AdvancedLinearRegression-NatarajaGodina

November 28, 2020

1 Advanced Regression Assignment

1.0.1 Predicting the features affecting the house prices using Advanced Regression

1.1 Data Understanding

[1]:
```
# Import the libraries and load the data in to data frame house_prices
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

house_prices = pd.read_csv('train.csv',
                     encoding='ISO-8859-1')
```

[2]:
```
house_prices.head()
```

[2]:
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN      Reg
1   2          20       RL         80.0     9600   Pave   NaN      Reg
2   3          60       RL         68.0    11250   Pave   NaN      IR1
3   4          70       RL         60.0     9550   Pave   NaN      IR1
4   5          60       RL         84.0    14260   Pave   NaN      IR1

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

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

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

[3]:
```
house_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id                 1460 non-null int64
MSSubClass         1460 non-null int64
MSZoning           1460 non-null object
LotFrontage        1201 non-null float64
LotArea            1460 non-null int64
Street             1460 non-null object
Alley              91 non-null object
LotShape           1460 non-null object
LandContour        1460 non-null object
Utilities          1460 non-null object
LotConfig          1460 non-null object
LandSlope          1460 non-null object
Neighborhood       1460 non-null object
Condition1         1460 non-null object
Condition2         1460 non-null object
BldgType           1460 non-null object
HouseStyle         1460 non-null object
OverallQual        1460 non-null int64
OverallCond        1460 non-null int64
YearBuilt          1460 non-null int64
YearRemodAdd       1460 non-null int64
RoofStyle          1460 non-null object
RoofMatl           1460 non-null object
Exterior1st        1460 non-null object
Exterior2nd        1460 non-null object
MasVnrType         1452 non-null object
MasVnrArea         1452 non-null float64
ExterQual          1460 non-null object
ExterCond          1460 non-null object
Foundation         1460 non-null object
BsmtQual           1423 non-null object
BsmtCond           1423 non-null object
BsmtExposure       1422 non-null object
BsmtFinType1       1423 non-null object
BsmtFinSF1         1460 non-null int64
BsmtFinType2       1422 non-null object
BsmtFinSF2         1460 non-null int64
BsmtUnfSF          1460 non-null int64
TotalBsmtSF        1460 non-null int64
Heating            1460 non-null object
```

```
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

data frame has 1460 rows and 81 columns

[4]:
```
# Check the ratio of nulls in the columns
```

```
print(round(100*(house_prices.isnull().sum()/len(house_prices.index))).
 ↪sort_values())
```

```
Id                0.0
BedroomAbvGr      0.0
HalfBath          0.0
FullBath          0.0
BsmtHalfBath      0.0
                 …
FireplaceQu      47.0
Fence            81.0
Alley            94.0
MiscFeature      96.0
PoolQC          100.0
Length: 81, dtype: float64
```

[5]:
```
# drop the columns with 80% missing values, as these do not add any value or␣
 ↪information
house_prices = house_prices.drop('Fence', axis=1)
house_prices = house_prices.drop('Alley', axis=1)
house_prices = house_prices.drop('MiscFeature', axis=1)
house_prices = house_prices.drop('PoolQC', axis=1)
```

[6]:
```
print(round(100*(house_prices.isnull().sum()/len(house_prices.index))).
 ↪sort_values())
```

```
Id               0.0
BedroomAbvGr     0.0
HalfBath         0.0
FullBath         0.0
BsmtHalfBath     0.0
                 …
GarageCond       6.0
GarageType       6.0
GarageFinish     6.0
LotFrontage     18.0
FireplaceQu     47.0
Length: 77, dtype: float64
```

[7]:
```
#Let us drop the column FireplaceQu as it has 47% of the rows numm.
house_prices = house_prices.drop('FireplaceQu', axis=1)
```

[8]:
```
print(round(100*(house_prices.isnull().sum()/len(house_prices.index))).
 ↪sort_values())
```

```
Id               0.0
BedroomAbvGr     0.0
HalfBath         0.0
```

```
FullBath         0.0
BsmtHalfBath     0.0
                 …
GarageCond       6.0
GarageFinish     6.0
GarageYrBlt      6.0
GarageType       6.0
LotFrontage     18.0
Length: 76, dtype: float64
```

[9]:
```
# Now let us delete the records with null values.
house_prices = house_prices.dropna(axis=0, how='any')
```

[10]:
```
print(round(100*(house_prices.isnull().sum()/len(house_prices.index))).
 ↪sort_values())
```

```
Id              0.0
Functional      0.0
TotRmsAbvGrd    0.0
KitchenQual     0.0
KitchenAbvGr    0.0
               …
Exterior1st     0.0
RoofMatl        0.0
RoofStyle       0.0
ExterCond       0.0
SalePrice       0.0
Length: 76, dtype: float64
```

Now we do not have missing values for any columns

[11]:
```
house_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1094 entries, 0 to 1459
Data columns (total 76 columns):
Id                1094 non-null int64
MSSubClass        1094 non-null int64
MSZoning          1094 non-null object
LotFrontage       1094 non-null float64
LotArea           1094 non-null int64
Street            1094 non-null object
LotShape          1094 non-null object
LandContour       1094 non-null object
Utilities         1094 non-null object
LotConfig         1094 non-null object
LandSlope         1094 non-null object
Neighborhood      1094 non-null object
```

```
Condition1        1094 non-null object
Condition2        1094 non-null object
BldgType          1094 non-null object
HouseStyle        1094 non-null object
OverallQual       1094 non-null int64
OverallCond       1094 non-null int64
YearBuilt         1094 non-null int64
YearRemodAdd      1094 non-null int64
RoofStyle         1094 non-null object
RoofMatl          1094 non-null object
Exterior1st       1094 non-null object
Exterior2nd       1094 non-null object
MasVnrType        1094 non-null object
MasVnrArea        1094 non-null float64
ExterQual         1094 non-null object
ExterCond         1094 non-null object
Foundation        1094 non-null object
BsmtQual          1094 non-null object
BsmtCond          1094 non-null object
BsmtExposure      1094 non-null object
BsmtFinType1      1094 non-null object
BsmtFinSF1        1094 non-null int64
BsmtFinType2      1094 non-null object
BsmtFinSF2        1094 non-null int64
BsmtUnfSF         1094 non-null int64
TotalBsmtSF       1094 non-null int64
Heating           1094 non-null object
HeatingQC         1094 non-null object
CentralAir        1094 non-null object
Electrical        1094 non-null object
1stFlrSF          1094 non-null int64
2ndFlrSF          1094 non-null int64
LowQualFinSF      1094 non-null int64
GrLivArea         1094 non-null int64
BsmtFullBath      1094 non-null int64
BsmtHalfBath      1094 non-null int64
FullBath          1094 non-null int64
HalfBath          1094 non-null int64
BedroomAbvGr      1094 non-null int64
KitchenAbvGr      1094 non-null int64
KitchenQual       1094 non-null object
TotRmsAbvGrd      1094 non-null int64
Functional        1094 non-null object
Fireplaces        1094 non-null int64
GarageType        1094 non-null object
GarageYrBlt       1094 non-null float64
GarageFinish      1094 non-null object
GarageCars        1094 non-null int64
```

```
GarageArea       1094 non-null int64
GarageQual       1094 non-null object
GarageCond       1094 non-null object
PavedDrive       1094 non-null object
WoodDeckSF       1094 non-null int64
OpenPorchSF      1094 non-null int64
EnclosedPorch    1094 non-null int64
3SsnPorch        1094 non-null int64
ScreenPorch      1094 non-null int64
PoolArea         1094 non-null int64
MiscVal          1094 non-null int64
MoSold           1094 non-null int64
YrSold           1094 non-null int64
SaleType         1094 non-null object
SaleCondition    1094 non-null object
SalePrice        1094 non-null int64
dtypes: float64(3), int64(35), object(38)
memory usage: 658.1+ KB
```

After data cleaning we have 1094 records with 76 columns

[12]:
```
house_prices.columns[house_prices.nunique() <= 1]
```

[12]:
```
Index(['Utilities'], dtype='object')
```

[13]:
```
house_prices.groupby('Utilities').count()
```

[13]:
```
             Id  MSSubClass  MSZoning  LotFrontage  LotArea  Street  LotShape  \
Utilities
AllPub     1094        1094      1094         1094     1094    1094      1094

           LandContour  LotConfig  LandSlope  …  EnclosedPorch  3SsnPorch  \
Utilities                                     …
AllPub            1094       1094       1094  …           1094       1094

           ScreenPorch  PoolArea  MiscVal  MoSold  YrSold  SaleType  \
Utilities
AllPub            1094      1094     1094    1094    1094      1094

           SaleCondition  SalePrice
Utilities
AllPub              1094       1094

[1 rows x 75 columns]
```

[14]:
```
# As we have only one value for column Utilities, Let us drop the column
house_prices = house_prices.drop('Utilities', axis=1)
```

[15]:
```python
house_prices.describe()
```

[15]:
```
                Id    MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1094.000000  1094.000000  1094.000000    1094.000000  1094.000000
mean    727.375686    56.128885    70.759598   10132.346435     6.247715
std     420.955488    41.976345    24.508859    8212.249621     1.366797
min       1.000000    20.000000    21.000000    1300.000000     2.000000
25%     366.500000    20.000000    60.000000    7606.750000     5.000000
50%     723.500000    50.000000    70.000000    9444.500000     6.000000
75%    1093.750000    70.000000    80.000000   11387.250000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  …  \
count  1094.000000  1094.000000   1094.000000  1094.000000  1094.000000  …
mean      5.575868  1972.412249   1985.915905   109.855576   448.191956  …
std       1.066500    31.189752     20.930772   190.667459   468.728095  …
min       2.000000  1880.000000   1950.000000     0.000000     0.000000  …
25%       5.000000  1953.000000   1967.000000     0.000000     0.000000  …
50%       5.000000  1975.000000   1995.000000     0.000000   384.500000  …
75%       6.000000  2003.000000   2005.000000   171.750000   712.750000  …
max       9.000000  2010.000000   2010.000000  1600.000000  5644.000000  …

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  ScreenPorch  \
count  1094.000000  1094.000000    1094.000000  1094.000000  1094.000000
mean     94.341865    46.946984      22.053016     3.266910    16.498172
std     122.624615    64.820019      61.570502    29.655973    58.455303
min       0.000000     0.000000       0.000000     0.000000     0.000000
25%       0.000000     0.000000       0.000000     0.000000     0.000000
50%       0.000000    28.000000       0.000000     0.000000     0.000000
75%     169.750000    68.000000       0.000000     0.000000     0.000000
max     857.000000   547.000000     552.000000   508.000000   480.000000

