0.0858,
 'MSZoning_RM': 0.0592,
 'Neighborhood_Crawfor': 0.0256,
 'Neighborhood_Edwards': -0.0111,
 'Neighborhood_MeadowV': -0.0082,
 'Neighborhood_NWAmes': -0.0065,
 'Neighborhood_NridgHt': 0.0223,
 'Neighborhood_OldTown': -0.015,
 'Neighborhood_SWISU': -0.0089,
 'Neighborhood_StoneBr': 0.017,
 'Exterior1st_BrkComm': -0.0093,
 'Exterior1st_CemntBd': -0.0107,
 'Exterior1st_Stucco': 0.0122,
 'Exterior1st_VinylSd': -0.0339,
 'Exterior1st_Wd Sdng': -0.0303,
 'Exterior2nd_CmentBd': 0.0097,
 'Exterior2nd_Stucco': -0.0166,
 'Exterior2nd_VinylSd': 0.0291,
 'Exterior2nd_Wd Sdng': 0.0192,
 'Foundation_CBlock': 0.0209,
 'Foundation_PConc': 0.0414,
 'Foundation_Slab': 0.017,
 'Foundation_Stone': 0.0061,
 'GarageType_CarPort': -0.0101}
```

In [114]:

```
#RFE

# Do an RFE to minimise the features to 15
X_train_ridge = X_train[ridge_df.Features]

lm = LinearRegression()
lm.fit(X_train_ridge, y_train)

# running RFE
rfe = RFE(lm, 15)
rfe = rfe.fit(X_train_ridge, y_train)
```

In [115]:

```
# Method to get the coefficient values

def find(x):
    return ridge_coeff_dict[x]

# Assign top 10 features to a temp dataframe for further display in the bar plot

temp1_df = pd.DataFrame(list(zip( X_train_ridge.columns, rfe.support_, rfe.ranking_)), colu
temp1_df = temp1_df.loc[temp1_df['rfe_support'] == True]
temp1_df.reset_index(drop=True, inplace=True)

temp1_df['Coefficient'] = temp1_df['Features'].apply(find)
temp1_df = temp1_df.sort_values(by=['Coefficient'], ascending=False)
temp1_df = temp1_df.head(10)
temp1_df
```

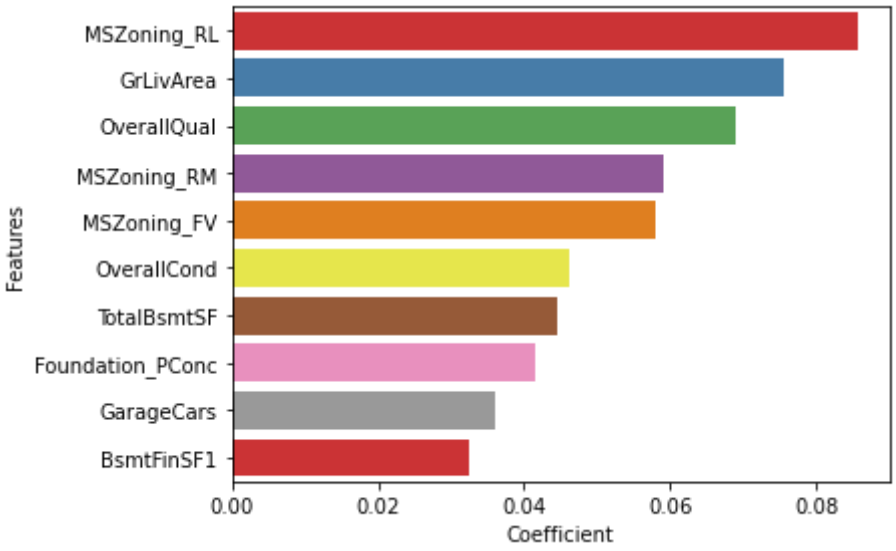
Out[115]:

	Features	rfe_support	rfe_ranking	Coefficient
10	MSZoning_RL	True	1	0.0858
5	GrLivArea	True	1	0.0755
1	OverallQual	True	1	0.0690
11	MSZoning_RM	True	1	0.0592
9	MSZoning_FV	True	1	0.0581
2	OverallCond	True	1	0.0463
4	TotalBsmtSF	True	1	0.0445
14	Foundation_PConc	True	1	0.0414
7	GarageCars	True	1	0.0361
3	BsmtFinSF1	True	1	0.0324

In [116]:

