1. Data Understanding and Exploration

Let's first have a look at the dataset and understand the size, attribute names etc.

In [1]:
```
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 import metrics

import os

# hide warnings
import warnings

warnings.filterwarnings('ignore')
```

In [2]:
```
# reading the dataset
house = pd.read_csv("/DOCUMENTS/COLLEGE/CLASSES/EXPERIMENT_NO_3/train.csv")
test = pd.read_csv("/DOCUMENTS/COLLEGE/CLASSES/EXPERIMENT_NO_3/test.csv")
```

In [3]:
```
# head
house.head()
```

Out[3]:

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

5 rows × 81 columns

Data Exploration

In [4]:
```
house.shape
```

Out[4]:
```
(1460, 81)
```

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

Out[5]:
```
(1459, 80)
```

In [6]:
```
# summary of the dataset: 1460 rows, 81 columns
house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [7]:
```
house.describe()  # other atributes of the dataframe
```

Out[7]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.00000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.63972
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.09809
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.00000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.00000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.50000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.25000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.00000

8 rows × 38 columns

In [8]:
```
# all numeric (float and int) variables in the dataset
house_numeric = house.select_dtypes(include=['float64', 'int64'])
house_numeric.head()
```

Out[8]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckSF	Op
0	1	60	65.0	8450	7	5	2003	2003	196.0	706	...	0	
1	2	20	80.0	9600	6	8	1976	1976	0.0	978	...	298	
2	3	60	68.0	11250	7	5	2001	2002	162.0	486	...	0	
3	4	70	60.0	9550	7	5	1915	1970	0.0	216	...	0	
4	5	60	84.0	14260	8	5	2000	2000	350.0	655	...	192	

5 rows × 38 columns

In [9]:
```
house_numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 38 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   LotFrontage    1201 non-null   float64
 3   LotArea        1460 non-null   int64
 4   OverallQual    1460 non-null   int64
 5   OverallCond    1460 non-null   int64
 6   YearBuilt      1460 non-null   int64
 7   YearRemodAdd   1460 non-null   int64
 8   MasVnrArea     1452 non-null   float64
 9   BsmtFinSF1     1460 non-null   int64
 10  BsmtFinSF2     1460 non-null   int64
 11  BsmtUnfSF      1460 non-null   int64
 12  TotalBsmtSF    1460 non-null   int64
 13  1stFlrSF       1460 non-null   int64
 14  2ndFlrSF       1460 non-null   int64
 15  LowQualFinSF   1460 non-null   int64
 16  GrLivArea      1460 non-null   int64
 17  BsmtFullBath   1460 non-null   int64
 18  BsmtHalfBath   1460 non-null   int64
 19  FullBath       1460 non-null   int64
 20  HalfBath       1460 non-null   int64
 21  BedroomAbvGr   1460 non-null   int64
 22  KitchenAbvGr   1460 non-null   int64
 23  TotRmsAbvGrd   1460 non-null   int64
 24  Fireplaces     1460 non-null   int64
 25  GarageYrBlt    1379 non-null   float64
 26  GarageCars     1460 non-null   int64
 27  GarageArea     1460 non-null   int64
 28  WoodDeckSF     1460 non-null   int64
 29  OpenPorchSF    1460 non-null   int64
 30  EnclosedPorch  1460 non-null   int64
 31  3SsnPorch      1460 non-null   int64
 32  ScreenPorch    1460 non-null   int64
 33  PoolArea       1460 non-null   int64
 34  MiscVal        1460 non-null   int64
 35  MoSold         1460 non-null   int64
 36  YrSold         1460 non-null   int64
 37  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35)
memory usage: 433.6 KB
```

Here, although some variables are numeric (int), we'd rather treat them as categorical since they have discrete values.

In [10]:
```
# dropping the columns we want to treat as categorical variables
house_numeric = house_numeric.drop([
    'MSSubClass', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
    'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
    'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars',
    'MoSold', 'YrSold'
],
                                    axis=1)
house_numeric.head()
```

Out[10]:

	Id	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	...	GrLivArea	GarageArea	V
0	1	65.0	8450	196.0	706	0	150	856	856	854	...	1710	548	
1	2	80.0	9600	0.0	978	0	284	1262	1262	0	...	1262	460	
2	3	68.0	11250	162.0	486	0	434	920	920	866	...	1786	608	
3	4	60.0	9550	0.0	216	0	540	756	961	756	...	1717	642	
4	5	84.0	14260	350.0	655	0	490	1145	1145	1053	...	2198	836	

5 rows × 21 columns

Outlier Treatment

In [11]:
```
house_numeric.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

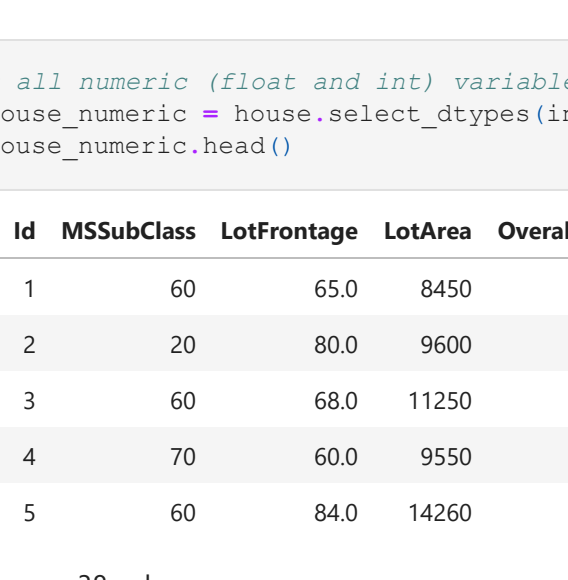
Out[11]:

	Id	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	...
count	1460.000000	1201.000000	1460.000000	1452.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	...
mean	730.500000	70.049958	10516.828082	103.685262	443.639726	46.549315	567.240411	1057.429452	1162.626712	346.992466	...
std	421.610009	24.284752	9981.264932	181.066207	456.098091	161.319273	441.866955	438.705324	386.587738	436.528436	...
min	1.000000	21.000000	1300.000000	0.000000	0.000000	0.000000	0.000000	0.000000	334.000000	0.000000	...
25%	365.750000	59.000000	7553.500000	0.000000	0.000000	0.000000	223.000000	795.750000	882.000000	0.000000	...
50%	730.500000	69.000000	9478.500000	0.000000	383.500000	0.000000	477.500000	991.500000	1087.000000	0.000000	...
75%	1095.250000	80.000000	11601.500000	166.000000	712.250000	0.000000	808.000000	1298.250000	1391.250000	728.000000	...
90%	1314.100000	96.000000	14381.700000	335.000000	1065.500000	117.200000	1232.000000	1602.200000	1680.000000	954.200000	...
95%	1387.050000	107.000000	17401.150000	456.000000	1274.000000	396.200000	1468.000000	1753.000000	1831.250000	1141.050000	...
99%	1445.410000	141.000000	37567.640000	791.920000	1572.410000	830.380000	1797.050000	2155.050000	2219.460000	1418.920000	...
max	1460.000000	313.000000	215245.000000	1600.000000	5644.000000	1474.000000	2336.000000	6110.000000	4692.000000	2065.000000	...

11 rows × 21 columns

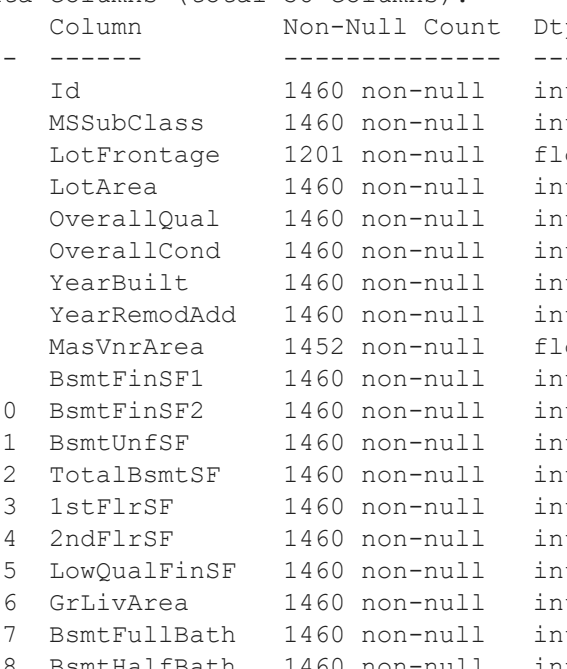
In [12]:
```
# outlier treatment
plt.boxplot(house['PoolArea'])
Q1 = house['PoolArea'].quantile(0.1)
Q3 = house['PoolArea'].quantile(0.9)
IQR = Q3 - Q1
house = house[(house['PoolArea'] >= Q1 - 1.5 * IQR)
              & (house['PoolArea'] <= Q3 + 1.5 * IQR)]
house.shape
```