          PoolArea      MiscVal       MoSold       YrSold      SalePrice
count  1094.000000  1094.000000  1094.000000  1094.000000    1094.000000
mean      3.007313    23.550274     6.335466  2007.786106  187033.263254
std      40.713175   167.135237     2.694558     1.334307   83165.332151
min       0.000000     0.000000     1.000000  2006.000000   35311.000000
25%       0.000000     0.000000     5.000000  2007.000000  132500.000000
50%       0.000000     0.000000     6.000000  2008.000000  165750.000000
75%       0.000000     0.000000     8.000000  2009.000000  221000.000000
max     648.000000  2500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]
```

1.2 Data Preparation

[16]:
```
house_prices.columns
```

[16]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Neighborhood',
       'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual',
       'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
       'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SaleType', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

[17]:
```
house_prices.columns = ['Id', 'MSSubClass', 'MSZoning', 'LotFrontage',␣
↪'LotArea', 'Street',
       'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Neighborhood',
       'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual',
       'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
       'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAC', 'Electrical',
       '1stFlourSF', '2ndFlourSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotalRoomsAboveGrade', 'Functional', 'Fireplaces',␣
↪'GarageType',
       'GarageYearBlt', 'GarageFinish', 'GarageCars', 'GarageArea',␣
↪'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SeasonPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MonthSold', 'YearSold', 'SaleType', 'SaleCondition', 'SalePrice']
```

Modify some of the columns to be easily readable and understandable

[18]:
```
house_prices.columns
```

[18]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Neighborhood',
       'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
```

```
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual',
       'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
       'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAC', 'Electrical',
       '1stFlourSF', '2ndFlourSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotalRoomsAboveGrade', 'Functional', 'Fireplaces',
       'GarageType', 'GarageYearBlt', 'GarageFinish', 'GarageCars',
       'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SeasonPorch', 'ScreenPorch',
       'PoolArea', 'MiscVal', 'MonthSold', 'YearSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Add derived columns

1. TotalSF = 1stFlourSF + 2ndFlourSF

[19]:
```
house_prices['TotalSF'] = house_prices['1stFlourSF'] +␣
↪house_prices['2ndFlourSF']
```

[20]:
```
house_prices['TotalSF'].describe()
```

[20]:
```
count    1094.000000
mean     1530.346435
std       522.649431
min       438.000000
25%      1150.500000
50%      1479.000000
75%      1775.750000
max      5642.000000
Name: TotalSF, dtype: float64
```

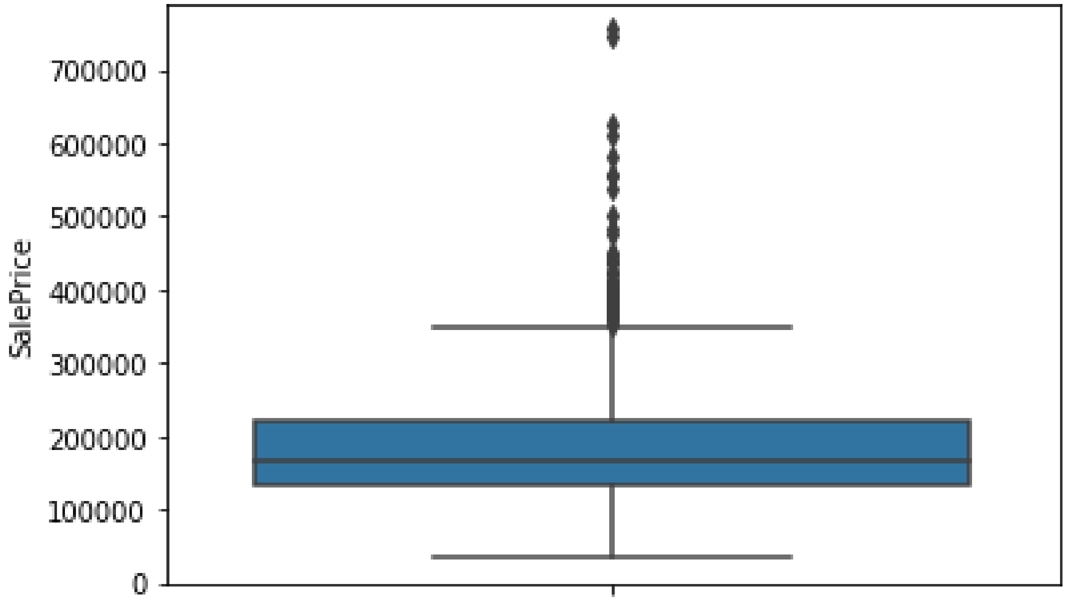
1.3 EDA

1.3.1 Univariate Analysis

[21]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

[22]:
```
sns.boxplot(y=house_prices['SalePrice'])
```

[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x123436d90>
```



[23]:
```
house_prices['SalePrice'].describe()
```

[23]:
```
count      1094.000000
mean     187033.263254
std       83165.332151
min       35311.000000
25%      132500.000000
50%      165750.000000
75%      221000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

[24]:
```
house_prices[(house_prices['SalePrice'] > 350000)]['SalePrice']
```

[24]:
```
53      385000
58      438780
112     383970
151     372402
161     412500
178     501837
185     475000
224     386250
231     403000
278     415298
309     360000
313     375000
```

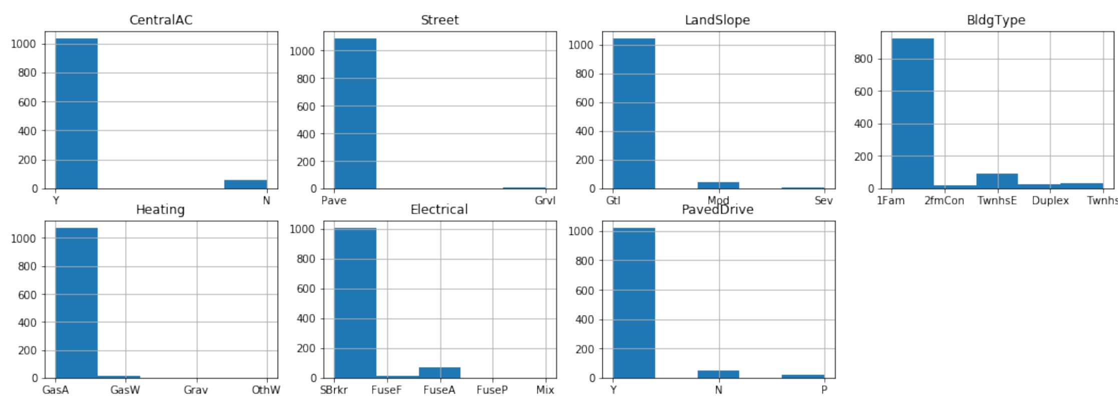
```
321     354000
336     377426
349     437154
378     394432
389     426000
440     555000
473     440000
477     380000
481     374000
515     402861
527     446261
585     369900
591     451950
608     359100
644     370878
661     402000
664     423000
678     372500
688     392000
691     755000
702     361919
769     538000
774     395000
798     485000
803     582933
825     385000
898     611657
987     395192
1046    556581
1142    424870
1169    625000
1181    392500
1182    745000
1228    367294
1267    378500
1353    410000
1388    377500
1437    394617
Name: SalePrice, dtype: int64
```

We have about 50 records with sale price above 350000 as outliers.

[25]:
```
fig = plt.figure(figsize=(18, 6))
plt.subplot(2,4,1)
plt.title('CentralAC')
house_prices['CentralAC'].hist(bins=5)
plt.subplot(2,4,2)
```

```
plt.title('Street')
house_prices['Street'].hist(bins=5)
plt.subplot(2,4,3)
plt.title('LandSlope')
house_prices['LandSlope'].hist(bins=5)
plt.subplot(2,4,4)
plt.title('BldgType')
house_prices['BldgType'].hist(bins=5)
plt.subplot(2,4,5)
plt.title('Heating')
house_prices['Heating'].hist(bins=5)
plt.subplot(2,4,6)
plt.title('Electrical')
house_prices['Electrical'].hist(bins=5)
plt.subplot(2,4,7)
plt.title('PavedDrive')
house_prices['PavedDrive'].hist(bins=5)
```

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1a26203350>



We can see that majority of the houses are with below items in respetive categories:

1. Central Air Conditioning - Yes
2. Type of Road access to property - Paved
3. Land Slope - Gentle
4. Building Type - 1Fam (Single Family - Detached)
5. Heating Type - GasA (Gas forced warm air furnace)
6. Electrical System - Standard Circuit Breakers & Romex
7. Paved Drive - Yes

1.3.2 Bivariate Analysis

- As our target is to find the variables which are significant in predicting the price of a house.
- We will perform Bivariate analysis of all other variables against Price.
- As we cannot see clearly anything in the pair plot of entire data set, Let us do in multiple batches

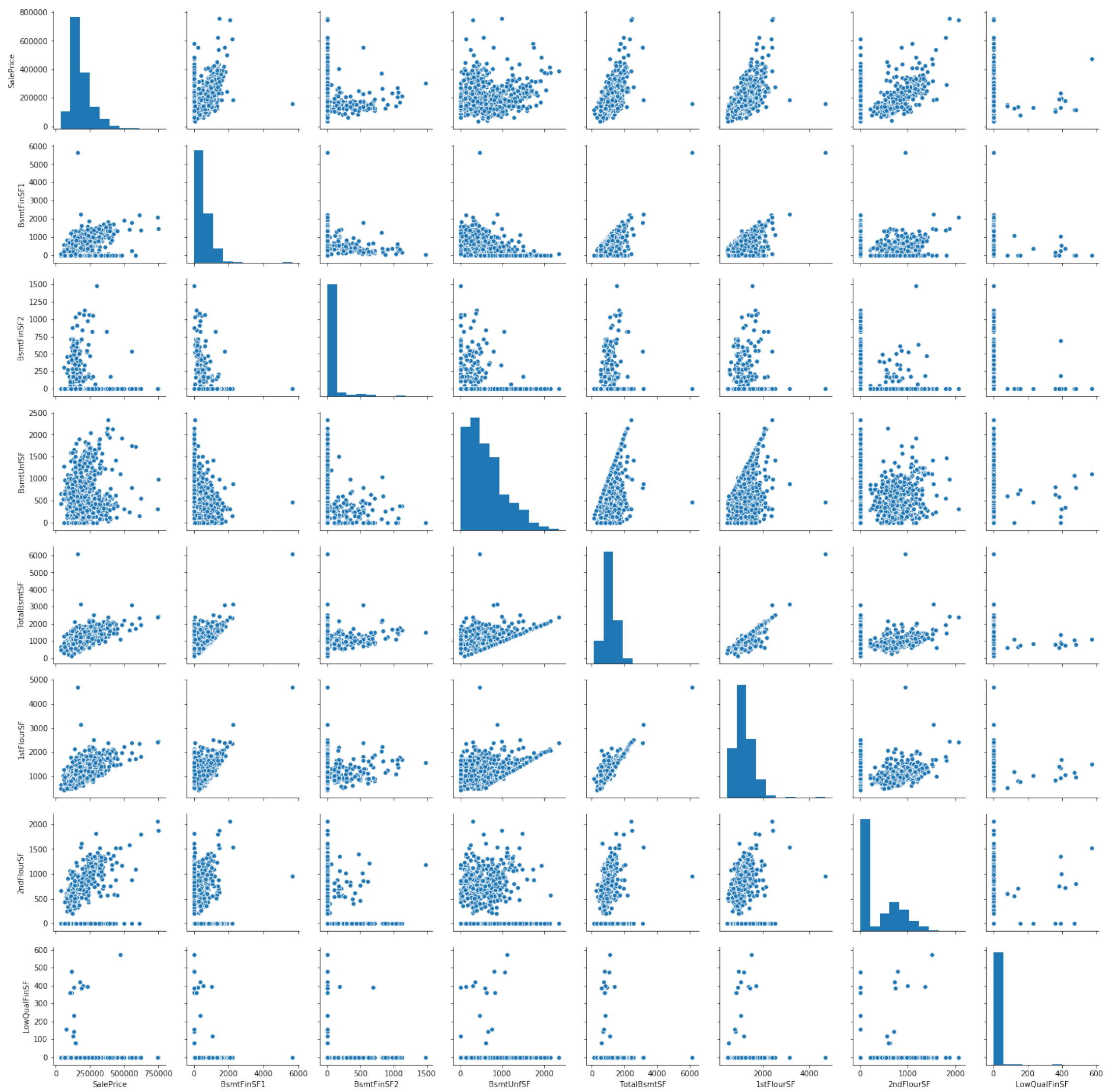
[26]:
```
##### Pairplot 1
pp1 = house_prices[['SalePrice','MSSubClass',  'LotFrontage', 'LotArea',␣
↪'OverallQual',
                     'OverallCond', 'MasVnrArea']]