```
# bar plot to determine the variables that would affect pricing most using ridge regression

plt.figure(figsize=(20,20))
plt.subplot(4,3,1)
sns.barplot(y = 'Features', x='Coefficient', palette='Set1', data = temp1_df)
plt.show()
```



In [ ]:

```
#The above graph displays the top 10 variables based on the Ridge Regression model that are
```

In [117]:

```
#Lasso


lasso = Lasso()

# list of alphas

params = {'alpha': [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.0

# cross validation

folds = 5
lasso_model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

lasso_model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done  55 out of  55 | elapsed:    1.4s finished
```

Out[117]:

```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [0.0001, 0.0002, 0.0003, 0.0004, 0.0005,
                                   0.001, 0.002, 0.003, 0.004, 0.005, 0.0
1]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [118]:

```
# display the mean scores

lasso_cv_results = pd.DataFrame(lasso_model_cv.cv_results_)
lasso_cv_results[['param_alpha', 'mean_train_score', 'mean_test_score', 'rank_test_score']]
```

Out[118]:

	param_alpha	mean_train_score	mean_test_score	rank_test_score
4	0.0005	-0.078045	-0.084196	1
3	0.0004	-0.077932	-0.084197	2
2	0.0003	-0.077840	-0.084253	3
1	0.0002	-0.077759	-0.084318	4
0	0.0001	-0.077699	-0.084416	5
5	0.001	-0.078700	-0.084681	6
6	0.002	-0.079565	-0.085242	7
7	0.003	-0.080113	-0.085610	8
8	0.004	-0.080610	-0.086044	9
9	0.005	-0.081198	-0.086506	10
10	0.01	-0.085115	-0.089263	11

In [119]:

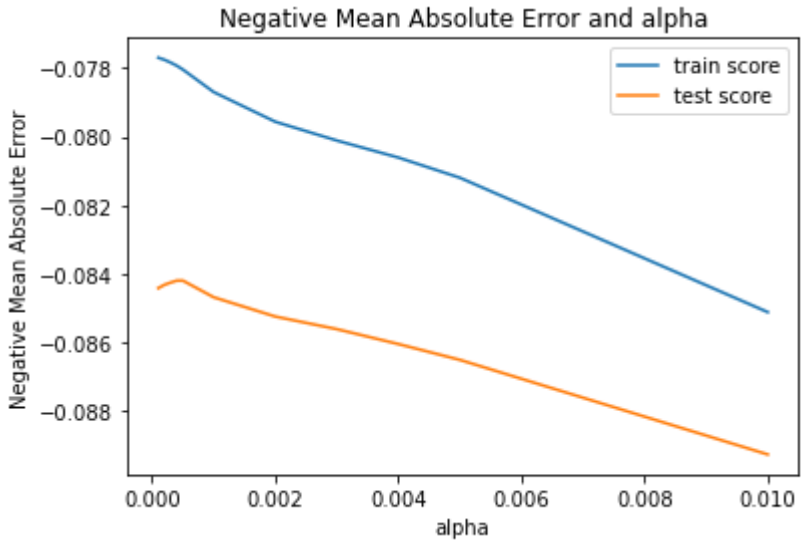
```
# plotting mean test and train scoes with alpha

lasso_cv_results['param_alpha'] = lasso_cv_results['param_alpha'].astype('float64')

# plotting

plt.plot(lasso_cv_results['param_alpha'], lasso_cv_results['mean_train_score'])
plt.plot(lasso_cv_results['param_alpha'], lasso_cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper right')
plt.show()
```



In [120]:

```
# get the best estimator for lambda

lasso_model_cv.best_estimator_
```

