In [1]:
```
####Assignment of Advanced Regression for house price predictions Using Ridge
 and Lasso algorithm
```

In [2]:
```
# Importing libraries
import os
import sys

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
```

In [3]:
```
#reading the dataset
housePrice=pd.read_csv('train.csv')
housePrice.info
```

Out[3]:
```
<bound method DataFrame.info of         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
5        6          50       RL         85.0    14115   Pave   NaN      IR1
6        7          20       RL         75.0    10084   Pave   NaN      Reg
7        8          60       RL          NaN    10382   Pave   NaN      IR1
8        9          50       RM         51.0     6120   Pave   NaN      Reg
9       10         190       RL         50.0     7420   Pave   NaN      Reg
10      11          20       RL         70.0    11200   Pave   NaN      Reg
11      12          60       RL         85.0    11924   Pave   NaN      IR1
12      13          20       RL          NaN    12968   Pave   NaN      IR2
13      14          20       RL         91.0    10652   Pave   NaN      IR1
14      15          20       RL          NaN    10920   Pave   NaN      IR1
15      16          45       RM         51.0     6120   Pave   NaN      Reg
16      17          20       RL          NaN    11241   Pave   NaN      IR1
17      18          90       RL         72.0    10791   Pave   NaN      Reg
18      19          20       RL         66.0    13695   Pave   NaN      Reg
19      20          20       RL         70.0     7560   Pave   NaN      Reg
20      21          60       RL        101.0    14215   Pave   NaN      IR1
21      22          45       RM         57.0     7449   Pave  Grvl      Reg
22      23          20       RL         75.0     9742   Pave   NaN      Reg
23      24         120       RM         44.0     4224   Pave   NaN      Reg
24      25          20       RL          NaN     8246   Pave   NaN      IR1
25      26          20       RL        110.0    14230   Pave   NaN      Reg
26      27          20       RL         60.0     7200   Pave   NaN      Reg
27      28          20       RL         98.0    11478   Pave   NaN      Reg
28      29          20       RL         47.0    16321   Pave   NaN      IR1
29      30          30       RM         60.0     6324   Pave   NaN      IR1
...    ...         ...      ...          ...      ...    ...   ...      ...
1430  1431          60       RL         60.0    21930   Pave   NaN      IR3
1431  1432         120       RL          NaN     4928   Pave   NaN      IR1
1432  1433          30       RL         60.0    10800   Pave  Grvl      Reg
1433  1434          60       RL         93.0    10261   Pave   NaN      IR1
1434  1435          20       RL         80.0    17400   Pave   NaN      Reg
1435  1436          20       RL         80.0     8400   Pave   NaN      Reg
1436  1437          20       RL         60.0     9000   Pave   NaN      Reg
1437  1438          20       RL         96.0    12444   Pave   NaN      Reg
1438  1439          20       RM         90.0     7407   Pave   NaN      Reg
1439  1440          60       RL         80.0    11584   Pave   NaN      Reg
1440  1441          70       RL         79.0    11526   Pave   NaN      IR1
1441  1442         120       RM          NaN     4426   Pave   NaN      Reg
1442  1443          60       FV         85.0    11003   Pave   NaN      Reg
1443  1444          30       RL          NaN     8854   Pave   NaN      Reg
1444  1445          20       RL         63.0     8500   Pave   NaN      Reg
1445  1446          85       RL         70.0     8400   Pave   NaN      Reg
1446  1447          20       RL          NaN    26142   Pave   NaN      IR1
1447  1448          60       RL         80.0    10000   Pave   NaN      Reg
1448  1449          50       RL         70.0    11767   Pave   NaN      Reg
1449  1450         180       RM         21.0     1533   Pave   NaN      Reg
1450  1451          90       RL         60.0     9000   Pave   NaN      Reg
1451  1452          20       RL         78.0     9262   Pave   NaN      Reg
1452  1453         180       RM         35.0     3675   Pave   NaN      Reg
1453  1454          20       RL         90.0    17217   Pave   NaN      Reg
```

```
1454  1455          20        FV          62.0     7500   Pave  Pave       Reg
1455  1456          60        RL          62.0     7917   Pave   NaN       Reg
1456  1457          20        RL          85.0    13175   Pave   NaN       Reg
1457  1458          70        RL          66.0     9042   Pave   NaN       Reg
1458  1459          20        RL          68.0     9717   Pave   NaN       Reg
1459  1460          20        RL          75.0     9937   Pave   NaN       Reg

     LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature MiscVal  \
0            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
2            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
3            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
4            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
5            Lvl    AllPub  ...        0    NaN  MnPrv        Shed     700
6            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
7            Lvl    AllPub  ...        0    NaN    NaN        Shed     350
8            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
9            Lvl    AllPub  ...        0    NaN    NaN         NaN       0
10           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
11           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
12           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
13           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
14           Lvl    AllPub  ...        0    NaN   GdWo         NaN       0
15           Lvl    AllPub  ...        0    NaN  GdPrv         NaN       0
16           Lvl    AllPub  ...        0    NaN    NaN        Shed     700
17           Lvl    AllPub  ...        0    NaN    NaN        Shed     500
18           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
19           Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0
20           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
21           Bnk    AllPub  ...        0    NaN  GdPrv         NaN       0
22           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
23           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
24           Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0
25           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
26           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
27           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
28           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
29           Lvl    AllPub  ...        0    NaN    NaN         NaN       0
...          ...       ...  ...      ...    ...    ...         ...     ...
1430         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1431         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1432         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1433         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1434         Low    AllPub  ...        0    NaN    NaN         NaN       0
1435         Lvl    AllPub  ...        0    NaN  GdPrv         NaN       0
1436         Lvl    AllPub  ...        0    NaN   GdWo         NaN       0
1437         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1438         Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0
1439         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1440         Bnk    AllPub  ...        0    NaN    NaN         NaN       0
1441         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1442         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1443         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1444         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1445         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1446         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1447         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
```

```
1448         Lvl    AllPub  ...         0    NaN   GdWo            NaN        0
1449         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1450         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1451         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1452         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1453         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1454         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1455         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1456         Lvl    AllPub  ...         0    NaN  MnPrv            NaN        0
1457         Lvl    AllPub  ...         0    NaN  GdPrv           Shed     2500
1458         Lvl    AllPub  ...         0    NaN    NaN            NaN        0
1459         Lvl    AllPub  ...         0    NaN    NaN            NaN        0

     MoSold YrSold  SaleType  SaleCondition  SalePrice
0         2   2008        WD         Normal     208500
1         5   2007        WD         Normal     181500
2         9   2008        WD         Normal     223500
3         2   2006        WD        Abnorml     140000
4        12   2008        WD         Normal     250000
5        10   2009        WD         Normal     143000
6         8   2007        WD         Normal     307000
7        11   2009        WD         Normal     200000
8         4   2008        WD        Abnorml     129900
9         1   2008        WD         Normal     118000
10        2   2008        WD         Normal     129500
11        7   2006       New        Partial     345000
12        9   2008        WD         Normal     144000
13        8   2007       New        Partial     279500
14        5   2008        WD         Normal     157000
15        7   2007        WD         Normal     132000
16        3   2010        WD         Normal     149000
17       10   2006        WD         Normal      90000
18        6   2008        WD         Normal     159000
19        5   2009       COD        Abnorml     139000
20       11   2006       New        Partial     325300
21        6   2007        WD         Normal     139400
22        9   2008        WD         Normal     230000
23        6   2007        WD         Normal     129900
24        5   2010        WD         Normal     154000
25        7   2009        WD         Normal     256300
26        5   2010        WD         Normal     134800
27        5   2010        WD         Normal     306000
28       12   2006        WD         Normal     207500
29        5   2008        WD         Normal      68500
...     ...    ...       ...            ...        ...
1430      7   2006        WD         Normal     192140
1431     10   2009        WD         Normal     143750
1432      8   2007        WD         Normal      64500
1433      5   2008        WD         Normal     186500
1434      5   2006        WD         Normal     160000
1435      7   2008       COD        Abnorml     174000
1436      5   2007        WD         Normal     120500
1437     11   2008       New        Partial     394617
1438      4   2010        WD         Normal     149700
1439     11   2007        WD         Normal     197000
1440      9   2008        WD         Normal     191000
1441      5   2008        WD         Normal     149300
```

```
1442       4   2009        WD        Normal     310000
1443       5   2009        WD        Normal     121000
1444      11   2007        WD        Normal     179600
1445       5   2007        WD        Normal     129000
1446       4   2010        WD        Normal     157900
1447      12   2007        WD        Normal     240000
1448       5   2007        WD        Normal     112000
1449       8   2006        WD       Abnorml      92000
1450       9   2009        WD        Normal     136000
1451       5   2009       New       Partial     287090
1452       5   2006        WD        Normal     145000
1453       7   2006        WD       Abnorml      84500
1454      10   2009        WD        Normal     185000
1455       8   2007        WD        Normal     175000
1456       2   2010        WD        Normal     210000
1457       5   2010        WD        Normal     266500
1458       4   2010        WD        Normal     142125
1459       6   2008        WD        Normal     147500

[1460 rows x 81 columns]>
```