sns.pairplot(pp1)
```

[26]: <seaborn.axisgrid.PairGrid at 0x1a26346a10>

From the above pair plot we can see that SalePrice positively correlated with 'Lot-Frontage', 'LotArea', 'OverallQual','OverallCond', 'MasVnrArea'

[27]:
```
##### Pairplot 2
pp2 = house_prices[['SalePrice', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
        'TotalBsmtSF','1stFlourSF', '2ndFlourSF', 'LowQualFinSF']]

sns.pairplot(pp2)
```

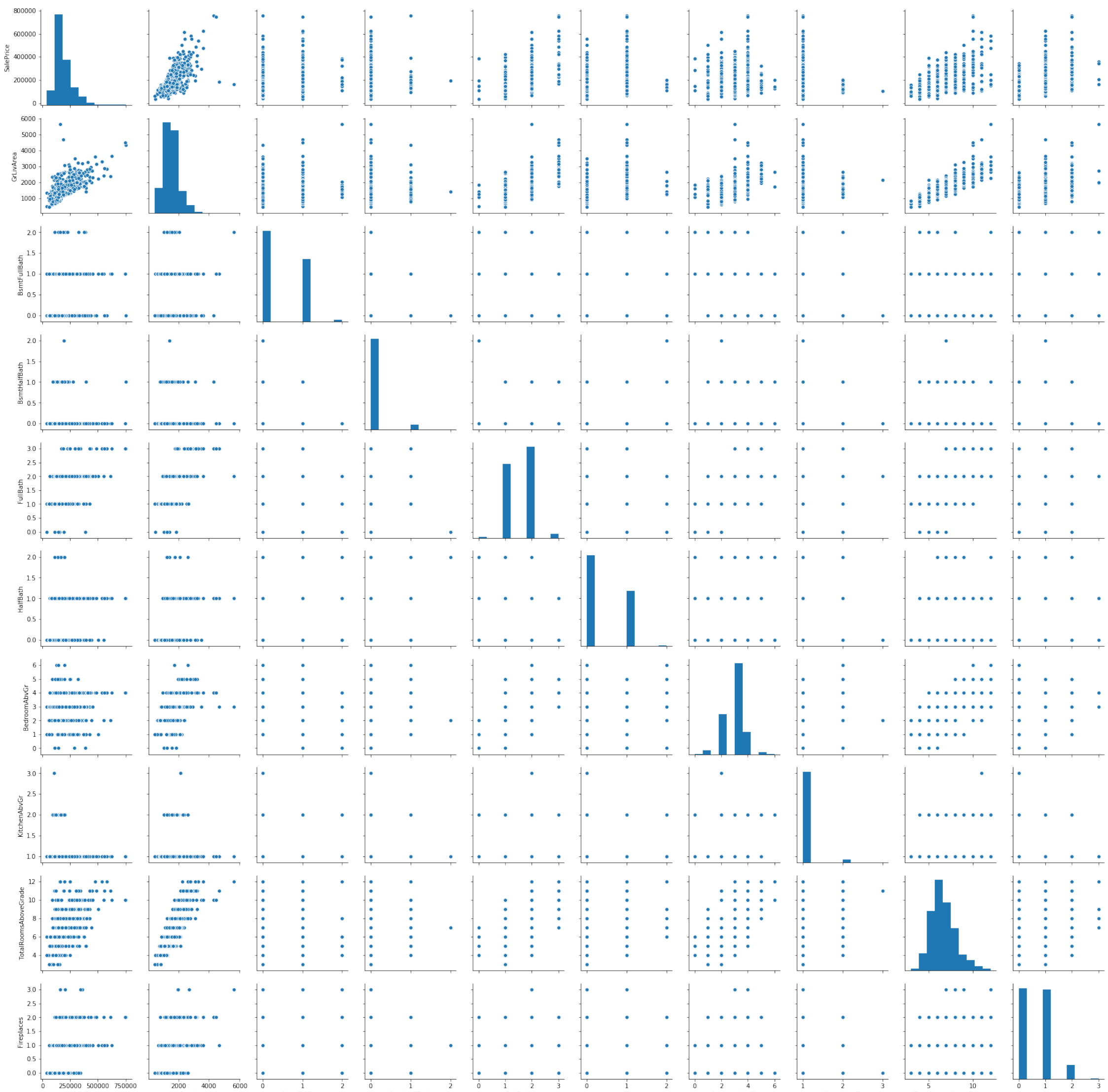
[27]:
```
<seaborn.axisgrid.PairGrid at 0x1a27ce7290>
```



From the above pair plot we can see that SalePrice positively correlated with 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF','1stFlourSF', '2ndFlourSF'

[28]:
```python
##### Pairplot 3
pp3 = house_prices[['SalePrice', 'GrLivArea' , 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
               'BedroomAbvGr', 'KitchenAbvGr', 'TotalRoomsAboveGrade', 'Fireplaces']]

sns.pairplot(pp3)
```

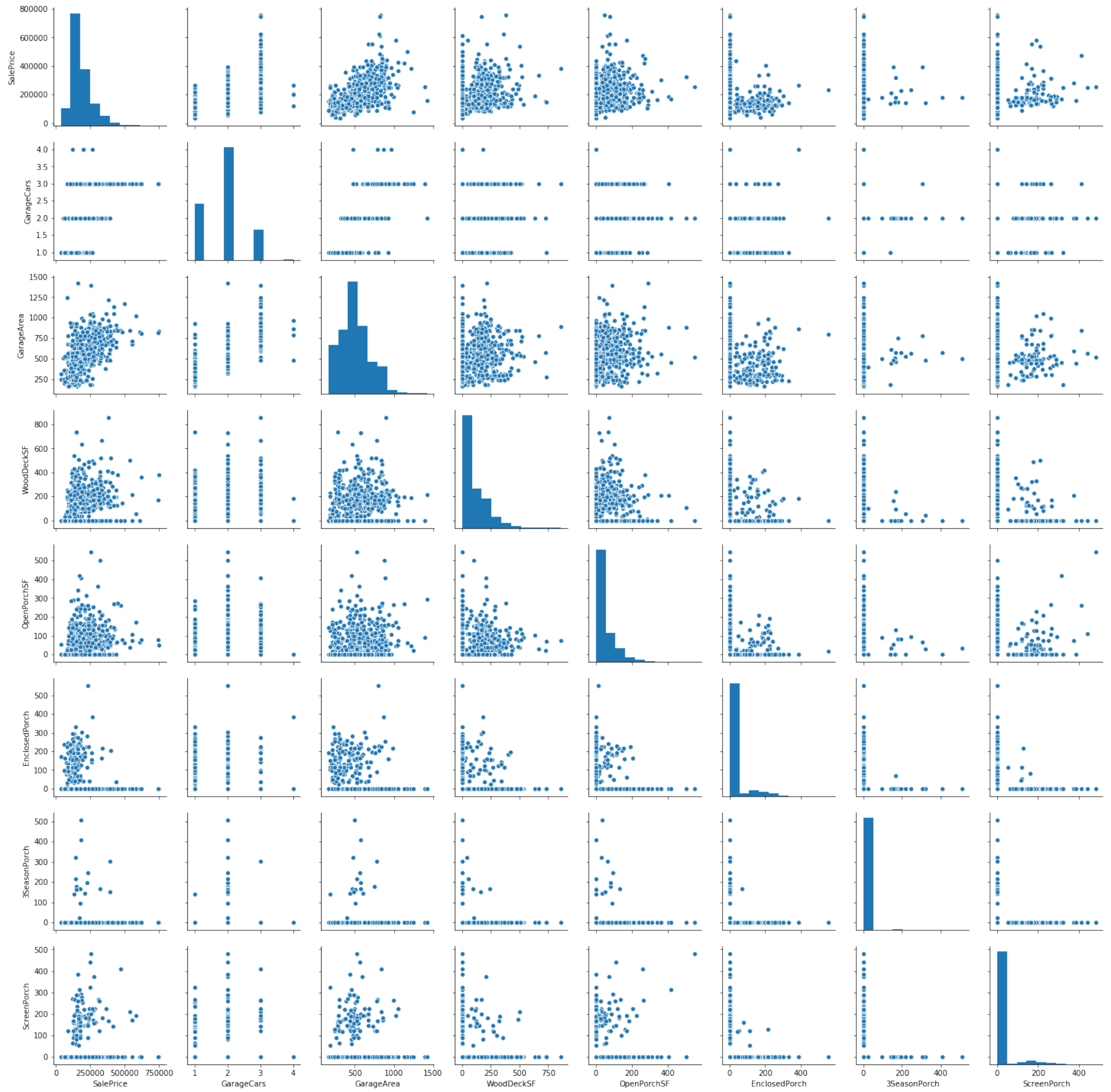
[28]:
```
<seaborn.axisgrid.PairGrid at 0x1a2a715490>
```



From the above pair plot we can see that SalePrice positively correlated with 'GrLivArea', 'TotalRoomsAboveGrade'

[29]:
```
##### Pairplot 4
pp4 = house_prices[['SalePrice', 'GarageCars', 'GarageArea', 'WoodDeckSF',
        'OpenPorchSF', 'EnclosedPorch', '3SeasonPorch', 'ScreenPorch']]