Out[12]:
```
(1453, 81)
```



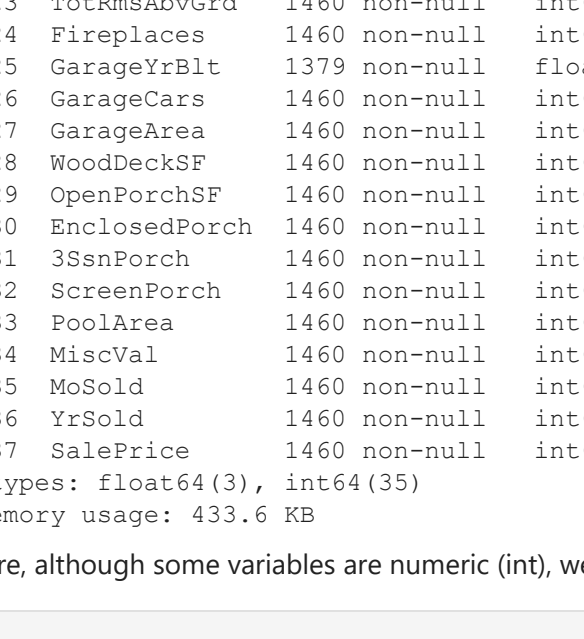
In [13]:
```
# outlier treatment
plt.boxplot(house['MiscVal'])
Q1 = house['MiscVal'].quantile(0.1)
Q3 = house['MiscVal'].quantile(0.9)
IQR = Q3 - Q1
house = house[(house['MiscVal'] >= Q1 - 1.5 * IQR)
              & (house['MiscVal'] <= Q3 + 1.5 * IQR)]
house.shape
```

Out[13]:
```
(1402, 81)
```



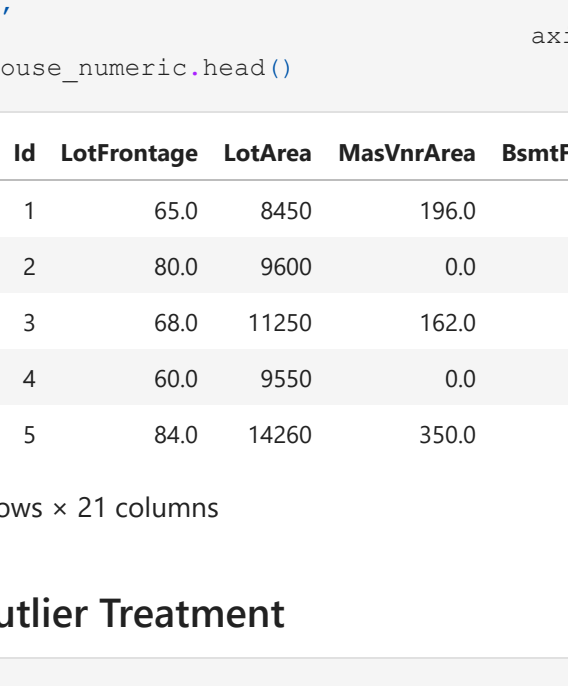
In [14]:
```
# outlier treatment
plt.boxplot(house['ScreenPorch'])
Q1 = house['ScreenPorch'].quantile(0.1)
Q3 = house['ScreenPorch'].quantile(0.9)
IQR = Q3 - Q1
house = house[(house['ScreenPorch'] >= Q1 - 1.5 * IQR)
              & (house['ScreenPorch'] <= Q3 + 1.5 * IQR)]
house.shape
```

Out[14]:
```
(1291, 81)
```



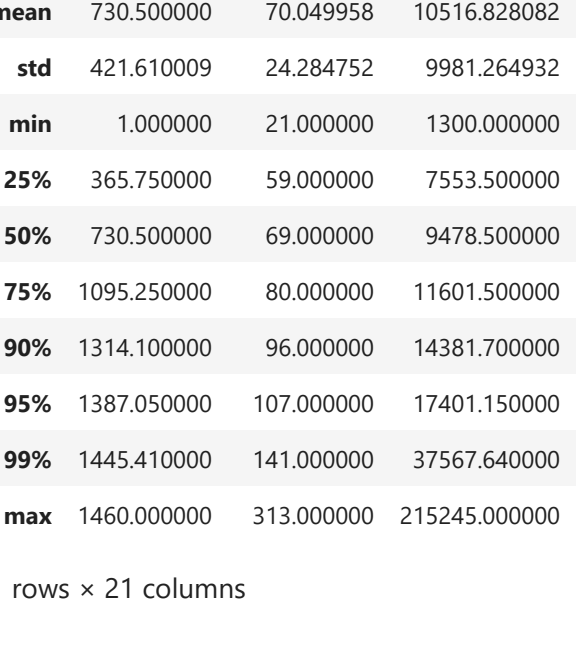
In [15]:
```
# outlier treatment
plt.boxplot(house['LotArea'])
Q1 = house['LotArea'].quantile(0.1)
Q3 = house['LotArea'].quantile(0.9)
IQR = Q3 - Q1
house = house[(house['LotArea'] >= Q1 - 1.5 * IQR)
              & (house['LotArea'] <= Q3 + 1.5 * IQR)]
house.shape
```

Out[15]:
```
(1274, 81)
```



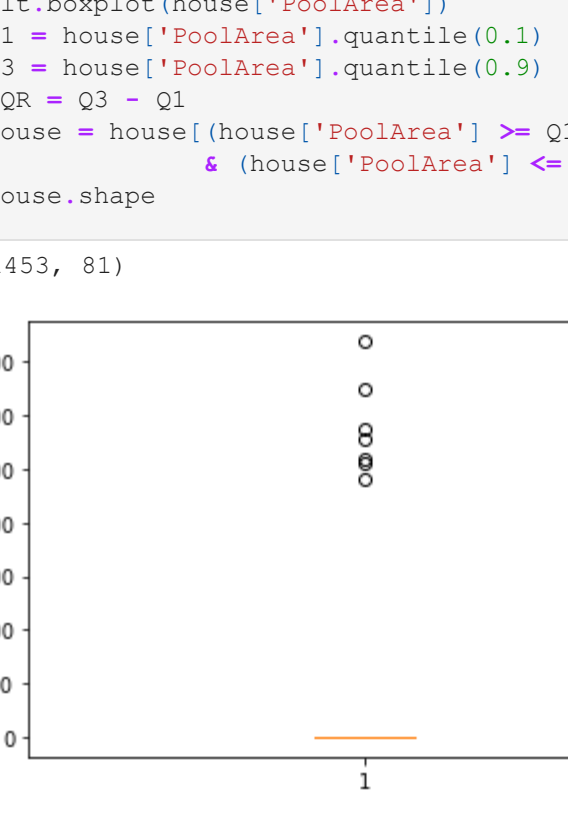
In [16]:
```
# outlier treatment
plt.boxplot(house['MasVnrArea'])
Q1 = house['MasVnrArea'].quantile(0.1)
Q3 = house['MasVnrArea'].quantile(0.9)
IQR = Q3 - Q1
house = house[(house['MasVnrArea'] >= Q1 - 1.5 * IQR)
              & (house['MasVnrArea'] <= Q3 + 1.5 * IQR)]
house.shape
```

Out[16]:
```
(1255, 81)
```



In [17]:
```
# outlier treatment
plt.boxplot(house['SalePrice'])
Q1 = house['SalePrice'].quantile(0.1)
Q3 = house['SalePrice'].quantile(0.9)
IQR = Q3 - Q1
house = house[(house['SalePrice'] >= Q1 - 1.5 * IQR)
              & (house['SalePrice'] <= Q3 + 1.5 * IQR)]
house.shape
```

Out[17]:
```
(1253, 81)
```



Checking the Correlation between the variables

In [18]:
```
# correlation matrix
cor = house_numeric.corr()
cor
```

Out[18]:

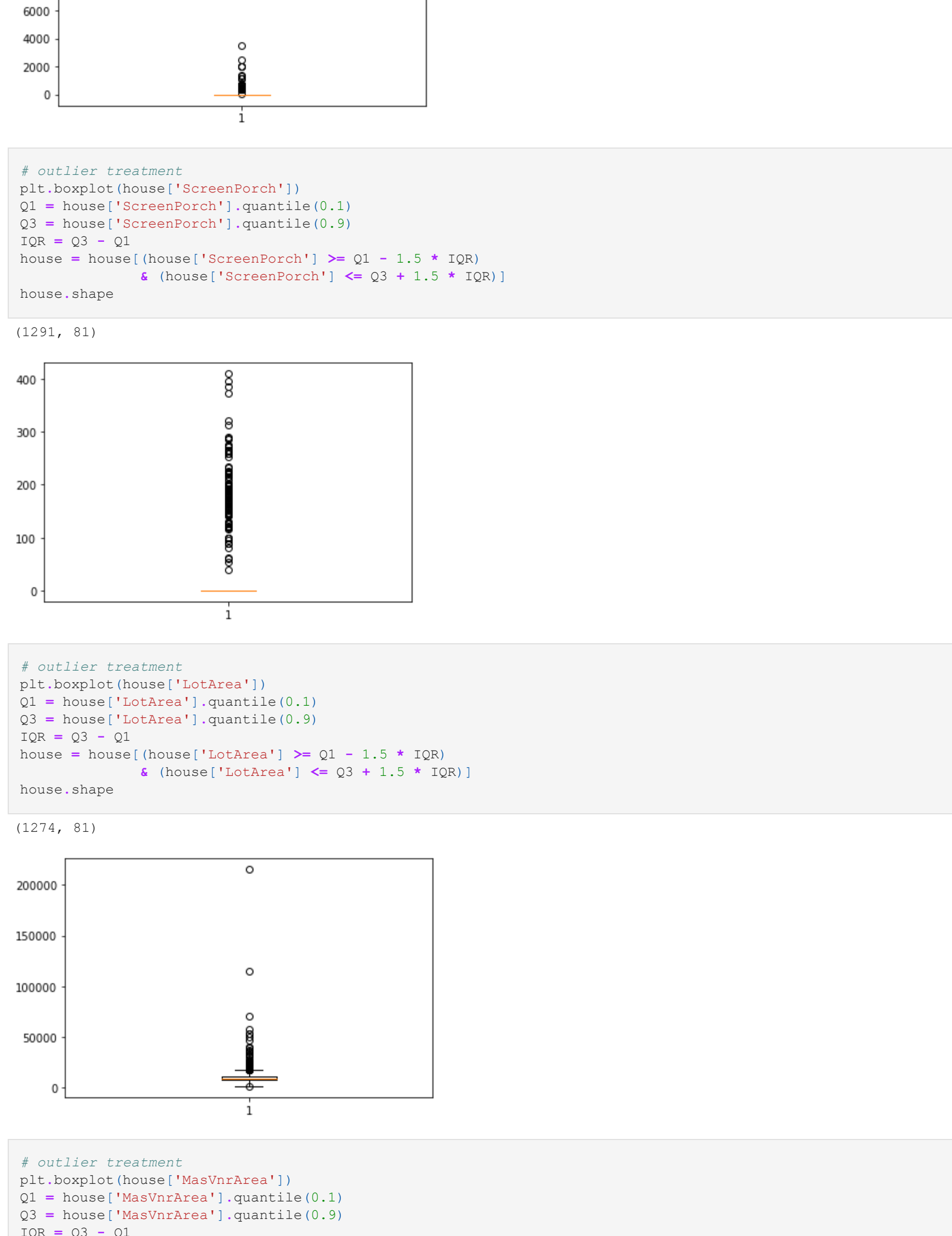
	Id	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	...
Id	1.000000	-0.010601	-0.033226	-0.050298	-0.005024	-0.005968	-0.007940	-0.015415	0.010496	0.005590	...
LotFrontage	-0.010601	1.000000	0.426095	0.193458	0.233633	0.049900	0.132644	0.392075	0.457181	0.080177	...
LotArea	-0.033226	0.426095	1.000000	0.104160	0.214103	0.111170	-0.002618	0.260833	0.299475	0.050986	...
MasVnrArea	-0.050298	0.193458	0.104160	1.000000	0.264736	-0.072319	0.114442	0.363936	0.344501	0.174561	...
BsmtFinSF1	-0.005024	0.233633	0.214103	0.264736	1.000000	-0.050117	-0.495251	0.522396	0.445863	-0.137079	...
BsmtFinSF2	-0.005968	0.049900	0.111170	-0.072319	-0.050117	1.000000	-0.209294	0.104810	0.097117	-0.099260	...
BsmtUnfSF	-0.007940	0.132644	-0.002618	0.114442	-0.495251	-0.209294	1.000000	0.415360	0.317987	0.004469	...
TotalBsmtSF	-0.015415	0.392075	0.260833	0.363936	0.522396	0.104810	0.415360	1.000000	0.819530	-0.174512	...
1stFlrSF	0.010496	0.457181	0.299475	0.344501	0.445863	0.097117	0.317987	0.819530	1.000000	-0.202646	...
2ndFlrSF	0.005590	0.080177	0.050986	0.174561	-0.137079	-0.099260	0.004469	-0.174512	-0.202646	1.000000	...
LowQualFinSF	-0.044230	0.038469	0.004779	-0.069071	-0.064503	0.014807	0.028167	-0.033245	-0.014241	0.063353	...
GrLivArea	0.008273	0.402797	0.263116	0.390857	0.208171	-0.009640	0.240257	0.454868	0.566024	0.687501	...
GarageArea	0.017634	0.344997	0.180403	0.373066	0.296970	-0.018227	0.183303	0.486665	0.489782	0.138347	...
WoodDeckSF	-0.029643	0.088521	0.171698	0.159718	0.204306	0.067898	-0.005316	0.232019	0.235459	0.092165	...
OpenPorchSF	-0.000477	0.151972	0.084774	0.125703	0.111761	0.003093	0.129005	0.247264	0.211671	0.208026	...
EnclosedPorch	0.002889	0.010700	-0.018340	-0.110204	-0.102303	0.036543	-0.002538	-0.095478	-0.065292	0.061989	...
3SsnPorch	-0.046635	0.070029	0.020423	0.018796	0.026451	-0.029993	0.020764	0.037384	0.056104	-0.024358	...
ScreenPorch	0.001330	0.041383	0.043160	0.061466	0.062021	0.088871	-0.012579	0.084489	0.088758	0.040606	...
PoolArea	0.057044	0.206167	0.077672	0.011723	0.140491	0.041709	-0.035092	0.126053	0.131525	0.081487	...
MiscVal	-0.006242	0.003368	0.038068	-0.029815	0.003571	0.004940	-0.023837	-0.018479	-0.021096	0.016197	...
SalePrice	-0.021917	0.351799	0.263843	0.477493	0.386420	-0.011378	0.214479	0.613581	0.605852	0.319334	...

21 rows × 21 columns

In [19]:
```
# plotting correlations on a heatmap

# figure size
plt.figure(figsize=(18, 10))

# heatmap
sns.heatmap(cor, annot=True)
plt.show()
```



2. Data Cleaning

Let's now conduct some data cleaning steps.

We've seen that there are some missing values in the dataset. We've also seen that variables are in the correct format, except some variables with distinct values, which should rather be categorical variables (so that dummy variable are created for the categories).

In [20]:
```
# variable formats
house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1253 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1253 non-null   int64
 1   MSSubClass     1253 non-null   int64
 2   MSZoning       1253 non-null   object
 3   LotFrontage    1040 non-null   float64
 4   LotArea        1253 non-null   int64
 5   Street         1253 non-null   object
 6   Alley          82 non-null     object
 7   LotShape       1253 non-null   object
 8   LandContour    1253 non-null   object
 9   Utilities      1253 non-null   object
 10  LotConfig      1253 non-null   object
 11  LandSlope      1253 non-null   object
 12  Neighborhood   1253 non-null   object
 13  Condition1     1253 non-null   object
 14  Condition2     1253 non-null   object
 15  BldgType       1253 non-null   object
 16  HouseStyle     1253 non-null   object
 17  OverallQual    1253 non-null   int64
 18  OverallCond    1253 non-null   int64
 19  YearBuilt      1253 non-null   int64
 20  YearRemodAdd   1253 non-null   int64
 21  RoofStyle      1253 non-null   object
 22  RoofMatl       1253 non-null   object
 23  Exterior1st    1253 non-null   object
 24  Exterior2nd    1253 non-null   object
 25  MasVnrType     1253 non-null   object
 26  MasVnrArea     1253 non-null   float64
 27  ExterQual      1253 non-null   object
 28  ExterCond      1253 non-null   object
 29  Foundation     1253 non-null   object
 30  BsmtQual       1219 non-null   object
 31  BsmtCond       1219 non-null   object
 32  BsmtExposure   1218 non-null   object
 33  BsmtFinType1   1219 non-null   object
 34  BsmtFinSF1     1253 non-null   int64
 35  BsmtFinType2   1218 non-null   object
 36  BsmtFinSF2     1253 non-null   int64
 37  BsmtUnfSF      1253 non-null   int64
 38  TotalBsmtSF    1253 non-null   int64
 39  Heating        1253 non-null   object
 40  HeatingQC      1253 non-null   object
 41  CentralAir     1253 non-null   object
 42  Electrical     1252 non-null   object
 43  1stFlrSF       1253 non-null   int64
 44  2ndFlrSF       1253 non-null   int64
 45  LowQualFinSF   1253 non-null   int64
 46  GrLivArea      1253 non-null   int64
 47  BsmtFullBath   1253 non-null   int64
 48  BsmtHalfBath   1253 non-null   int64
 49  FullBath       1253 non-null   int64
 50  HalfBath       1253 non-null   int64
 51  BedroomAbvGr   1253 non-null   int64
 52  KitchenAbvGr   1253 non-null   int64
 53  KitchenQual    1253 non-null   object
 54  TotRmsAbvGrd   1253 non-null   int64
 55  Functional     1253 non-null   object
 56  Fireplaces     1253 non-null   int64
 57  FireplaceQu    621 non-null    object
 58  GarageType     1175 non-null   object
 59  GarageYrBlt    1175 non-null   float64
 60  GarageFinish   1175 non-null   object
 61  GarageCars     1253 non-null   int64
 62  GarageArea     1253 non-null   int64
 63  GarageQual     1175 non-null   object
 64  GarageCond     1175 non-null   object
 65  PavedDrive     1253 non-null   object
 66  WoodDeckSF     1253 non-null   int64
 67  OpenPorchSF    1253 non-null   int64
 68  EnclosedPorch  1253 non-null   int64
 69  3SsnPorch      1253 non-null   int64
 70  ScreenPorch    1253 non-null   int64
 71  PoolArea       1253 non-null   int64
 72  PoolQC         0 non-null      object
 73  Fence          232 non-null    object
 74  MiscFeature    2 non-null      object
 75  MiscVal        1253 non-null   int64
 76  MoSold         1253 non-null   int64
 77  YrSold         1253 non-null   int64
 78  SaleType       1253 non-null   object
 79  SaleCondition  1253 non-null   object
 80  SalePrice      1253 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 802.7+ KB
```

In [21]:
```
house.isnull().sum()  # checking the number of null values in the dataset
```

Out[21]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      213
LotArea            0
                ...
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

In [22]:
```
# Checking the percentage of missing values
round(100 * (house.isnull().sum() / len(house.index)), 2)
```

Out[22]:
```
Id               0.0
MSSubClass       0.0
MSZoning         0.0
LotFrontage     17.0
LotArea          0.0
                ...
MoSold           0.0
YrSold           0.0
SaleType         0.0
SaleCondition    0.0
SalePrice        0.0
Length: 81, dtype: float64
```

Null value treatment

Instead of dropping the null values which will result in a data loss, we will impute the null values according to the domain understanding and the data dictionary provided with the data.

In [23]:
```
house.shape
```

Out[23]:
```
(1253, 81)
```

In [24]:
```
house = pd.concat((house, test))
```

In [25]:
```
# NA in Alley column means No Alley, so we will replace NA by it.
house['Alley'].fillna('No Alley', inplace=True)
```

In [26]:
```
house['MasVnrType'].fillna('None', inplace=True)
```

In [27]:
```
# NA in FireplaceQu column means No Fireplace, so we will replace NA by it.
house['FireplaceQu'].fillna('No Fireplace', inplace=True)
```

In [28]:
```
# NA in PoolQC column means No Pool, so we will replace NA by it.
house['PoolQC'].fillna('No Pool', inplace=True)
```

In [29]:
```
# NA in Fence column means No Fence, so we will replace NA by it.
house['Fence'].fillna('No Fence', inplace=True)
```

In [30]:
```
house['MasVnrArea'].fillna(0, inplace=True)
```

In [31]:
```
house['LotFrontage'].fillna(0, inplace=True)
```

In [32]:
```
# NA in GarageType, GarageFinish, GarageQual, GarageCond columns mean No Garage, so we will replace NA by it.

house['GarageType'].fillna('No Garage', inplace=True)
house['GarageFinish'].fillna('No Garage', inplace=True)
house['GarageQual'].fillna('No Garage', inplace=True)
house['GarageCond'].fillna('No Garage', inplace=True)
```

In [33]:
```
# MiscFeature column has almost 99% null values so we will drop it
house = house.drop('MiscFeature', axis=1)
```

In [34]:
```
house.isnull().sum()
```

Out[34]:
```
Id                  0
MSSubClass          0
MSZoning            4
LotFrontage         0
LotArea             0
                 ...
MoSold              0
YrSold              0
SaleType            1
SaleCondition       0
SalePrice        1459
Length: 80, dtype: int64
```

In [35]:
```
# converting year to number of years
house['YearBuilt'] = 2019 - house['YearBuilt']
house['YearRemodAdd'] = 2019 - house['YearRemodAdd']
house['GarageYrBlt'] = 2019 - house['GarageYrBlt']
house['YrSold'] = 2019 - house['YrSold']
```

In [36]:
```
# converting from int type to object to treat the variables as categorical variables
house['MSSubClass'] = house['MSSubClass'].astype('object')
house['OverallQual'] = house['OverallQual'].astype('object')
house['OverallCond'] = house['OverallCond'].astype('object')
house['BsmtFullBath'] = house['BsmtFullBath'].astype('object')
house['BsmtHalfBath'] = house['BsmtHalfBath'].astype('object')
house['FullBath'] = house['FullBath'].astype('object')
house['HalfBath'] = house['HalfBath'].astype('object')
house['BedroomAbvGr'] = house['BedroomAbvGr'].astype('object')
house['KitchenAbvGr'] = house['KitchenAbvGr'].astype('object')
house['TotRmsAbvGrd'] = house['TotRmsAbvGrd'].astype('object')
house['Fireplaces'] = house['Fireplaces'].astype('object')
house['GarageCars'] = house['GarageCars'].astype('object')
```

In [37]:
```
house.shape
```

Out[37]:
```
(2712, 80)
```

In [38]:
```
final = house
```

Dummy Variables

In [39]:
```
# List of variables to map

varlist1 = ['Street']

# Defining the map function


def binary_map(x):
    return x.map({'Pave': 1, "Grvl": 0})


# Applying the function to the Lead list
final[varlist1] = final[varlist1].apply(binary_map)
```

In [40]:
```
# List of variables to map

varlist2 = ['Utilities']

# Defining the map function


def binary_map(x):
    return x.map({'AllPub': 1, "NoSeWa": 0})


# Applying the function to the Lead list
final[varlist2] = final[varlist2].apply(binary_map)
```

In [41]:
```
# List of variables to map

varlist3 = ['CentralAir']

# Defining the map function


def binary_map(x):
    return x.map({'Y': 1, "N": 0})


# Applying the function to the Lead list
final[varlist3] = final[varlist3].apply(binary_map)
```

3. Data Preparation

Data Preparation

Let's now prepare the data and build the model.

In [42]:
```
# split into X and y
X = final.drop(['Id'], axis=1)
```

In [43]:
```
# creating dummy variables for categorical variables

# subset all categorical variables
house_categorical = X.select_dtypes(include=['object'])
house_categorical.head()
```

Out[43]:

	MSSubClass	MSZoning	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	...	GarageType	Gar
0	60	RL	No Alley	Reg	Lvl	Inside	Gtl	CollgCr	Norm	Norm	...	Attchd	
1	20	RL	No Alley	Reg	Lvl	FR2	Gtl	Veenker	Feedr	Norm	...	Attchd	
2	60	RL	No Alley	IR1	Lvl	Inside	Gtl	CollgCr	Norm	Norm	...	Attchd	
3	70	RL	No Alley	IR1	Lvl	Corner	Gtl	Crawfor	Norm	Norm	...	Detchd	
4	60	RL	No Alley	IR1	Lvl	FR2	Gtl	NoRidge	Norm	Norm	...	Attchd	

5 rows × 51 columns

In [44]:
```
# convert into dummies
house_dummies = pd.get_dummies(house_categorical, drop_first=True)
house_dummies.head()
```

Out[44]:

	MSSubClass_30	MSSubClass_40	MSSubClass_45	MSSubClass_50	MSSubClass_60	MSSubClass_70	MSSubClass_75	MSSubClass_80	MSSubCla
0	0	0	0	0	1	0	0	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	0	1	0	0	0	
3	0	0	0	0	0	1	0	0	
4	0	0	0	0	1	0	0	0	

5 rows × 281 columns

In [45]:
```
# drop categorical variables
final = final.drop(list(house_categorical.columns), axis=1)
```

In [46]:
```
# concat dummy variables with X
final = pd.concat([final, house_dummies], axis=1)
```

In [47]:
```
final.shape
```

Out[47]: (2712, 310)

In [48]:
```
test = final.tail(1459)
```

In [49]:
```
test.shape
```

Out[49]: (1459, 310)

In [50]:
```
X = final.head(1253)
y = np.log(X.SalePrice)
X = X.drop("SalePrice", 1)  # take out the target variable
```

In [51]:
```
test = test.fillna(test.interpolate())
```

In [52]:
```
X = X.fillna(X.interpolate())
```

In [53]:
```
test = test.drop("SalePrice", 1)  # take out the target variable
```

In [54]:
```
# scaling the features

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
```

Out[54]: StandardScaler()

In [55]:
```
# scaling the features

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(test)
```

Out[55]: StandardScaler()

In [56]:
```
# split into train and test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    train_size=0.7,
                                                    test_size=0.3,
                                                    random_state=100)
```

4. Model Building and Evaluation

Ridge and Lasso Regression

Let's now try predicting car prices, a dataset used in simple linear regression, to perform ridge and lasso regression.

Ridge Regression

In [57]:
```
# list of alphas to tune
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)
```

Out[57]:
```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
GridSearchCV(cv=5, estimator=Ridge(),
             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, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [58]:
```
# checking the value of optimum number of parameters
print(model_cv.best_params_)
print(model_cv.best_score_)
```

```
{'alpha': 4.0}
-0.07490031919810557
```

In [59]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results = cv_results[cv_results['param_alpha'] <= 1000]
cv_results
```

Out[59]:

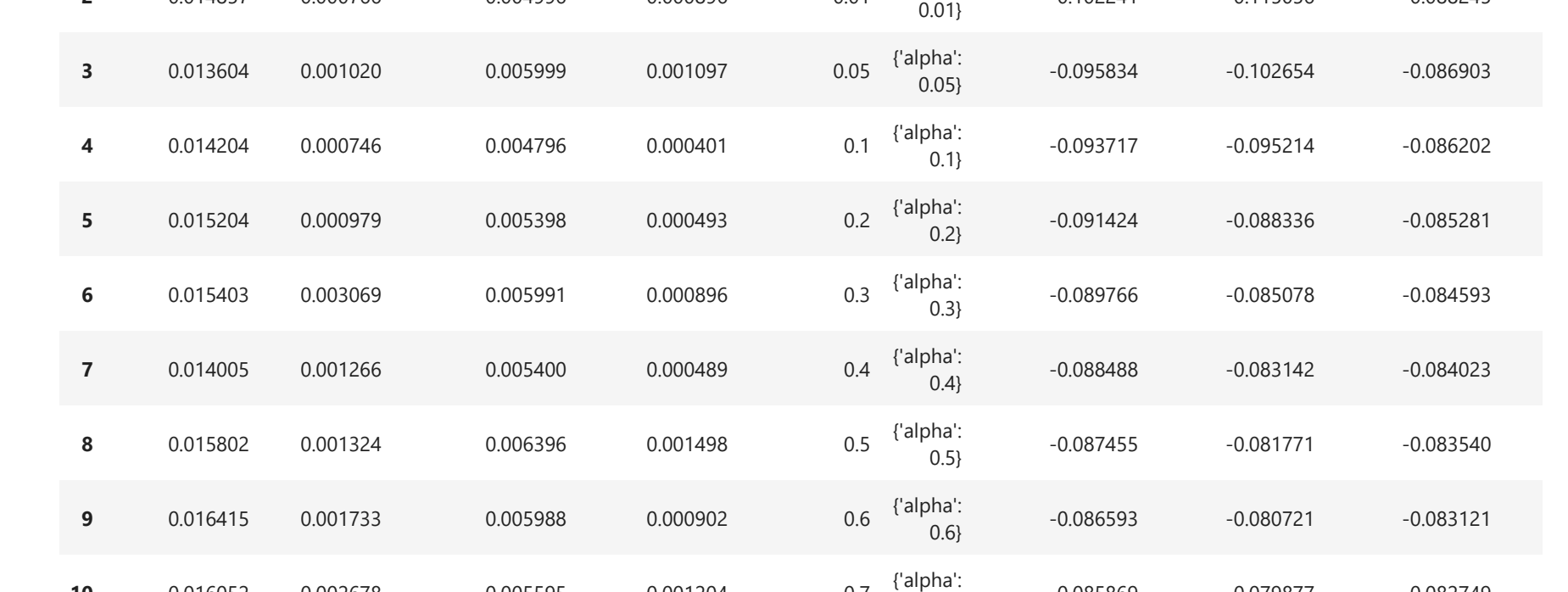
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	spli
0	0.034397	0.013811	0.010199	0.002640	0.0001	{'alpha': 0.0001}	-0.106722	-0.119848	-0.089229	
1	0.045404	0.023622	0.012597	0.005855	0.001	{'alpha': 0.001}	-0.106177	-0.119162	-0.089101	
2	0.014857	0.000766	0.004996	0.000896	0.01	{'alpha': 0.01}	-0.102241	-0.115056	-0.088245	
3	0.013604	0.001020	0.005994	0.001097	0.05	{'alpha': 0.05}	-0.095834	-0.102654	-0.086903	
4	0.014204	0.000746	0.004796	0.000401	0.1	{'alpha': 0.1}	-0.093717	-0.095214	-0.086202	
5	0.015204	0.000979	0.005398	0.000493	0.2	{'alpha': 0.2}	-0.091424	-0.088336	-0.085281	
6	0.015403	0.003069	0.005991	0.000896	0.3	{'alpha': 0.3}	-0.089766	-0.085078	-0.084593	
7	0.014005	0.001266	0.005400	0.000489	0.4	{'alpha': 0.4}	-0.088488	-0.083142	-0.084023	
8	0.015802	0.001324	0.006396	0.001498	0.5	{'alpha': 0.5}	-0.087455	-0.081771	-0.083540	
9	0.016415	0.001733	0.005988	0.000902	0.6	{'alpha': 0.6}	-0.086593	-0.080721	-0.083121	
10	0.016052	0.002678	0.005595	0.001204	0.7	{'alpha': 0.7}	-0.085869	-0.079877	-0.082749	
11	0.013791	0.001173	0.004992	0.000007	0.8	{'alpha': 0.8}	-0.085252	-0.079189	-0.082432	
12	0.014609	0.000488	0.004996	0.000636	0.9	{'alpha': 0.9}	-0.084728	-0.078621	-0.082149	
13	0.015002	0.003098	0.006197	0.002995	1.0	{'alpha': 1.0}	-0.084260	-0.078135	-0.081898	
14	0.013008	0.001096	0.004993	0.000003	2.0	{'alpha': 2.0}	-0.081425	-0.075830	-0.080821	
15	0.015007	0.000888	0.004594	0.000494	3.0	{'alpha': 3.0}	-0.080188	-0.075488	-0.080255	
16	0.014403	0.000488	0.005599	0.000795	4.0	{'alpha': 4.0}	-0.079646	-0.075591	-0.080057	
17	0.015086	0.006294	0.007448	0.007025	5.0	{'alpha': 5.0}	-0.079570	-0.076060	-0.080177	
18	0.014895	0.006477	0.004125	0.006067	6.0	{'alpha': 6.0}	-0.079679	-0.076558	-0.080443	
19	0.014200	0.000744	0.005391	0.000798	7.0	{'alpha': 7.0}	-0.079912	-0.077004	-0.080709	
20	0.015307	0.000648	0.004123	0.006067	8.0	{'alpha': 8.0}	-0.080222	-0.077483	-0.080931	
21	0.021437	0.006374	0.008250	0.006324	9.0	{'alpha': 9.0}	-0.080511	-0.077993	-0.081135	
22	0.013689	0.006280	0.008052	0.006409	10.0	{'alpha': 10.0}	-0.080860	-0.078451	-0.081319	
23	0.013074	0.003919	0.008266	0.008791	20	{'alpha': 20}	-0.083678	-0.081646	-0.082790	
24	0.012501	0.006250	0.006248	0.007652	50	{'alpha': 50}	-0.088107	-0.086251	-0.086715	
25	0.014897	0.000751	0.005031	0.003210	100	{'alpha': 100}	-0.091797	-0.090109	-0.090689	
26	0.011654	0.005912	0.005321	0.005715	500	{'alpha': 500}	-0.099158	-0.097570	-0.099983	
27	0.012500	0.006250	0.006250	0.007655	1000	{'alpha': 1000}	-0.101533	-0.100637	-0.103563	

28 rows × 21 columns

In [60]:
```
# 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()
```



from the above graph and the best*param* score we got optimum lambda to be 4

In [61]:
```
alpha = 4
ridge = Ridge(alpha=alpha)

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

Out[61]:
```
array([-1.08442335e-05, -1.29476419e-04,  8.64982823e-04, -2.08529167e-02,
        0.00000000e+00, -2.80948988e-03, -1.17578887e-03,  2.14798318e-05,
        5.96453070e-03,  6.69155956e-03,  1.28352002e-05,  1.03621466e-04,
        3.77558156e-02,  7.70920820e-05,  1.14943376e-04, -8.32110397e-06,
        1.89743975e-03,  5.25785534e-05,  1.39361439e-04,  1.18398461e-06,
        2.24007508e-04,  1.19009185e-04,  2.24073414e-04,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00, -1.06780542e-02,  3.26827206e-03,
       -1.15084205e-02,  9.11182461e-03,  2.20253648e-02,  5.56730762e-03,
       -1.39914305e-02,  8.85513943e-02,  2.26983416e-02, -2.19904813e-02,
       -1.56052054e-02, -1.72139518e-02,  2.21571433e-03,  0.00000000e+00,
       -4.16049885e-02, -1.65480922e-02,  8.60619766e-03,  6.97223400e-03,
        1.14903868e-02,  2.89003386e-02,  3.36333979e-03,  2.09400529e-02,
        3.67571189e-02, -1.14787143e-02, -4.86678885e-02,  5.43167335e-03,
       -1.61830286e-02, -8.33988590e-02,  1.06142471e-02,  2.85968505e-02,
       -4.26612994e-02, -2.84381465e-02, -8.27957009e-03, -9.31114608e-03,
        1.72350306e-02,  0.00000000e+00,  2.23760674e-03,  1.67604452e-03,
        2.49955100e-02,  2.29799037e-04,  9.47299308e-02, -5.67634698e-02,
       -9.12295428e-02, -3.57704143e-02, -8.37498842e-02, -2.83723813e-02,
       -1.71181647e-03,  4.52079110e-02, -1.84444016e-02,  1.26475666e-02,
        2.25575629e-02, -6.90260229e-02,  1.03482114e-02,  1.34366653e-02,
       -8.82611219e-03,  3.22979967e-02,  8.60616138e-02, -1.00033671e-02,
        1.32202974e-02,  1.88685476e-02,  3.24288196e-02,  9.67311688e-03,
        4.18914911e-02, -4.32915731e-02,  1.15782954e-02, -1.97085989e-02,
        1.88848318e-02, -1.31359555e-02,  2.48534670e-03,  2.71453080e-02,
        0.00000000e+00,  0.00000000e+00,  2.77856202e-03, -1.03983932e-02,
       -1.72135018e-02,  2.08538444e-02, -1.87975777e-02, -6.57111710e-04,
       -1.98464441e-02, -1.96353742e-02,  2.56672514e-02, -1.94984519e-02,
       -1.08889174e-02, -2.52547527e-02, -8.21099818e-02, -3.86749117e-02,
       -2.98545947e-02,  3.64213701e-03,  4.78782982e-03,  1.69379088e-02,
        5.70423153e-02,  9.36461787e-02,  2.51420744e-02, -4.29178245e-02,
       -1.45908602e-02,  5.79956335e-02, -1.69899221e-02,  9.93561517e-03,
        6.26023202e-02,  8.75062731e-02,  1.01767773e-01, -4.04407235e-02,
        1.46112849e-02, -2.99015773e-04,  8.19835436e-04,  8.95359383e-03,
        8.16943679e-03,  0.00000000e+00,  2.34108245e-02, -1.84368567e-02,
       -2.64502394e-02, -8.82135915e-04, -4.59655144e-02,  7.85870949e-02,
       -1.07858837e-02, -1.83011610e-02, -1.92626546e-02,  8.63620055e-03,
        8.89461024e-02, -1.61273837e-02, -1.28551456e-02,  4.63704143e-02,
       -6.55763323e-05, -2.07590637e-02,  1.61924500e-02,  7.20786747e-03,
       -1.21713443e-02, -1.83843570e-02, -1.08758837e-02,  1.22337742e-02,
       -2.62196655e-03, -1.17106558e-02,  3.98070148e-03,  2.36498832e-02,
        1.50055542e-02, -1.38412050e-02,  5.03236415e-03,  2.39422728e-02,
        1.65444518e-03,  1.33997229e-02,  5.30744710e-03,  3.63965086e-02,
       -3.26995874e-02, -2.48291843e-02, -3.16336836e-02, -2.45713786e-02,
       -8.70275387e-03, -4.35980499e-03, -4.61248620e-03, -1.25429978e-02,
        2.29521708e-03, -7.47226128e-03,  2.56453322e-03, -4.78730470e-02,
        2.37984926e-03,  4.32159498e-03, -5.51788905e-02,  3.96118781e-02,
       -2.56676639e-03,  3.27929108e-02,  5.32434388e-02, -8.92678946e-03,
       -5.12629774e-03,  2.63511141e-02,  2.97978082e-02, -3.07409353e-02,
       -1.66498067e-03,  2.59956564e-03,  1.40859413e-02, -5.40874869e-04,
        3.41503768e-03,  1.96861883e-03,  1.91397395e-02,  1.49087825e-02,
        7.35580650e-03,  0.00000000e+00,  4.04614423e-04,  2.43324246e-02,
       -3.05363411e-02,  1.25325533e-02,  4.41771211e-03, -2.66182013e-02,
        1.36955409e-02, -5.06037562e-03,  2.71246639e-02, -5.19063487e-03,
       [illegible]  2.16044153e-03,
       [illegible]  5.23782890e-02,
       [illegible] -5.60063378e-02,
       [illegible] -1.82613244e-02,
       [illegible] -2.49813105e-02,
       [illegible] -4.27235797e-02,
       [illegible]  2.20829674e-02,
       [illegible]  1.00342952e-02,
       [illegible]  0.00000000e+00,
       [illegible]  9.62548319e-03,
       -1.89640650e-02,  0.00000000e+00,  6.21099354e-02,  9.74100606e-03,
        6.29512235e-04,  0.00000000e+00,  0.00000000e+00,  6.04450356e-03,
        4.26731433e-05, -2.05100070e-02, -3.22864278e-02, -6.04226332e-03,
       -4.25599262e-02, -8.78926827e-03,  2.97123476e-03,  4.31077486e-04,
        2.03663205e-02, -4.53031555e-03, -4.53031555e-03, -1.24626423e-02,
       -2.82319901e-02, -2.78921981e-02,  6.44165081e-04,  1.35744800e-02,
        1.82038686e-02,  0.00000000e+00, -5.61766542e-02,  4.17650480e-02,
       -4.53031555e-03, -8.01273483e-03,  1.57810165e-03,  2.37909771e-03,
       -3.49314807e-03, -4.53031555e-03, -2.74345851e-02,  7.70239613e-03,
       -8.63556355e-03,  2.04542120e-02,  0.00000000e+00,  0.00000000e+00,
       -1.05544580e-02,  3.07617681e-02, -4.51030194e-02,  1.58745813e-02,
        3.08303668e-02,  1.62154496e-02,  4.19676530e-02, -4.21529646e-03,
        1.59991868e-02,  3.39156417e-02,  4.85500074e-02, -7.66122776e-03,
        1.35220760e-02,  2.97537840e-02,  3.36043854e-02,  4.90495109e-02,
        3.39156417e-02])
```

In [62]:
```
# lets predict the R-squared value of test and train data
y_train_pred = ridge.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
```

```
0.9543534179988409
```

Lasso

In [63]:
```
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)
```

Out[63]:
```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
GridSearchCV(cv=5, estimator=Lasso(),
             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, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [64]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[64]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	spli
0	0.158327	0.054745	0.004601	0.000483	0.0001	{'alpha': 0.0001}	-0.084353	-0.075172	-0.080517	
1	0.023593	0.002730	0.004005	0.000008	0.001	{'alpha': 0.001}	-0.083513	-0.082241	-0.086239	
2	0.012000	0.000009	0.003995	0.000009	0.01	{'alpha': 0.01}	-0.102980	-0.103797	-0.107636	
3	0.037590	0.005541	0.004056	0.000109	0.05	{'alpha': 0.05}	-0.104493	-0.107558	-0.110196	
4	0.032600	0.003138	0.004199	0.000397	0.1	{'alpha': 0.1}	-0.103231	-0.107880	-0.110775	
5	0.037979	0.010170	0.005400	0.002337	0.2	{'alpha': 0.2}	-0.101457	-0.108667	-0.111706	
6	0.031039	0.012721	0.006200	0.002639	0.3	{'alpha': 0.3}	-0.101585	-0.109304	-0.112625	
7	0.027724	0.009580	0.001600	0.001959	0.4	{'alpha': 0.4}	-0.102175	-0.109999	-0.113650	
8	0.016652	0.000923	0.007672	0.004763	0.5	{'alpha': 0.5}	-0.102913	-0.110751	-0.114802	
9	0.011334	0.006103	0.006244	0.007648	0.6	{'alpha': 0.6}	-0.103793	-0.111564	-0.116410	
10	0.011004	0.004451	0.003774	0.004352	0.7	{'alpha': 0.7}	-0.104541	-0.112549	-0.118140	
11	0.012501	0.006250	0.003124	0.006248	0.8	{'alpha': 0.8}	-0.105447	-0.113752	-0.120131	
12	0.015626	0.000001	0.006250	0.007655	0.9	{'alpha': 0.9}	-0.106494	-0.115133	-0.122382	
13	0.012501	0.006250	0.006251	0.007655	1.0	{'alpha': 1.0}	-0.107763	-0.116750	-0.124783	
14	0.012499	0.006249	0.003125	0.006250	2.0	{'alpha': 2.0}	-0.124119	-0.131889	-0.145178	
15	0.009375	0.007655	0.006250	0.007655	3.0	{'alpha': 3.0}	-0.137520	-0.145926	-0.160454	
16	0.009375	0.007655	0.003125	0.006250	4.0	{'alpha': 4.0}	-0.140376	-0.147105	-0.162543	
17	0.012500	0.006250	0.003125	0.006251	5.0	{'alpha': 5.0}	-0.142106	-0.147917	-0.163933	
18	0.009375	0.007655	0.003125	0.006250	6.0	{'alpha': 6.0}	-0.143827	-0.149095	-0.165856	
19	0.012501	0.006251	0.003125	0.006249	7.0	{'alpha': 7.0}	-0.145599	-0.150416	-0.167971	
20	0.012500	0.006250	0.003126	0.006251	8.0	{'alpha': 8.0}	-0.146804	-0.151205	-0.170272	
21	0.003125	0.006250	0.006250	0.007655	9.0	{'alpha': 9.0}	-0.148186	-0.152041	-0.172089	
22	0.008449	0.007102	0.007084	0.007140	10.0	{'alpha': 10.0}	-0.149603	-0.152883	-0.173938	
23	0.015625	0.000004	0.000000	0.000000	20	{'alpha': 20}	-0.170920	-0.166302	-0.195126	
24	0.006251	0.007656	0.000000	0.000000	50	{'alpha': 50}	-0.219630	-0.195860	-0.232301	
25	0.000000	0.000000	0.012500	0.006250	100	{'alpha': 100}	-0.296166	-0.261723	-0.298255	
26	0.008206	0.006613	0.003926	0.006054	500	{'alpha': 500}	-0.321071	-0.284968	-0.322368	
27	0.012872	0.008703	0.002998	0.001548	1000	{'alpha': 1000}	-0.324391	-0.290967	-0.326605	

28 rows × 21 columns

In [65]:
```
# lets find out the R-squared value of the lasso model
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)
```

Out[65]:
```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
GridSearchCV(cv=5, estimator=Lasso(),
             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, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='r2', verbose=1)
```

In [66]:
```
# cv results
cv_results1 = pd.DataFrame(model_cv1.cv_results_)
cv_results1
```

Out[66]:

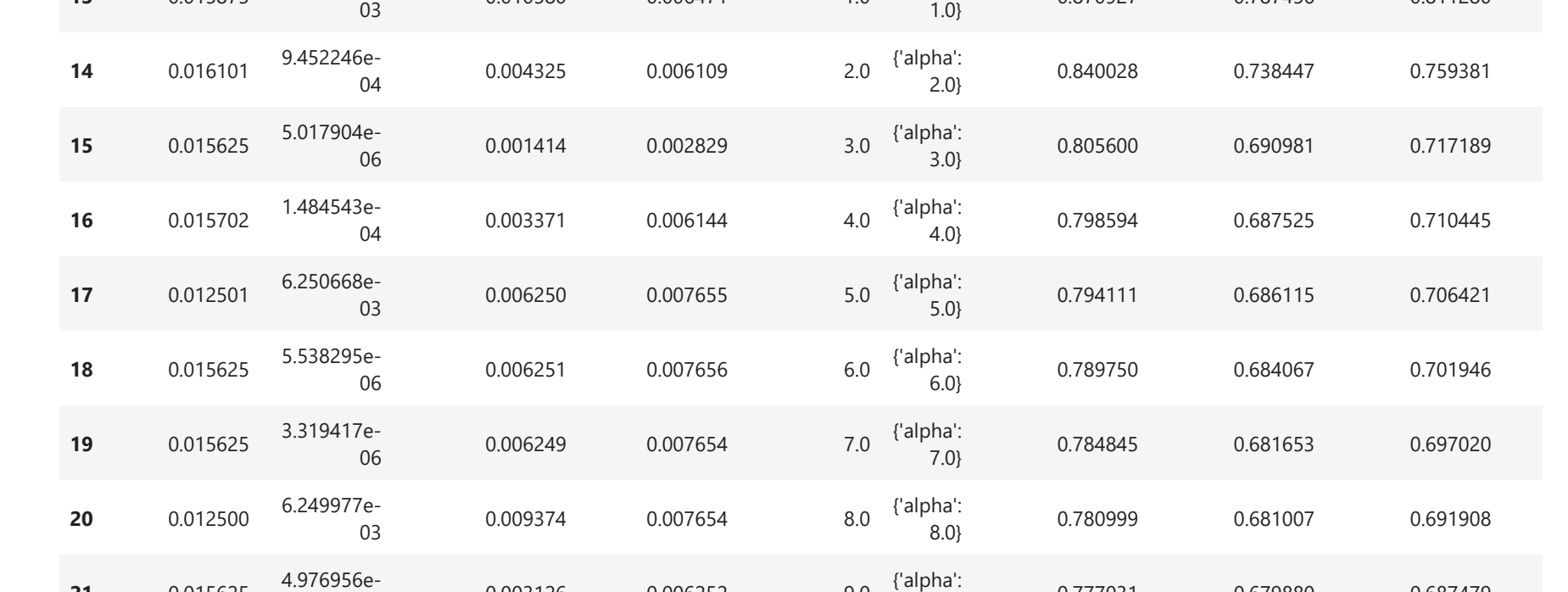
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	spli
0	0.131903	4.250767e-02	0.003402	0.001745	0.0001	{'alpha': 0.0001}	0.908154	0.898766	0.918740	
1	0.017423	7.375206e-03	0.003125	0.006250	0.001	{'alpha': 0.001}	0.915172	0.896825	0.905315	
2	0.009375	7.654695e-03	0.003125	0.006251	0.01	{'alpha': 0.01}	0.879692	0.835426	0.860768	
3	0.038514	9.038504e-03	0.003976	0.006054	0.05	{'alpha': 0.05}	0.875506	0.823165	0.852224	
4	0.026448	6.308962e-03	0.003125	0.006250	0.1	{'alpha': 0.1}	0.877591	0.822462	0.850550	
5	0.025000	7.654558e-03	0.000000	0.000000	0.2	{'alpha': 0.2}	0.880167	0.819668	0.847828	
6	0.024999	7.656563e-03	0.000000	0.000000	0.3	{'alpha': 0.3}	0.880831	0.816835	0.844628	
7	0.016097	9.429990e-04	0.006944	0.007199	0.4	{'alpha': 0.4}	0.880534	0.813327	0.840908	
8	0.017528	2.319519e-03	0.000000	0.000000	0.5	{'alpha': 0.5}	0.879737	0.809349	0.836593	
9	0.015625	1.676406e-06	0.003125	0.006250	0.6	{'alpha': 0.6}	0.878517	0.805988	0.831923	
10	0.015625	7.893059e-07	0.000000	0.000000	0.7	{'alpha': 0.7}	0.877350	0.802121	0.827330	
11	0.021875	7.653020e-03	0.006250	0.007655	0.8	{'alpha': 0.8}	0.875598	0.797743	0.822353	
12	0.015625	1.628244e-06	0.000000	0.000000	0.9	{'alpha': 0.9}	0.873434	0.792856	0.817008	
13	0.015875	5.012572e-03	0.010580	0.006471	1.0	{'alpha': 1.0}	0.870927	0.787456	0.811280	
14	0.016101	9.452246e-04	0.004325	0.006109	2.0	{'alpha': 2.0}	0.840028	0.738447	0.759381	
15	0.015625	5.017904e-06	0.001414	0.002829	3.0	{'alpha': 3.0}	0.805600	0.690981	0.717189	
16	0.015702	1.484543e-04	0.003371	0.006144	4.0	{'alpha': 4.0}	0.798594	0.687525	0.710445	
17	0.012501	6.250668e-03	0.006250	0.007655	5.0	{'alpha': 5.0}	0.794111	0.686115	0.706421	
18	0.015625	5.538295e-06	0.006251	0.007656	6.0	{'alpha': 6.0}	0.789750	0.684067	0.701946	
19	0.015625	3.319417e-06	0.006249	0.007656	7.0	{'alpha': 7.0}	0.784845	0.681653	0.697020	
20	0.015000	6.249977e-03	0.000000	0.000000	8.0	{'alpha': 8.0}	0.780999	0.681007	0.691908	
21	0.015577	4.976956e-06	0.003126	0.006252	9.0	{'alpha': 9.0}	0.777031	0.679880	0.687479	
22	0.015577	9.507054e-05	0.004320	0.006106	10.0	{'alpha': 10.0}	0.772687	0.678270	0.682760	
23	0.015620	6.107328e-06	0.004327	0.006112	20	{'alpha': 20}	0.708547	0.635656	0.619568	
24	0.009377	7.656174e-03	0.006249	0.007654	50	{'alpha': 50}	0.549293	0.523380	0.480244	
25	0.012168	6.118052e-03	0.007507	0.007014	100	{'alpha': 100}	0.183096	0.198867	0.152330	
26	0.009376	7.655356e-03	0.006250	0.007655	500	{'alpha': 500}	0.021539	0.036963	0.018989	
27	0.009905	8.145802e-03	0.007451	0.007026	1000	{'alpha': 1000}	-0.000106	-0.005716	-0.009400	

28 rows × 21 columns

In [67]:
```
# 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[67]: <matplotlib.legend.Legend at 0x24d2877fd30>



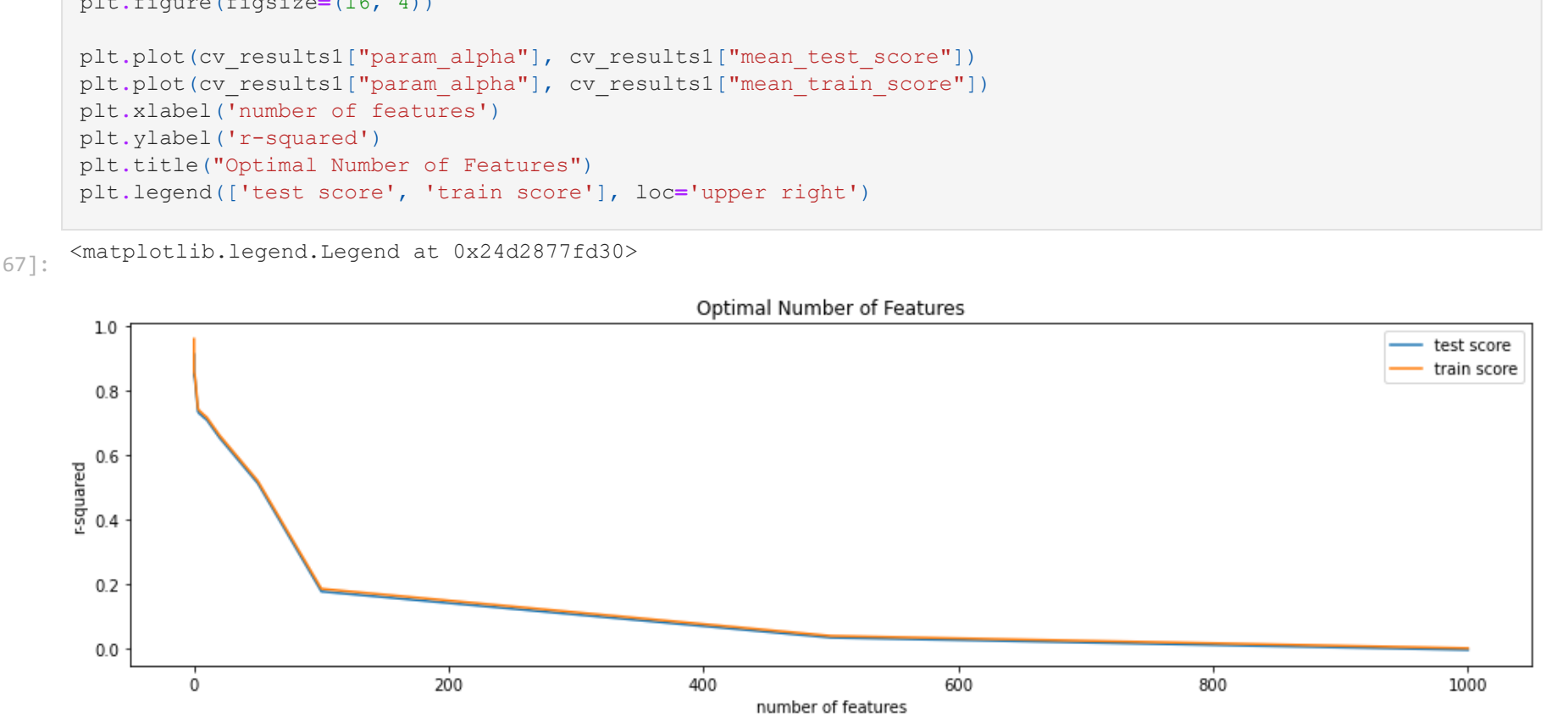
In [68]:
```
# checking the value of optimum number of parameters
print(model_cv.best_params_)
print(model_cv.best_score_)
```

```
{'alpha': 0.0001}
-0.07630744921613342
```

In [69]:
```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.figure(figsize=(16, 5))
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()
```



from the above graph and the best*param* score we got optimum lambda to be 0.0001

In [70]:
```
alpha = 0.0001

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

Out[70]: Lasso(alpha=0.0001)

In [71]:
```
# lets predict the R-squared value of test and train data
y_train_pred = lasso.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
```

```
0.9575119031855726
```

In [72]:
```
alpha = 0.0001

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

Out[72]: Lasso(alpha=0.0001)

In [73]:
```
# lets predict the R-squared value of test and train data
y_test_pred = lasso.predict(X_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred))
```

```
0.8818744660675766
```

In [74]:
```
from sklearn.metrics import mean_squared_error

print('RMSE is: \n', mean_squared_error(y_test, y_test_pred))
```

```
RMSE is:
 0.0200219184646192
```

In [75]:
```
alpha = 0.0001

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
preds = lasso.predict(test)
final_predictions = np.exp(preds)
```

In [76]:
```
test.index = test.index + 1461
```

In [77]:
```
submission = pd.DataFrame({'Id': test.index, 'SalePrice': final_predictions})
```

In [78]:
```
submission
```

Out[78]:

	Id	SalePrice
0	1461	119599.156045
1	1462	155560.639747
2	1463	202537.317501
3	1464	201609.123333
4	1465	181245.355148
...	...	...
1454	2915	86622.391864
1455	2916	81521.090708
1456	2917	176374.006362
1457	2918	127247.038023
1458	2919	200820.166938

1459 rows × 2 columns

predictions using ridge

In [79]:
```
alpha = 4

ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)
preds1 = ridge.predict(test)
final_predictions1 = np.exp(preds1)
```

In [80]:
```
submission1 = pd.DataFrame({'Id': test.index, 'SalePrice': final_predictions1})
```

In [81]:
```
submission1
```

Out[81]:

	Id	SalePrice
0	1461	119670.376779
1	1462	158159.031833
2	1463	203458.842110
3	1464	202362.661692
4	1465	177606.073857
...	...	...
1454	2915	88734.466119
1455	2916	82643.485386
1456	2917	176256.568426
1457	2918	127696.707670
1458	2919	202855.625089

1459 rows × 2 columns

-----X------X----------X----------X----------X----------X------X----------X----------X-----
---