Problem Statement :

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 at a higher price. 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.**

Based on various market surveys, the consulting firm has gathered a large dataset of different types of cars across the American market.

Business Goal :

- **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.**
- **Determine the optimal value of lambda for ridge and lasso regression.**
- **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.**
- **The model will be a good way for the management to understand the pricing dynamics of a new market.**

Step 1: Reading and Understanding the Data¶

Let us first import NumPy and Pandas and read the dataset

In [1]:

```
# Supress Warnings

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

In [2]:

```
# 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
```

```
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:30: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `f
loat` by itself. Doing this will not modify any behavior and is safe. If you specifically
wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
```

```
  method='lar', copy_X=True, eps=np.finfo(np.float).eps,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:167: DeprecationWarning
: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  method='lar', copy_X=True, eps=np.finfo(np.float).eps,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:284: DeprecationWarning
: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_Gram=True, verbose=0,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:862: DeprecationWarning
: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:1101: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:1127: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, positive=False):
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:1362: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:1602: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
D:\anaconda\lib\site-packages\sklearn\linear_model\least_angle.py:1738: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you specifical
ly wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, positive=False):
D:\anaconda\lib\site-packages\sklearn\decomposition\online_lda.py:29: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `f
loat` by itself. Doing this will not modify any behavior and is safe. If you specifically
wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  EPS = np.finfo(np.float).eps
```

In [3]:

```
# Importing dataset

housingInfo = pd.read_csv('train.csv', encoding = 'latin')
housingInfo.head()
```

Out[3]:

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

5 rows × 81 columns

In [4]:

```
# 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()
```

```
*********************************  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
None
*********************************  Shape ******************************************
(1460, 81)
**************************** Columns having null values *****************************
Id                False
MSSubClass        False
MSZoning          False
LotFrontage        True
LotArea           False
                  ...
MoSold            False
YrSold            False
SaleType          False
SaleCondition     False
SalePrice         False
Length: 81, dtype: bool
**************************** Describe *****************************
```

Out[4]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVı
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.(

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasV
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.6
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.0
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.0
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.0
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.0
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.0
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.0

8 rows × 38 columns

Step 2: Data Cleaning

Removing/Imputing NaN values in Categorical attributes

In [5]:

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

housingInfo.select_dtypes(include='object').isnull().sum()[housingInfo.select_dtypes(incl
ude='object').isnull().sum()>0]
```

Out[5]:

```
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 [6]:

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

for col in ('Alley', 'MasVnrType', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1
', 'BsmtFinType2', 'FireplaceQu',
            'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence',
'MiscFeature', 'Electrical'):

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

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

In [7]:

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

# 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[7]:

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

5 rows × 60 columns

In [8]:

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

housingInfo.select_dtypes(include='object').isnull().sum()[housingInfo.select_dtypes(incl
ude='object').isnull().sum()>0]
```

Out[8]:

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

No more null values in the categorical variables

Removing null values in Numerical attributes

In [10]:

```
# check the null values in the numerical data

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

Out[10]:

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

In [11]:

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

housingInfo['LotFrontage'] = housingInfo['LotFrontage'].replace(np.nan, housingInfo['Lot
Frontage'].median())
housingInfo['MasVnrArea'] = housingInfo['MasVnrArea'].replace(np.nan, housingInfo['MasVn
rArea'].median())
```

In [12]:

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

```
umn further below

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

In [13]:

```
# Create a new column named IsRemodelled - This column would determine whether the house has been remodelled or not based on
# 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[13]:

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

5 rows × 61 columns

In [14]:

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

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[14]:

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

5 rows × 62 columns

In [15]:

```
# 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(getGarageConstructionPer
iod)
housingInfo.head()
```

Out[15]:

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

5 rows × 63 columns

In [16]:

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

housingInfo.drop(['YearBuilt', 'YearRemodAdd', 'YrSold', 'GarageYrBlt'], axis = 1, inpla
ce = True)
```

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

In [17]:

```
# Drop the following columns that have more than 85% values associated to a specific valu
e
# 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[17]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	BldgType	HouseStyle	...	Garage(
0	1	60	RL	65.0	8450	Reg	Inside	CollgCr	1Fam	2Story	...	

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	BldgType	HouseStyle	...	Garage(
1	2	20	RL	80.0	9600	Reg	FR2	Veenker	1Fam	1Story	...	
2	3	60	RL	68.0	11250	IR1	Inside	CollgCr	1Fam	2Story	...	
3	4	70	RL	60.0	9550	IR1	Corner	Crawfor	1Fam	2Story	...	
4	5	60	RL	84.0	14260	IR1	FR2	NoRidge	1Fam	2Story	...	

5 rows × 49 columns

In [18]:

```
# 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
```

Hence there are no null values in the dataset

Check for Duplicates

In [19]:

```
# Check if there are any duplicate values in the dataset

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

Out[19]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LotConfig	Neighborhood	BldgType	HouseStyle	...	GarageC

0 rows × 49 columns

No duplicate entries found !!!

Outlier Treatment

In [20]:

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

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

Out[20]:

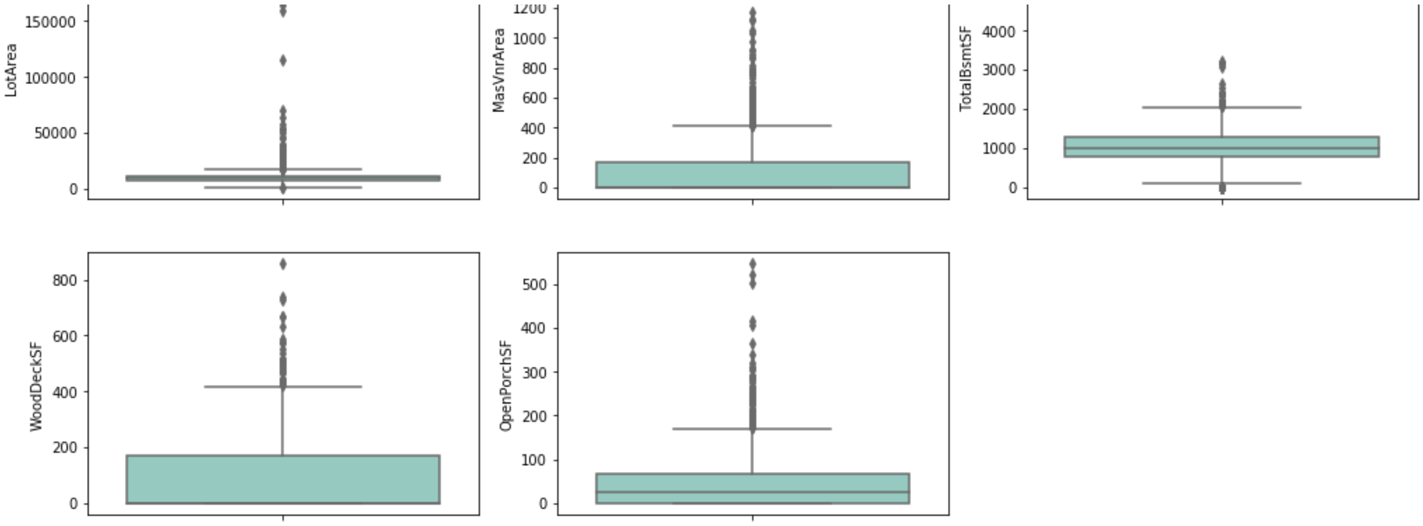
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtUnf
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.0000
mean	730.500000	56.897260	69.863699	10516.828082	6.099315	5.575342	103.117123	443.639726	567.2404
std	421.610009	42.300571	22.027677	9981.264932	1.382997	1.112799	180.731373	456.098091	441.8669
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	0.000000	0.000000	0.0000
25%	365.750000	20.000000	60.000000	7553.500000	5.000000	5.000000	0.000000	0.000000	223.0000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	0.000000	383.500000	477.5000
75%	1095.250000	70.000000	79.000000	11601.500000	7.000000	6.000000	164.250000	712.250000	808.0000
90%	1314.100000	120.000000	92.000000	14381.700000	8.000000	7.000000	335.000000	1065.500000	1232.0000
95%	1387.050000	160.000000	104.000000	17401.150000	8.000000	8.000000	456.000000	1274.000000	1468.0000
99%	1445.410000	190.000000	137.410000	37567.640000	10.000000	9.000000	791.280000	1572.410000	1797.0500
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	1600.000000	5644.000000	2336.0000

11 rows × 27 columns

In [21]:

```
# 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 [22]:

```
# 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 [23]:

```
# Determine the percentage of data retained

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

```
93.01
```

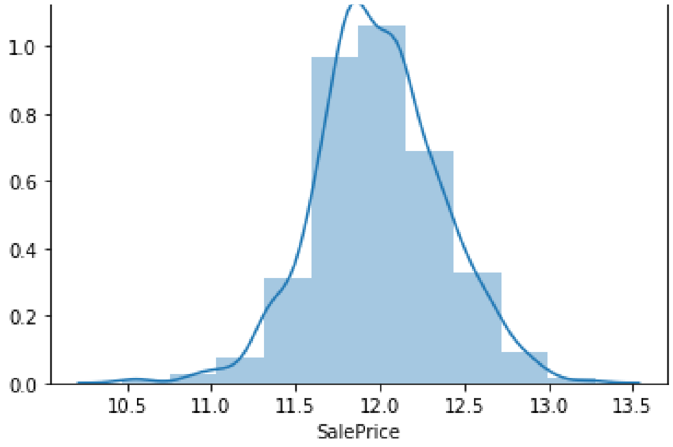
Step 3: Data Visualization

In [24]:

```
# 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()
```



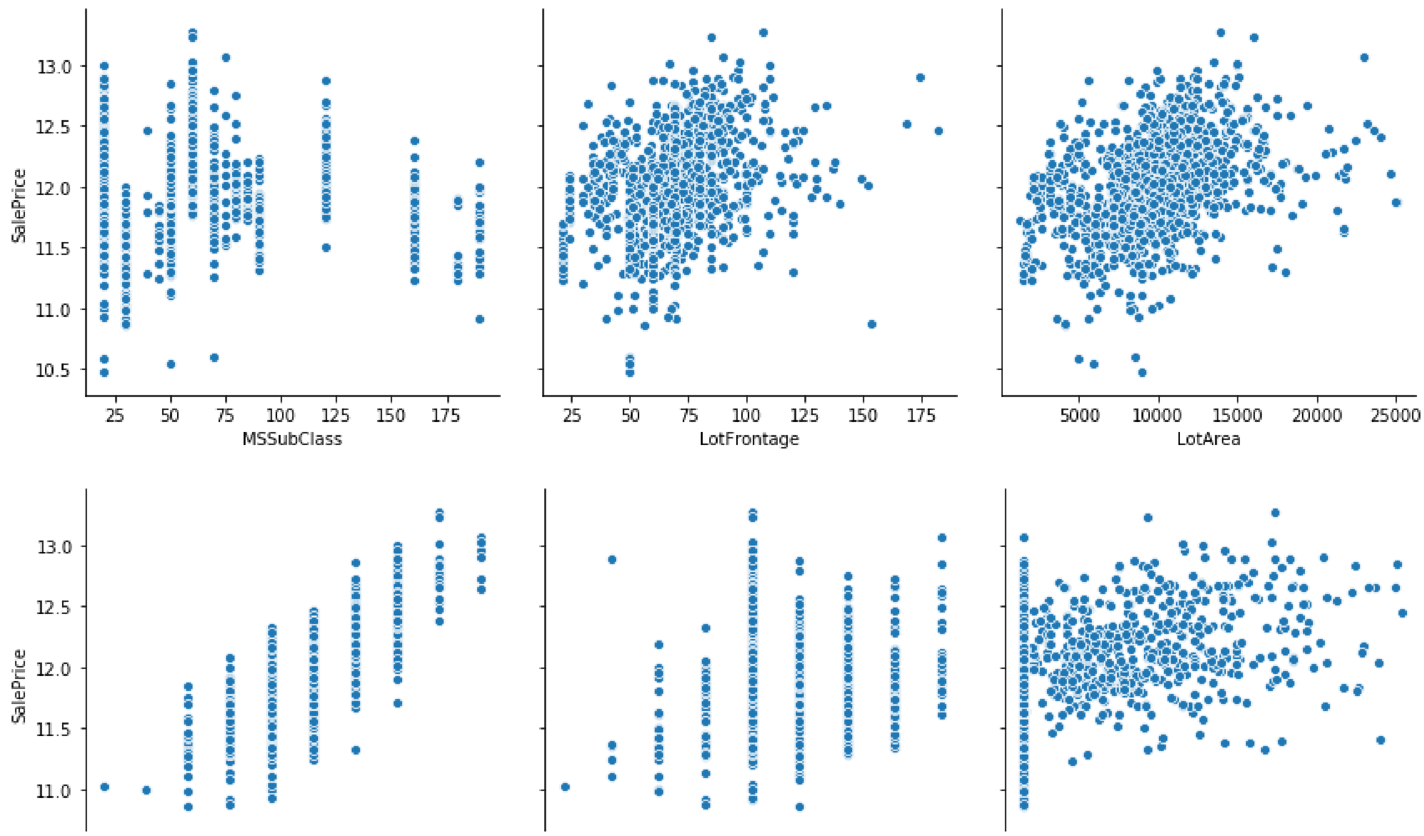
The target value seems to be normalized with some noise.