sns.pairplot(pp4)
```

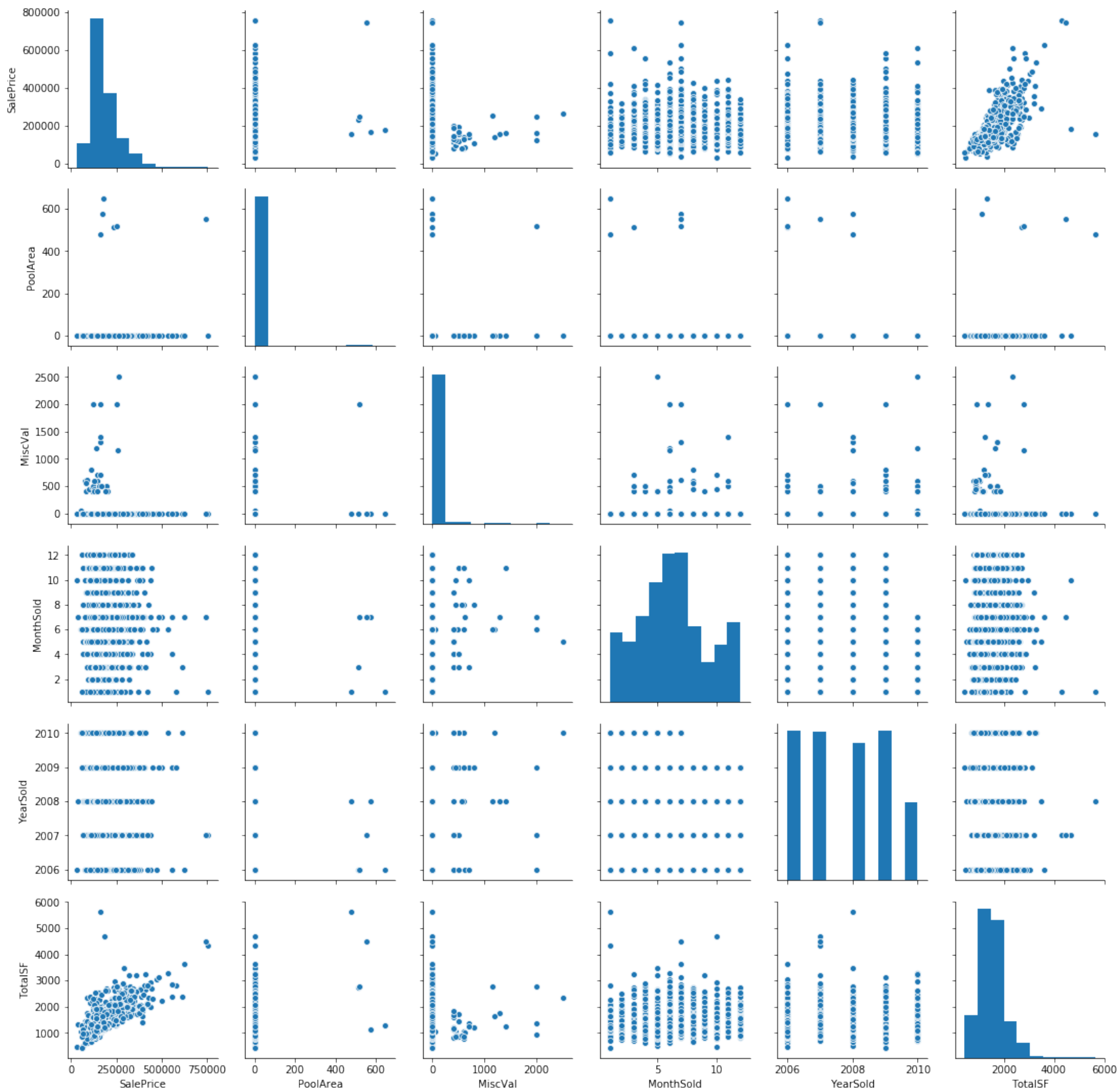
[29]: <seaborn.axisgrid.PairGrid at 0x1a2e08c690>



From the above pair plot we can see that SalePrice mostly positively correlated with 'GarageArea', 'WoodDeckSF', 'OpenPorchSF'

[30]:
```
##### Pairplot 5
pp5 = house_prices[['SalePrice', 'PoolArea', 'MiscVal', 'MonthSold',␣
↪'YearSold', 'SaleType',
        'SaleCondition', 'TotalSF']]

sns.pairplot(pp5)
```

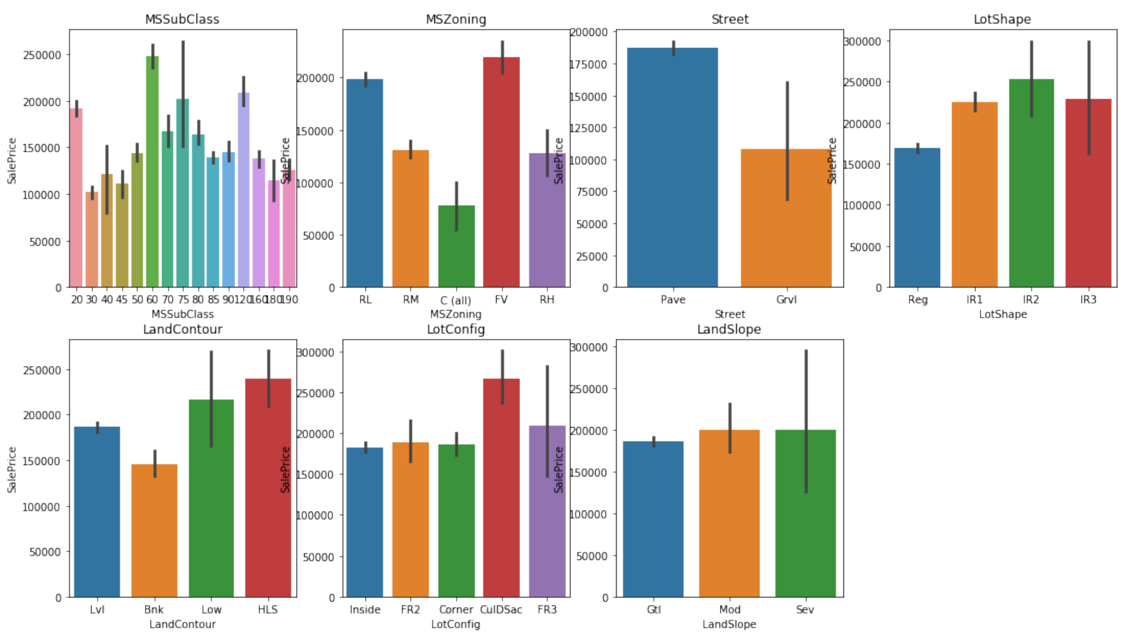
[30]: <seaborn.axisgrid.PairGrid at 0x123d9f5d0>



From the above pair plot we can see that SalePrice mostly positively correlated with¶ ‘TotalSF’

[31]:
```
fig = plt.figure(figsize=(18, 10))
plt.subplot(2,4,1)
plt.title('MSSubClass')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['MSSubClass'])
plt.subplot(2,4,2)
plt.title('MSZoning')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['MSZoning'])
plt.subplot(2,4,3)
plt.title('Street')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['Street'])
plt.subplot(2,4,4)
plt.title('LotShape')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['LotShape'])
plt.subplot(2,4,5)
plt.title('LandContour')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['LandContour'])
plt.subplot(2,4,6)
plt.title('LotConfig')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['LotConfig'])
plt.subplot(2,4,7)
plt.title('LandSlope')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['LandSlope'])
```

[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1a3338bed0>

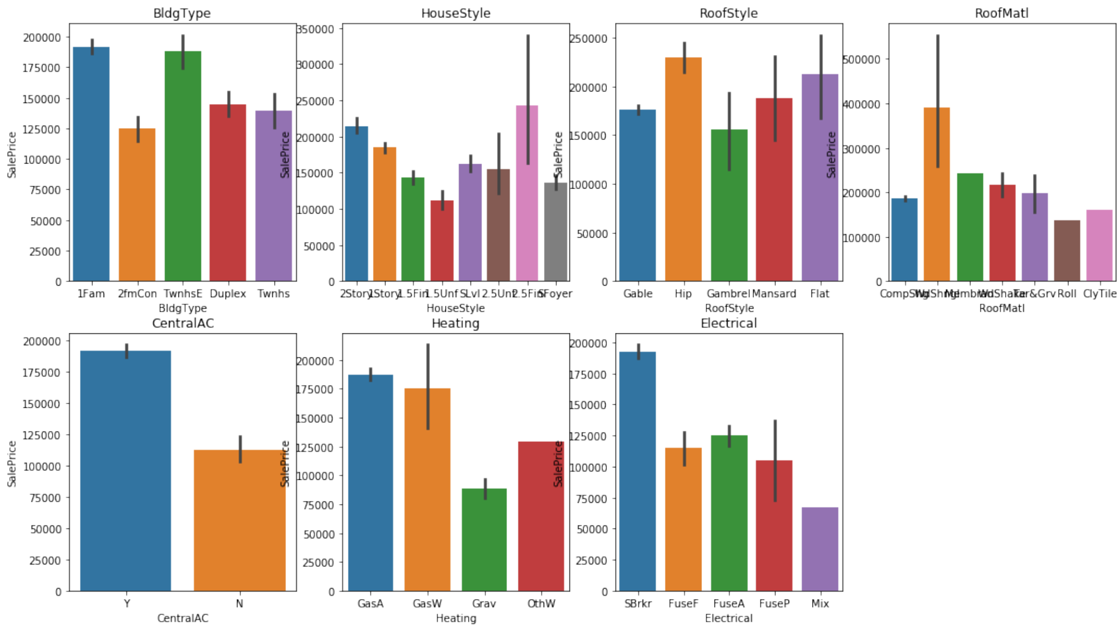


From the above plots we can see that average SalePrice is higher for:

1. MSSubClass of 60 (2-STORY 1946 & NEWER)
2. MSZoning of RL and FV (Residential Low Density and Floating Village Residential)
3. Street of Paved (Paved type of road access to property)
4. LandContour of HLS (Hillside)
5. LotConfig of Cul-de-sac

[32]:
```
fig = plt.figure(figsize=(18, 10))
plt.subplot(2,4,1)
plt.title('BldgType')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['BldgType'])
plt.subplot(2,4,2)
plt.title('HouseStyle')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['HouseStyle'])
plt.subplot(2,4,3)
plt.title('RoofStyle')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['RoofStyle'])
plt.subplot(2,4,4)
plt.title('RoofMatl')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['RoofMatl'])
plt.subplot(2,4,5)
plt.title('CentralAC')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['CentralAC'])
plt.subplot(2,4,6)
plt.title('Heating')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['Heating'])
plt.subplot(2,4,7)
plt.title('Electrical')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['Electrical'])
```

[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a33a5b950>
```