In [4]:
```
housePrice.head()
```

Out[4]:

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

5 rows × 81 columns

In [5]:
```
housePrice.shape
```

Out[5]: (1460, 81)

In [6]:
```
housePrice.describe()
```

Out[6]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	Yea
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.0(
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.2(
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.2(
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.0(
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.0(
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.0(
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.0(
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.0(

8 rows × 38 columns

In [7]:
```
housePrice.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
```

In [8]:

```
# Data Cleaning - Missing Value, Duplicates, Imputing, Dropping, Deleting, Exp
loration
```

In [9]:

```
#Cleaning up variable Alley (Replacing NA => No Alley Access)
housePrice['Alley'].replace({np.nan:'No Alley Access'},inplace=True)
100*(housePrice['Alley'].value_counts()/housePrice['Alley'].count())
```

Out[9]:

```
No Alley Access    93.767123
Grvl                3.424658
Pave                2.808219
Name: Alley, dtype: float64
```

In [10]:

```
# As 94% of Alley is "No Alley access" it can be dropped
housePrice=housePrice.drop(['Alley'],axis=1)
```

In [11]:
```
#Checking the dataset for the amount of nulls present
round(housePrice.isnull().sum()/len(housePrice.index),2).sort_values(ascending
=False).head(18)
```

Out[11]:
```
PoolQC          1.00
MiscFeature     0.96
Fence           0.81
FireplaceQu     0.47
LotFrontage     0.18
GarageType      0.06
GarageCond      0.06
GarageYrBlt     0.06
GarageFinish    0.06
GarageQual      0.06
BsmtFinType1    0.03
BsmtExposure    0.03
BsmtCond        0.03
BsmtQual        0.03
BsmtFinType2    0.03
MasVnrArea      0.01
MasVnrType      0.01
Exterior2nd     0.00
dtype: float64
```

In [12]:
```
#Considering 10% as my threshold and dropping the column
round(housePrice.isnull().sum()/len(housePrice.index),2)[round(housePrice.isnu
ll().sum()/len(housePrice.index),2).values>0.10]
```

Out[12]:
```
LotFrontage    0.18
FireplaceQu    0.47
PoolQC         1.00
Fence          0.81
MiscFeature    0.96
dtype: float64
```

In [13]:
```
housePrice = housePrice.drop(['LotFrontage','FireplaceQu','PoolQC','Fence','Mi
scFeature'],axis='columns')
```

In [14]:
```
#verifying the columns for the missing values between 0-10%
round(housePrice.isnull().sum()/len(housePrice.index),2)[round(housePrice.isnu
ll().sum()/len(housePrice.index),2).values>0.00]
```

Out[14]:
```
MasVnrType      0.01
MasVnrArea      0.01
BsmtQual        0.03
BsmtCond        0.03
BsmtExposure    0.03
BsmtFinType1    0.03
BsmtFinType2    0.03
GarageType      0.06
GarageYrBlt     0.06
GarageFinish    0.06
GarageQual      0.06
GarageCond      0.06
dtype: float64
```