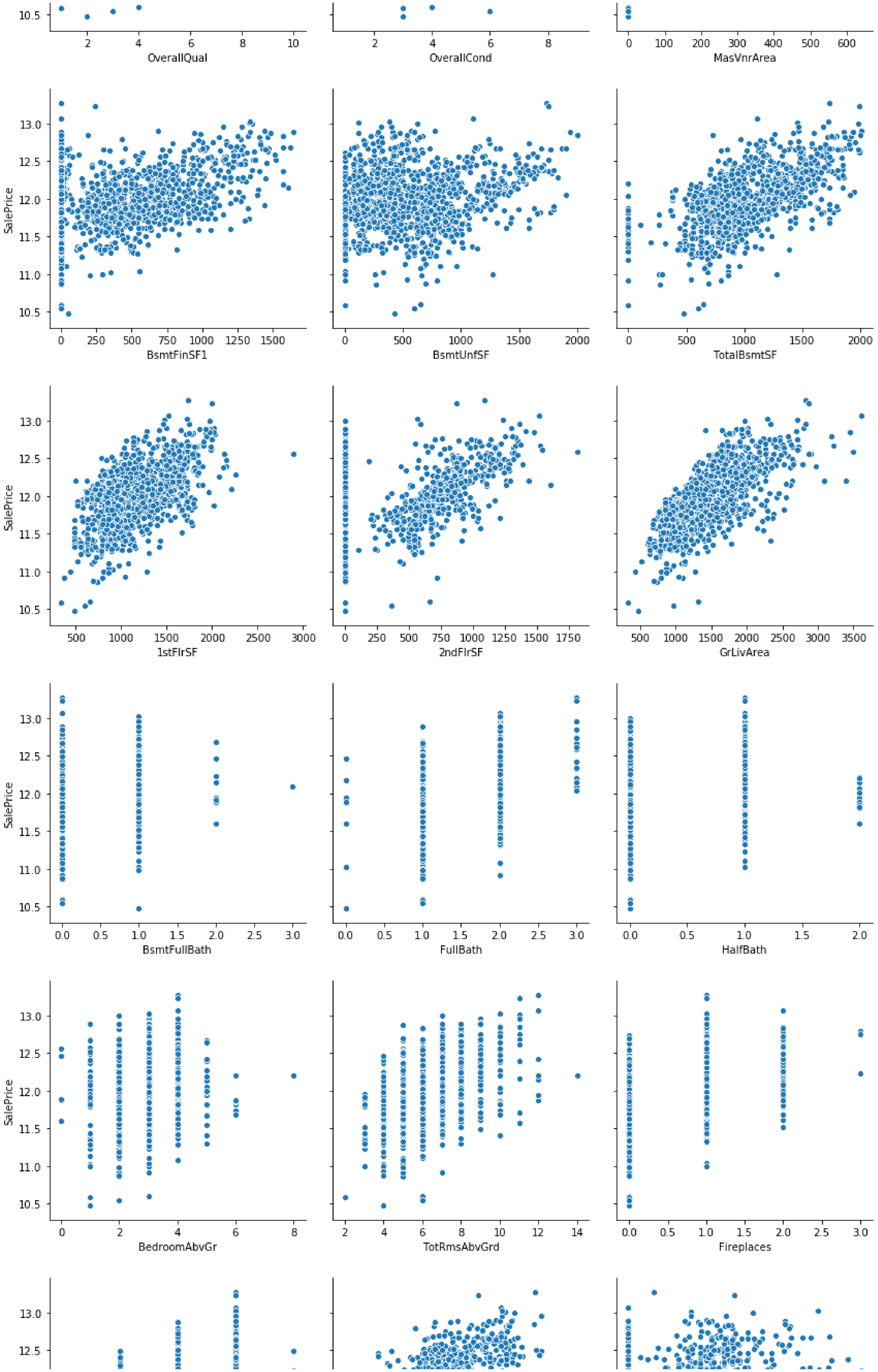
In [25]:

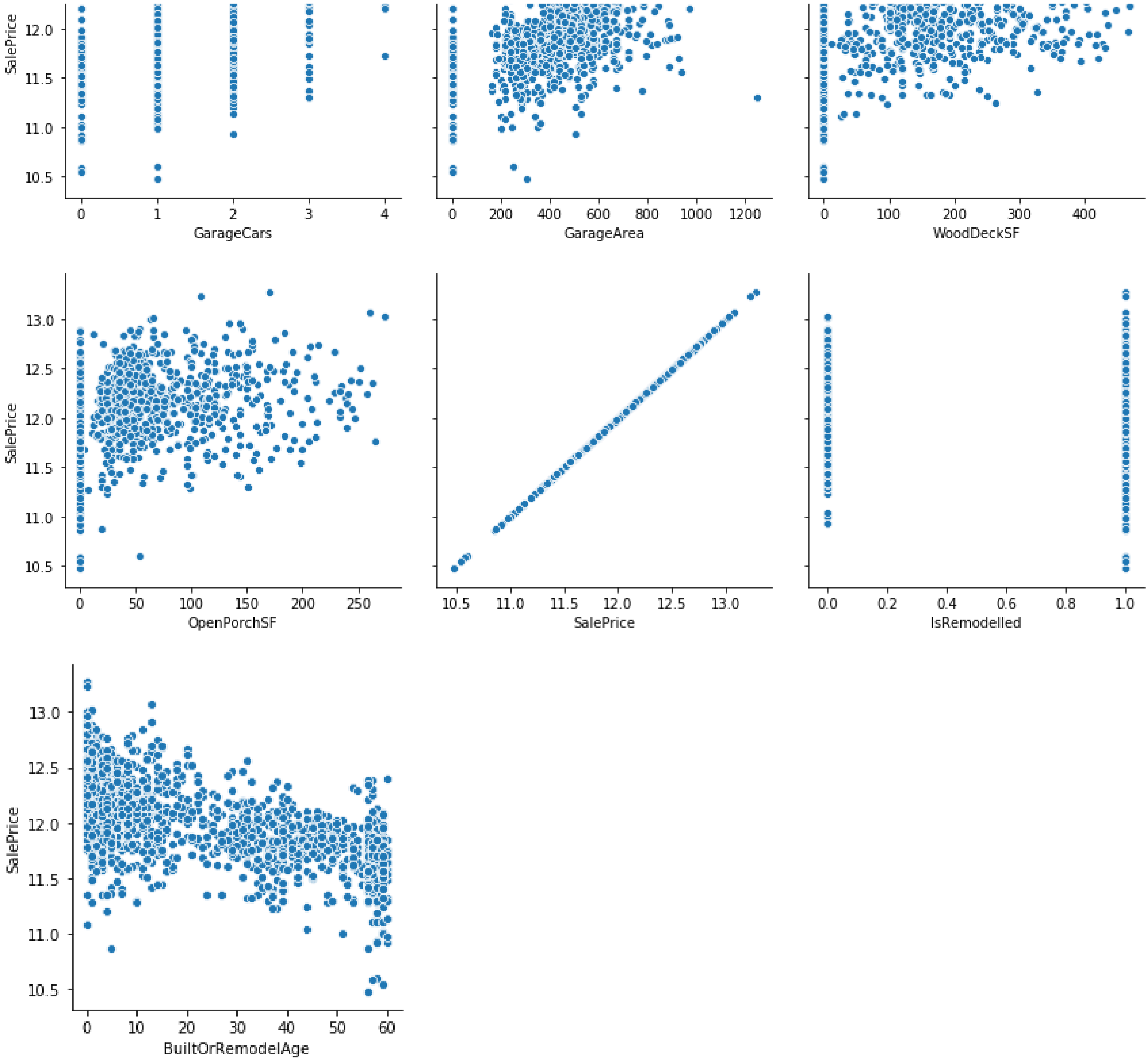
```
# Check the numerical values using pairplots

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