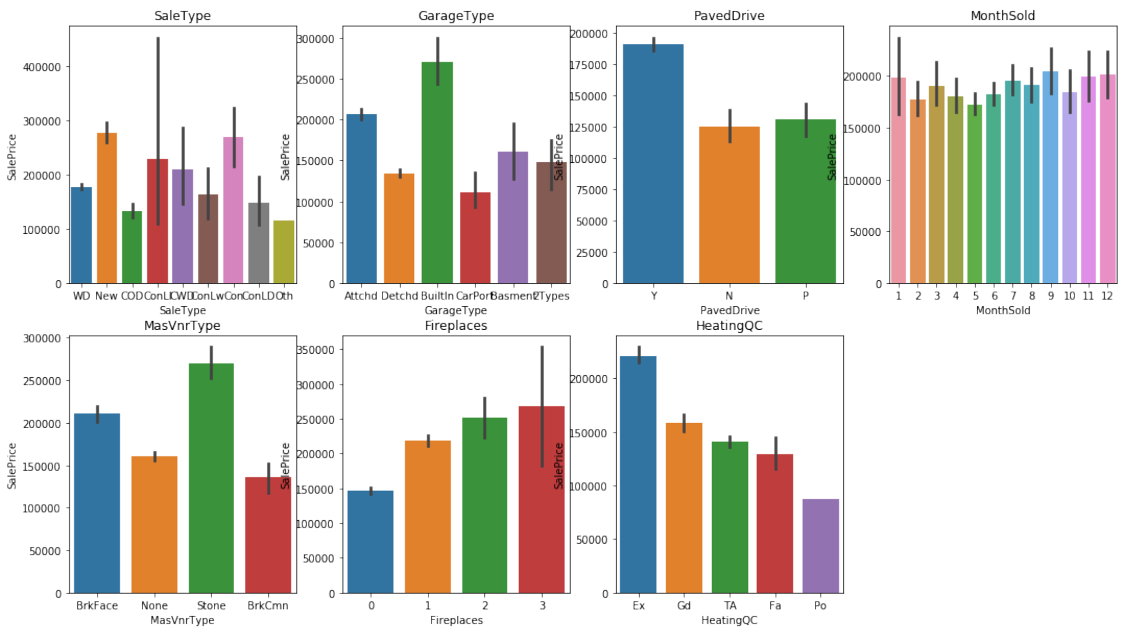
From the above plots we can see that average SalePrice is higher for:

1. BldgType of 1Fam and TwnhsE (Single-family Detached and Townhouse End Unit)
2. Housetyle of 2Story and 2.5Fin (Two story, and Two and one-half story: 2nd level finished)
3. RoofStyle of Hip and Flat
4. RoofMatl of Standard (Composite) Shingle
5. Centralized Air Conditioning
6. Heating of GasA and GasW (Gas forced warm air furnace and Gas hot water or steam heat)
7. Electrical system of Standard Circuit Breakers & Romex

[33]:
```
fig = plt.figure(figsize=(18, 10))
plt.subplot(2,4,1)
plt.title('SaleType')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['SaleType'])
plt.subplot(2,4,2)
plt.title('GarageType')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['GarageType'])
plt.subplot(2,4,3)
plt.title('PavedDrive')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['PavedDrive'])
plt.subplot(2,4,4)
plt.title('MonthSold')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['MonthSold'])
plt.subplot(2,4,5)
plt.title('MasVnrType')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['MasVnrType'])
plt.subplot(2,4,6)
```

```
plt.title('Fireplaces')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['Fireplaces'])
plt.subplot(2,4,7)
plt.title('HeatingQC')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['HeatingQC'])
```

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1a340927d0>



From the above plots we can see that average SalePrice is higher for:

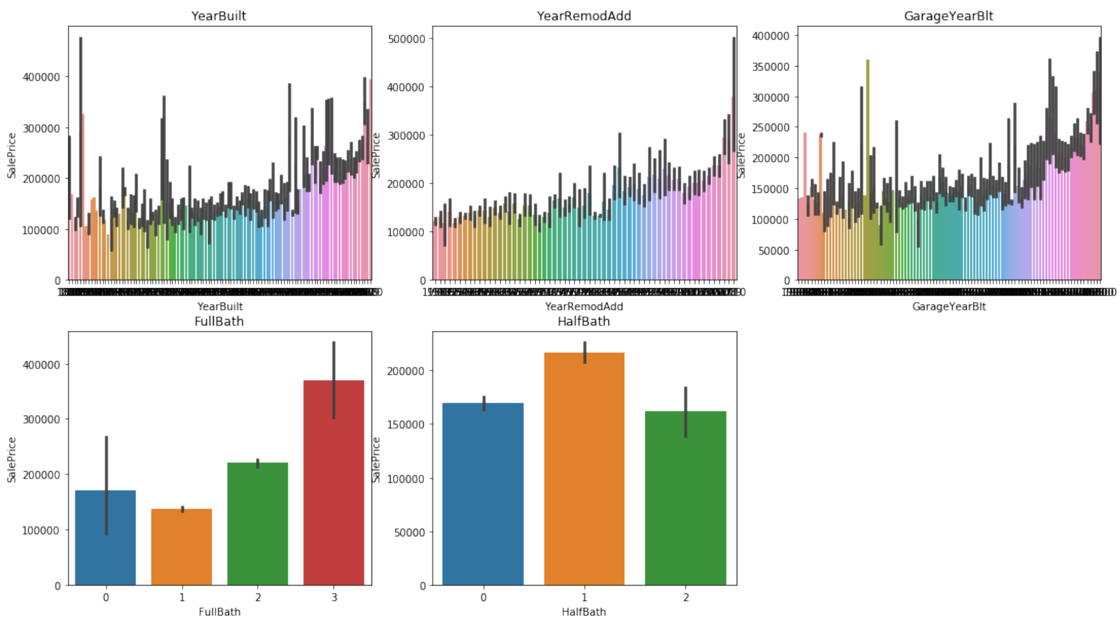
1. GarageType of Builtin
2. Paved Drive - Yes
3. Masonry Veneer Type - Stone
4. HeatingQuality - Excellent

— We can also see that average SalePrice does not change by the month of sale. — Saleprice increases with the number of fireplaces

[34]:
```
fig = plt.figure(figsize=(18, 10))
plt.subplot(2,3,1)
plt.title('YearBuilt')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['YearBuilt'])
plt.subplot(2,3,2)
plt.title('YearRemodAdd')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['YearRemodAdd'])
plt.subplot(2,3,3)
plt.title('GarageYearBlt')
```

```
sns.barplot(y=house_prices['SalePrice'], x=house_prices['GarageYearBlt'])
plt.subplot(2,3,4)
plt.title('FullBath')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['FullBath'])
plt.subplot(2,3,5)
plt.title('HalfBath')
sns.barplot(y=house_prices['SalePrice'], x=house_prices['HalfBath'])
```

[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a34d6aad0>
```



From the above plots we can see that average SalePrice is higher for: — SalePrice increases for newly built houses, newly remodeled/refurbished houses and newly built garages. Older the house, lower the price. — Saleprice increases with the number of FullBath's

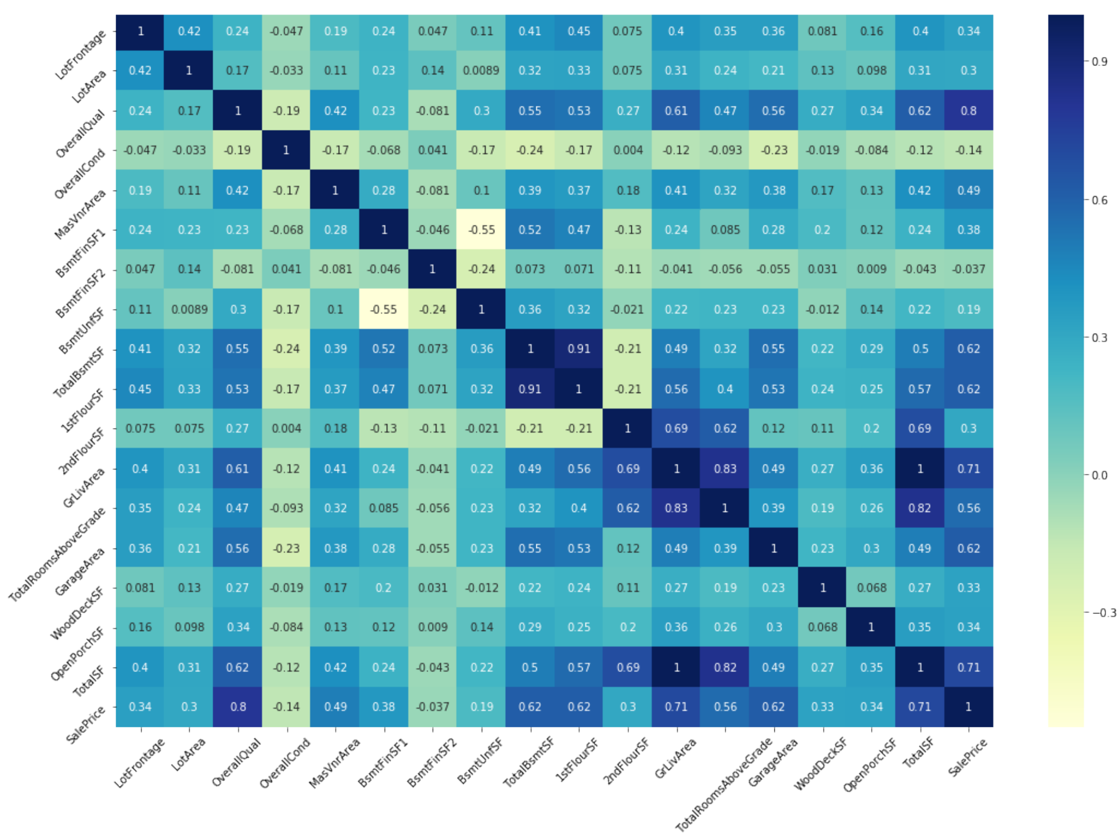
[35]:
```
house_prices.columns
```

[35]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
        'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Neighborhood',
        'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
        'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl',
        'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual',
        'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
        'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF',
        'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAC', 'Electrical',
        '1stFlourSF', '2ndFlourSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
        'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
```

```
       'KitchenQual', 'TotalRoomsAboveGrade', 'Functional', 'Fireplaces',
       'GarageType', 'GarageYearBlt', 'GarageFinish', 'GarageCars',
       'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SeasonPorch', 'ScreenPorch',
       'PoolArea', 'MiscVal', 'MonthSold', 'YearSold', 'SaleType',
       'SaleCondition', 'SalePrice', 'TotalSF'],
      dtype='object')
```

[36]:
```
df_corr = house_prices[['LotFrontage', 'LotArea', 'OverallQual','OverallCond',␣
 ↪'MasVnrArea',
                        'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',␣
 ↪'TotalBsmtSF','1stFlourSF', '2ndFlourSF',
                        'GrLivArea', 'TotalRoomsAboveGrade','GarageArea',␣
 ↪'WoodDeckSF', 'OpenPorchSF',
                        'TotalSF', 'SalePrice']]
plt.figure(figsize=(18,12))
ax = plt.subplot(1,1,1)
sns.heatmap(df_corr.corr(), annot=True, cmap='YlGnBu')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
ax.set_ylim(18, 0.1)
```

[36]:
```
(18, 0.1)
```



From the heat map, we can see SalePrice is highly positively correlated with below variables in that order:

1. Overall Quality
2. Total SFT, GrLivArea (Living Area SFT)
3. Garage Area, First Floor SFT, Total Basement SFT
4. Total Rooms above grade.

Let us add dummy variables and map 1's and 0's for variables with two values.

[37]:
```
house_prices['CentralAC'].value_counts()
```

[37]:
```
Y    1036
N      58
Name: CentralAC, dtype: int64
```

[38]:
```
house_prices['Street'].value_counts()
```

[38]:
```
Pave    1090
Grvl       4
Name: Street, dtype: int64
```

[39]:
```
# List of variables to map - CentralAC, Street
house_prices['CentralAC'] = house_prices['CentralAC'].map({'Y': 1, 'N': 0})
house_prices['Street'] = house_prices['Street'].map({'Pave': 1, 'Grvl': 0})
```

Dummy variables

[40]:
```
# List of category variables with more than one level
varlist= ['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope',␣
 ↪'BldgType', 'HouseStyle',
          'RoofStyle', 'RoofMatl', 'Heating', 'Electrical', 'SaleType',␣
 ↪'GarageType', 'PavedDrive',
          'MasVnrType', 'HeatingQC', 'Neighborhood', 'Condition1','Condition2',␣
 ↪'Exterior1st','Exterior2nd',
         ␣
 ↪'ExterQual','ExterCond','Foundation','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1',
          'BsmtFinType2','KitchenQual','Functional','GarageFinish',␣
 ↪'GarageQual','GarageCond',
          'SaleCondition']

for i in varlist:
    # Add dummy variables
    status = pd.get_dummies(house_prices[i], prefix = i, drop_first = True)
    # Add the results to the original housing dataframe
```

```
    house_prices = pd.concat([house_prices, status], axis = 1)
    # Drop the original column
    house_prices.drop([i], axis = 1, inplace = True)
```

[41]:
```
house_prices.head()
```

[41]:
```
   Id  MSSubClass  LotFrontage  LotArea  Street  OverallQual  OverallCond  \
0   1          60         65.0     8450       1            7            5
1   2          20         80.0     9600       1            6            8
2   3          60         68.0    11250       1            7            5
3   4          70         60.0     9550       1            7            5
4   5          60         84.0    14260       1            8            5

   YearBuilt  YearRemodAdd  MasVnrArea  …  GarageQual_TA  GarageCond_Fa  \
0       2003          2003       196.0  …              1              0
1       1976          1976         0.0  …              1              0
2       2001          2002       162.0  …              1              0
3       1915          1970         0.0  …              1              0
4       2000          2000       350.0  …              1              0

   GarageCond_Gd  GarageCond_Po  GarageCond_TA  SaleCondition_AdjLand  \
0              0              0              1                      0
1              0              0              1                      0
2              0              0              1                      0
3              0              0              1                      0
4              0              0              1                      0

   SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
0                     0                     0                     1
1                     0                     0                     1
2                     0                     0                     1
3                     0                     0                     0
4                     0                     0                     1

   SaleCondition_Partial
0                      0
1                      0
2                      0
3                      0
4                      0

[5 rows x 224 columns]
```

1.4 Model Building and Evaluation

1.4.1 Linear Regression - Least Squares Fitting

[42]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

[43]:
```
house_prices_X = house_prices.drop('SalePrice', axis=1)
house_prices_y = house_prices['SalePrice'].values.reshape(-1,1)
lm = LinearRegression()
MSEs = cross_val_score(lm, house_prices_X, house_prices_y,
                      scoring = 'neg_mean_squared_error', cv=5)
mean_MSE = np.mean(MSEs)
print('mean_MSE : ', mean_MSE)

lm.fit(house_prices_X, house_prices_y)
house_prices_pred = lm.predict(house_prices_X)
print('Mean Squared Error : ',np.sqrt(mean_squared_error(house_prices_y,␣
 ↪house_prices_pred)))
print('R2 Score : ',r2_score(house_prices_y, house_prices_pred))
```

```
mean_MSE :  -3463351332.2262397
Mean Squared Error :  21623.558696434407
R2 Score :  0.9323345699549781
```

1.4.2 Ridge Regression

[44]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
```

[45]:
```
ridge = Ridge()
parameters = {'alpha': [ 1e-2, 1, 3, 5, 8, 10, 20, 50]}
ridge_regressor = GridSearchCV(estimator = ridge,
                               param_grid = parameters,
                               scoring = 'neg_mean_squared_error',
                               cv=5,
                               return_train_score=True,
                               verbose = 1)
ridge_regressor.fit(house_prices_X, house_prices_y)

print('Best Parameter : ', ridge_regressor.best_params_)
print('Best score : ', ridge_regressor.best_score_)

house_prices_pred_ridge = ridge_regressor.predict(house_prices_X)
print('Mean Squared Error : ',np.sqrt(mean_squared_error(house_prices_y,␣
 ↪house_prices_pred_ridge)))
```

```
print('R2 Score : ',r2_score(house_prices_y, house_prices_pred_ridge))
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best Parameter :  {'alpha': 8}
Best score :  -1271898439.6906643
Mean Squared Error :  27090.782716103156
R2 Score :  0.8937923955455443

[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:    0.4s finished
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:814: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

Best Parameter for ridge regression is : {'alpha': 8}

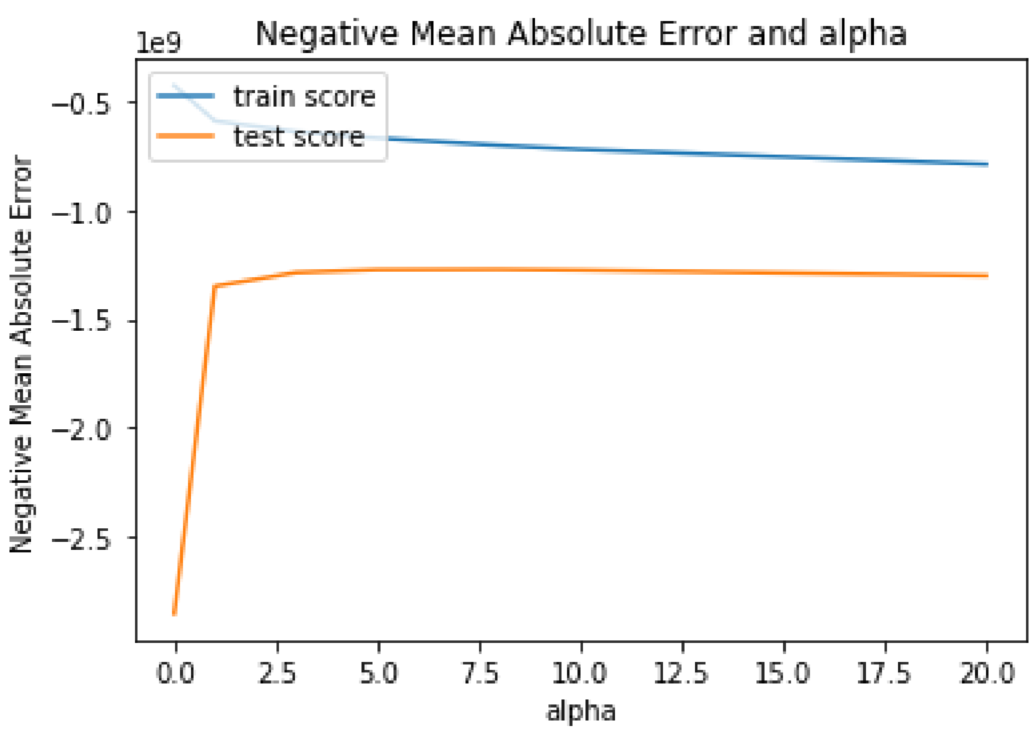
[46]:
```
cv_results = pd.DataFrame(ridge_regressor.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<=20]
cv_results.head()
```

[46]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time param_alpha  \
0       0.015308      0.007338         0.003372        0.001642        0.01
1       0.006058      0.001326         0.001388        0.000323           1
2       0.004430      0.000116         0.001094        0.000018           3
3       0.009610      0.005818         0.001946        0.000763           5
4       0.008439      0.002356         0.001849        0.000398           8

             params  split0_test_score  split1_test_score  split2_test_score  \
0  {'alpha': 0.01}      -3.131078e+09      -5.485413e+09      -2.073675e+09
1     {'alpha': 1}      -6.529956e+08      -1.301786e+09      -1.288566e+09
2     {'alpha': 3}      -6.070646e+08      -1.271233e+09      -1.175712e+09
3     {'alpha': 5}      -5.917897e+08      -1.264559e+09      -1.166726e+09
4     {'alpha': 8}      -5.812925e+08      -1.265670e+09      -1.177524e+09

   split3_test_score  …  mean_test_score  std_test_score  rank_test_score  \
0      -6.837652e+08  …    -2.855663e+09    1.570588e+09                8
1      -8.524257e+08  …    -1.348297e+09    6.968919e+08                6
2      -8.013432e+08  …    -1.286047e+09    6.898166e+08                4
3      -7.775054e+08  …    -1.273065e+09    6.927298e+08                2
4      -7.601534e+08  …    -1.271898e+09    7.007827e+08                1

   split0_train_score  split1_train_score  split2_train_score  \
0       -4.866166e+08       -4.071896e+08       -3.301947e+08
1       -6.832788e+08       -5.960382e+08       -5.627355e+08
```