In [15]:
```
#convert the Year columns with the age to fill these columns with number
housePrice['YearBuiltOld'] = housePrice.YearBuilt.max()-housePrice.YearBuilt
housePrice['YearRemodAddOld'] = housePrice.YearRemodAdd.max()-housePrice.YearR
emodAdd
housePrice['GarageYrBltOld'] = housePrice.GarageYrBlt.max()-housePrice.GarageY
rBlt
housePrice['YrSoldOld'] = housePrice.YrSold.max()-housePrice.YrSold
housePrice[['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold','YearBuiltOld',
'YearRemodAddOld','GarageYrBltOld','YrSoldOld']].sample(10)
```

Out[15]:

	YearBuilt	YearRemodAdd	GarageYrBlt	YrSold	YearBuiltOld	YearRemodAddOld	GarageYr
484	1962	2001	1963.0	2007	48	9	
1015	2001	2001	2001.0	2009	9	9	
1131	1991	1992	NaN	2007	19	18	
406	1936	1950	1936.0	2008	74	60	
1069	1949	2003	1985.0	2007	61	7	
88	1915	1982	NaN	2009	95	28	
174	1986	1986	1986.0	2008	24	24	
592	1982	2003	1985.0	2008	28	7	
205	1990	1990	1990.0	2009	20	20	
651	1940	1950	1940.0	2009	70	60	

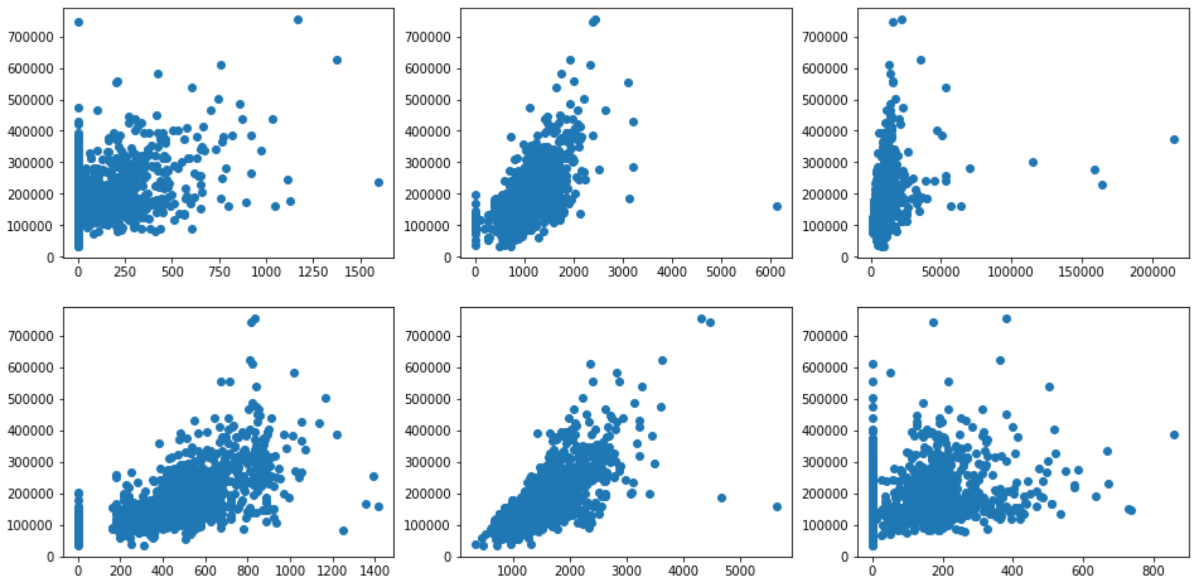
In [16]:
```
#Lets drop original Year columns
housePrice = housePrice.drop(['YearBuilt','YearRemodAdd','GarageYrBlt','YrSol
d'],axis='columns')
```

In [17]:
```
#Imputing Missing values
housePrice.MasVnrType.fillna('None',inplace=True)
housePrice.MasVnrArea.fillna(housePrice.MasVnrArea.mean(),inplace=True)
housePrice.BsmtQual.fillna('TA',inplace=True)
housePrice.BsmtCond.fillna('TA',inplace=True)
housePrice.BsmtExposure.fillna('No',inplace=True)
housePrice.BsmtFinType1.fillna('Unf',inplace=True)
housePrice.BsmtFinType2.fillna('Unf',inplace=True)
housePrice.GarageType.fillna('Attchd',inplace=True)
housePrice.GarageFinish.fillna('Unf',inplace=True)
housePrice.GarageQual.fillna('TA',inplace=True)
housePrice.GarageCond.fillna('TA',inplace=True)
housePrice.GarageYrBltOld.fillna(-1,inplace=True)
```

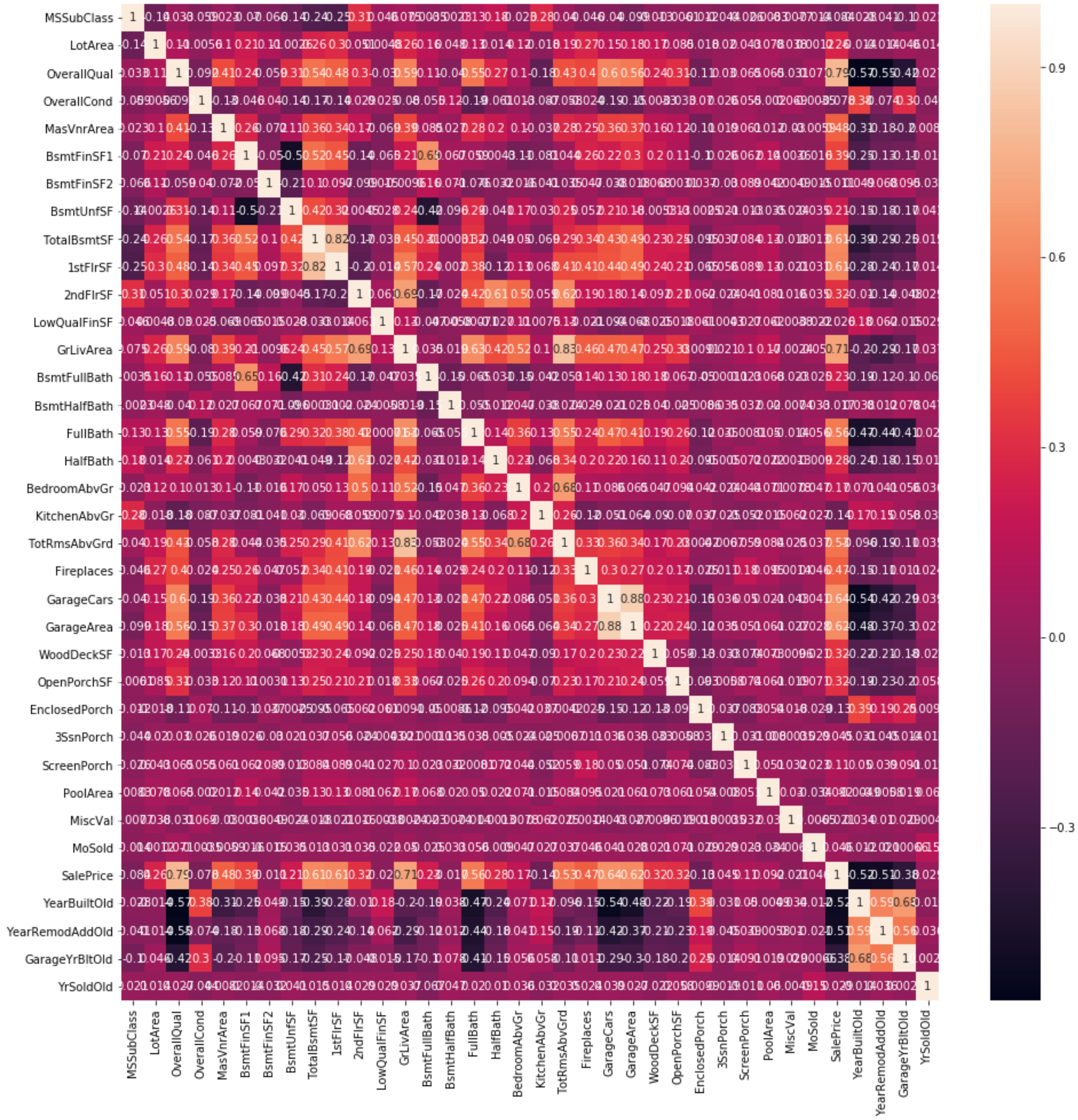
In [18]:
```
#dropping Id, street and utilities as no impact
housePrice = housePrice.drop(['Id','Street','Utilities'],axis='columns')
```

In [19]:
```
#EDA Analysis with graphs
plt.figure(figsize=(16,8))
plt.subplot(2,3,1)
plt.scatter(housePrice.MasVnrArea,housePrice.SalePrice)
plt.subplot(2,3,2)
plt.scatter(housePrice.TotalBsmtSF,housePrice.SalePrice)
plt.subplot(2,3,3)
plt.scatter(housePrice['LotArea'],housePrice.SalePrice)
plt.subplot(2,3,4)
plt.scatter(housePrice['GarageArea'],housePrice.SalePrice)
plt.subplot(2,3,5)
plt.scatter(housePrice['GrLivArea'],housePrice.SalePrice)
plt.subplot(2,3,6)
plt.scatter(housePrice['WoodDeckSF'],housePrice.SalePrice)
```

Out[19]: <matplotlib.collections.PathCollection at 0x1d1e8b71d30>



In [20]:
```
plt.figure(figsize=(16,16))
sns.heatmap(housePrice[list(housePrice.dtypes[housePrice.dtypes!='object'].ind
ex)].corr(),annot=True)
plt.show()
```



In [21]:
```
#Handling the outliers by considering the lower and upper quantile as 0.25 &
 0.99
housePrice.shape
```

Out[21]:
```
(1460, 72)
```

In [22]:
```
numCol = list(housePrice.dtypes[housePrice.dtypes !='object'].index)
numCol = ['LotArea','MasVnrArea','BsmtFinSF1','BsmtFinSF2','TotalBsmtSF','1stF
lrSF','GrLivArea','OpenPorchSF',
           'EnclosedPorch','3SsnPorch',
           'ScreenPorch' ,'PoolArea','MiscVal','SalePrice']
def dropOutliers(x):
    list = []
    for col in numCol:
        Q1 = x[col].quantile(.25)
        Q3 = x[col].quantile(.99)
        IQR = Q3-Q1
        x =  x[(x[col] >= (Q1-(1.5*IQR))) & (x[col] <= (Q3+(1.5*IQR)))]
    return x

housePrice = dropOutliers(housePrice)
```

In [23]:
```
housePrice.shape
housePrice[list(housePrice.dtypes[housePrice.dtypes=='object'].index)].head()
```

Out[23]:

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

5 rows × 36 columns

In [24]:
```
#Below columns have some kind of order and hence check ordinal in nature
housePrice[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposure','Bsmt
FinType1','BsmtFinType2','HeatingQC',
              'CentralAir',  'KitchenQual','GarageFinish','GarageQual','GarageC
ond','ExterCond','LotShape']].head()
```

Out[24]:

	LandSlope	ExterQual	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinType2	H
0	Gtl	Gd	Gd	TA	No	GLQ	Unf	
1	Gtl	TA	Gd	TA	Gd	ALQ	Unf	
2	Gtl	Gd	Gd	TA	Mn	GLQ	Unf	
3	Gtl	TA	TA	Gd	No	ALQ	Unf	
4	Gtl	Gd	Gd	TA	Av	GLQ	Unf	

In [25]:
```
housePrice['LandSlope'] = housePrice.LandSlope.map({'Gtl':0,'Mod':1,'Sev':2})
housePrice['ExterQual'] = housePrice.ExterQual.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
housePrice['BsmtQual'] = housePrice.BsmtQual.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5})
housePrice['BsmtCond'] = housePrice.BsmtCond.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5})
housePrice['BsmtExposure'] = housePrice.BsmtExposure.map({'NA':0,'No':1,'Mn':2,'Av':3,'Gd':4})
housePrice['BsmtFinType1'] = housePrice.BsmtFinType1.map({'NA':0,'Unf':1,'LwQ':2,'Rec':3,'BLQ':4,'ALQ':5,'GLQ':6})
housePrice['BsmtFinType2'] = housePrice.BsmtFinType2.map({'NA':0,'Unf':1,'LwQ':2,'Rec':3,'BLQ':4,'ALQ':5,'GLQ':6})
housePrice['HeatingQC'] = housePrice.HeatingQC.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
housePrice['CentralAir'] = housePrice.CentralAir.map({'N':0,'Y':1})
housePrice['KitchenQual'] = housePrice.KitchenQual.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
housePrice['GarageFinish'] = housePrice.GarageFinish.map({'NA':0,'Unf':1,'RFn':2,'Fin':3})
housePrice['GarageQual'] = housePrice.GarageQual.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5})
housePrice['GarageCond'] = housePrice.GarageCond.map({'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5})
housePrice['ExterCond'] = housePrice.ExterCond.map({'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4})
housePrice['LotShape'] = housePrice.LotShape.map({'IR1':0,'IR2':1,'IR3':2,'Reg':3})
```

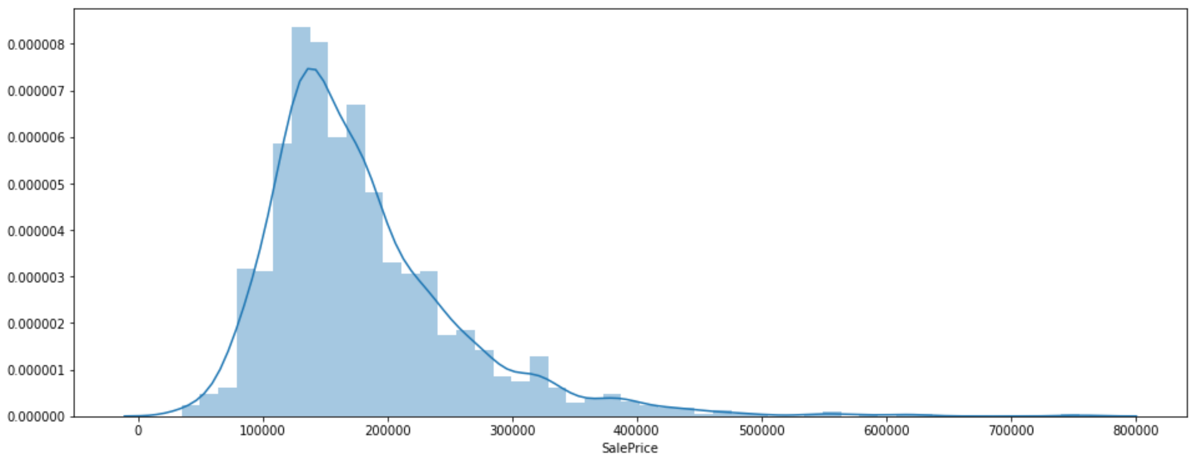
In [26]:
```
#check converted columns
housePrice[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','HeatingQC',
             'CentralAir',  'KitchenQual','GarageFinish','GarageQual','GarageCond','ExterCond','LotShape']].head()
```

Out[26]:

	LandSlope	ExterQual	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinType2	H
0	0	3	4	3	1	6	1	
1	0	2	4	3	4	5	1	
2	0	3	4	3	2	6	1	
3	0	2	3	4	1	5	1	
4	0	3	4	3	3	6	1	

In [27]:
```
#Filling dummy column with the actual dataset
dummyCol = pd.get_dummies(housePrice[['MSZoning','LandContour','LotConfig','Ne
ighborhood','Condition1','Condition2','BldgType',
             'HouseStyle','RoofStyle','RoofMatl','Exterior1st',  'Exterior2nd'
,'MasVnrType','Foundation','Heating','Electrical',
             'Functional','GarageType','PavedDrive','SaleType','SaleCondition'
]],drop_first=True)

housePrice = pd.concat([housePrice,dummyCol],axis='columns')

housePrice = housePrice.drop(['MSZoning','LandContour','LotConfig','Neighborho
od','Condition1','Condition2','BldgType',
             'HouseStyle','RoofStyle','RoofMatl','Exterior1st',  'Exterior2nd'
,'MasVnrType','Foundation','Heating',
             'Electrical','Functional','GarageType','PavedDrive','SaleType','S
aleCondition'],axis='columns')
```

In [28]:
```
#verify the distribution of target variable
plt.figure(figsize=(16,6))
sns.distplot(housePrice.SalePrice)
plt.show()
```



In [29]:
```
#Creating train and test dataset for verification
from sklearn.model_selection import train_test_split
df_train,df_test = train_test_split(housePrice,train_size=0.7,test_size=0.3,random_state=42)

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

Out[29]:

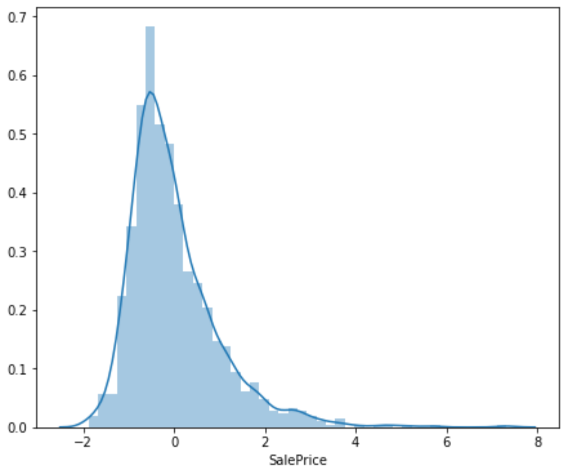
	LandSlope	ExterQual	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinType2	H
0	0	3	4	3	1	6	1	
1	0	2	4	3	4	5	1	
2	0	3	4	3	2	6	1	
3	0	2	3	4	1	5	1	
4	0	3	4	3	3	6	1	

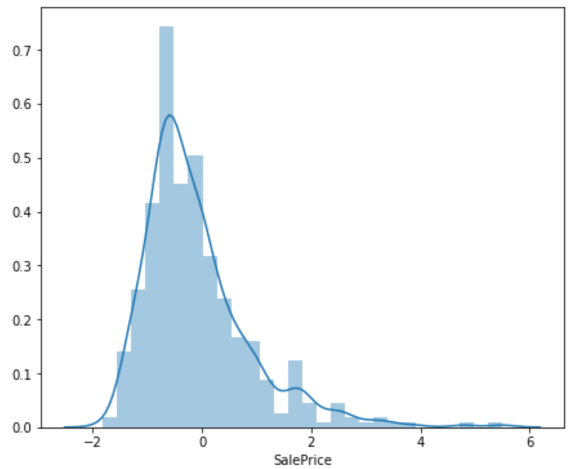
In [30]:
```
#Scaling the train dataset with dependent variable
from sklearn.preprocessing import StandardScaler

numCol = ['MSSubClass','LotArea','OverallQual','OverallCond','MasVnrArea','Bsm
tFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF',
          '1stFlrSF','2ndFlrSF','LowQualFinSF','GrLivArea','BsmtFullBath','Bsm
tHalfBath','FullBath','HalfBath','BedroomAbvGr',
          'KitchenAbvGr','TotRmsAbvGrd','Fireplaces','GarageCars','GarageArea'
,'WoodDeckSF','OpenPorchSF','EnclosedPorch',
          '3SsnPorch',
           'ScreenPorch','PoolArea','MiscVal','SalePrice']

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

```
E:\Python\lib\site-packages\ipykernel_launcher.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  # This is added back by InteractiveShellApp.init_path()
E:\Python\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
E:\Python\lib\site-packages\ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  if sys.path[0] == '':
E:\Python\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

In [31]:
```
#check the distribution
plt.figure(figsize=(16,6))
plt.subplot(121)
sns.distplot(df_train.SalePrice)
plt.subplot(122)
sns.distplot(df_test.SalePrice)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1d1e9f91e48>





In [32]:
```
#Spliting the dependent and independent variable
y_train = df_train.pop('SalePrice')
X_train = df_train

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

In [36]:
```
#Model Building with Ridge
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}


ridge = Ridge()

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

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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:    1.7s finished
```

Out[36]:
```
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=Non
e,
                             solver='auto', tol=0.001),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.
0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 5
0,
                                   100, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_absolute_error', verbose=1)
```

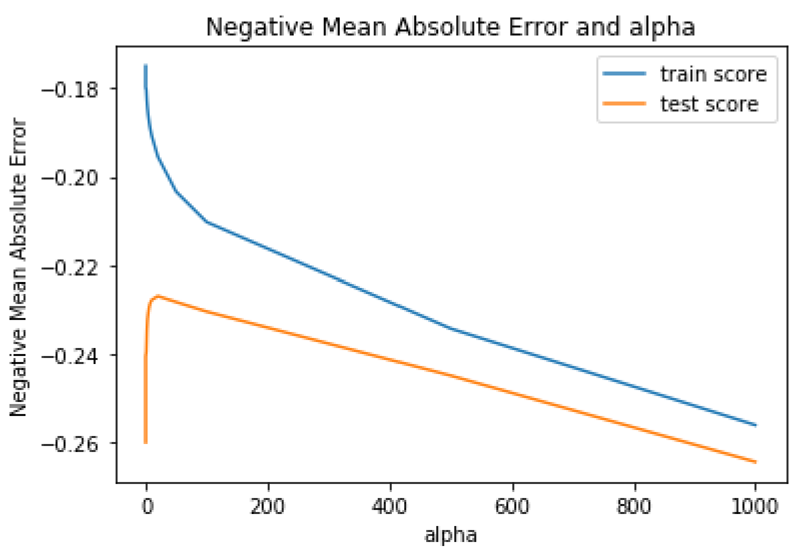
In [47]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<=1000]
cv_results.head()
```

Out[47]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_te
0	0.011098	0.002158	0.002798	0.000749	0.0001	{'alpha': 0.0001}	-
1	0.009395	0.001958	0.002002	0.000898	0.001	{'alpha': 0.001}	-
2	0.009195	0.001165	0.001799	0.000400	0.01	{'alpha': 0.01}	-
3	0.009994	0.000632	0.002198	0.000400	0.05	{'alpha': 0.05}	-
4	0.008995	0.001095	0.001798	0.000400	0.1	{'alpha': 0.1}	-

5 rows × 21 columns

In [48]:
```
# 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 right')
plt.show()
```



In [49]:
```
model_cv.best_params_
```

Out[49]: {'alpha': 20}

In [50]:
```
alpha = 20
ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)
#Predictor Variables from the Model built using Ridge Regression:
ridge.coef_
```