Out[120]:

```
Lasso(alpha=0.0005)
```

In [121]:

```
# check the coefficient values with lambda = 0.0004

alpha = 0.0004

lasso = Lasso(alpha=alpha)

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

Out[121]:

```
array([ 0.02206294,  0.07021892,  0.04589482,  0.03282556,  0.04527864,
        0.00655867, -0.        ,  0.09900629,  0.01092904,  0.01056894,
        0.01961002,  0.02098685,  0.03696972,  0.009914  , -0.01196011,
       -0.01772065,  0.01316803,  0.01336422,  0.0160723 ,  0.01799361,
        0.01629107,  0.01519127, -0.01937469,  0.01457282,  0.06898952,
        0.02728223,  0.10738587,  0.0769138 ,  0.02518834, -0.01083779,
       -0.00779288, -0.0058438 ,  0.02191322, -0.01528071, -0.00970672,
        0.01642764, -0.00923118, -0.00035289,  0.01163966, -0.03062255,
       -0.02962691,  0.        , -0.01602983,  0.02593184,  0.0179302 ,
        0.01989064,  0.04133948,  0.01613226,  0.00637348, -0.00913834])
```

In [122]:

```
# Check the mean squared error

mean_squared_error(y_test, lasso.predict(X_test))
```

Out[122]:

```
0.013533659502462882
```

In [123]:

```
# Put the shortlisted Features and coefficienst in a dataframe

lasso_df = pd.DataFrame({'Features':X_train.columns, 'Coefficient':lasso.coef_.round(4)})
lasso_df = lasso_df[lasso_df['Coefficient'] != 0.00]
lasso_df.reset_index(drop=True, inplace=True)
lasso_df
```

Out[123]:

	Features	Coefficient
0	LotArea	0.0221
1	OverallQual	0.0702
2	OverallCond	0.0459
3	BsmtFinSF1	0.0328
4	TotalBsmtSF	0.0453
5	1stFlrSF	0.0066
6	GrLivArea	0.0990
7	BsmtFullBath	0.0109
8	FullBath	0.0106
9	HalfBath	0.0196
10	Fireplaces	0.0210
11	GarageCars	0.0370
12	WoodDeckSF	0.0099
13	IsRemodelled	-0.0120
14	BuiltOrRemodelAge	-0.0177
15	OldOrNewGarage	0.0132
16	d_BsmtQual	0.0134
17	d_BsmtExposure	0.0161
18	d_HeatingQC	0.0180
19	d_KitchenQual	0.0163
20	d_GarageFinish	0.0152
21	d_BldgType	-0.0194
22	d_SaleCondition	0.0146
23	MSZoning_FV	0.0690
24	MSZoning_RH	0.0273
25	MSZoning_RL	0.1074
26	MSZoning_RM	0.0769
27	Neighborhood_Crawfor	0.0252
28	Neighborhood_Edwards	-0.0108
29	Neighborhood_MeadowV	-0.0078

	Features	Coefficient
30	Neighborhood_NWAmes	-0.0058
31	Neighborhood_NridgHt	0.0219
32	Neighborhood_OldTown	-0.0153
33	Neighborhood_SWISU	-0.0097
34	Neighborhood_StoneBr	0.0164
35	Exterior1st_BrkComm	-0.0092
36	Exterior1st_CemntBd	-0.0004
37	Exterior1st_Stucco	0.0116
38	Exterior1st_VinylSd	-0.0306
39	Exterior1st_Wd Sdng	-0.0296
40	Exterior2nd_Stucco	-0.0160
41	Exterior2nd_VinylSd	0.0259
42	Exterior2nd_Wd Sdng	0.0179
43	Foundation_CBlock	0.0199
44	Foundation_PConc	0.0413
45	Foundation_Slab	0.0161
46	Foundation_Stone	0.0064
47	GarageType_CarPort	-0.0091

In [124]:

```
# Put the Features and Coefficients in dictionary

lasso_coeff_dict = dict(pd.Series(lasso.coef_, index = X_train.columns))
lasso_coeff_dict
```

Out[124]:

```
{'LotArea': 0.022062943657806817,
 'OverallQual': 0.07021892032148519,
 'OverallCond': 0.045894819270617185,
 'BsmtFinSF1': 0.03282555663101663,
 'TotalBsmtSF': 0.04527863853305824,
 '1stFlrSF': 0.006558671210433306,
 '2ndFlrSF': -0.0,
 'GrLivArea': 0.09900629383174138,
 'BsmtFullBath': 0.010929044096718203,
 'FullBath': 0.010568936891077033,
 'HalfBath': 0.019610015612012706,
 'Fireplaces': 0.020986846775147682,
 'GarageCars': 0.03696972257759497,
 'WoodDeckSF': 0.009913999664875663,
 'IsRemodelled': -0.011960109671270612,
 'BuiltOrRemodelAge': -0.01772064986305161,
 'OldOrNewGarage': 0.013168028065982355,
 'd_BsmtQual': 0.013364221830997716,
 'd_BsmtExposure': 0.016072301871762663,
 'd_HeatingQC': 0.01799361158284386,
 'd_KitchenQual': 0.016291071217230352,
 'd_GarageFinish': 0.015191271389084984,
 'd_BldgType': -0.01937469353046299,
 'd_SaleCondition': 0.014572816652899747,
 'MSZoning_FV': 0.06898951586372384,
 'MSZoning_RH': 0.027282230431137054,
 'MSZoning_RL': 0.1073858736014483,
 'MSZoning_RM': 0.07691379565064048,
 'Neighborhood_Crawfor': 0.02518833901870593,
 'Neighborhood_Edwards': -0.010837785569905405,
 'Neighborhood_MeadowV': -0.007792878454139047,
 'Neighborhood_NWAmes': -0.00584379771207938,
 'Neighborhood_NridgHt': 0.021913223954609047,
 'Neighborhood_OldTown': -0.015280712747488765,
 'Neighborhood_SWISU': -0.009706720435173797,
 'Neighborhood_StoneBr': 0.016427638546201488,
 'Exterior1st_BrkComm': -0.009231175993689712,
 'Exterior1st_CemntBd': -0.00035288911339149737,
 'Exterior1st_Stucco': 0.011639660494078368,
 'Exterior1st_VinylSd': -0.03062255484446454,
 'Exterior1st_Wd Sdng': -0.029626910988984376,
 'Exterior2nd_CmentBd': 0.0,
 'Exterior2nd_Stucco': -0.016029830024977237,
 'Exterior2nd_VinylSd': 0.02593183841795492,
 'Exterior2nd_Wd Sdng': 0.01793020338090469,
 'Foundation_CBlock': 0.0198906406157244,
 'Foundation_PConc': 0.04133947830477581,
 'Foundation_Slab': 0.01613226014986512,
 'Foundation_Stone': 0.006373482123769002,
 'GarageType_CarPort': -0.009138342702628604}
```

In [125]:

```
#RFE

# Do an RFE to minimise the features to 15

X_train_lasso = X_train[lasso_df.Features]

lm = LinearRegression()
lm.fit(X_train_lasso, y_train)

# running RFE

rfe = RFE(lm, 15)
rfe = rfe.fit(X_train_lasso, y_train)
```

In [126]:

```
# Method to get the coefficient values

def find(x):
    return lasso_coeff_dict[x]

# Assign top 10 features to a temp dataframe for further display in the bar plot

temp2_df = pd.DataFrame(list(zip( X_train_lasso.columns, rfe.support_, rfe.ranking_)), colu
temp2_df = temp2_df.loc[temp2_df['rfe_support'] == True]
temp2_df.reset_index(drop=True, inplace=True)

temp2_df['Coefficient'] = temp2_df['Features'].apply(find)
temp2_df = temp2_df.sort_values(by=['Coefficient'], ascending=False)
temp2_df = temp2_df.head(10)
temp2_df
```

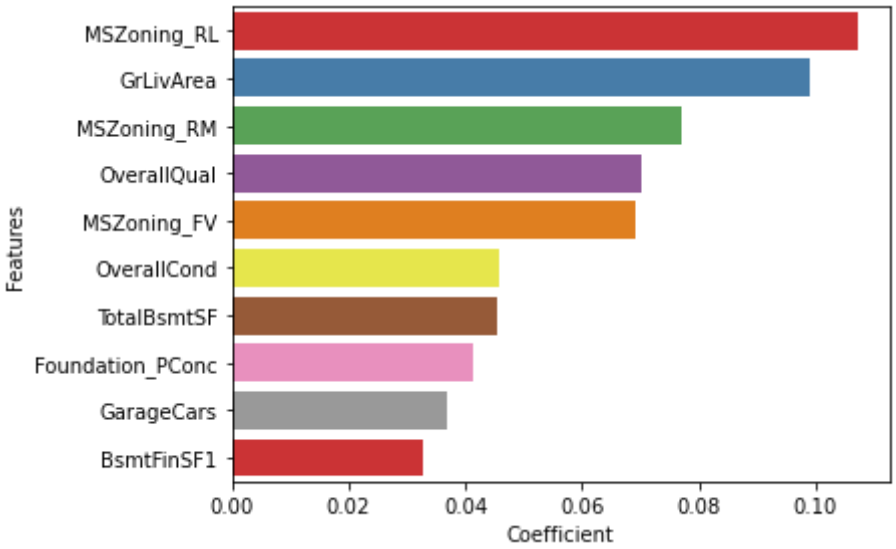
Out[126]:

	Features	rfe_support	rfe_ranking	Coefficient
11	MSZoning_RL	True	1	0.107386
5	GrLivArea	True	1	0.099006
12	MSZoning_RM	True	1	0.076914
1	OverallQual	True	1	0.070219
9	MSZoning_FV	True	1	0.068990
2	OverallCond	True	1	0.045895
4	TotalBsmtSF	True	1	0.045279
14	Foundation_PConc	True	1	0.041339
7	GarageCars	True	1	0.036970
3	BsmtFinSF1	True	1	0.032826

In [127]:

```
# bar plot to determine the variables that would affect pricing most using ridge regression

plt.figure(figsize=(20,20))
plt.subplot(4,3,1)
sns.barplot(y = 'Features', x='Coefficient', palette='Set1', data = temp2_df)
plt.show()
```



In [ ]:

```
'''The above graph displays the top 10 variables based on the Lasso Regression
model that are significant in predicting the price of a house.
'''
```

In [128]:

```
'''
Conclusion :
The optimal lambda value in case of Ridge and Lasso is as below:

Ridge - 10
Lasso - 0.0004
The Mean Squared error in case of Ridge and Lasso are:

Ridge - 0.013743
Lasso - 0.013556
The Mean Squared Error of Lasso is slightly lower than that of Ridge

Also, since Lasso helps in feature reduction (as the coefficient value of one
of the feature became 0), Lasso has a better edge over Ridge.

Hence based on Lasso, the factors that generally affect the price are the
Zoning classification, Living area square feet, Overall quality and condition
of the house, Foundation type of the house, Number of cars that can be
accomodated in the garage, Total basement area in square feet and the Basement
finished square feet area

Therefore, the variables predicted by Lasso in the above bar chart as
significant variables for predicting the price of a house.
'''
```

Out[128]:

```
'\nConclusion :\nThe optimal lambda value in case of Ridge and Lasso is as b
elow:\n\nRidge - 10\nLasso - 0.0004\nThe Mean Squared error in case of Ridge
and Lasso are:\n\nRidge - 0.013743\nLasso - 0.013556\nThe Mean Squared Error
of Lasso is slightly lower than that of Ridge\n\nAlso, since Lasso helps in
feature reduction (as the coefficient value of one \nof the feature became
0), Lasso has a better edge over Ridge.\n\nHence based on Lasso, the factors
that generally affect the price are the \nZoning classification, Living area
square feet, Overall quality and condition \nof the house, Foundation type o
f the house, Number of cars that can be \naccomodated in the garage, Total b
asement area in square feet and the Basement \nfinished square feet area\n\n
Therefore, the variables predicted by Lasso in the above bar chart as \nsign
ificant variables for predicting the price of a house.\n'
```