```
2        -7.359186e+08       -6.408170e+08       -6.298955e+08
3        -7.672608e+08       -6.670085e+08       -6.641260e+08
4        -8.034244e+08       -6.969855e+08       -6.994766e+08

   split3_train_score  split4_train_score  mean_train_score  std_train_score
0        -4.652837e+08       -4.390802e+08     -4.256730e+08     5.462715e+07
1        -6.338600e+08       -4.630405e+08     -5.877906e+08     7.416031e+07
2        -6.855673e+08       -4.947101e+08     -6.373817e+08     8.056601e+07
3        -7.184298e+08       -5.182044e+08     -6.670059e+08     8.348128e+07
4        -7.564375e+08       -5.461551e+08     -7.004958e+08     8.666969e+07

[5 rows x 21 columns]
```

[47]:

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

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



1.4.3 Lasso Regression

[48]:
```
from sklearn.linear_model import Lasso
```

[49]:
```
lasso = Lasso(tol=0.001, max_iter=100)
parameters = {'alpha': [20, 50, 100, 110, 120, 125, 128, 130, 133, 135, 140,␣
↪145, 150, 200, 250, 300]}
lasso_regressor = GridSearchCV(estimator = lasso,
                               param_grid = parameters,
                               scoring = 'neg_mean_squared_error',
                               cv=5,
                               return_train_score=True,
                               verbose = 1)
lasso_regressor.fit(house_prices_X, house_prices_y)

print('Best Parameter : ', lasso_regressor.best_params_)
print('Best score : ', lasso_regressor.best_score_)

house_prices_pred_lasso = lasso_regressor.predict(house_prices_X)
print('Mean Squared Error : ',np.sqrt(mean_squared_error(house_prices_y,␣
↪house_prices_pred_lasso)))
print('R2 Score : ',r2_score(house_prices_y, house_prices_pred_lasso))
```

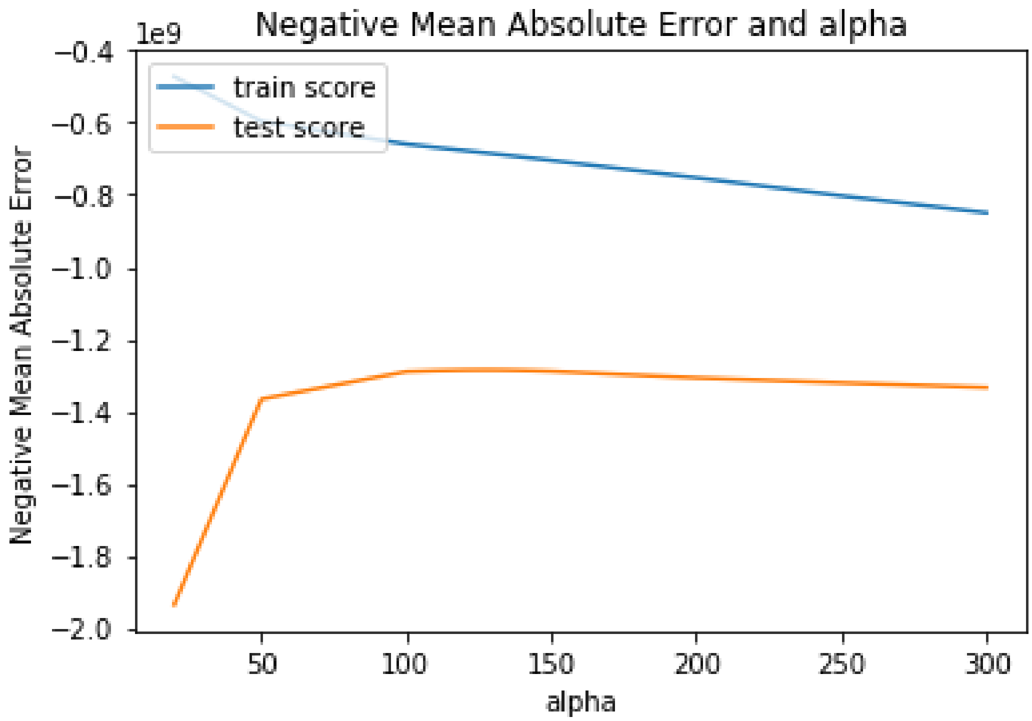
```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best Parameter :  {'alpha': 128}
Best score :  -1285405522.4791458
Mean Squared Error :  27219.875014219506
R2 Score :  0.8927777884343187

[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    1.5s finished
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:814: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[50]:
```
cv_results = pd.DataFrame(lasso_regressor.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<=500]
cv_results.head()
```

[50]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time param_alpha  \
0       0.024923      0.006109         0.001821        0.000261          20
1       0.014866      0.002270         0.001397        0.000364          50
```

```
2        0.030290      0.010278          0.001907         0.000323          100
3        0.012658      0.001094          0.001155         0.000097          110
4        0.012584      0.001006          0.001173         0.000029          120

           params  split0_test_score  split1_test_score  split2_test_score  \
0   {'alpha': 20}      -1.780951e+09      -3.105635e+09      -1.488734e+09
1   {'alpha': 50}      -6.533619e+08      -1.381605e+09      -1.377087e+09
2  {'alpha': 100}      -5.368075e+08      -1.238030e+09      -1.260826e+09
3  {'alpha': 110}      -5.361581e+08      -1.232180e+09      -1.249369e+09
4  {'alpha': 120}      -5.364797e+08      -1.228911e+09      -1.239883e+09

   split3_test_score  …  mean_test_score  std_test_score  rank_test_score  \
0      -6.934877e+08  …    -1.935453e+09    8.477646e+08               16
1      -8.041660e+08  …    -1.366043e+09    6.914755e+08               15
2      -7.559616e+08  …    -1.289681e+09    7.396230e+08               11
3      -7.513943e+08  …    -1.287193e+09    7.443581e+08                8
4      -7.479536e+08  …    -1.285678e+09    7.480440e+08                5

   split0_train_score  split1_train_score  split2_train_score  \
0       -5.285944e+08       -4.542915e+08       -3.985466e+08
1       -6.679963e+08       -5.923737e+08       -5.740736e+08
2       -7.626003e+08       -6.695604e+08       -6.360567e+08
3       -7.730468e+08       -6.783568e+08       -6.456949e+08
4       -7.828618e+08       -6.873277e+08       -6.551956e+08

   split3_train_score  split4_train_score  mean_train_score  std_train_score
0       -5.232207e+08       -4.587330e+08     -4.726773e+08     4.839256e+07
1       -6.630214e+08       -4.850223e+08     -5.964975e+08     6.705828e+07
2       -7.137486e+08       -5.167073e+08     -6.597347e+08     8.320199e+07
3       -7.226740e+08       -5.233968e+08     -6.686339e+08     8.430530e+07
4       -7.320414e+08       -5.296531e+08     -6.774159e+08     8.547352e+07

[5 rows x 21 columns]
```

[51]:
```python
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

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



Best parameter: alpha = 128 and hence let us run Lasso regression with alpha = 128

[52]:
```
alpha = 128

lasso = Lasso(alpha=alpha, tol=0.001, max_iter=100)

lasso.fit(house_prices_X, house_prices_y)
```

[52]:
```
Lasso(alpha=128, copy_X=True, fit_intercept=True, max_iter=100, normalize=False,
      positive=False, precompute=False, random_state=None, selection='cyclic',
      tol=0.001, warm_start=False)
```

[53]:
```
# Let us look at the coefficients of the model
coef = pd.Series(lasso.coef_, index=house_prices_X.columns)
coef
```

[53]:
```
Id                        -0.821584
MSSubClass              -227.137932
LotFrontage              -96.799531
LotArea                    0.617619
Street                     0.000000
                             …
SaleCondition_AdjLand      0.000000
SaleCondition_Alloca      -0.000000
```

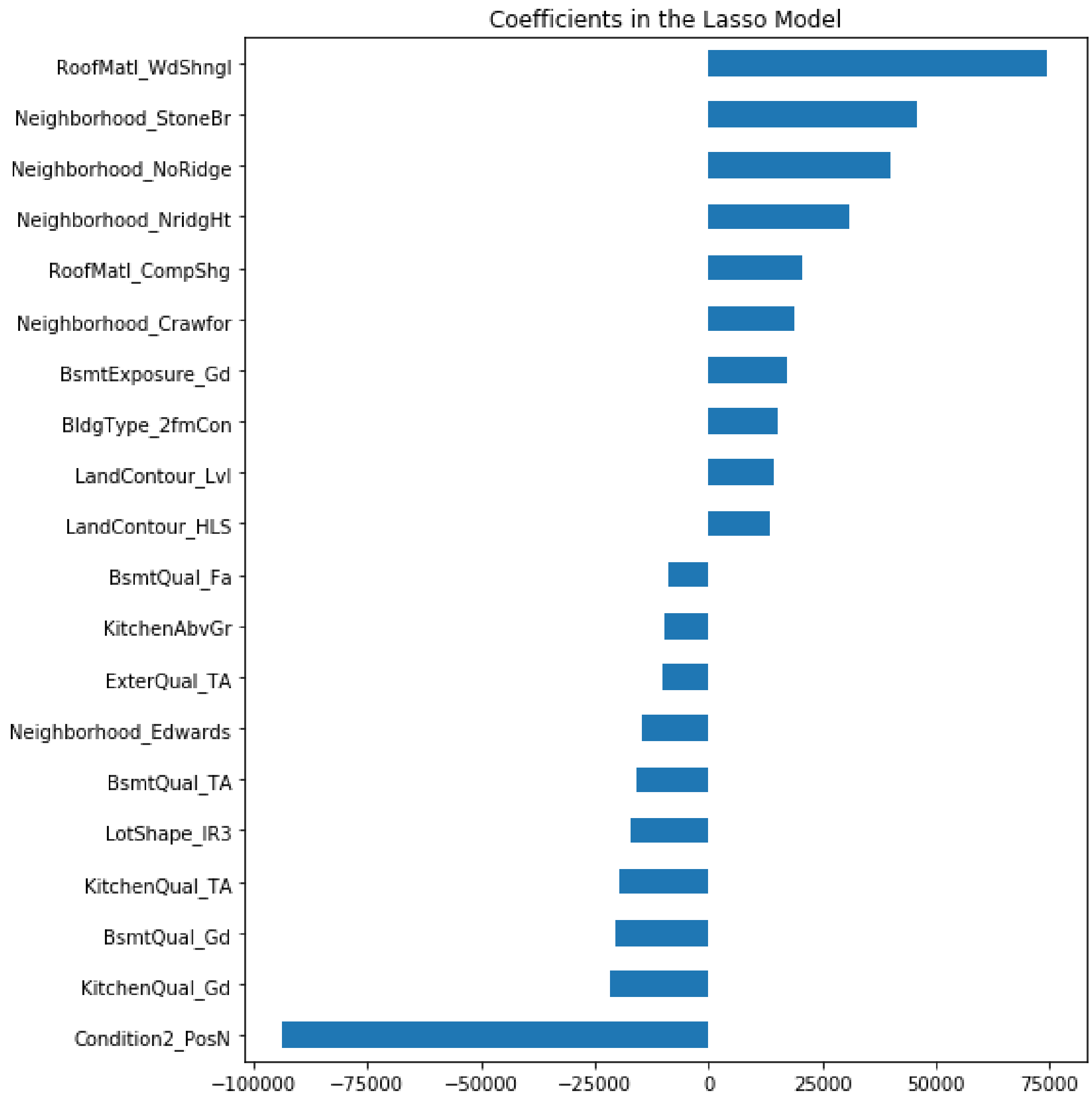
```
SaleCondition_Family      -0.000000
SaleCondition_Normal     960.458659
SaleCondition_Partial      0.000000
Length: 223, dtype: float64
```

[54]:
```
print('Lasso picked ' + str(sum(coef != 0)) +
      ' variables and eliminated the other ' + str(sum(coef == 0)) + "␣
 ↪variables")
```

```
Lasso picked 108 variables and eliminated the other 115 variables
```

[55]:
```
# Let us look at a chart about the coefficients
import matplotlib
imp_coef = pd.concat([coef.sort_values().head(10), coef.sort_values().tail(10)])
matplotlib.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind = 'barh')
plt.title('Coefficients in the Lasso Model')
```

[55]:
```
Text(0.5, 1.0, 'Coefficients in the Lasso Model')
```



1.4.4 From the above plot, we can see that SalePrice is positively correlated with below 10

1.4.5 variables in the given order (high to low):

- Roof Material - Wood Shingles
- Neighborhood - Stone Brook
- Neighborhood - Northridge
- Neighborhood - Northridge Heights
- Roof Material - Standard (Composite) Shingle
- Neighborhood - Crawford
- walkout or garden level walls - Good Exposure
- Building Type - Two-family Conversion; originally built as one-family dwelling

- Flatness of the property - Near Flat/Level

- Flatness of the property - Hillside - Significant slope from side to side ### And Sale price is negatively correlated with the below 10 variables in that order (high to low):
- Proximity - Near positive off-site feature–park, greenbelt, etc.
- Kitchen Quality - Good
- Height of the basement - Good (90-99 inches) - Typical (80-89 inches)
- Kitchen Quality - Typical/Average
- General shape of property - Irregular
- Height of the basement - Typical (80-89 inches)
- Neighborhood - Edwards
- Quality of the material on the exterior - Average/Typical
- Kitchens above grade
- Height of the basement - Fair (70-79 inches)

1.4.6 Below model is created to answer the third Subjective question

[63]:
```
# Let is remove the five most important predictor variables from the model 1
columns_to_drop =␣
↪['RoofMatl_WdShngl','Neighborhood_StoneBr','Neighborhood_NoRidge',
                  'Neighborhood_NridgHt','RoofMatl_CompShg']
model2_X = house_prices_X.drop(columns_to_drop, axis=1)
model2_X.head()
```

[63]:
```
   Id  MSSubClass  LotFrontage  LotArea  Street  OverallQual  OverallCond  \
0   1          60         65.0     8450       1            7            5
1   2          20         80.0     9600       1            6            8
2   3          60         68.0    11250       1            7            5
3   4          70         60.0     9550       1            7            5
4   5          60         84.0    14260       1            8            5

   YearBuilt  YearRemodAdd  MasVnrArea  …  GarageQual_TA  GarageCond_Fa  \
0       2003          2003       196.0  …              1              0
1       1976          1976         0.0  …              1              0
2       2001          2002       162.0  …              1              0
3       1915          1970         0.0  …              1              0
4       2000          2000       350.0  …              1              0

   GarageCond_Gd  GarageCond_Po  GarageCond_TA  SaleCondition_AdjLand  \
0              0              0              1                      0
1              0              0              1                      0
2              0              0              1                      0
3              0              0              1                      0
4              0              0              1                      0

   SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
0                     0                     0                     1
```

```
1                      0                      0                      1
2                      0                      0                      1
3                      0                      0                      0
4                      0                      0                      1

   SaleCondition_Partial
0                      0
1                      0
2                      0
3                      0
4                      0

[5 rows x 218 columns]
```

[64]:
```
# With the optimal alpha of 128, let us perform lasso regression
alpha = 128

lasso = Lasso(alpha=alpha, tol=0.001, max_iter=100)

lasso.fit(model2_X, house_prices_y)
```

[64]:
```
Lasso(alpha=128, copy_X=True, fit_intercept=True, max_iter=100, normalize=False,
      positive=False, precompute=False, random_state=None, selection='cyclic',
      tol=0.001, warm_start=False)
```

[66]:
```
# Let us look at the coefficients of the model 2
coef = pd.Series(lasso.coef_, index=model2_X.columns)
coef
```

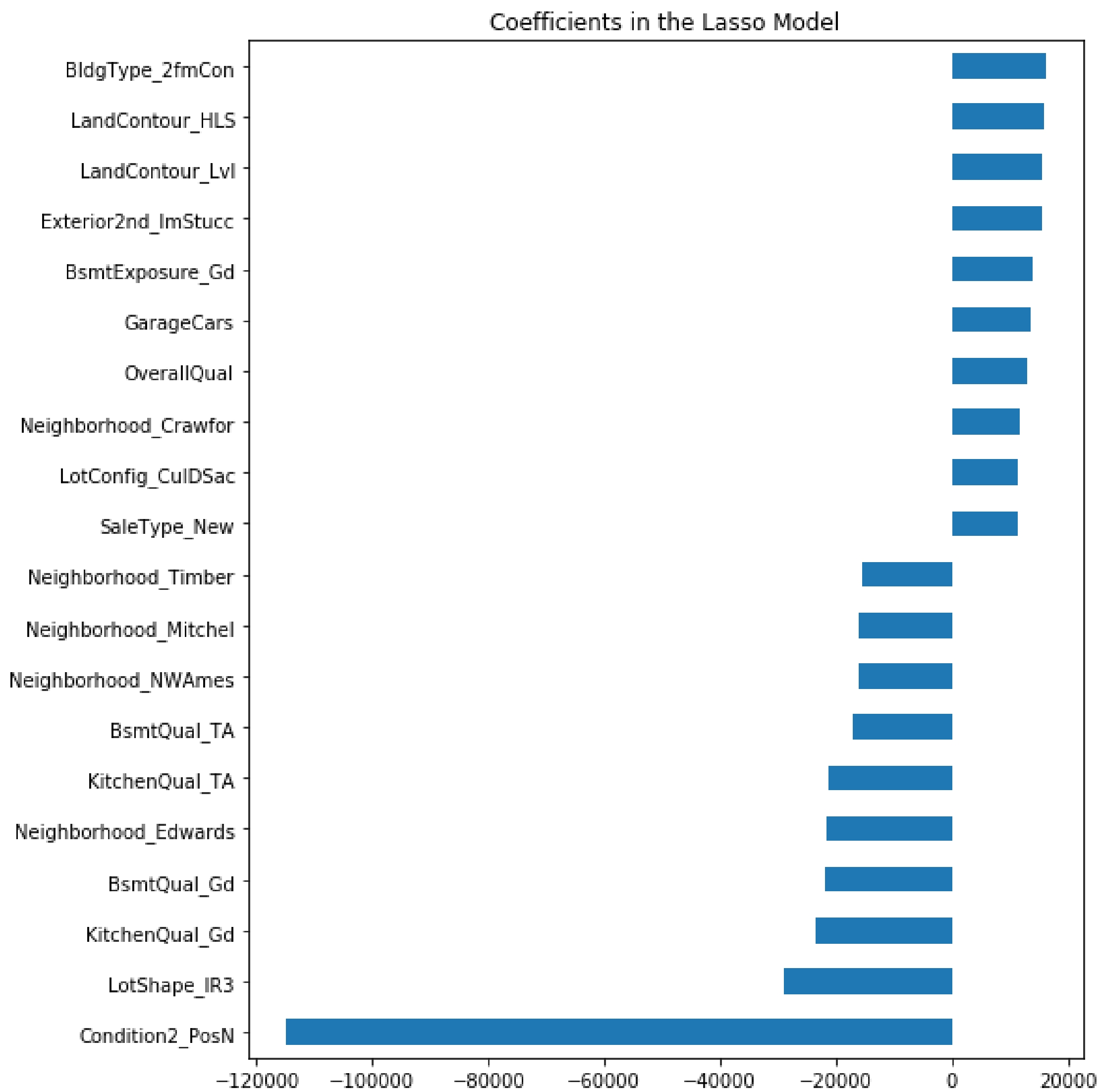
[66]:
```
Id                          -1.171019
MSSubClass                -241.510373
LotFrontage               -120.861673
LotArea                      0.766002
Street                       0.000000
                              …
SaleCondition_AdjLand        0.000000
SaleCondition_Alloca        -0.000000
SaleCondition_Family        -0.000000
SaleCondition_Normal      1402.844752
SaleCondition_Partial        0.000000
Length: 218, dtype: float64
```

[67]:
```
print('Lasso picked ' + str(sum(coef != 0)) +
      ' variables and eliminated the other ' + str(sum(coef == 0)) + "␣
 ↪variables")
```

```
Lasso picked 103 variables and eliminated the other 115 variables
```

[68]:
```
# Let us look at a chart about the coefficients for the model 2
import matplotlib
imp_coef = pd.concat([coef.sort_values().head(10), coef.sort_values().tail(10)])
matplotlib.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind = 'barh')
plt.title('Coefficients in the Lasso Model')
```

[68]:
```
Text(0.5, 1.0, 'Coefficients in the Lasso Model')
```



Five most important predictor variable from model 2 positively correlated with SalePrice are as below:

- Building Type - Two-family Conversion; originally built as one-family dwelling

- Flatness of the property - Hillside - Significant slope from side to side
- Flatness of the property - Near Flat/Level
- Exterior covering on house - Imitation Stucco
- walkout or garden level walls - Good Exposure

[ ]:
```

```