```
<Figure size 720x360 with 0 Axes>
```




OverallQual
OverallCond
MasVnrArea
SalePrice
BsmtFinSF1
BsmtUnfSF
TotalBsmtSF
SalePrice
1stFlrSF
2ndFlrSF
GrLivArea
SalePrice
BsmtFullBath
FullBath
HalfBath
SalePrice
BedroomAbvGr
TotRmsAbvGrd
Fireplaces



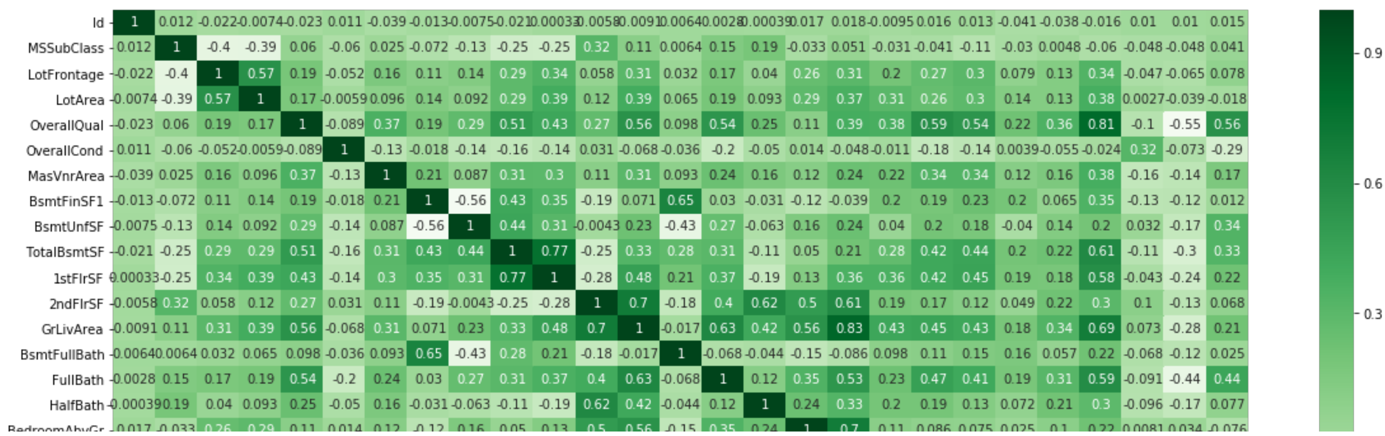
Observations :

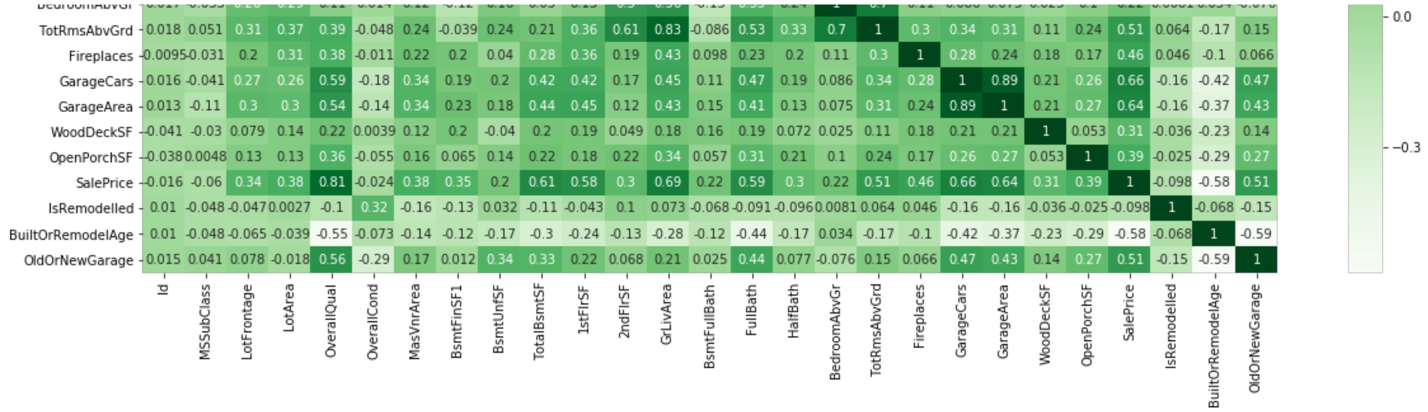
- **1stFlrSF, GrLivArea seems to be showing correlation towards right**
- **Rest of the variables are too scattered and hence can be understood during further analysis**

In [26]:

```
# Check the correlation of numerical columns

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





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 [27]:

```
# Removing the highly correlated variables

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

In [28]:

```
# Check the shape of the dataframe

housingInfo.shape
```

Out[28]:

```
(1358, 47)
```

Step 4: Data Preparation

- **Converting categorical data into numercal data**
- **Creating Dummies**

In [29]:

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

# 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': 0})
housingInfo['d_ExterQual'] = housingInfo['ExterQual'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0 })
housingInfo['d_BsmtQual'] = housingInfo['BsmtQual'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
housingInfo['d_BsmtExposure'] = housingInfo['BsmtExposure'].map({'Gd': 4, 'Av': 3, 'Mn': 2, 'No': 1, 'None': 0})
housingInfo['d_BsmtFinType1'] = housingInfo['BsmtFinType1'].map({'GLQ': 6, 'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1,
                                                                 'None': 0})
housingInfo['d_HeatingQC'] = housingInfo['HeatingQC'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
housingInfo['d_KitchenQual'] = housingInfo['KitchenQual'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
housingInfo['d_FireplaceQu'] = housingInfo['FireplaceQu'].map({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
housingInfo['d_GarageFinish'] = housingInfo['GarageFinish'].map({'Fin': 3, 'RFn': 2, 'Un
```

```
f': 1, 'None': 0 })
housingInfo['d_BldgType'] = housingInfo['BldgType'].map({'Twnhs': 5, 'TwnhsE': 4, 'Duple
x': 3, '2fmCon': 2, '1Fam': 1,
                                                                   'None': 0 })
housingInfo['d_HouseStyle'] = housingInfo['HouseStyle'].map({'SLvl': 8, 'SFoyer': 7, '2.
5Fin': 6, '2.5Unf': 5, '2Story': 4,
                                                                   '1.5Fin': 3, '1.5Unf':
2, '1Story': 1, 'None': 0 })
housingInfo['d_Fence'] = housingInfo['Fence'].map({'GdPrv': 4, 'GdWo': 3, 'MnPrv': 2, 'M
nWw': 1, 'None': 0 })
housingInfo['d_LotConfig'] = housingInfo['LotConfig'].map({'Inside': 5, 'Corner': 4, 'Cu
lDSac': 3, 'FR2': 2, 'FR3': 1,
                                                             'None': 0  })
housingInfo['d_MasVnrType'] = housingInfo['MasVnrType'].map({'BrkCmn': 1, 'BrkFace': 1,
'CBlock': 1, 'Stone': 1, 'None': 0 })
housingInfo['d_SaleCondition'] = housingInfo['SaleCondition'].map({'Normal': 1, 'Partial
': 1, 'Abnorml': 0, 'Family': 0,
                                                                     'Alloca': 0, 'AdjLan
d': 0, 'None': 0})
housingInfo.head()
```

Out[29]:

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

5 rows × 62 columns

In [30]:

```
# 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
', 'BsmtFinType1', 'HeatingQC',
                                'KitchenQual', 'FireplaceQu', 'GarageFinish', 'BldgType'
, 'HouseStyle', 'Fence',
                                'LotConfig', 'MasVnrType', 'SaleCondition'], axis=1)

housingInfo.head()
```

Out[30]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Neighborhood	OverallQual	OverallCond	RoofStyle	Exterior1st	Exterior2n
0	60	RL	65.0	8450	CollgCr	7	5	Gable	VinylSd	VinylS
1	20	RL	80.0	9600	Veenker	6	8	Gable	MetalSd	MetalS
2	60	RL	68.0	11250	CollgCr	7	5	Gable	VinylSd	VinylS
3	70	RL	60.0	9550	Crawfor	7	5	Gable	Wd Sdng	Wd Shn
4	60	RL	84.0	14260	NoRidge	8	5	Gable	VinylSd	VinylS

5 rows × 46 columns

In [31]:

```
# 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_first = True)
housingInfo = pd.concat([housingInfo, d_Neighborhood], axis = 1)

# Creating dummies for RoofStyle

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

# Creating dummies for Exterior1st

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

# Creating dummies for Exterior2nd

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

# Creating dummies for Foundation

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

# Creating dummies for GarageType

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

housingInfo.head()
```

Out[31]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Neighborhood	OverallQual	OverallCond	RoofStyle	Exterior1st	Exterior2n
0	60	RL	65.0	8450	CollgCr	7	5	Gable	VinylSd	VinylS
1	20	RL	80.0	9600	Veenker	6	8	Gable	MetalSd	MetalS
2	60	RL	68.0	11250	CollgCr	7	5	Gable	VinylSd	VinylS
3	70	RL	60.0	9550	Crawfor	7	5	Gable	Wd Sdng	Wd Shn
4	60	RL	84.0	14260	NoRidge	8	5	Gable	VinylSd	VinylS

5 rows × 119 columns

In [32]:

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

housingInfo = housingInfo.drop(['MSZoning', 'Neighborhood', 'RoofStyle', 'Exterior1st', 'Exterior2nd', 'Foundation',
                                'GarageType'], axis=1)

housingInfo.head()
```

Out[32]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtUnfSF	TotalBsmtSF	1stFlrSI

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtUnfSF	TotalBsmtSF	1stFlrSl
0	60	65.0	8450	7	5	196.0	706	150	856	85
1	20	80.0	9600	6	8	0.0	978	284	1262	126
2	60	68.0	11250	7	5	162.0	486	434	920	92
3	70	60.0	9550	7	5	0.0	216	540	756	96
4	60	84.0	14260	8	5	350.0	655	490	1145	114

5 rows × 112 columns

In [33]:

```
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
```

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

Step 5: Train Test Split

In [34]:

```
# Putting all feature variable to X

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

Out[34]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtUnfSF	TotalBsmtSF	1stFlrSl
0	60	65.0	8450	7	5	196.0	706	150	856	85
1	20	80.0	9600	6	8	0.0	978	284	1262	126
2	60	68.0	11250	7	5	162.0	486	434	920	92
3	70	60.0	9550	7	5	0.0	216	540	756	96
4	60	84.0	14260	8	5	350.0	655	490	1145	114

5 rows × 111 columns

In [35]:

```
# Putting response variable to y

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

Out[35]:

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

Scaling the features

In [36]:

```
# 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[36]:

```
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 [38]:

```
# split into train and test

#from sklearn.cross_validation import train_test_split
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, random_state=42)
```

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 [39]:

```
# 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)
```

```
D:\anaconda\lib\site-packages\sklearn\feature_selection\rfe.py:167: DeprecationWarning: `
np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool
` by itself. Doing this will not modify any behavior and is safe. If you specifically wan
ted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  support_ = np.ones(n_features, dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\feature_selection\rfe.py:168: DeprecationWarning: `
np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` b
y itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, y
ou may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to
review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  ranking_ = np.ones(n_features, dtype=np.int)
```

In [40]:

```
# 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=['Va
riable', 'rfe_support', 'rfe_ranking'])
temp_df = temp_df.loc[temp_df['rfe_support'] == True]
temp_df.reset_index(drop=True, inplace=True)

temp_df
```

Out[40]:

	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_BsmtFinType1	True	1
20	d_HeatingQC	True	1
21	d_KitchenQual	True	1
22	d_GarageFinish	True	1
23	d_BldgType	True	1
24	d_SaleCondition	True	1
25	MSZoning_FV	True	1
26	MSZoning_RH	True	1
27	MSZoning_RL	True	1
28	MSZoning_RM	True	1
29	Neighborhood_Crawfor	True	1
30	Neighborhood_Edwards	True	1
31	Neighborhood_MeadowV	True	1
32	Neighborhood_NridgHt	True	1
33	Neighborhood_OldTown	True	1
34	Neighborhood_SWISU	True	1

	Variable	rfe_support	rfe_ranking
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 [41]:

```
# Assign the 50 columns to X_train_rfe

X_train_rfe = X_train[col]
```

In [42]:

```
# 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]
```

Step 6: Model Building and Evaluation¶

Ridge

In [43]:

```
# 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, 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
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
```

```
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:442: DeprecationWarning:
`np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int`
by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`,
you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish t
o review your current use, check the release note link for additional information.
```

```
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 135 out of 135 | elapsed:    1.2s finished
D:\anaconda\lib\site-packages\sklearn\model_selection\_search.py:793: DeprecationWarning:
`np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int`
by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`,
you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish t
o review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  dtype=np.int)
```

Out[43]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='warn', n_jobs=None,
             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, 500,
                                   1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_absolute_error', verbose=1)
```

In [44]:

```
# 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'
]].sort_values(by = ['rank_test_score'])
```

Out[44]:

	param_alpha	mean_train_score	mean_test_score	rank_test_score
16	5.0	-0.077755	-0.083854	1
17	6.0	-0.077806	-0.083862	2
18	7.0	-0.077856	-0.083877	3
19	8.0	-0.077903	-0.083890	4
15	4.0	-0.077707	-0.083897	5
20	9.0	-0.077949	-0.083902	6
21	10.0	-0.077992	-0.083919	7
14	3.0	-0.077660	-0.083976	8
22	20	-0.078320	-0.084037	9
13	2.0	-0.077612	-0.084070	10
12	1.0	-0.077573	-0.084209	11
11	0.9	-0.077572	-0.084229	12
10	0.8	-0.077570	-0.084249	13
9	0.7	-0.077570	-0.084270	14
8	0.6	-0.077571	-0.084292	15
7	0.5	-0.077573	-0.084314	16
6	0.4	-0.077575	-0.084338	17
23	50	-0.078887	-0.084339	18
5	0.3	-0.077577	-0.084363	19
4	0.2	-0.077580	-0.084390	20
3	0.1	-0.077584	-0.084417	21
2	0.01	-0.077589	-0.084442	22
1	0.001	-0.077590	-0.084445	23
0	0.0001	-0.077590	-0.084445	24
24	100	-0.079584	-0.084811	25
25	500	-0.085651	-0.089621	26