Out[50]:
```
array([-6.94874307e-02,  6.83512111e-02,  1.91163796e-02, -1.02702444e-02,
        1.69177621e-01,  7.59735368e-02,  9.37878124e-02,  1.00951315e-01,
       -1.11529249e-02,  8.27290150e-02, -1.35542545e-02,  6.58750835e-02,
        1.60753428e-03,  9.21320034e-02,  1.17982760e-02,  5.08248498e-03,
       -1.53962364e-02,  8.21954396e-02,  4.63484144e-03, -3.54677093e-02,
        6.39963352e-02,  1.46891973e-01,  1.08964374e-02,  1.72539399e-01,
        2.06440938e-02, -5.02669431e-03,  2.59694899e-02,  1.97819805e-02,
       -5.03812058e-02, -4.04939921e-02,  8.95019648e-02,  7.21267497e-02,
        2.40734374e-02,  1.36170197e-02,  4.83059580e-03,  6.85625925e-02,
        6.16986069e-02, -2.45860787e-02,  2.28999932e-02, -1.30097967e-02,
        1.35028841e-02,  7.58619451e-03,  3.36025838e-02,  0.00000000e+00,
       -4.07271359e-03, -1.02827315e-02, -1.85121795e-03, -1.64410803e-04,
       -9.60771185e-04,  6.91584564e-03,  2.08214420e-02,  3.49275831e-02,
        4.82259423e-02, -3.18697276e-02,  8.06871322e-02, -3.47126628e-02,
        5.05855973e-02,  1.10930858e-01, -4.70889857e-02, -7.75907203e-03,
       -1.57394740e-02,  2.45608543e-04,  3.43221032e-02,  7.19757048e-02,
       -3.45941629e-02, -1.09112253e-01,  1.48506586e-01, -1.41909483e-01,
       -8.02841881e-02, -2.90214414e-02,  1.74144431e-02, -8.26559908e-02,
       -6.93283902e-02,  2.67996594e-02, -7.00582810e-02,  1.61384230e-01,
        2.54346495e-01, -2.98807221e-02, -2.73171305e-02, -6.04124333e-03,
       -6.93813240e-02,  1.30055933e-02,  1.48670146e-01, -7.71950019e-02,
       -4.07863977e-03,  1.82166076e-02,  1.03919457e-01, -1.67682756e-02,
       -1.27841279e-01, -5.04813786e-02,  5.93816353e-02, -1.16641606e-02,
        1.33847424e-02,  1.63602130e-02,  1.41462954e-01,  0.00000000e+00,
       -1.93899300e-01,  9.87957071e-03,  1.33846667e-02,  3.88079377e-02,
        2.21131648e-02, -3.56898732e-02, -4.93955317e-02,  3.34636235e-02,
        7.34048485e-02, -3.22792200e-02, -4.65354447e-02, -3.04358903e-02,
        4.93346628e-03,  2.13837511e-02,  4.29644975e-03, -1.23927570e-03,
        3.51990874e-02,  1.21160654e-03,  3.02273288e-03,  1.29004201e-02,
        1.10227925e-02, -1.33936957e-02, -7.47487025e-02, -3.46888385e-02,
        2.06887943e-01,  1.87261643e-03,  2.33314919e-03,  1.28730853e-01,
       -3.04277590e-03, -3.33460311e-02, -1.66277311e-02,  0.00000000e+00,
       -3.90708220e-03, -2.15745888e-02, -1.50192590e-02,  2.49182424e-02,
        1.59202310e-02, -7.76094444e-02,  9.57879261e-03, -6.68078338e-03,
        2.06605100e-02,  2.27103059e-02, -3.04277590e-03, -3.39037942e-02,
       -2.71061895e-02,  3.82163280e-03, -1.79230350e-03,  1.25930547e-04,
       -5.63756551e-03,  1.62524418e-02,  1.16859136e-02, -6.85104120e-03,
        5.16772016e-02, -2.72520870e-02, -1.34547869e-02,  8.02323895e-02,
        2.46876453e-02, -1.76462878e-02,  3.55264003e-02,  7.65662851e-02,
       -1.93195656e-02, -1.05262778e-02,  4.67688370e-04, -1.47557056e-02,
        8.71171730e-04,  0.00000000e+00,  1.34168455e-02,  2.02123572e-02,
        1.38968808e-02,  3.57640879e-03, -2.73266363e-02, -9.64010069e-03,
       -2.73774393e-02, -4.08590395e-03, -1.51603097e-02, -2.97011907e-02,
        1.16491645e-01,  1.48724747e-02, -5.25909620e-03,  5.87908978e-03,
       -7.44720566e-04,  2.80639578e-02, -1.15113633e-03, -4.93349148e-03,
       -7.51490023e-03,  0.00000000e+00,  3.30093172e-02, -3.24694050e-02,
       -1.06579328e-02,  1.17115657e-01,  2.12358070e-02, -1.76128421e-03,
        1.73514542e-02,  2.28541527e-02, -1.95437570e-02,  5.10665855e-02,
        6.91423986e-02])
```

In [53]:
```python
#R-squared value of test and train data
from sklearn import metrics
y_train_pred = ridge.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
```

```
0.9031439595017332
```

In [54]:
```
####Best alpha value for Ridge : {'alpha': 0.9}
#Build model with Lasso
lasso = Lasso()

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

model_cv.fit(X_train, y_train)
```

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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
E:\Python\lib\site-packages\sklearn\linear_model\coordinate_descent.py:475: C
onvergenceWarning: Objective did not converge. You might want to increase the
number of iterations. Duality gap: 0.13920341294216598, tolerance: 0.08344794
86660015
  positive)
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:    2.6s finished
```

Out[54]:
```
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=Fals
e),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.
0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 5
0,
                                   100, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_absolute_error', verbose=1)
```

In [55]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[55]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_t
0	0.147509	3.312605e-02	0.001799	3.998281e-04	0.0001	{'alpha': 0.0001}	
1	0.075354	1.395784e-02	0.001599	4.898045e-04	0.001	{'alpha': 0.001}	
2	0.015390	1.958292e-03	0.001799	7.478695e-04	0.01	{'alpha': 0.01}	
3	0.012792	3.865033e-03	0.001999	6.322590e-04	0.05	{'alpha': 0.05}	
4	0.008795	1.165215e-03	0.001399	4.897269e-04	0.1	{'alpha': 0.1}	
5	0.008395	4.888698e-04	0.001599	4.892985e-04	0.2	{'alpha': 0.2}	
6	0.008395	1.019244e-03	0.001599	8.003594e-04	0.3	{'alpha': 0.3}	
7	0.007196	9.786937e-04	0.001599	4.890259e-04	0.4	{'alpha': 0.4}	
8	0.006596	4.897466e-04	0.001599	4.895324e-04	0.5	{'alpha': 0.5}	
9	0.007795	7.482523e-04	0.001998	6.321838e-04	0.6	{'alpha': 0.6}	
10	0.007195	7.479080e-04	0.001998	6.316563e-04	0.7	{'alpha': 0.7}	
11	0.007396	7.995849e-04	0.001798	7.484049e-04	0.8	{'alpha': 0.8}	
12	0.007595	7.997277e-04	0.001799	3.998284e-04	0.9	{'alpha': 0.9}	
13	0.007195	7.475518e-04	0.001798	3.995899e-04	1	{'alpha': 1.0}	
14	0.006996	6.319574e-04	0.001599	4.896873e-04	2	{'alpha': 2.0}	
15	0.007196	7.477679e-04	0.001598	4.894733e-04	3	{'alpha': 3.0}	
16	0.007397	4.887729e-04	0.001598	4.893175e-04	4	{'alpha': 4.0}	
17	0.007396	7.985843e-04	0.001798	3.995449e-04	5	{'alpha': 5.0}	
18	0.006797	7.481499e-04	0.001798	3.995672e-04	6	{'alpha': 6.0}	
19	0.006196	3.990416e-04	0.001798	3.996617e-04	7	{'alpha': 7.0}	
20	0.006796	7.479332e-04	0.001398	4.894731e-04	8	{'alpha': 8.0}	
21	0.006796	3.996373e-04	0.000999	1.168008e-07	9	{'alpha': 9.0}	
22	0.005997	5.309834e-07	0.001198	3.995420e-04	10	{'alpha': 10.0}	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_t
23	0.006196	3.991366e-04	0.001799	3.998779e-04	20	{'alpha': 20}	
24	0.005997	2.780415e-07	0.001198	3.992560e-04	50	{'alpha': 50}	
25	0.006796	3.998050e-04	0.001598	4.891238e-04	100	{'alpha': 100}	
26	0.006597	7.995964e-04	0.001198	3.993280e-04	500	{'alpha': 500}	
27	0.006996	8.941504e-04	0.001399	4.886566e-04	1000	{'alpha': 1000}	

28 rows × 21 columns

In [56]:

```
#R-squared value
model_cv1 = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv1.fit(X_train, y_train)
```

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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
E:\Python\lib\site-packages\sklearn\linear_model\coordinate_descent.py:475: C
onvergenceWarning: Objective did not converge. You might want to increase the
number of iterations. Duality gap: 0.13920341294216598, tolerance: 0.08344794
86660015
  positive)
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:    2.8s finished
```

Out[56]:

```
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=Fals
e),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.
0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 5
0,
                                   100, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='r2', verbose=1)
```

In [57]:
```
cv_results1 = pd.DataFrame(model_cv1.cv_results_)
cv_results1
```

Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_t
0	0.144710	6.498708e-02	0.002399	4.899015e-04	0.0001	{'alpha': 0.0001}	
1	0.071556	1.157906e-02	0.001998	3.504023e-07	0.001	{'alpha': 0.001}	
2	0.013792	1.600295e-03	0.001799	3.997327e-04	0.01	{'alpha': 0.01}	
3	0.010394	1.742979e-03	0.001999	6.324853e-04	0.05	{'alpha': 0.05}	
4	0.011794	1.325800e-03	0.002997	2.463915e-06	0.1	{'alpha': 0.1}	
5	0.011193	7.475394e-04	0.002398	4.894150e-04	0.2	{'alpha': 0.2}	
6	0.011193	3.997335e-04	0.002399	4.895706e-04	0.3	{'alpha': 0.3}	
7	0.008195	1.165280e-03	0.001799	7.479460e-04	0.4	{'alpha': 0.4}	
8	0.008795	1.164560e-03	0.001798	3.994466e-04	0.5	{'alpha': 0.5}	
9	0.009394	1.018552e-03	0.002398	7.992510e-04	0.6	{'alpha': 0.6}	
10	0.008595	4.895901e-04	0.002199	7.479210e-04	0.7	{'alpha': 0.7}	
11	0.010394	7.993107e-04	0.002199	4.002097e-04	0.8	{'alpha': 0.8}	
12	0.010194	9.793940e-04	0.002198	3.998520e-04	0.9	{'alpha': 0.9}	
13	0.006996	6.317313e-04	0.001798	3.994942e-04	1	{'alpha': 1.0}	
14	0.007395	1.019356e-03	0.001599	4.894537e-04	2	{'alpha': 2.0}	
15	0.006596	4.883268e-04	0.001798	3.996376e-04	3	{'alpha': 3.0}	
16	0.007795	9.781292e-04	0.001640	5.290908e-04	4	{'alpha': 4.0}	
17	0.006996	6.318821e-04	0.001598	4.891618e-04	5	{'alpha': 5.0}	
18	0.007595	1.019048e-03	0.001599	4.898237e-04	6	{'alpha': 6.0}	
19	0.006397	7.992745e-04	0.001598	4.896874e-04	7	{'alpha': 7.0}	
20	0.006797	7.470669e-04	0.001598	4.885974e-04	8	{'alpha': 8.0}	
21	0.006196	3.995181e-04	0.001399	4.895705e-04	9	{'alpha': 9.0}	
22	0.006196	3.994704e-04	0.001599	4.894926e-04	10	{'alpha': 10.0}	

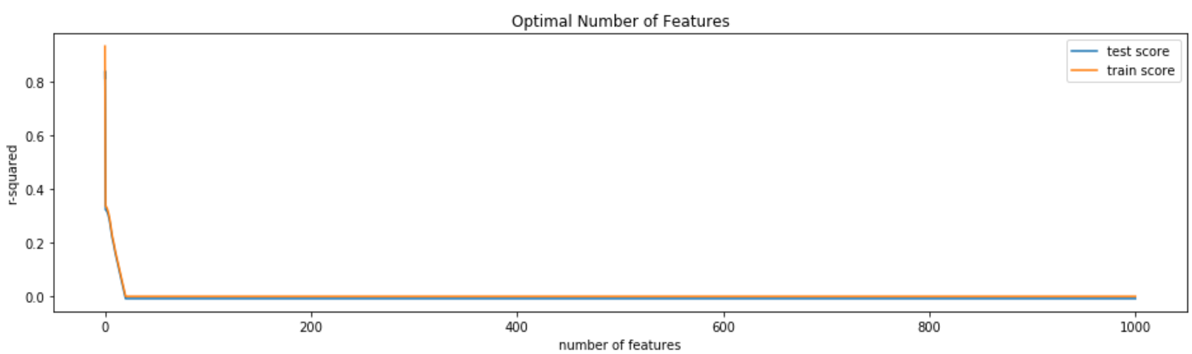
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_t
23	0.006597	7.996321e-04	0.001398	4.893370e-04	20	{'alpha': 20}	
24	0.006596	4.902713e-04	0.001798	3.996856e-04	50	{'alpha': 50}	
25	0.005997	3.371748e-07	0.001598	4.898236e-04	100	{'alpha': 100}	
26	0.007196	7.477295e-04	0.001598	4.901165e-04	500	{'alpha': 500}	
27	0.006196	3.986837e-04	0.001399	4.894343e-04	1000	{'alpha': 1000}	

28 rows × 21 columns

In [58]:

```
# plotting cv results
plt.figure(figsize=(16,4))

plt.plot(cv_results1["param_alpha"], cv_results1["mean_test_score"])
plt.plot(cv_results1["param_alpha"], cv_results1["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper right')
```

Out[58]: `<matplotlib.legend.Legend at 0x1d1e9422f60>`



In [60]:

```
#value of optimum number of parameters
print(model_cv.best_params_)
print(model_cv.best_score_)
```

```
{'alpha': 0.001}
-0.22686016003961607
```

In [ ]:

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