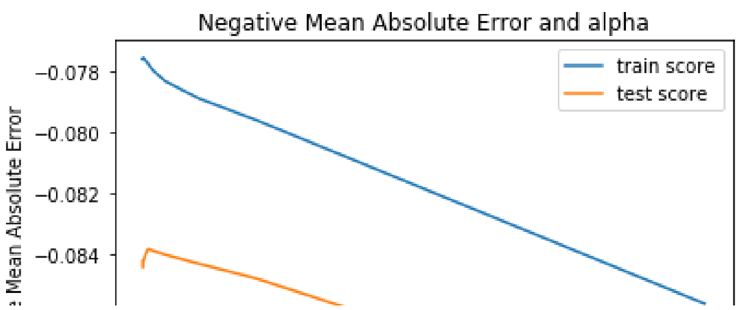
In [45]:

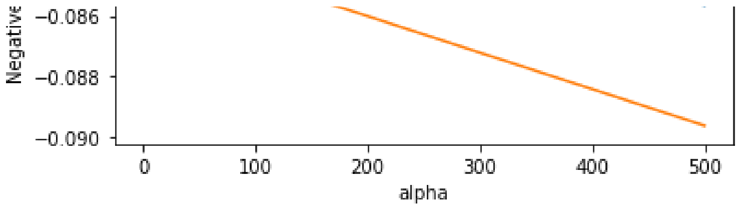
```
# plotting mean test and train scoes with alpha

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 [46]:

```
# get the best estimator for lambda

ridge_model_cv.best_estimator_
```

Out[46]:

```
Ridge(alpha=5.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [47]:

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

alpha = 10
ridge = Ridge(alpha=alpha)

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

Out[47]:

```
array([ 0.02239287,  0.06836485,  0.04532654,  0.02767768,  0.04533362,
        0.0237079 ,  0.02005459,  0.07625324,  0.01099668,  0.01094735,
        0.02058157,  0.02117717,  0.03591107,  0.00987019, -0.01217035,
       -0.01780087,  0.01417665,  0.01277901,  0.01630917,  0.00776765,
        0.01872641,  0.01669692,  0.01456865, -0.01972157,  0.01492893,
        0.05806234,  0.02207887,  0.08511546,  0.05883699,  0.0261742 ,
       -0.01071093, -0.00847713,  0.02315571, -0.01446403, -0.00886549,
        0.01724907, -0.00936039, -0.01056044,  0.01257372, -0.03418348,
       -0.03011869,  0.01031355, -0.01684323,  0.03000763,  0.01985691,
        0.02038818,  0.04195944,  0.01755802,  0.00602015, -0.00983821])
```

In [48]:

```
# Check the mean squared error

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

Out[48]:

```
0.013592887428853766
```

In [49]:

```
# 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[49]:

	Features	Coefficient
0	LotArea	0.0224
1	OverallQual	0.0684
2	OverallCond	0.0453
3	BsmtFinSF1	0.0277
4	TotalBsmtSF	0.0453

	Features	Coefficient
5	1stFlrSF	0.0237
6	2ndFlrSF	0.0201
7	GrLivArea	0.0763
8	BsmtFullBath	0.0110
9	FullBath	0.0109
10	HalfBath	0.0206
11	Fireplaces	0.0212
12	GarageCars	0.0359
13	WoodDeckSF	0.0099
14	IsRemodelled	-0.0122
15	BuiltOrRemodelAge	-0.0178
16	OldOrNewGarage	0.0142
17	d_BsmtQual	0.0128
18	d_BsmtExposure	0.0163
19	d_BsmtFinType1	0.0078
20	d_HeatingQC	0.0187
21	d_KitchenQual	0.0167
22	d_GarageFinish	0.0146
23	d_BldgType	-0.0197
24	d_SaleCondition	0.0149
25	MSZoning_FV	0.0581
26	MSZoning_RH	0.0221
27	MSZoning_RL	0.0851
28	MSZoning_RM	0.0588
29	Neighborhood_Crawfor	0.0262
30	Neighborhood_Edwards	-0.0107
31	Neighborhood_MeadowV	-0.0085
32	Neighborhood_NridgHt	0.0232
33	Neighborhood_OldTown	-0.0145
34	Neighborhood_SWISU	-0.0089
35	Neighborhood_StoneBr	0.0172
36	Exterior1st_BrkComm	-0.0094
37	Exterior1st_CemntBd	-0.0106
38	Exterior1st_Stucco	0.0126
39	Exterior1st_VinylSd	-0.0342
40	Exterior1st_Wd Sdng	-0.0301
41	Exterior2nd_CmentBd	0.0103
42	Exterior2nd_Stucco	-0.0168
43	Exterior2nd_VinylSd	0.0300
44	Exterior2nd_Wd Sdng	0.0199
45	Foundation_CBlock	0.0204
46	Foundation_PConc	0.0420
47	Foundation_Slab	0.0176
48	Foundation_Stone	0.0060
49	GarageType_CarPort	-0.0098

In [50]:

```
# Assign the Features and their coefficient values to a dictionary which would be used wh
ile plotting the bar plot

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

Out[50]:

```
{'LotArea': 0.0224,
 'OverallQual': 0.0684,
 'OverallCond': 0.0453,
 'BsmtFinSF1': 0.0277,
 'TotalBsmtSF': 0.0453,
 '1stFlrSF': 0.0237,
 '2ndFlrSF': 0.0201,
 'GrLivArea': 0.0763,
 'BsmtFullBath': 0.011,
 'FullBath': 0.0109,
 'HalfBath': 0.0206,
 'Fireplaces': 0.0212,
 'GarageCars': 0.0359,
 'WoodDeckSF': 0.0099,
 'IsRemodelled': -0.0122,
 'BuiltOrRemodelAge': -0.0178,
 'OldOrNewGarage': 0.0142,
 'd_BsmtQual': 0.0128,
 'd_BsmtExposure': 0.0163,
 'd_BsmtFinType1': 0.0078,
 'd_HeatingQC': 0.0187,
 'd_KitchenQual': 0.0167,
 'd_GarageFinish': 0.0146,
 'd_BldgType': -0.0197,
 'd_SaleCondition': 0.0149,
 'MSZoning_FV': 0.0581,
 'MSZoning_RH': 0.0221,
 'MSZoning_RL': 0.0851,
 'MSZoning_RM': 0.0588,
 'Neighborhood_Crawfor': 0.0262,
 'Neighborhood_Edwards': -0.0107,
 'Neighborhood_MeadowV': -0.0085,
 'Neighborhood_NridgHt': 0.0232,
 'Neighborhood_OldTown': -0.0145,
 'Neighborhood_SWISU': -0.0089,
 'Neighborhood_StoneBr': 0.0172,
 'Exterior1st_BrkComm': -0.0094,
 'Exterior1st_CemntBd': -0.0106,
 'Exterior1st_Stucco': 0.0126,
 'Exterior1st_VinylSd': -0.0342,
 'Exterior1st_Wd Sdng': -0.0301,
 'Exterior2nd_CmentBd': 0.0103,
 'Exterior2nd_Stucco': -0.0168,
 'Exterior2nd_VinylSd': 0.03,
 'Exterior2nd_Wd Sdng': 0.0199,
 'Foundation_CBlock': 0.0204,
 'Foundation_PConc': 0.042,
 'Foundation_Slab': 0.0176,
 'Foundation_Stone': 0.006,
 'GarageType_CarPort': -0.0098}
```

RFE

In [52]:

```
# 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)
```

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

```
D:\anaconda\lib\site-packages\sklearn\feature_selection\rfe.py:167: DeprecationWarning: `
np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool
` by itself. Doing this will not modify any behavior and is safe. If you specifically wan
ted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  support_ = np.ones(n_features, dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\feature_selection\rfe.py:168: DeprecationWarning: `
np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` b
y itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, y
ou may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to
review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  ranking_ = np.ones(n_features, dtype=np.int)
```

In [53]:

```python
# 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_)), co
lumns=['Features', 'rfe_support', 'rfe_ranking'])
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
```

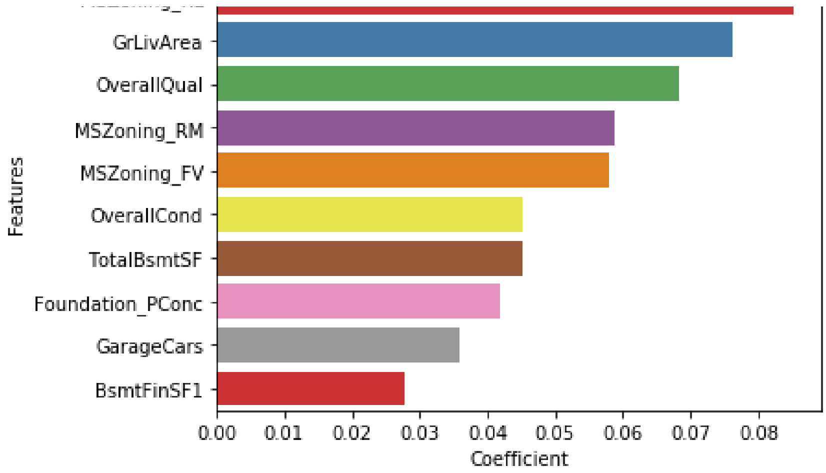
Out[53]:

	Features	rfe_support	rfe_ranking	Coefficient
10	MSZoning_RL	True	1	0.0851
5	GrLivArea	True	1	0.0763
1	OverallQual	True	1	0.0684
11	MSZoning_RM	True	1	0.0588
9	MSZoning_FV	True	1	0.0581
2	OverallCond	True	1	0.0453
4	TotalBsmtSF	True	1	0.0453
14	Foundation_PConc	True	1	0.0420
7	GarageCars	True	1	0.0359
3	BsmtFinSF1	True	1	0.0277

In [54]:

```python
# bar plot to determine the variables that would affect pricing most using ridge regressi
on

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



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

Lasso

In [55]:

```
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,
0.005, 0.01]}

# 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
```

```
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:442: DeprecationWarning:
`np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int`
by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`,
you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish t
o review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
```

```
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\model_selection\_split.py:102: DeprecationWarning:
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `boo
l` by itself. Doing this will not modify any behavior and is safe. If you specifically wa
nted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  55 out of  55 | elapsed:    1.1s finished
D:\anaconda\lib\site-packages\sklearn\model_selection\_search.py:793: DeprecationWarning:
`np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int`
by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`,
you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish t
o review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  dtype=np.int)
```

Out[55]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, positive=False,
                             precompute=False, random_state=None,
                             selection='cyclic', tol=0.0001, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.0002, 0.0003, 0.0004, 0.0005,
                                   0.001, 0.002, 0.003, 0.004, 0.005, 0.01]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_absolute_error', verbose=1)
```

In [56]:

```
# 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'
]].sort_values(by = ['rank_test_score'])
```

Out[56]:

	param_alpha	mean_train_score	mean_test_score	rank_test_score
3	0.0004	-0.077845	-0.084082	1
2	0.0003	-0.077752	-0.084122	2
4	0.0005	-0.077966	-0.084124	3
1	0.0002	-0.077675	-0.084196	4
0	0.0001	-0.077606	-0.084286	5
5	0.001	-0.078658	-0.084649	6
6	0.002	-0.079585	-0.085257	7
7	0.003	-0.080151	-0.085618	8
8	0.004	-0.080639	-0.086056	9
9	0.005	-0.081208	-0.086511	10
10	0.01	-0.085104	-0.089259	11

In [57]:

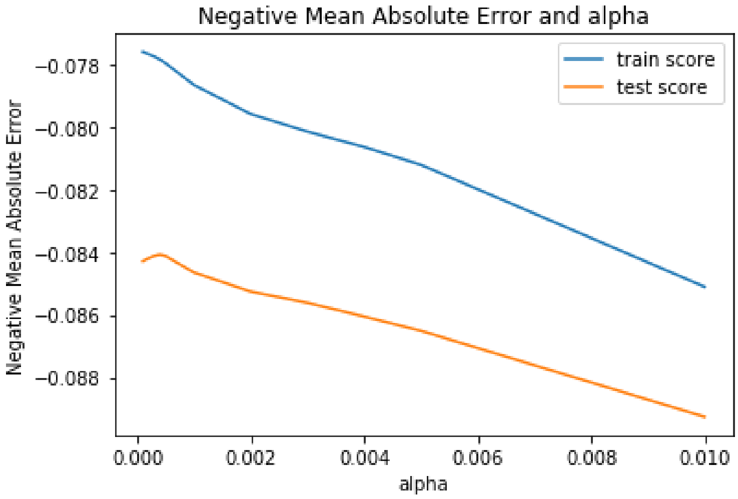
```
# plotting mean test and train scoes with alpha

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 [58]:

```
# get the best estimator for lambda

lasso_model_cv.best_estimator_
```

Out[58]:

```
Lasso(alpha=0.0004, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [59]:

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

alpha = 0.0004

lasso = Lasso(alpha=alpha)

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

Out[59]:

```
array([ 0.02209   ,  0.06965615,  0.04499187,  0.02857128,  0.04617356,
        0.00589288, -0.        ,  0.10018134,  0.01012931,  0.00903681,
        0.01910648,  0.02061525,  0.03680238,  0.00969557, -0.01141099,
       -0.01760434,  0.01381252,  0.01187995,  0.01644408,  0.00693116,
        0.01831145,  0.01614689,  0.01498323, -0.01936956,  0.01447596,
        0.06889636,  0.02711608,  0.10656541,  0.07644758,  0.02578236,
       -0.01048374, -0.0078987 ,  0.02272991, -0.01484452, -0.0096969 ,
        0.01670877, -0.00927381, -0.        ,  0.01212285, -0.03116947,
       -0.02947411,  0.        , -0.01640651,  0.02694071,  0.01853022,
        0.01949665,  0.04186531,  0.01670137,  0.00626478, -0.0089319 ])
```

In [60]:

```
# Check the mean squared error
```

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

Out[60]:

```
0.013468208254564653
```

In [61]:

```
# 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[61]:

	Features	Coefficient
0	LotArea	0.0221
1	OverallQual	0.0697
2	OverallCond	0.0450
3	BsmtFinSF1	0.0286
4	TotalBsmtSF	0.0462
5	1stFlrSF	0.0059
6	GrLivArea	0.1002
7	BsmtFullBath	0.0101
8	FullBath	0.0090
9	HalfBath	0.0191
10	Fireplaces	0.0206
11	GarageCars	0.0368
12	WoodDeckSF	0.0097
13	IsRemodelled	-0.0114
14	BuiltOrRemodelAge	-0.0176
15	OldOrNewGarage	0.0138
16	d_BsmtQual	0.0119
17	d_BsmtExposure	0.0164
18	d_BsmtFinType1	0.0069
19	d_HeatingQC	0.0183
20	d_KitchenQual	0.0161
21	d_GarageFinish	0.0150
22	d_BldgType	-0.0194
23	d_SaleCondition	0.0145
24	MSZoning_FV	0.0689
25	MSZoning_RH	0.0271
26	MSZoning_RL	0.1066
27	MSZoning_RM	0.0764
28	Neighborhood_Crawfor	0.0258
29	Neighborhood_Edwards	-0.0105
30	Neighborhood_MeadowV	-0.0079
31	Neighborhood_NridgHt	0.0227

	Features	Coefficient
32	Neighborhood_OldTown	-0.0148
33	Neighborhood_SWISU	-0.0097
34	Neighborhood_StoneBr	0.0167
35	Exterior1st_BrkComm	-0.0093
36	Exterior1st_Stucco	0.0121
37	Exterior1st_VinylSd	-0.0312
38	Exterior1st_Wd Sdng	-0.0295
39	Exterior2nd_Stucco	-0.0164
40	Exterior2nd_VinylSd	0.0269
41	Exterior2nd_Wd Sdng	0.0185
42	Foundation_CBlock	0.0195
43	Foundation_PConc	0.0419
44	Foundation_Slab	0.0167
45	Foundation_Stone	0.0063
46	GarageType_CarPort	-0.0089

In [62]:

```
# Put the Features and Coefficients in dictionary

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

Out[62]:

```
{'LotArea': 0.022089999224203623,
 'OverallQual': 0.0696561509104974,
 'OverallCond': 0.04499186911162908,
 'BsmtFinSF1': 0.028571275909719095,
 'TotalBsmtSF': 0.046173555214106064,
 '1stFlrSF': 0.00589287821707483,
 '2ndFlrSF': -0.0,
 'GrLivArea': 0.10018133511162267,
 'BsmtFullBath': 0.010129306271065019,
 'FullBath': 0.0090368095005422,
 'HalfBath': 0.019106482229354464,
 'Fireplaces': 0.020615253870428053,
 'GarageCars': 0.036802382758918256,
 'WoodDeckSF': 0.009695570943808554,
 'IsRemodelled': -0.01141099402995129,
 'BuiltOrRemodelAge': -0.017604335236860075,
 'OldOrNewGarage': 0.01381252338587617,
 'd_BsmtQual': 0.01187994739930228,
 'd_BsmtExposure': 0.016444083613428136,
 'd_BsmtFinType1': 0.006931160628862921,
 'd_HeatingQC': 0.01831145337027134,
 'd_KitchenQual': 0.016146894247203846,
 'd_GarageFinish': 0.014983230938384554,
 'd_BldgType': -0.019369558316358487,
 'd_SaleCondition': 0.01447596454312684,
 'MSZoning_FV': 0.0688963581396626,
 'MSZoning_RH': 0.027116084289799544,
 'MSZoning_RL': 0.10656541288581745,
 'MSZoning_RM': 0.07644758077475355,
 'Neighborhood_Crawfor': 0.025782364243311588,
 'Neighborhood_Edwards': -0.010483742506651982,
 'Neighborhood_MeadowV': -0.007898699336534287,
 'Neighborhood_NridgHt': 0.02272990924527991,
 'Neighborhood_OldTown': -0.014844524239235567,
 'Neighborhood_SWISU': -0.009696898984417581,
 'Neighborhood_StoneBr': 0.016708769759 24731,
 'Exterior1st_BrkComm': -0.009273810448459211,
 'Exterior1st_CemntBd': -0.0,
 'Exterior1st_Stucco': 0.01212285338121778,
```

```
 'Exterior1st_VinylSd': -0.031169467789715456,
 'Exterior1st_Wd Sdng': -0.02947411164957756,
 'Exterior2nd_CmentBd': 0.0,
 'Exterior2nd_Stucco': -0.016406507175428953,
 'Exterior2nd_VinylSd': 0.026940711361319803,
 'Exterior2nd_Wd Sdng': 0.01853021852925134,
 'Foundation_CBlock': 0.019496645236393915,
 'Foundation_PConc': 0.041865310807535305,
 'Foundation_Slab': 0.016701367476420377,
 'Foundation_Stone': 0.006264784358337689,
 'GarageType_CarPort': -0.008931900866462653}
```

RFE

In [63]:

```
# 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)
```

```
D:\anaconda\lib\site-packages\sklearn\feature_selection\rfe.py:167: DeprecationWarning: `
np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool
` by itself. Doing this will not modify any behavior and is safe. If you specifically wan
ted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  support_ = np.ones(n_features, dtype=np.bool)
D:\anaconda\lib\site-packages\sklearn\feature_selection\rfe.py:168: DeprecationWarning: `
np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` b
y itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, y
ou may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to
review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
  ranking_ = np.ones(n_features, dtype=np.int)
```

In [64]:

```
# 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_)), co
lumns=['Features', 'rfe_support', 'rfe_ranking'])
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[64]:

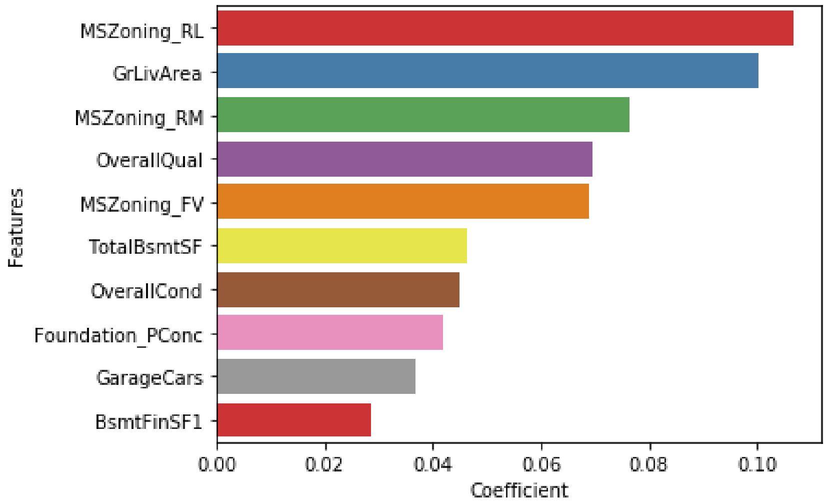
	Features	rfe_support	rfe_ranking	Coefficient
11	MSZoning_RL	True	1	0.106565
5	GrLivArea	True	1	0.100181

	Features	rfe_support	rfe_ranking	Coefficient
12	MSZoning_RM	True	1	0.076448
1	OverallQual	True	1	0.069656
9	MSZoning_FV	True	1	0.068896
4	TotalBsmtSF	True	1	0.046174
2	OverallCond	True	1	0.044992
14	Foundation_PConc	True	1	0.041865
7	GarageCars	True	1	0.036802
3	BsmtFinSF1	True	1	0.028571

In [65]:

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

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



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

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.**

In [ ]: