Project 1 - Regression Problem

Dataset Description

Link to the dataset used: [https://www.kaggle.com/sameersmahajan/seattle-house-sales-prices (https://www.kaggle.com/sameersmahajan/seattle-house-sales-prices)](https://www.kaggle.com/sameersmahajan/seattle-house-sales-prices)

ID - Unique ID for each home sold

Date - Date of the home sale

Price - Price of the home sale

Bedrooms - Number of bedrooms

Bathrooms - Number of bathrooms (0.5 accounts for a room with a toilet but no shower)

Sqft_living - Square footage of the apartments

Sqft_lot - Square footage of the land space

Floors - Number of floors

Waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not

View - Index from 0 to 4 of how good the view of the property was

Condition - An index from 1 to 5 on the condition of the apartment

Grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design

Sqft_above - The square footage of the interior housing space that is above ground level

Sqft_basement - The square footage of the interior housing space that is below ground level

Yr_built - The year the house was intially built

Yr_renovated - The year of the house's last renovation

Zipcode - What zipcode area the house is in

Lat - Latitude

Long - Longitude

Sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors

Sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

To print out the output of all codes in a cell

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Loading Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV
```

Loading Dataset

In [4]:
```
bm = pd.read_csv("house_sales.csv")
```

Overview of Dataset

In [5]:
```
bm.shape
bm.head(10)
```

Out[5]: (21613, 21)

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	7129300520	20141013T000000	221900	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000	3	2.00	1680	8080	1.0	
5	7237550310	20140512T000000	1225000	4	4.50	5420	101930	1.0	
6	1321400060	20140627T000000	257500	3	2.25	1715	6819	2.0	
7	2008000270	20150115T000000	291850	3	1.50	1060	9711	1.0	
8	2414600126	20150415T000000	229500	3	1.00	1780	7470	1.0	
9	3793500160	20150312T000000	323000	3	2.50	1890	6560	2.0	

10 rows × 21 columns

In [6]:
```
bm.columns
bm.dtypes
```

Out[6]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

Out[6]:
```
id                 int64
date              object
price              int64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [7]:
```
bm.head().T
```

Out[7]:

	0	1	2	3	
id	7129300520	6414100192	5631500400	2487200875	19544
date	20141013T000000	20141209T000000	20150225T000000	20141209T000000	20150218T(
price	221900	538000	180000	604000	5
bedrooms	3	3	2	4	
bathrooms	1	2.25	1	3	
sqft_living	1180	2570	770	1960	
sqft_lot	5650	7242	10000	5000	
floors	1	2	1	1	
waterfront	0	0	0	0	
view	0	0	0	0	
condition	3	3	3	5	
grade	7	7	6	7	
sqft_above	1180	2170	770	1050	
sqft_basement	0	400	0	910	
yr_built	1955	1951	1933	1965	
yr_renovated	0	1991	0	0	
zipcode	98178	98125	98028	98136	
lat	47.5112	47.721	47.7379	47.5208	4
long	-122.257	-122.319	-122.233	-122.393	-1
sqft_living15	1340	1690	2720	1360	
sqft_lot15	5650	7639	8062	5000	

Data Preprocessing

Converting the data types of columns accordingly for exploring the dataset easier

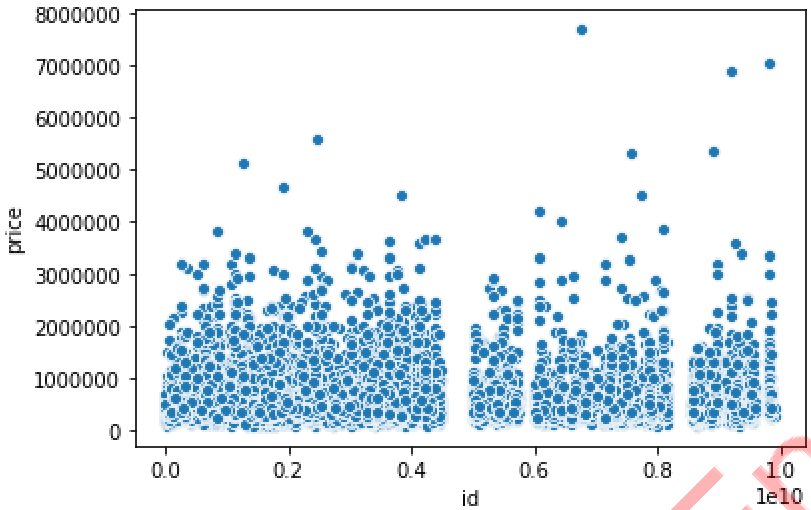
Columns
'id','bedrooms','bathrooms','floors','waterfront','view','condition','grade','yr_built','yr_renovated','zipcod
are categorical

In [8]:
```
bm[['id','bedrooms','bathrooms','floors','waterfront','view','condition','grade',
```

Dropping the column 'id' as it does not affect the prediction of house prices.

In [9]:
```
sns.scatterplot(x='id',y='price',data=bm)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9790e320>



In [10]:
```
bm_1 = bm.drop(['id'],axis=1)
```

Creating and Handling Null Values

Null values are created randomly across the dataset

Creating a dataframe without column 'Price' so that null values are not created under the dependent column 'Price'

In [11]:
```
bm_price = bm_1['price']
bm_x = bm_1.drop(['price'],axis=1)
bm_x.shape
```

Out[11]: (21613, 19)

In [12]:
```
np.random.seed(0)
bm_x = bm_x.mask(np.random.random(bm_x.shape) < .051)
```

In [13]:
```
bm_x.isnull().sum()
```

Out[13]:
```
date             1128
bedrooms         1099
bathrooms        1140
sqft_living      1120
sqft_lot         1069
floors           1061
waterfront       1096
view             1119
condition        1078
grade            1087
sqft_above       1127
sqft_basement    1080
yr_built         1031
yr_renovated     1177
zipcode          1128
lat              1087
long             1137
sqft_living15    1097
sqft_lot15       1123
dtype: int64
```

In [14]:
```
bm_x.isnull().sum().sum()
```

Out[14]:
```
20984
```

20984 null values (approx. 5-6% of whole dataset) are created across the dataset

In [15]:
```
bm_x[bm_x.isnull().any(axis=1)]
```

Out[15]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
0	20141013T000000	3	1	1180.0	5650.0	1	0	0	
1	20141209T000000	3	2.25	2570.0	7242.0	2	0	0	
3	20141209T000000	4	3	1960.0	5000.0	1	0	0	
5	20140512T000000	4	NaN	5420.0	NaN	1	0	0	
6	20140627T000000	3	2.25	1715.0	6819.0	2	0	0	
8	20150415T000000	3	1	1780.0	7470.0	1	0	0	
9	20150312T000000	3	NaN	1890.0	6560.0	2	0	0	
10	20150403T000000	3	2.5	3560.0	9796.0	1	0	0	
11	20140527T000000	2	1	1160.0	6000.0	NaN	0	0	
12	20140528T000000	3	1	1430.0	19901.0	NaN	0	0	
13	20141007T000000	3	1.75	1370.0	9680.0	1	0	0	
14	20150312T000000	5	2	1810.0	4850.0	1.5	0	0	
16	20140731T000000	3	2	1890.0	14040.0	NaN	0	0	
18	20141205T000000	NaN	1	1200.0	9850.0	1	0	0	
20	20140514T000000	4	1.75	1620.0	4980.0	1	0	0	
21	20140826T000000	3	2.75	3050.0	44867.0	1	0	4	
23	20140516T000000	2	1.5	1070.0	9643.0	1	NaN	0	
26	20140626T000000	3	1.75	2450.0	2691.0	2	0	0	
27	20141201T000000	3	1	1400.0	1581.0	1.5	0	0	
28	20140624T000000	3	1.75	1520.0	6380.0	1	0	0	
30	20141110T000000	NaN	2.5	2320.0	3980.0	2	0	0	
35	20140613T000000	3	2.5	2300.0	3060.0	1.5	0	0	
36	20140528T000000	4	1	1660.0	34848.0	1	0	NaN	
38	NaN	4	1	1220.0	8075.0	1	0	0	
39	20140620T000000	NaN	2.5	2620.0	7553.0	2	0	0	
42	20140707T000000	5	NaN	3595.0	5639.0	2	0	0	
43	20141028T000000	3	1	1570.0	NaN	2	0	0	
44	20140729T000000	NaN	1	1280.0	9656.0	1	0	0	
48	20150428T000000	3	1.75	1250.0	5963.0	1	0	0	
49	NaN	3	2.5	2753.0	65005.0	1	1	2	
...	...	...	...	...	...	...	...	...	
21568	20150130T000000	4	3.5	3830.0	8963.0	2	0	NaN	
21569	NaN	2	2.5	980.0	1020.0	3	0	0	
21570	20140513T000000	3	2.5	1450.0	5008.0	1	0	0	
21572	20140707T000000	3	1.75	1140.0	1201.0	2	0	0	

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
21573	20140520T000000	4	3.5	3070.0	4684.0	2	0	0	
21574	20140507T000000	3	3	1680.0	1570.0	3	0	0	
21575	NaN	3	2.5	3087.0	5002.0	2	0	0	
21577	NaN	4	3.25	1900.0	2631.0	2	0	0	
21580	20141003T000000	3	3	2780.0	6000.0	2	0	NaN	
21581	20150504T000000	3	3	NaN	6000.0	2	0	0	
21583	20140610T000000	2	NaN	710.0	1157.0	NaN	0	0	
21585	20140828T000000	3	2.5	NaN	5000.0	2	0	0	
21586	NaN	2	2.5	1430.0	1201.0	3	0	0	
21587	20150305T000000	3	2.5	1520.0	1488.0	3	0	0	
21589	20140910T000000	3	NaN	2540.0	4760.0	2	0	0	
21590	20140514T000000	4	3.5	4910.0	9444.0	1.5	0	0	
21591	20141002T000000	4	2.75	2770.0	3852.0	NaN	0	0	
21593	20150317T000000	5	3.75	4170.0	8142.0	2	0	NaN	
21594	20141017T000000	4	2.75	2500.0	5995.0	NaN	0	0	
21597	20150421T000000	4	3.25	3410.0	10125.0	2	0	0	
21598	20141013T000000	4	2.5	3118.0	7866.0	2	0	2	
21600	20141015T000000	NaN	3.75	NaN	NaN	2	0	0	
21601	20150407T000000	3	2.5	1425.0	1179.0	3	0	0	
21605	20141014T000000	NaN	2.5	2520.0	6023.0	2	0	0	
21606	20150326T000000	4	3.5	3510.0	7200.0	2	0	0	
21608	20140521T000000	3	2.5	1530.0	1131.0	NaN	0	0	
21609	20150223T000000	4	2.5	NaN	5813.0	2	0	0	
21610	20140623T000000	2	0.75	1020.0	1350.0	NaN	0	0	
21611	20150116T000000	3	2.5	1600.0	2388.0	2	0	NaN	
21612	20141015T000000	2	0.75	1020.0	NaN	NaN	0	0	

13637 rows × 19 columns

There are 13,637 rows with null values. Hence, it is not advisable to drop all the rows with null values. Better method would be impute the null values wherever possible.

Combining the dataset with null values with the column "price"

In [16]:
```
bm_final = pd.concat([bm_x,bm_price],axis=1)
```

In [17]:
```
bm_final.shape
bm_final.isnull().sum()
```

Out[17]:
```
(21613, 20)
```

Out[17]:
```
date             1128
bedrooms         1099
bathrooms        1140
sqft_living      1120
sqft_lot         1069
floors           1061
waterfront       1096
view             1119
condition        1078
grade            1087
sqft_above       1127
sqft_basement    1080
yr_built         1031
yr_renovated     1177
zipcode          1128
lat              1087
long             1137
sqft_living15    1097
sqft_lot15       1123
price               0
dtype: int64
```

Creating two columns from the 'date' column - sale year and month

In [18]:
```
bm_final['sale_year'] = bm_final['date'].str[:4]
```

In [19]:
```
bm_final['sale_month'] = bm_final['date'].str[4:6]
```

In [20]:
```
bm_final.sale_year.fillna('nan',inplace=True)
bm_final.sale_month.fillna('nan',inplace=True)
```

In [21]:
```
bm_final['sale_year'].value_counts()
bm_final['sale_year'].unique()
bm_final['sale_month'].value_counts()
bm_final['sale_month'].unique()
```

Out[21]:
```
2014    13890
2015     6595
nan      1128
Name: sale_year, dtype: int64
```

Out[21]:
```
array(['2014', '2015', 'nan'], dtype=object)
```

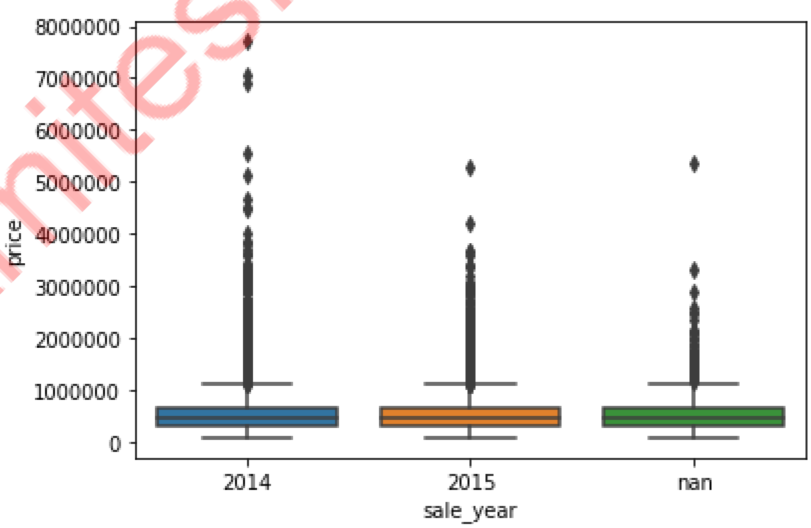
Out[21]:
```
05     2282
04     2110
06     2083
07     2081
08     1851
10     1779
03     1769
09     1682
12     1404
11     1330
02     1188
nan    1128
01      926
Name: sale_month, dtype: int64
```

Out[21]:
```
array(['10', '12', '02', '05', '06', '01', '04', '03', '07', '08', '11',
       'nan', '09'], dtype=object)
```

'Sale_Year' and 'Sale_Month'

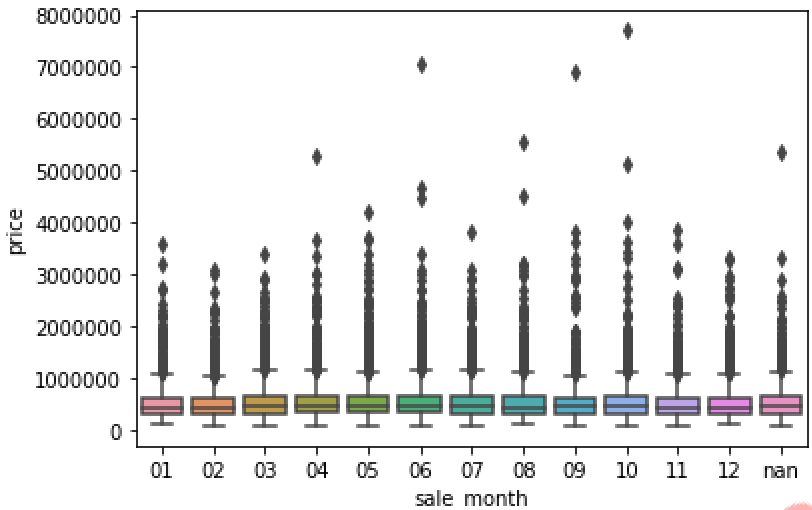
In [22]:
```
sns.boxplot(x='sale_year',y='price',data=bm_final)
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd99d7ad68>
```



In [23]:
```
sns.boxplot(x='sale_month',y='price',data=bm_final)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9ac1a4a8>



In [24]:
```
bm_month = bm_final[['sale_month','price']]
bm_month.groupby('sale_month').mean()
```

Out[24]:

sale_month	price
01	522241.234341
02	510867.186027
03	544219.409836
04	559923.530806
05	550857.780456
06	559200.218435
07	544410.164344
08	537774.264722
09	526874.860285
10	538100.658797
11	519752.912782
12	525533.121083
nan	545605.439716

We can infer that the mean price of the houses over the month does not vary a lot. Hence, dropping the null values in the column 'sale_month'

In [25]:
```
bm_final = bm_final[bm_final.sale_month != 'nan']
```

In [26]:
```
bm_year = bm_final[['sale_year','price']]
bm_year.groupby('sale_year').mean()
```

Out[26]:

sale_year	price
2014	539228.268035
2015	540955.486277

We can observe that the mean price of houses sold in 2015 is greater than those sold in 2014

From the analysis of "sale_year" column, we get to know that sales are made between the years 2014-2015. Also, from the plot, we can infer that the sales price are similar to those houses sold at 2015.

It would be advisable to impute the 'nan' values with 2015. But, before that we have to check the "year built" column so as to ensure year built is 'nan' or not for the rows with 'nan' under sale_year column

In [27]:
```
bm_final[(bm_final['sale_year']=='nan') & (bm_final['yr_built']=='nan')]
```

Out[27]:

date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	...

0 rows × 22 columns

No rows with 'nan' in both columns 'yr_built' and 'sale_year'. Therefore, replacing null under 'sale_year' with '2015'

In [28]:
```
bm_final['sale_year'] = bm_final['sale_year'].replace({'nan':'2015'})
```

Only 2 years are present: 2014, 2015

Hence, assigning 0 and 1 respectively

In [29]:
```
bm_final['sale_year'] = bm_final['sale_year'].replace({'2014':'0'})
bm_final['sale_year'] = bm_final['sale_year'].replace({'2015':'1'})
```

In [30]:
```
bm_final['sale_year'].value_counts()
```

Out[30]:
```
0    13890
1     6595
Name: sale_year, dtype: int64
```

'Bedrooms' and 'Bathrooms'

In [31]:
```
bm_final.bedrooms.fillna('nan',inplace=True)
bm_final.bathrooms.fillna('nan',inplace=True)
```

In [32]:
```
bm_final['bedrooms'].unique()
bm_final['bedrooms'].value_counts()
bm_final['bathrooms'].unique()
bm_final['bathrooms'].value_counts()
```

Out[32]:
```
array([3, 2, 4, 5, 'nan', 1, 6, 7, 0, 8, 9, 11, 10, 33], dtype=object)
```

Out[32]:
```
3      8863
4      6199
2      2452
5      1431
nan    1046
6       243
1       180
7        36
8        13
0        12
9         5
10        3
33        1
11        1
Name: bedrooms, dtype: int64
```

Out[32]:
```
array([1.0, 2.25, 3.0, 2.0, 'nan', 1.5, 2.5, 1.75, 2.75, 3.25, 4.0, 3.5,
       0.75, 4.75, 5.0, 4.25, 4.5, 3.75, 0.0, 1.25, 5.25, 6.0, 0.5, 5.5,
       6.75, 5.75, 8.0, 7.5, 7.75, 6.25, 6.5], dtype=object)
```

Out[32]:
```
2.5     4845
1.0     3466
1.75    2747
2.25    1837
2.0     1726
1.5     1286
nan     1073
2.75    1071
3.0      671
3.5      663
3.25     517
3.75     139
4.0      119
4.5       87
4.25      73
0.75      67
4.75      19
5.0       19
5.25      13
0.0        9
5.5        9
1.25       7
6.0        6
5.75       4
0.5        4
8.0        2
6.75       2
7.75       1
6.25       1
```

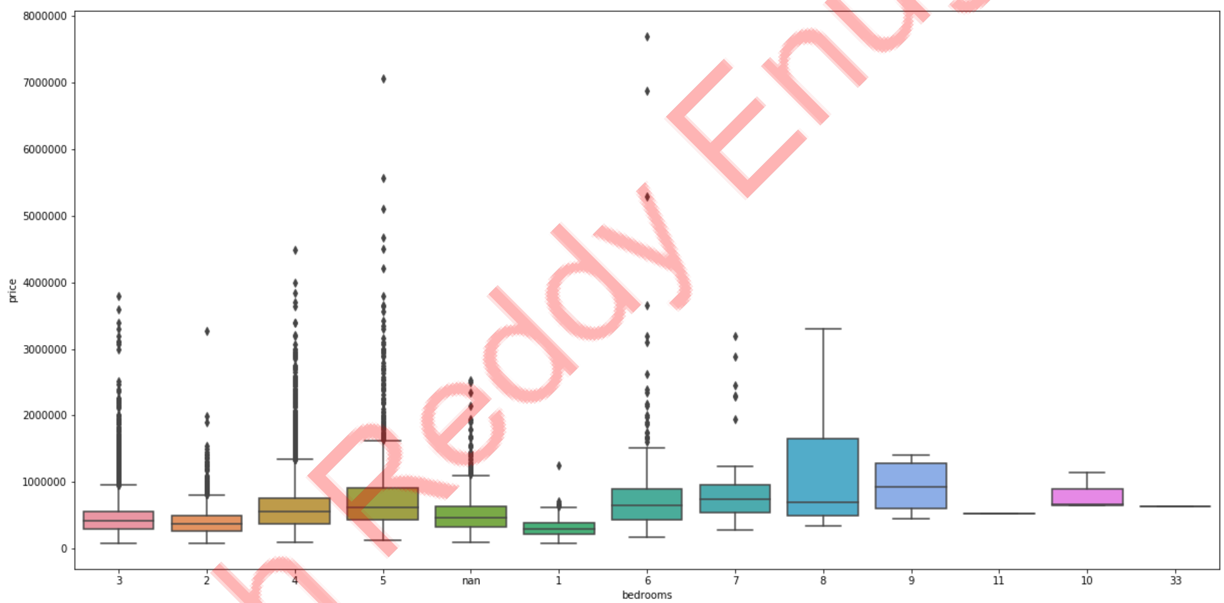
```
7.5        1
6.5        1
Name: bathrooms, dtype: int64
```

From above description of the number of bedrooms and bathrooms, we are able to find that there are houses with zero bedrooms and bathrooms, which is practically not possible. Hence, those rows can be dropped.

In [33]:
```
bm_final = bm_final[bm_final.bedrooms != 0]
bm_final = bm_final[bm_final.bathrooms != 0]
```

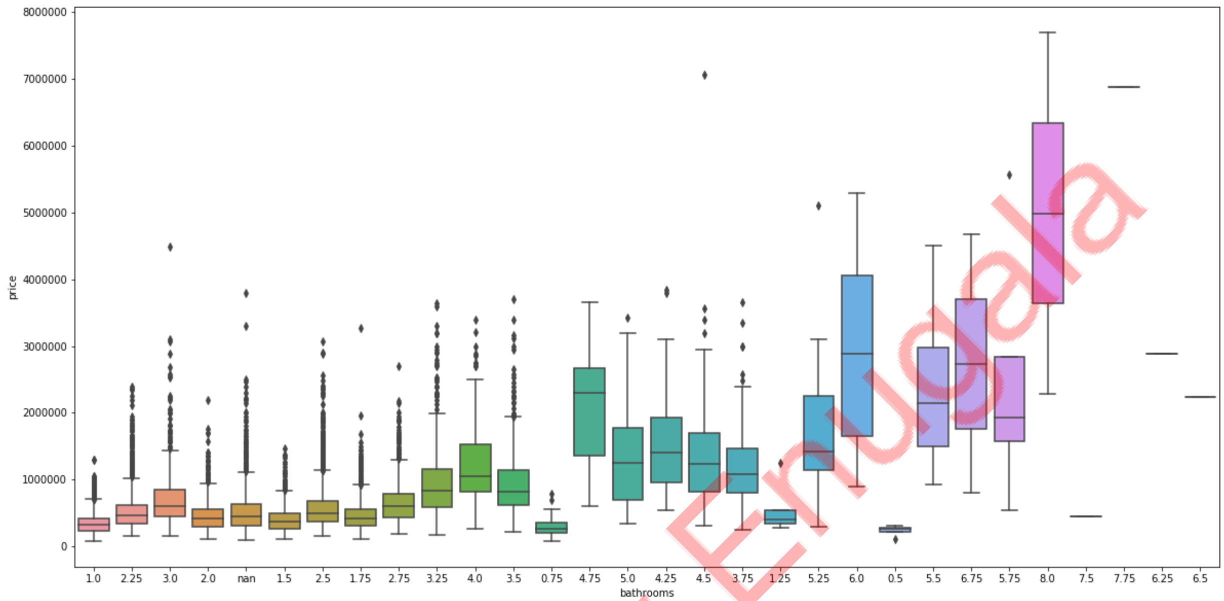
In [34]:
```
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(x='bedrooms',y='price',data=bm_final)
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9afb3748>



In [35]:
```
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(x='bathrooms',y='price',data=bm_final)
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9ac40438>



Based on the plots, we can infer that the null values for the bedrooms can be replaced with '3' and for bathrooms with '2.25'

In [36]:
```
bm_final['bedrooms'] = bm_final['bedrooms'].replace({'nan':3})
bm_final['bathrooms'] = bm_final['bathrooms'].replace({'nan':2.25})
```

In [37]:
```
bm_final['bedrooms'].unique()
bm_final['bedrooms'].value_counts()
bm_final['bathrooms'].unique()
bm_final['bathrooms'].value_counts()
bm_final.shape
```

Out[37]:
```
array([ 3,  2,  4,  5,  1,  6,  7,  8,  9, 11, 10, 33], dtype=int64)
```

Out[37]:
```
3     9908
4     6199
2     2452
5     1431
6      243
1      178
7       36
8       13
9        5
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

Out[37]:
```
array([1.  , 2.25, 3.  , 2.  , 1.5 , 2.5 , 1.75, 2.75, 3.25, 4.  , 3.5 ,
       0.75, 4.75, 5.  , 4.25, 4.5 , 3.75, 1.25, 5.25, 6.  , 0.5 , 5.5 ,
       6.75, 5.75, 8.  , 7.5 , 7.75, 6.25, 6.5 ])
```

In [38]:
```
bm_final.isnull().sum()
```

Out[38]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living      1062
sqft_lot         1022
floors           1000
waterfront       1025
view             1078
condition        1019
grade            1031
sqft_above       1079
sqft_basement    1030
yr_built          971
yr_renovated     1102
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
```

"sqft_living"

In [39]:
```
bm_final['sqft_living'].isnull().sum()
bm_final['sqft_living'].describe()
bm_final['sqft_living'].median()
```

Out[39]:
```
1062
```

Out[39]:
```
count    19408.000000
mean      2081.284367
std        917.777180
min        370.000000
25%       1430.000000
50%       1920.000000
75%       2550.000000
max      13540.000000
Name: sqft_living, dtype: float64
```

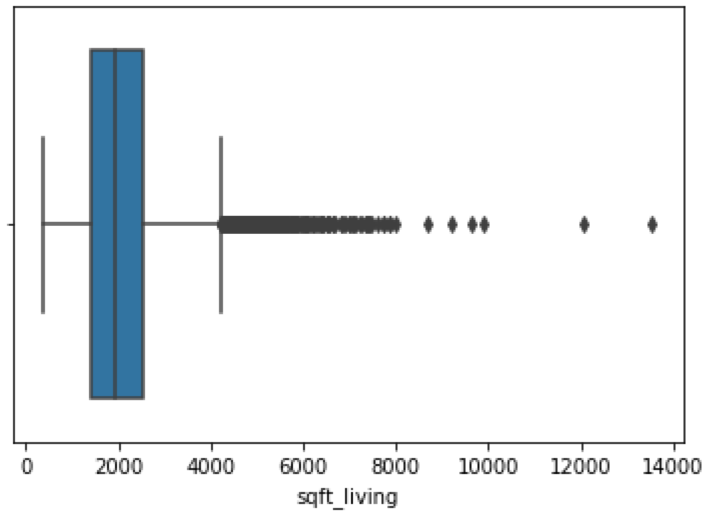
Out[39]:
```
1920.0
```

In [40]:
```
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.scatterplot(x='sqft_living',y='price',data=bm_final)
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9b4807f0>
```

In [41]:
```
sns.boxplot(x='sqft_living',data=bm_final)
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9ad55ac8>



Since, we are replacing the null values with median as there are outliers

In [42]:
```
bm_final['sqft_living']=bm_final['sqft_living'].fillna(bm_final['sqft_living'].m
```

In [43]:
```
bm_final['sqft_living'].isnull().sum()
```

Out[43]: 0

In [44]:
```
bm_final.isnull().sum()
```

Out[44]:
```
date              0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot       1022
floors         1000
waterfront     1025
view           1078
condition      1019
grade          1031
sqft_above     1079
sqft_basement  1030
yr_built        971
yr_renovated   1102
zipcode        1060
lat            1027
long           1078
sqft_living15  1041
sqft_lot15     1059
```

'sqft_lot'

In [45]:
```
bm_final['sqft_lot'].isnull().sum()
bm_final['sqft_lot'].describe()
bm_final['sqft_lot'].median()
```

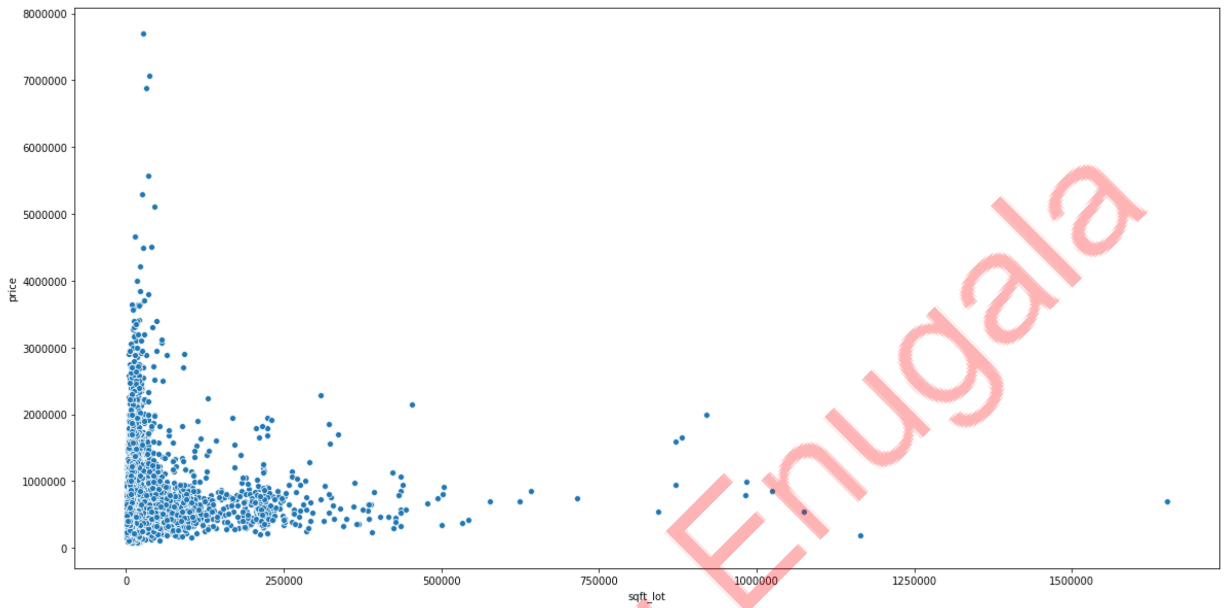
Out[45]:
```
1022
```

Out[45]:
```
count    1.944800e+04
mean     1.506629e+04
std      4.177920e+04
min      5.200000e+02
25%      5.053750e+03
50%      7.620000e+03
75%      1.070400e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```

Out[45]:
```
7620.0
```

In [46]:
```
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.scatterplot(x='sqft_lot',y='price',data=bm_final)
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9ada3128>
```



Since, outliers are present, imputing the null values with median of the column 'sqft_lot'

In [47]:
```
bm_final['sqft_lot']=bm_final['sqft_lot'].fillna(bm_final['sqft_lot'].median())
bm_final['sqft_lot'].isnull().sum()
```

Out[47]: 0

In [48]:
```
bm_final.isnull().sum()
```

Out[48]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors           1000
waterfront       1025
view             1078
condition        1019
grade            1031
sqft_above       1079
sqft_basement    1030
yr_built          971
yr_renovated     1102
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
```

'floors'

In [49]:
```
bm_final.floors.fillna('nan',inplace=True)
```

In [50]:
```
bm_final['floors'].value_counts()
bm_price_floors = bm_final[['floors','price']]
bm_price_floors.groupby('floors').mean()
```

Out[50]:
```
1.0    9602
2.0    7436
1.5    1731
nan    1000
3.0     549
2.5     145
3.5       7
Name: floors, dtype: int64
```

Out[50]:

	price
floors	
1.0	4.415906e+05
1.5	5.584669e+05
2.0	6.483285e+05
2.5	1.069188e+06
3.0	5.889418e+05
3.5	9.102143e+05
nan	5.388322e+05

We can see from the above table that the mean of those 'nan' floors is close to that of floors with

1.5

Hence, it is advisable to impute the nan's with 1.5

In [51]:
```
bm_final['floors'] = bm_final['floors'].replace({'nan':1.5})
```

In [52]:
```
bm_final['floors'].value_counts()
```

Out[52]:
```
1.0    9602
2.0    7436
1.5    2731
3.0     549
2.5     145
3.5       7
Name: floors, dtype: int64
```

In [53]:
```
bm_final.isnull().sum()
```

Out[53]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       1025
view             1078
condition        1019
grade            1031
sqft_above       1079
sqft_basement    1030
yr_built          971
yr_renovated     1102
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
price               0
sale_year           0
sale_month          0
dtype: int64
```

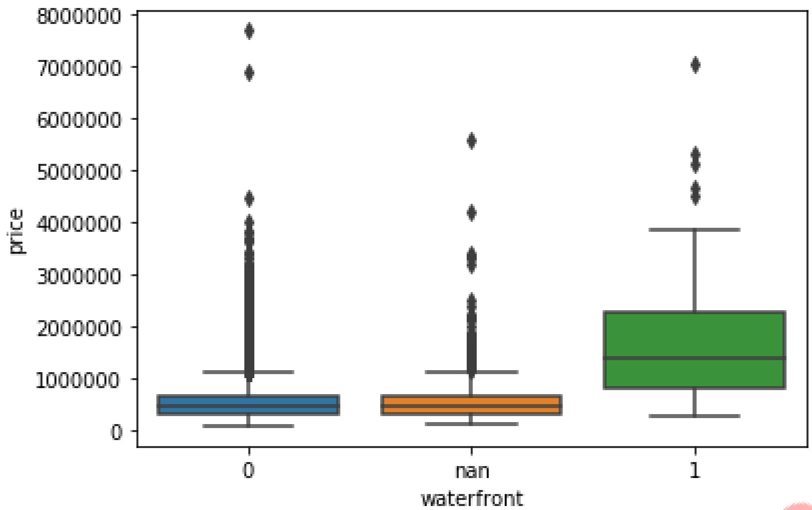
'waterfront'

In [54]:
```
bm_final.waterfront.fillna('nan',inplace=True)
bm_final['waterfront'].value_counts()
```

Out[54]:
```
0      19302
nan     1025
1        143
Name: waterfront, dtype: int64
```

In [55]:
```
sns.boxplot(x='waterfront',y='price',data=bm_final)
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9ae1cdd8>



Since, for both 'nan' and '0', they have the similar price ranges. We are imputing the nan's with 0

In [56]:
```
bm_final['waterfront'] = bm_final['waterfront'].replace({'nan':0})
bm_final['waterfront'].value_counts()
```

Out[56]:
```
0    20327
1      143
Name: waterfront, dtype: int64
```

In [57]:
```
bm_final.isnull().sum()
```

Out[57]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view             1078
condition        1019
grade            1031
sqft_above       1079
sqft_basement    1030
yr_built          971
yr_renovated     1102
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
price               0
sale_year           0
sale_month          0
dtype: int64
```

'view'

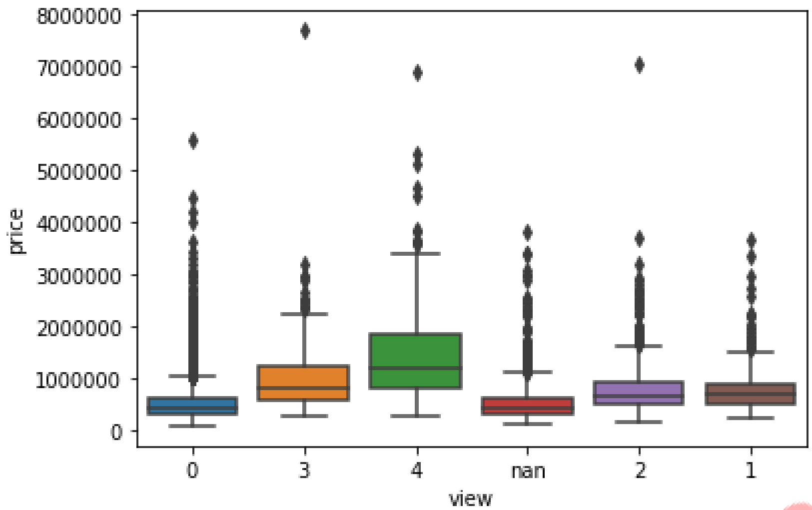
In [58]:
```
bm_final.view.fillna('nan',inplace=True)
```

In [59]:
```
bm_final['view'].value_counts()
```

Out[59]:
```
0      17520
nan     1078
2        846
3        444
1        299
4        283
Name: view, dtype: int64
```

In [60]:
```
sns.boxplot(x='view',y='price',data=bm_final)
```

Out[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9b06ecc0>
```



Since, for both 'nan' and '0', they have the similar price ranges. We are imputing the nan's with 0

In [61]:
```
bm_final['view'] = bm_final['view'].replace({'nan':0})
bm_final['view'].value_counts()
```

Out[61]:
```
0    18598
2      846
3      444
1      299
4      283
Name: view, dtype: int64
```

In [62]:
```
bm_final.isnull().sum()
```

Out[62]:
```
date               0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront         0
view               0
condition       1019
grade           1031
sqft_above      1079
sqft_basement   1030
yr_built         971
yr_renovated    1102
zipcode         1060
lat             1027
long            1078
sqft_living15   1041
sqft_lot15      1059
```

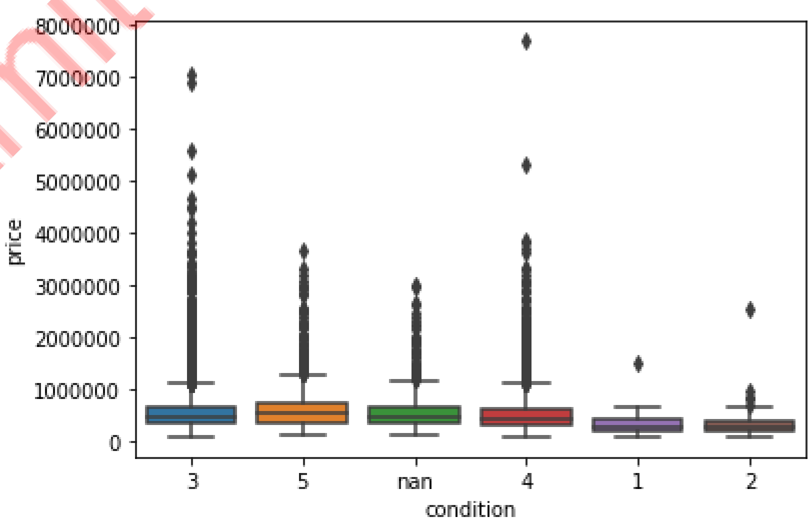
'condition'

In [63]:
```
bm_final.condition.fillna('nan',inplace=True)
bm_final['condition'].value_counts()
```

Out[63]:
```
3      12584
4       5147
5       1544
nan     1019
2        147
1         29
Name: condition, dtype: int64
```

In [64]:
```
sns.boxplot(x='condition',y='price',data=bm_final)
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9b543208>



In [65]:
```
bm_price_condition = bm_final[['condition','price']]
bm_price_condition.groupby('condition').mean()
```

Out[65]:

condition	price
1	341067.241379
2	318611.489796
3	540408.542832
4	522513.078104
5	611373.721503
nan	550373.646712

Since, for both 'nan' and '3', they have the similar price ranges. We are imputing the nan's with 3

In [66]:
```
bm_final['condition'] = bm_final['condition'].replace({'nan':3})
bm_final['condition'].value_counts()
bm_final.isnull().sum()
```

Out[66]:
```
3    13603
4     5147
5     1544
2      147
1       29
Name: condition, dtype: int64
```

Out[66]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade            1031
sqft_above       1079
sqft_basement    1030
yr_built          971
yr_renovated     1102
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
price               0
sale_year           0
sale_month          0
dtype: int64
```

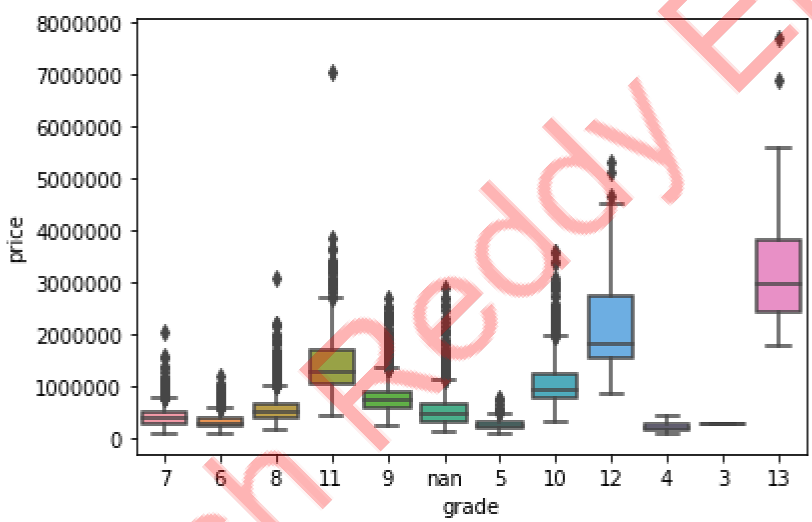
'grade'

In [67]:
```
bm_final.grade.fillna('nan',inplace=True)
bm_final['grade'].value_counts()
```

Out[67]:
```
7      8098
8      5477
9      2330
6      1818
nan    1031
10     1027
11      352
5       218
12       79
4        26
13       13
3         1
Name: grade, dtype: int64
```

In [68]:
```
sns.boxplot(x='grade',y='price',data=bm_final)
```

Out[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9b0fd5f8>
```

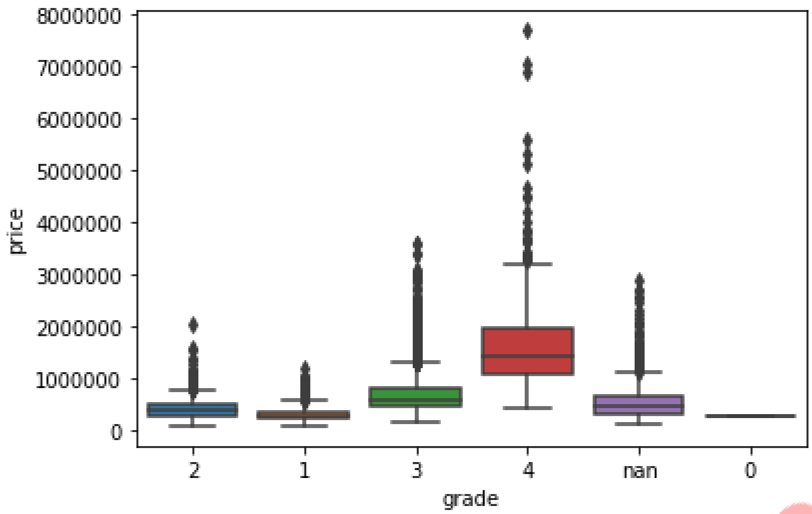


In [69]:
```
bm_final['grade'] = bm_final['grade'].replace({1:0,3:0,4:1,5:1,6:1,7:2,8:3,9:3,1

bm_final['grade'].unique()
```

Out[69]:
```
array([2, 1, 3, 4, 'nan', 0], dtype=object)
```

In [70]:
```
sns.boxplot(x='grade',y='price',data=bm_final)
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9b607240>



In [71]:
```
bm_price_grade = bm_final[['grade','price']]
bm_price_grade.groupby('grade').mean()
```

Out[71]:

grade	price
0	2.620000e+05
1	2.956212e+05
2	4.026813e+05
3	6.656492e+05
4	1.690680e+06
nan	5.331155e+05

In [72]:
```
bm_final['grade'].value_counts()
```

Out[72]:
```
3      8834
2      8098
1      2062
nan    1031
4       444
0         1
Name: grade, dtype: int64
```

Since, grade 'nan' has a different mean price. And 'grade' is an ordinal variable. Assuming higher the grade, higher the price.

We can impute nan as a new category '2.5'

In [73]:
```
bm_final['grade'] = bm_final['grade'].replace({'nan':2.5})
```

In [74]:
```
bm_final['grade'].value_counts()
```

Out[74]:
```
3.0    8834
2.0    8098
1.0    2062
2.5    1031
4.0     444
0.0       1
Name: grade, dtype: int64
```

In [75]:
```
bm_final.isnull().sum()
```

Out[75]:
```
date               0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront         0
view               0
condition          0
grade              0
sqft_above      1079
sqft_basement   1030
yr_built         971
yr_renovated    1102
zipcode         1060
lat             1027
long            1078
sqft_living15   1041
sqft_lot15      1059
price              0
sale_year          0
sale_month         0
dtype: int64
```

'sqft_above', 'sqft_basement'

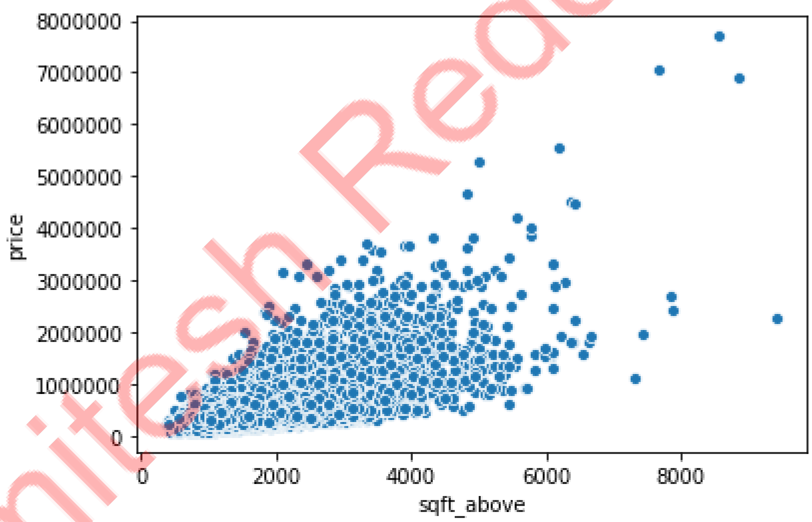
In [76]:
```
bm_final['sqft_above'].describe()
bm_final['sqft_basement'].describe()
```

Out[76]:
```
count    19391.000000
mean      1788.368934
std        827.641928
min        380.000000
25%       1190.000000
50%       1560.000000
75%       2210.000000
max       9410.000000
Name: sqft_above, dtype: float64
```

Out[76]:
```
count    19440.000000
mean       290.629835
std        442.046942
min          0.000000
25%          0.000000
50%          0.000000
75%        560.000000
max       4820.000000
Name: sqft_basement, dtype: float64
```

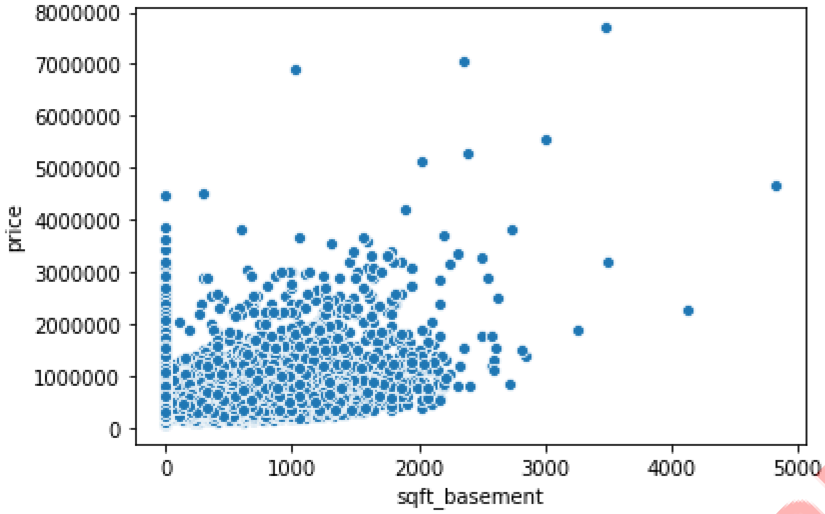
In [77]:
```
sns.scatterplot(x='sqft_above',y = 'price',data=bm_final)
```

Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9b9d2be0>



In [78]:
```
sns.scatterplot(x='sqft_basement', y='price',data=bm_final)
```

Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9ba48f98>



Filling the null values with mean and median

In [79]:
```
bm_final['sqft_basement']=bm_final['sqft_basement'].fillna(bm_final['sqft_basemen
bm_final['sqft_above']=bm_final['sqft_above'].fillna(bm_final['sqft_above'].mean
```

In [80]:
```
bm_final.isnull().sum()
```

Out[80]:
```
date              0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
view              0
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built        971
yr_renovated   1102
zipcode        1060
lat            1027
long           1078
sqft_living15  1041
sqft_lot15     1059
price             0
sale_year         0
sale_month        0
dtype: int64
```

'yr_built'

Dropping the nan under yr_built column as the price range is same over the years and does not have an effect on the price except for some cases

In [81]:
```
bm_final.yr_built.fillna('nan',inplace=True)
```

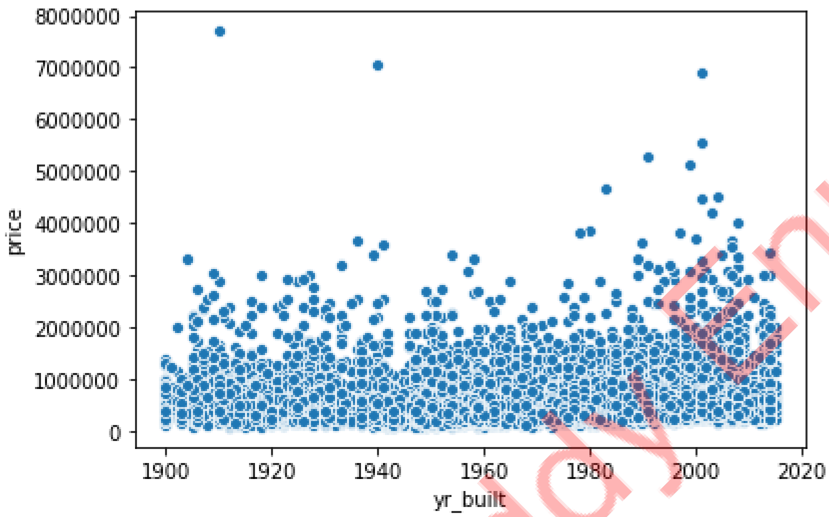
In [82]:
```
bm_final['yr_built'].value_counts()
```

Out[82]:
```
nan     971
2014    503
2005    410
2006    398
2004    385
1977    382
2003    366
2007    363
1968    350
1978    333
2008    327
1967    314
1979    308
1959    302
1990    287
1962    283
1954    282
2001    279
1987    267
1989    256
1969    255
1955    244
1988    244
1947    243
1999    241
1994    239
1976    233
1950    233
1963    231
1966    224
       ...
1945     82
1906     82
1909     81
1919     80
1908     78
1900     78
1923     71
1916     71
1912     70
1905     69
1921     68
1911     68
1937     61
1907     59
1931     55
1915     54
1917     51
1913     49
1914     48
1938     46
1904     44
1903     43
1936     39
2015     36
```

```
1932      34
1933      29
1901      27
1902      25
1935      22
1934      21
Name: yr_built, Length: 117, dtype: int64
```

In [83]:
```
sns.scatterplot(x='yr_built',y='price',data=bm_final)
```

Out[83]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9bab6a58>
```



Dropping the column 'yr_built' as the price for the houses built across the years have the same sales price

In [84]:
```
bm_final = bm_final.drop(['yr_built'],axis=1)
```

In [85]:
```
bm_final.shape
bm_final.isnull().sum()
```

Out[85]:
```
(20470, 21)
```

Out[85]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_renovated     1102
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
price               0
sale_year           0
sale_month          0
dtype: int64
```

'yr_renovated'

In [86]:
```
bm_final.yr_renovated.fillna('nan',inplace=True)
bm_final['yr_renovated'].value_counts()
```

Out[86]:
```
0       18559
nan      1102
2014       81
2003       33
2013       31
2005       30
2000       29
2007       26
2004       24
1990       23
2006       22
1989       21
2009       21
2002       20
1998       19
2001       19
1987       18
1984       18
1991       18
1983       17
1994       17
2008       17
2010       17
1993       16
1997       15
1985       15
1992       14
1986       13
2015       13
1988       13
         ...
1968        5
1958        5
1981        5
1978        4
1965        4
1964        4
1973        4
1972        4
1969        3
1953        3
1955        3
1956        3
1974        3
1957        3
1960        3
1963        3
1940        2
1962        2
1967        2
1971        2
1976        2
1950        2
1946        2
```

```
1945        2
1934        1
1954        1
1944        1
1948        1
1951        1
1959        1
Name: yr_renovated, Length: 71, dtype: int64
```

In [87]:
```
bm_final['yr_renovated'].unique()
```

Out[87]:
```
array([0, 1991, 'nan', 2002, 2010, 1992, 2013, 2005, 2008, 2003, 1994,
       1984, 1954, 2014, 2011, 1974, 1999, 1983, 1990, 1988, 1957, 1977,
       1981, 1995, 1978, 2000, 1998, 1970, 1989, 2004, 1986, 2009, 2007,
       1987, 1973, 2006, 2001, 1980, 1971, 1945, 1979, 1997, 1950, 1948,
       2015, 2012, 1968, 1963, 1951, 1993, 1962, 1996, 1972, 1985, 1953,
       1955, 1982, 1956, 1969, 1940, 1946, 1975, 1958, 1964, 1976, 1959,
       1960, 1967, 1965, 1934, 1944], dtype=object)
```

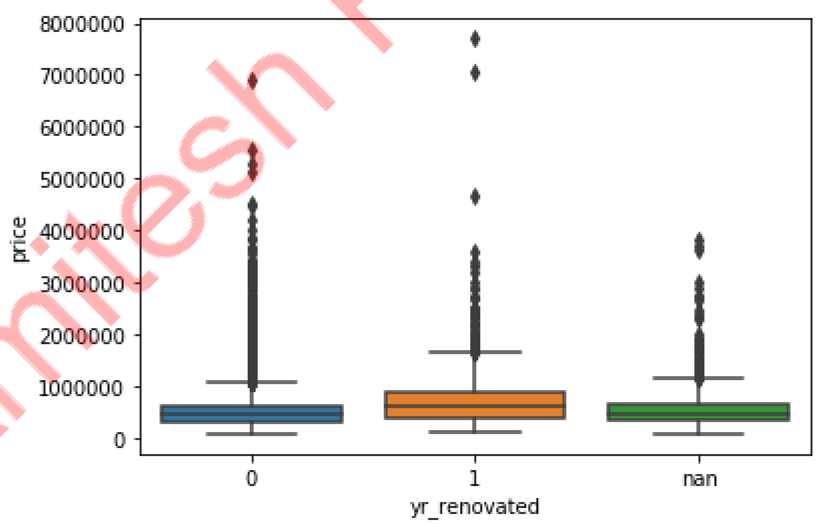
In [88]:
```
bm_final.loc[((bm_final.yr_renovated != 0) & (bm_final.yr_renovated != 'nan')),'y
```

In [89]:
```
bm_final['yr_renovated'].value_counts()
```

Out[89]:
```
0      18559
nan     1102
1        809
Name: yr_renovated, dtype: int64
```

In [90]:
```
sns.boxplot(x='yr_renovated',y='price',data=bm_final)
```

Out[90]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9bae2860>
```



We can infer that the price of houses which are renovated is higher when compared to the non-renovated houses

Imputing nan with value '0' as they both have compartively similar range

In [91]:
```
bm_final['yr_renovated'] = bm_final['yr_renovated'].replace({'nan':0})
```

In [92]:
```
bm_final['yr_renovated'].value_counts()
bm_final.isnull().sum()
```

Out[92]:
```
0    19661
1      809
Name: yr_renovated, dtype: int64
```

Out[92]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_renovated        0
zipcode          1060
lat              1027
long             1078
sqft_living15    1041
sqft_lot15       1059
price               0
sale_year           0
sale_month          0
dtype: int64
```

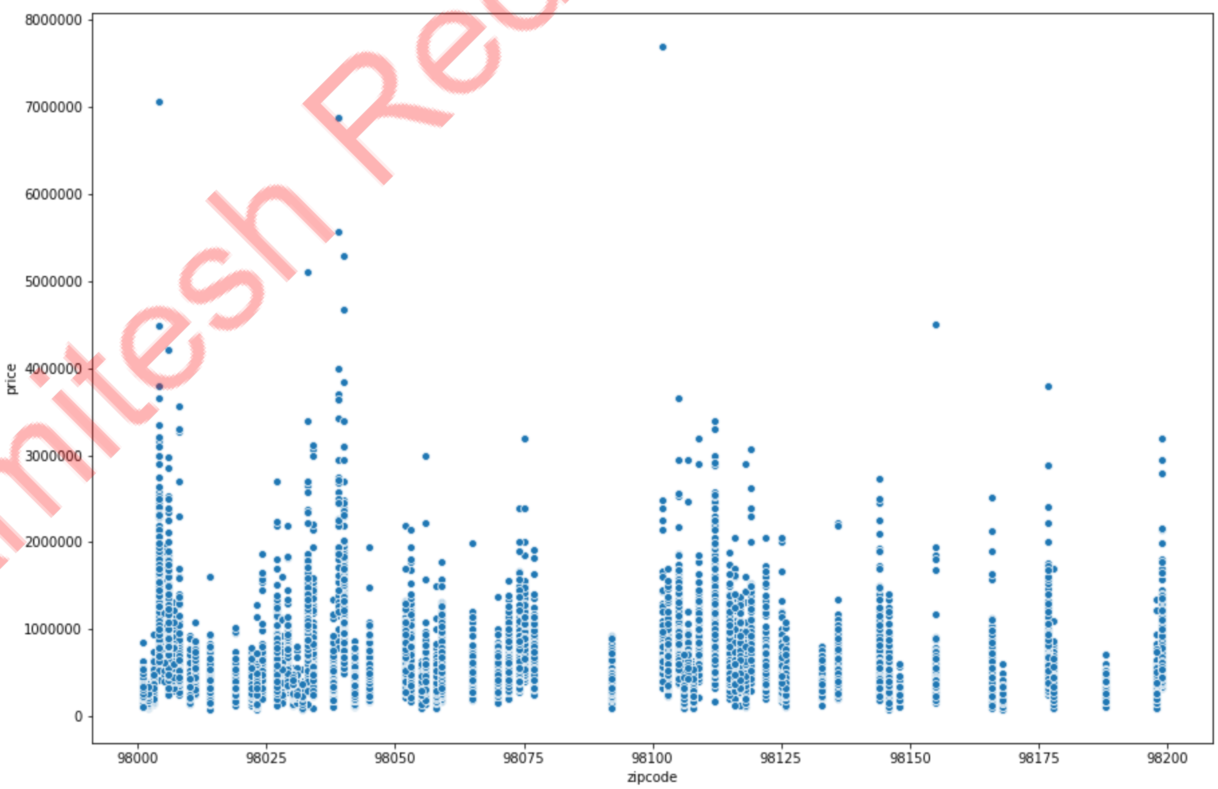
'zipcode'

In [93]:
```
bm_final.zipcode.fillna('nan',inplace=True)
bm_final['zipcode'].value_counts()
```

Out[93]:
```
nan      1060
98103     545
98038     537
98115     522
98052     516
98034     502
98117     500
98042     488
98118     457
98023     453
98006     446
98133     442
98059     419
98058     413
98155     402
98033     402
98074     397
98027     372
98125     361
```

In [94]:
```
a4_dims = (15,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.scatterplot(x='zipcode',y='price',data=bm_final)
```

Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9bb92eb8>



For the purpose of grouping the areas, we are calculating the price for each zipcode to form a group of zipcodes for easier analysis

In [95]:
```
bm_final['zipcode'].value_counts()
bm_price_floors = bm_final[['zipcode','price']]
```

Out[95]:
```
nan      1060
98103     545
98038     537
98115     522
98052     516
98034     502
98117     500
98042     488
98118     457
98023     453
98006     446
98133     442
98059     419
98058     413
98155     402
98033     402
98074     397
98027     372
98125     361
98056     360
98053     358
98126     323
98001     321
98075     318
98144     310
98106     306
98092     305
98116     298
98029     297
98004     294
         ...
98112     242
98031     242
98168     238
98055     236
98107     231
98178     230
98177     229
98030     227
98166     225
98022     207
98105     203
98045     200
98077     182
98002     182
98019     173
98011     171
98108     164
98119     162
98005     154
98188     124
98007     117
98014     110
98032     107
```

```
98070     102
98109     101
98102      91
98010      85
98024      69
98148      55
98039      44
Name: zipcode, Length: 71, dtype: int64
```

In [96]:
```
data = bm_price_floors.groupby('zipcode').mean()
data = pd.DataFrame(data)
data['zipcode'] = data.index
data.columns
data.sort_values('price')
sns.scatterplot(x='zipcode',y='price',data=data)
```

Out[96]:
```
Index(['price', 'zipcode'], dtype='object')
```

In [97]:
```
data.loc[((data.price < 300000)),'price']= 1
data.loc[((data.price >= 300000) & (data.price <= 600000)),'price']= 2
data.loc[((data.price > 600000) & (data.price <= 900000)),'price']= 3
data.loc[(data.price >= 900000),'price']= 4
```

In [98]:
```
data['price']. value_counts()
```

Out[98]:
```
2.0    36
3.0    20
1.0    10
4.0     5
Name: price, dtype: int64
```

In [99]:
```
a = data.loc[data['price']==1]
a1 = a['zipcode'].unique()
a1
b = data.loc[data['price']==2]
b1 = b['zipcode'].unique()
b1
c = data.loc[data['price']==3]
c1 = c['zipcode'].unique()
c1
d = data.loc[data['price']==4]
d1 = d['zipcode'].unique()
d1
```

Out[99]:
```
array([98001, 98002, 98003, 98023, 98030, 98031, 98032, 98148, 98168,
       98188], dtype=object)
```

Out[99]:
```
array([98010, 98011, 98014, 98019, 98022, 98024, 98028, 98034, 98038,
       98042, 98045, 98055, 98056, 98058, 98059, 98065, 98070, 98072,
       98092, 98103, 98106, 98107, 98108, 98117, 98118, 98125, 98126,
       98133, 98136, 98144, 98146, 98155, 98166, 98178, 98198, 'nan'],
      dtype=object)
```

Out[99]:
```
array([98005, 98006, 98007, 98008, 98027, 98029, 98033, 98052, 98053,
       98074, 98075, 98077, 98105, 98109, 98115, 98116, 98119, 98122,
       98177, 98199], dtype=object)
```

Out[99]:
```
array([98004, 98039, 98040, 98102, 98112], dtype=object)
```

We have identifed the zipcodes to be combined

Dropping the null values and then proceeding with grouping of zipcodes

In [100]:
```
bm_final = bm_final[bm_final.zipcode != 'nan']
bm_final['zipcode'].isnull().sum()
```

Out[100]:
```
0
```

In [101]:
```
bm_final['zipcode'] = bm_final['zipcode'].astype(object)
```

In [102]:
```
bm_final['zipcode'] = bm_final['zipcode'].replace(a1, '1')
bm_final['zipcode'] = bm_final['zipcode'].replace(b1, '2')
bm_final['zipcode'] = bm_final['zipcode'].replace(c1, '3')
bm_final['zipcode'] = bm_final['zipcode'].replace(d1, '4')
```

In [103]:
```
bm_final['zipcode'].value_counts()
```

Out[103]:
```
2    10396
3     5880
1     2207
4      927
Name: zipcode, dtype: int64
```

In [104]:
```
bm_final.shape
bm_final.isnull().sum()
```

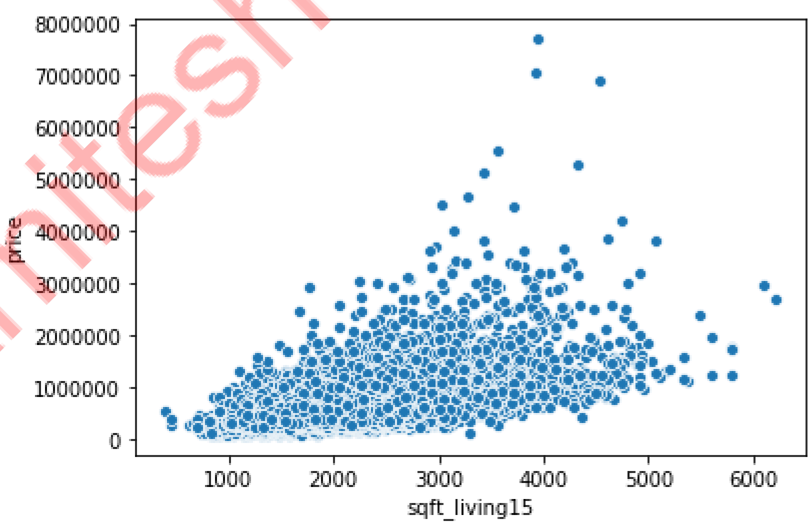
Out[104]:
```
(19410, 21)
```

Out[104]:
```
date                0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_renovated        0
zipcode             0
lat               976
long             1017
sqft_living15     984
sqft_lot15       1007
price               0
sale_year           0
sale_month          0
dtype: int64
```

'sqft_living15' and 'sqft_lot15'

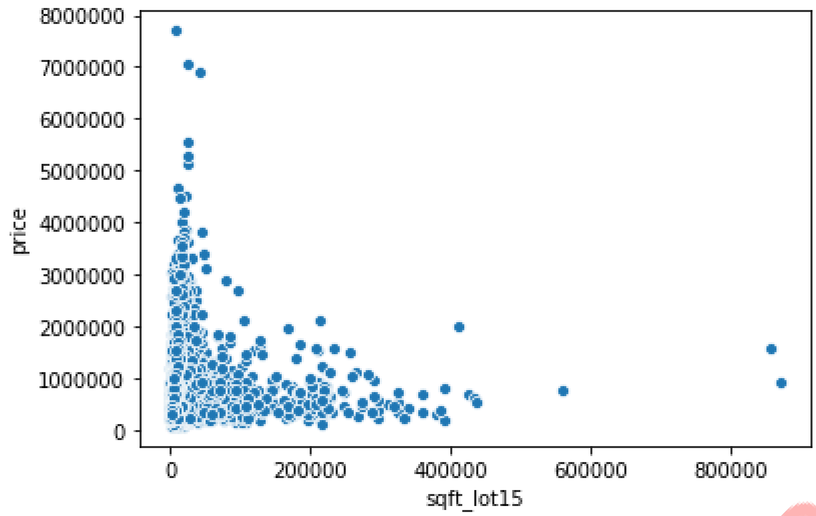
In [105]:
```
sns.scatterplot(x='sqft_living15',y='price',data=bm_final)
```

Out[105]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd9bc3d6a0>
```



In [106]:
```
sns.scatterplot(x='sqft_lot15',y='price',data=bm_final)
```

Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9bcea630>



In [107]:
```
bm_final['sqft_living15']=bm_final['sqft_living15'].fillna(bm_final['sqft_living1
bm_final['sqft_lot15']=bm_final['sqft_lot15'].fillna(bm_final['sqft_lot15'].mean
```

In [108]:
```
bm_final.isnull().sum()
```

Out[108]:
```
date              0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
view              0
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_renovated      0
zipcode           0
lat             976
long           1017
sqft_living15     0
sqft_lot15        0
price             0
sale_year         0
sale_month        0
dtype: int64
```

Dropping the date, lat,long,yr_built column

In [109]:
```
bm_final = bm_final.drop(['date','lat','long'],axis=1)
```

Dataset after cleaning null values

In [110]:
```
bm_final.shape
```

Out[110]:
```
(19410, 18)
```

In [111]:
```
bm_final.columns
```

Out[111]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_renovated', 'zipcode', 'sqft_living15',
       'sqft_lot15', 'price', 'sale_year', 'sale_month'],
      dtype='object')
```

In [112]:
```
bm_final.head()
```

Out[112]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_abov
0	3	1.00	1180.0	5650.0	1.0	0	0	3	2.0	1180
1	3	2.25	2570.0	7242.0	2.0	0	0	3	2.0	2170
2	2	1.00	770.0	10000.0	1.0	0	0	3	1.0	770
3	4	3.00	1960.0	5000.0	1.0	0	0	5	2.0	1050
4	3	2.00	1680.0	8080.0	1.0	0	0	3	3.0	1680

In [113]:
```
bm_final.dtypes
```

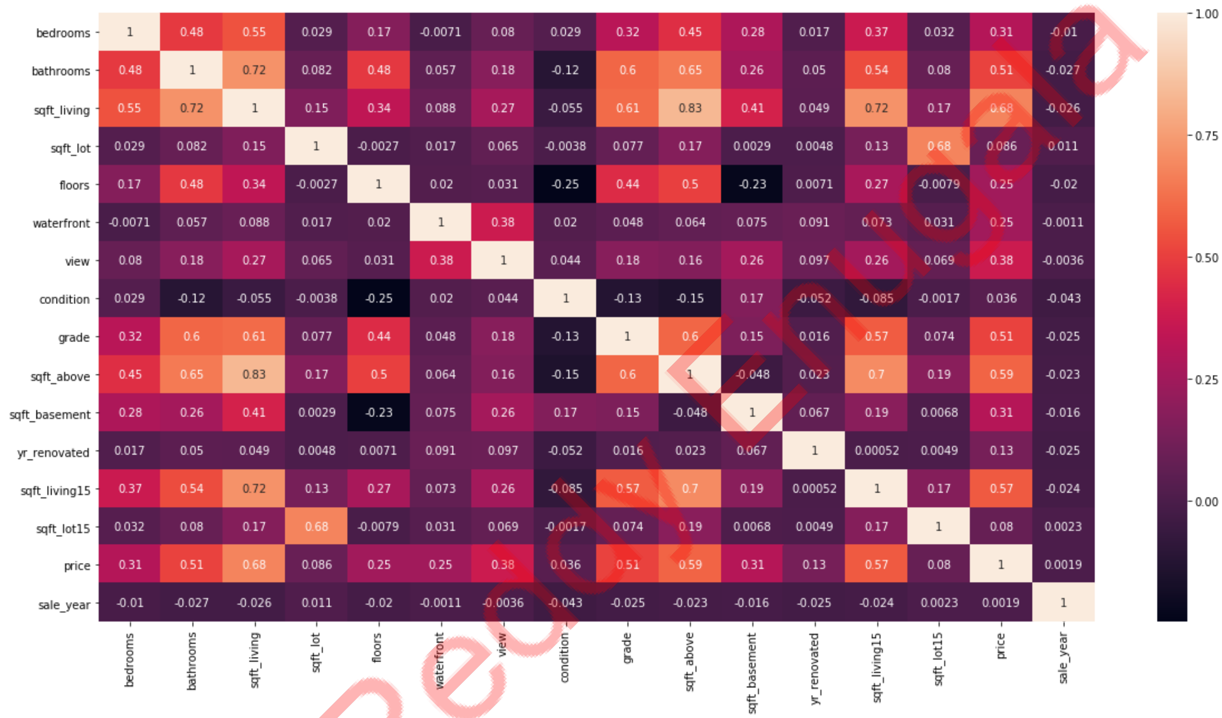
Out[113]:
```
bedrooms           int64
bathrooms        float64
sqft_living      float64
sqft_lot         float64
floors           float64
waterfront         int64
view               int64
condition          int64
grade            float64
sqft_above       float64
sqft_basement    float64
yr_renovated      object
zipcode           object
sqft_living15    float64
sqft_lot15       float64
price              int64
sale_year         object
sale_month        object
dtype: object
```

Columns 'zipcode', 'sale_month' should have dtype as object (i.e.) categorical

In [114]:
```
bm_final['yr_renovated'] = bm_final['yr_renovated'].astype(int)
bm_final['sale_year'] = bm_final['sale_year'].astype(int)
```

In [115]:
```
bmcor = bm_final.corr()
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.heatmap(bmcor,annot=True)
```

Out[115]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd9bcea320>



There is no correlation of greater than 0.7 between price and any other variable

Perfoming One-hot vector encoding for the column 'zipcode'

In [116]:
```
house_sales = pd.get_dummies(bm_final)
```

In [117]:
```
house_sales.shape
house_sales.head(10)
```

Out[117]: (19410, 32)

Out[117]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_abo
0	3	1.00	1180.0	5650.0	1.0	0	0	3	2.0	118
1	3	2.25	2570.0	7242.0	2.0	0	0	3	2.0	217
2	2	1.00	770.0	10000.0	1.0	0	0	3	1.0	77
3	4	3.00	1960.0	5000.0	1.0	0	0	5	2.0	105
4	3	2.00	1680.0	8080.0	1.0	0	0	3	3.0	168
5	4	2.25	5420.0	7620.0	1.0	0	0	3	4.0	389
6	3	2.25	1715.0	6819.0	2.0	0	0	3	2.0	171
7	3	1.50	1060.0	9711.0	1.0	0	0	3	2.0	106
9	3	2.25	1890.0	6560.0	2.0	0	0	3	2.0	189
11	2	1.00	1160.0	6000.0	1.5	0	0	4	2.0	86

10 rows × 32 columns

Train and test split

In [118]:
```
X = house_sales.drop(['price'],axis=1)
y = house_sales['price']

X_train_org, X_test_org, y_train, y_test = train_test_split(X, y, test_size=0.25
y_train.shape
y_test.shape
X_train_org.shape
X_test_org.shape
```

Out[118]: (14557,)

Out[118]: (4853,)

Out[118]: (14557, 31)

Out[118]: (4853, 31)

Scaling

In our dataset, there are outliers present. Hence, we are using StandardScaler as the MinMax scaler is very sensitive to the presence of outliers.

In [119]:
```
scaler = StandardScaler()
```

In [120]:
```
X_train_scale = scaler.fit_transform(X_train_org)
```

In [121]:
```
X_test_scale = scaler.transform(X_test_org)
```

Linear Regression

In [122]:
```
from sklearn.model_selection import GridSearchCV
model = LinearRegression()
parameters = {'normalize':[True,False]}
grid_search_lr = GridSearchCV(model,parameters, cv=6, return_train_score=True)
grid_search_lr.fit(X_train_org, y_train)
print("Best parameters: {}".format(grid_search_lr.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search_lr.best_score_))
```

Out[122]:
```
GridSearchCV(cv=6, error_score='raise-deprecating',
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='warn', n_jobs=None, param_grid={'normalize': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)

Best parameters: {'normalize': False}
Best cross-validation score: 0.7038
```

In [123]:
```
results = pd.DataFrame(grid_search_lr.cv_results_)
results
```

Out[123]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_normalize	params	spli
0	0.015868	0.003134	0.001048	0.001518	True	{'normalize': True}	
1	0.020165	0.001392	0.004478	0.000699	False	{'normalize': False}	

2 rows × 23 columns

In [124]:
```
lreg = LinearRegression(normalize = True)
lreg.fit(X_train_org, y_train)
print(lreg.score(X_train_org, y_train))
print(lreg.score(X_test_org, y_test))
```

Out[124]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)

0.7070004856644225
0.7057053631724195
```

In [125]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(lreg , X_train_org, y
scores = cross_val_score(lreg , X_train_org, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.70963421 0.71608895 0.71988844 0.69750333 0.68327092 0.69657358]
0.7038265695949354
```

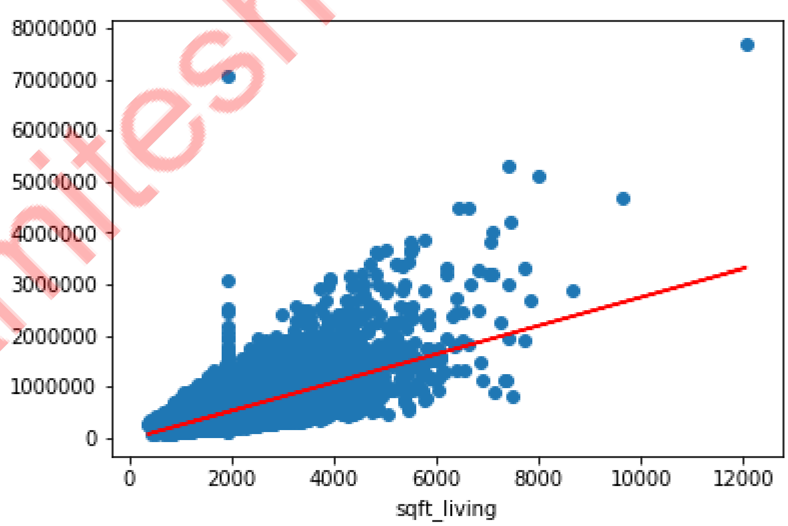
In [126]:
```
#PLOT

%matplotlib inline
import matplotlib.pyplot as plt

X_train_array = X_train_org.to_numpy()

X_train_rm = X_train_array[:,2].reshape(-1,1)
lreg.fit(X_train_rm, y_train)
y_predict = lreg.predict(X_train_rm)

plt.plot(X_train_rm, y_predict, c = 'r')
plt.scatter(X_train_rm,y_train)
plt.xlabel('sqft_living')
```

Out[126]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)

Out[126]: [<matplotlib.lines.Line2D at 0x1fd9e9eaef0>]

Out[126]: <matplotlib.collections.PathCollection at 0x1fd9e98f828>

Out[126]: Text(0.5, 0, 'sqft_living')



Linear Regression Result:

Best parameter: {'normalize': False}

Average Cross validation score: 0.7038

Test score: 0.7057

KNN Regression

In [127]:
```
grid_parms_knn = {'n_neighbors':[1,5,10,15,20]}
```

In [128]:
```
knn = KNeighborsRegressor()
grid_search_knn = GridSearchCV(knn, grid_parms_knn,cv=6,return_train_score=True,
grid_search_knn.fit(X_train_scale, y_train)
```

Out[128]:
```
GridSearchCV(cv=6, error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=None,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='warn', n_jobs=-1,
             param_grid={'n_neighbors': [1, 5, 10, 15, 20]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [129]:
```
print("Best parameters: {}".format(grid_search_knn.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search_knn.best_score_))
pd.DataFrame(grid_search_knn.cv_results_)
```

```
Best parameters: {'n_neighbors': 10}
Best cross-validation score: 0.7051
```

Out[129]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	params
0	0.355068	0.024509	4.756456	0.158985	1	{'n_neighbors': 1}
1	0.371450	0.033257	6.437256	0.259872	5	{'n_neighbors': 5}
2	0.394010	0.030371	7.297763	0.196907	10	{'n_neighbors': 10}
3	0.392970	0.030417	7.835041	0.268176	15	{'n_neighbors': 15}
4	0.392819	0.024051	8.136292	0.180613	20	{'n_neighbors': 20}

5 rows × 23 columns

In [130]:
```
knn = KNeighborsRegressor(n_neighbors = 10)
knn.fit(X_train_scale, y_train)
print(knn.score(X_train_scale, y_train))
print(knn.score(X_test_scale, y_test))
```

Out[130]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                    weights='uniform')

0.7596335157792412
0.7091584457948231
```

In [131]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(knn , X_train_scale,
scores = cross_val_score(knn , X_train_scale, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.7102932  0.71560329 0.71878694 0.71472988 0.6644096  0.70653242]
0.7050592204685852
```

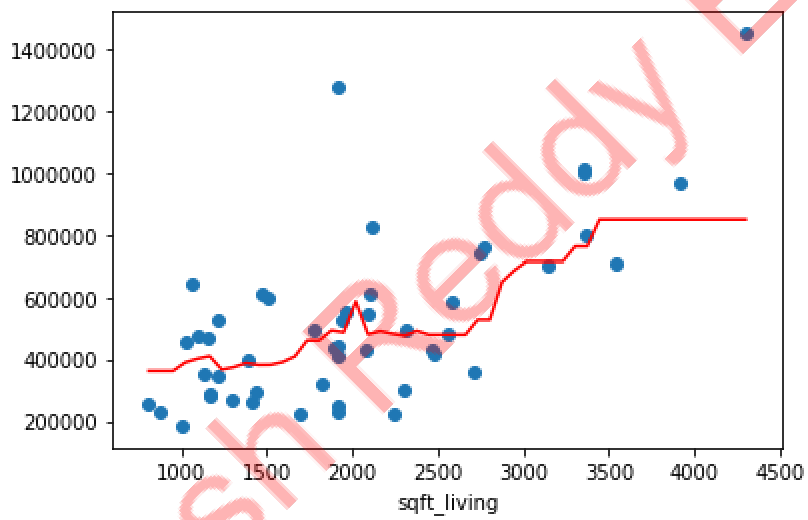
In [132]:
```
X_b = X_train_array[:50,2].reshape(-1,1)
y_b = y_train[:50]

knn_reg = KNeighborsRegressor(10)
knn_reg.fit(X_b, y_b)

X_new=np.linspace(X_b.min(), X_b.max(), 50).reshape(50, 1)
y_predict = knn_reg.predict(X_new)

plt.plot(X_new, y_predict, c = 'r')
plt.scatter(X_b, y_b)
plt.xlabel('sqft_living')
```

Out[132]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                    weights='uniform')
```

Out[132]:
```
[<matplotlib.lines.Line2D at 0x1fd9ea95a20>]
```

Out[132]:
```
<matplotlib.collections.PathCollection at 0x1fd9ea95eb8>
```

Out[132]:
```
Text(0.5, 0, 'sqft_living')
```



KNN Regression Result:

Best parameter: {n_neighbors: 10}

Average Cross validation score: 0.7050

Test score: 0.7091

Ridge Regression

In [133]:
```
grid_parms_ridge = {'alpha': [0.01, 0.1, 1, 10, 100]}
```

In [134]:
```
ridge = Ridge()
grid_search_ridge = GridSearchCV(estimator = ridge,param_grid = grid_parms_ridge
grid_search_ridge.fit(X_train_org, y_train)
print("Best parameters: {}".format(grid_search_ridge.best_params_))

print("Best cross-validation score: {:.4f}".format(grid_search_ridge.best_score_
```

Out[134]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='warn', n_jobs=-1,
             param_grid={'alpha': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

```
Best parameters: {'alpha': 1}
Best cross-validation score: 0.7044
```

In [135]:
```
ridge = Ridge(alpha = 1)
ridge.fit(X_train_org, y_train)
print(ridge.score(X_train_org, y_train))
print(ridge.score(X_test_org, y_test))
```

Out[135]:
```
Ridge(alpha=1, copy_X=True, fit_intercept=True, max_iter=None, normalize=False,
      random_state=None, solver='auto', tol=0.001)

0.7069981831976242
0.7058582864583789
```

In [136]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(ridge , X_train_org,
scores = cross_val_score(ridge , X_train_org, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.70978705 0.71599989 0.71998249 0.6977393  0.68300811 0.69661804]
0.7038558124738913
```

In [137]:
```
result_ridge = pd.DataFrame(grid_search_ridge.cv_results_)
result_ridge
```

Out[137]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_
0	0.016812	0.005477	0.003242	0.000977	0.01	{'alpha': 0.01}	0.7
1	0.018213	0.001392	0.003540	0.001892	0.1	{'alpha': 0.1}	0.7
2	0.018341	0.000349	0.002567	0.002119	1	{'alpha': 1}	0.7
3	0.016607	0.001885	0.004855	0.002448	10	{'alpha': 10}	0.7
4	0.017873	0.001684	0.001624	0.001988	100	{'alpha': 100}	0.7

5 rows × 21 columns

In [138]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(range(result_ridge.shape[0]), result_ridge['mean_train_score'], label =
plt.plot(range(result_ridge.shape[0]), result_ridge['mean_test_score'], label =
plt.xticks(range(result_ridge.shape[0]), result_ridge['param_alpha'], rotation =
plt.plot([grid_search_ridge.best_index_], result_ridge['mean_train_score'][grid_
plt.plot([grid_search_ridge.best_index_], result_ridge['mean_test_score'][grid_s
plt.grid()
plt.legend()
plt.xlabel('Alpha')
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x1fd9f88d9e8>]
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x1fd9f8664a8>]
```

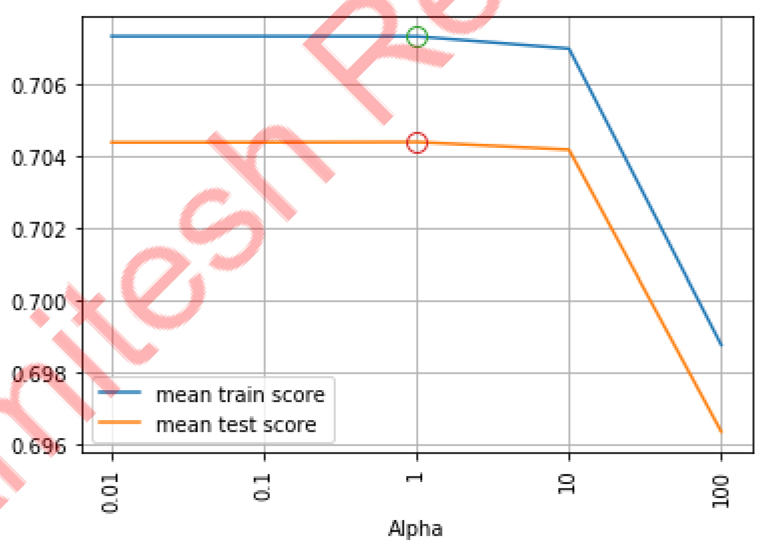
Out[138]:
```
([<matplotlib.axis.XTick at 0x1fd9ea587f0>,
  <matplotlib.axis.XTick at 0x1fd9ea56c88>,
  <matplotlib.axis.XTick at 0x1fd9ea565f8>,
  <matplotlib.axis.XTick at 0x1fd9f8983c8>,
  <matplotlib.axis.XTick at 0x1fd9f898898>],
 <a list of 5 Text xticklabel objects>)
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x1fd9f8667f0>]
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x1fd9f86d940>]
```

Out[138]:
```
<matplotlib.legend.Legend at 0x1fd9f898240>
```

Out[138]:
```
Text(0.5, 0, 'Alpha')
```



Ridge Regression Result:

Best parameter: {'alpha': 1}

Average Cross validation score: 0.7038

Test score: 0.7058

Lasso Regression

In [139]:
```
grid_parms_lasso = {'alpha': [0.01, 0.1, 1, 10,100]}
```

In [140]:
```
lasso = Lasso()
grid_search_lasso = GridSearchCV(estimator = lasso,param_grid = grid_parms_lasso,
grid_search_lasso.fit(X_train_org, y_train)
print("Best parameters: {}".format(grid_search_lasso.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search_lasso.best_score_
```

Out[140]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, positive=False,
                             precompute=False, random_state=None,
                             selection='cyclic', tol=0.0001, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'alpha': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)

Best parameters: {'alpha': 10}
Best cross-validation score: 0.7044
```

In [141]:
```
lass = Lasso(alpha = 10)
lass.fit(X_train_org, y_train)
print(lass.score(X_train_org, y_train))
print(lass.score(X_test_org, y_test))
```

Out[141]:
```
Lasso(alpha=10, copy_X=True, fit_intercept=True, max_iter=1000, normalize=Fals
e,
      positive=False, precompute=False, random_state=None, selection='cyclic',
      tol=0.0001, warm_start=False)

0.7069999591510197
0.7057416447269227
```

In [142]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(lass , X_train_org, y
scores = cross_val_score(lass , X_train_org, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.7096228  0.7160749  0.71994196 0.69756932 0.68323548 0.69658378]
0.7038380413758031
```

In [143]:
```
result_lasso = pd.DataFrame(grid_search_lasso.cv_results_)
result_lasso
```

Out[143]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_
0	3.353209	0.186871	0.007256	0.001912	0.01	{'alpha': 0.01}	0.7
1	3.576854	0.068750	0.007105	0.003076	0.1	{'alpha': 0.1}	0.7
2	1.758889	0.584525	0.005954	0.002353	1	{'alpha': 1}	0.7
3	0.592850	0.103912	0.009030	0.011775	10	{'alpha': 10}	0.7
4	0.282584	0.028904	0.006506	0.002697	100	{'alpha': 100}	0.7

5 rows × 21 columns

In [144]:
```
%matplotlib inline

plt.plot(range(result_lasso.shape[0]), result_lasso['mean_train_score'], label =
plt.plot(range(result_lasso.shape[0]), result_lasso['mean_test_score'], label =
plt.xticks(range(result_lasso.shape[0]), result_lasso['param_alpha'], rotation =
plt.plot([grid_search_lasso.best_index_], result_lasso['mean_train_score'][grid_s
plt.plot([grid_search_lasso.best_index_], result_lasso['mean_test_score'][grid_se
plt.grid()
plt.legend()
plt.xlabel('Alpha')
```

Out[144]: [<matplotlib.lines.Line2D at 0x1fd9f90f518>]

Out[144]: [<matplotlib.lines.Line2D at 0x1fd9f8ddc50>]

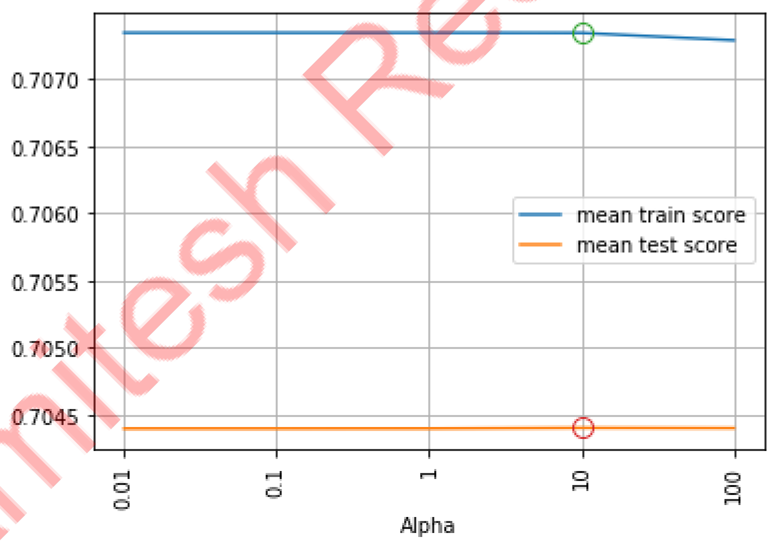
Out[144]:
```
([<matplotlib.axis.XTick at 0x1fd9f8c6c88>,
  <matplotlib.axis.XTick at 0x1fd9f8ac8d0>,
  <matplotlib.axis.XTick at 0x1fd9f8d4550>,
  <matplotlib.axis.XTick at 0x1fd9f90feb8>,
  <matplotlib.axis.XTick at 0x1fd9f9173c8>],
 <a list of 5 Text xticklabel objects>)
```

Out[144]: [<matplotlib.lines.Line2D at 0x1fd9f90fb70>]

Out[144]: [<matplotlib.lines.Line2D at 0x1fd9f8d4048>]

Out[144]: <matplotlib.legend.Legend at 0x1fd9f9200f0>

Out[144]: Text(0.5, 0, 'Alpha')



Lasso Regression Result:

Best parameter: {'alpha': 100}

Average Cross validation score: 0.7038

Test score: 0.7057

Polynominal Regression

In [145]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

In [146]:
```
param_grid_poly = {'polynomialfeatures__degree': np.arange(3)}

grid_poly = GridSearchCV(PolynomialRegression(), param_grid_poly,return_train_sc
```

In [147]:
```
grid_poly.fit(X_train_org, y_train)
```

Out[147]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('polynomialfeatures',
                                        PolynomialFeatures(degree=2,
                                                           include_bias=True,
                                                           interaction_only=Fal
se,
                                                           order='C')),
                                       ('linearregression',
                                        LinearRegression(copy_X=True,
                                                         fit_intercept=True,
                                                         n_jobs=None,
                                                         normalize=False))],
                                verbose=False),
             iid='warn', n_jobs=-1,
             param_grid={'polynomialfeatures__degree': array([0, 1, 2])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [148]:
```
print("Best parameters: {}".format(grid_poly.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_poly.best_score_))
```

```
Best parameters: {'polynomialfeatures__degree': 2}
Best cross-validation score: 0.7915
```

In [149]:
```
pol = PolynomialFeatures(degree = 2)
X_pol = pol.fit_transform(X_train_org)
Xt_pol = pol.fit_transform(X_test_org)
pol_reg = LinearRegression()
pol_reg.fit(X_pol,y_train)
print(pol_reg.score(X_pol, y_train))
print(pol_reg.score(Xt_pol, y_test))
```

Out[149]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

0.8278011097032606
0.7942424473052565
```

In [150]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(pol_reg , X_pol, y_tı
scores = cross_val_score(pol_reg , X_pol, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.78502321 0.7979335  0.78108883 0.79162343 0.809837   0.76740372]
0.7888182811260895
```

In [151]:
```
result_poly = pd.DataFrame(grid_poly.cv_results_)
result_poly
```

Out[151]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_polynomialfeatures__degree
0	0.012437	0.002573	0.002110	0.001110	0
1	0.048420	0.004095	0.007297	0.002061	1
2	1.099465	0.046975	0.064668	0.014644	2

3 rows × 21 columns

In [152]:

```
plt.plot(range(result_poly.shape[0]), result_poly['mean_train_score'], label = 'r
plt.plot(range(result_poly.shape[0]), result_poly['mean_test_score'], label = 'me
plt.xticks(range(result_poly.shape[0]), result_poly['param_polynomialfeatures__de
plt.plot([grid_poly.best_index_], result_poly['mean_train_score'][grid_poly.best_
plt.plot([grid_poly.best_index_], result_poly['mean_test_score'][grid_poly.best_i
plt.grid()
plt.xlabel('Degree')
plt.legend()
```

Out[152]:
```
[<matplotlib.lines.Line2D at 0x1fd9f990dd8>]
```

Out[152]:
```
[<matplotlib.lines.Line2D at 0x1fd9f99c240>]
```

Out[152]:
```
([<matplotlib.axis.XTick at 0x1fd9f9783c8>,
  <matplotlib.axis.XTick at 0x1fd9f976cc0>,
  <matplotlib.axis.XTick at 0x1fd9f8ddbe0>],
 <a list of 3 Text xticklabel objects>)
```

Out[152]:
```
[<matplotlib.lines.Line2D at 0x1fd9f99c630>]
```

Out[152]:
```
[<matplotlib.lines.Line2D at 0x1fd9f8dd7b8>]
```

Out[152]:
```
Text(0.5, 0, 'Degree')
```

Out[152]:
```
<matplotlib.legend.Legend at 0x1fd9f99cdd8>
```

Polynominal Regression Result:

Best parameters: {'polynomialfeatures__degree': 2}

Average Cross validation score: 0.7888

Test score: 0.7942

Linear (Simple) SVR

In [153]:
```
grid_parms_svrl = {'C': [0.01, 0.1, 1, 10, 100], 'epsilon' : [0.01, 0.1, 1, 10, 1
```

In [154]:
```
linearsvr = LinearSVR()
grid_svrl = GridSearchCV(estimator = linearsvr,param_grid = grid_parms_svrl,retu
```

In [155]:
```
grid_svrl.fit(X_train_scale,y_train)
```

Out[155]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=LinearSVR(C=1.0, dual=True, epsilon=0.0,
                                 fit_intercept=True, intercept_scaling=1.0,
                                 loss='epsilon_insensitive', max_iter=1000,
                                 random_state=None, tol=0.0001, verbose=0),
             iid='warn', n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100],
                         'epsilon': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [156]:
```
print("Best parameters: {}".format(grid_svrl.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svrl.best_score_))
```

```
Best parameters: {'C': 100, 'epsilon': 10}
Best cross-validation score: 0.5549
```

In [157]:
```
lsvr = LinearSVR(C = 100, epsilon = 1)

lsvr.fit(X_train_scale, y_train)

print(lsvr.score(X_train_scale, y_train))
print(lsvr.score(X_test_scale, y_test))
```

Out[157]:
```
LinearSVR(C=100, dual=True, epsilon=1, fit_intercept=True,
          intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000,
          random_state=None, tol=0.0001, verbose=0)
```

```
0.5676259465714262
0.5714589745919514
```

In [158]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=10)
print("Cross-validation scores:\n{}".format(cross_val_score(lsvr , X_train_scale
scores = cross_val_score(lsvr, X_train_scale, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.5438185  0.57203006 0.56323625 0.555753   0.58539792 0.58296943
 0.55636097 0.51814406 0.52171033 0.54886689]
0.5548712794044903
```

In [159]:
```
result_linearsvr = pd.DataFrame(grid_svrl.cv_results_)
result_linearsvr
```

Out[159]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	param
0	0.064522	0.005136	0.002878	0.001977	0.01	0.01	{'C 0.0 'epsilon 0.0
1	0.065998	0.004664	0.002690	0.002005	0.01	0.1	{'C 0.0 'epsilon 0.
2	0.060206	0.002599	0.002000	0.001946	0.01	1	{'C 0.0 'epsilon
3	0.064692	0.005485	0.001503	0.001773	0.01	10	{'C 0.0 'epsilon 10
4	0.063528	0.003081	0.001998	0.003066	0.01	100	{'C 0.0 'epsilon 100
5	0.061185	0.006544	0.002440	0.001994	0.1	0.01	{'C': 0. 'epsilon 0.0
6	0.065425	0.009028	0.001870	0.001979	0.1	0.1	{'C': 0. 'epsilon 0.
7	0.066129	0.007779	0.001425	0.002004	0.1	1	{'C': 0. 'epsilon
8	0.064609	0.012102	0.001606	0.002114	0.1	10	{'C': 0. 'epsilon 10
9	0.058471	0.006269	0.001238	0.001697	0.1	100	{'C': 0. 'epsilon 100
10	0.051752	0.004463	0.002725	0.001898	1	0.01	{'C': 'epsilon 0.0
11	0.054823	0.006824	0.001762	0.001708	1	0.1	{'C': 'epsilon 0.
12	0.068827	0.004589	0.002457	0.002999	1	1	{'C': 'epsilon
13	0.066519	0.004266	0.002116	0.002260	1	10	{'C': 'epsilon 10
14	0.068104	0.005309	0.002134	0.002108	1	100	{'C': 'epsilon 100

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	param
15	0.075731	0.006666	0.002213	0.001950	10	0.01	{'C': 1 'epsilon 0.0
16	0.079332	0.012926	0.002581	0.001618	10	0.1	{'C': 1 'epsilon 0.
17	0.076175	0.011934	0.001921	0.002220	10	1	{'C': 1 'epsilon
18	0.079074	0.013626	0.002255	0.001957	10	10	{'C': 1 'epsilon 1
19	0.069037	0.009133	0.002244	0.002093	10	100	{'C': 1 'epsilon 10
20	0.101198	0.016539	0.002969	0.002135	100	0.01	{'C 10 'epsilon 0.0
21	0.094971	0.011703	0.001738	0.001974	100	0.1	{'C 10 'epsilon 0.
22	0.089644	0.012913	0.001816	0.002303	100	1	{'C 10 'epsilon
23	0.086846	0.007896	0.002433	0.002096	100	10	{'C 10 'epsilon 1
24	0.086774	0.016526	0.001602	0.001690	100	100	{'C 10 'epsilon 10

25 rows × 32 columns

In [160]:
```
plt.plot(range(result_linearsvr.shape[0]), result_linearsvr['mean_train_score'],
plt.plot(range(result_linearsvr.shape[0]), result_linearsvr['mean_test_score'], 
plt.xticks(range(result_linearsvr.shape[0]), result_linearsvr['param_C'], rotati
plt.plot([grid_svrl.best_index_], result_linearsvr['mean_train_score'][grid_svrl
plt.plot([grid_svrl.best_index_], result_linearsvr['mean_test_score'][grid_svrl.
plt.grid()
plt.legend()
plt.xlabel('Alpha')
```

Out[160]:
```
[<matplotlib.lines.Line2D at 0x1fd9fa47550>]
```

Out[160]:
```
[<matplotlib.lines.Line2D at 0x1fd9fa47908>]
```

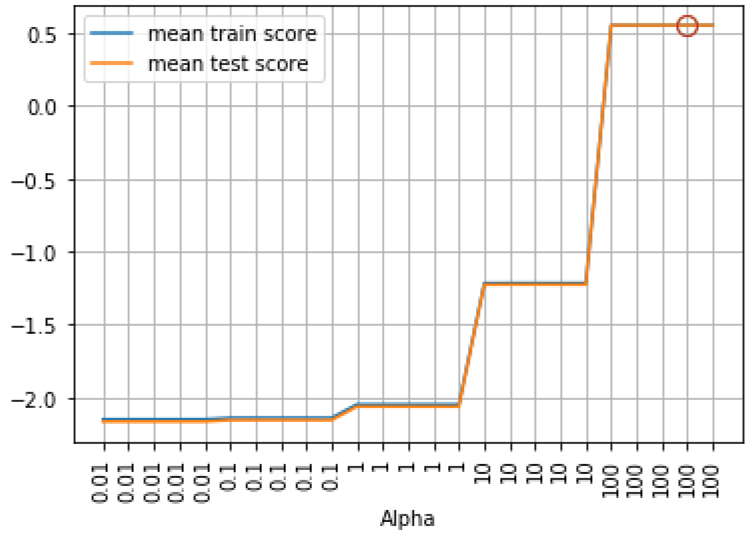
Out[160]:
```
([<matplotlib.axis.XTick at 0x1fd9f9fb9b0>,
  <matplotlib.axis.XTick at 0x1fd9f9fbcc0>,
  <matplotlib.axis.XTick at 0x1fd9f9326d8>,
  <matplotlib.axis.XTick at 0x1fd9fa47f60>,
  <matplotlib.axis.XTick at 0x1fd9fa51400>,
  <matplotlib.axis.XTick at 0x1fd9fa51940>,
  <matplotlib.axis.XTick at 0x1fd9fa51e10>,
  <matplotlib.axis.XTick at 0x1fd9fa57320>,
  <matplotlib.axis.XTick at 0x1fd9fa577f0>,
  <matplotlib.axis.XTick at 0x1fd9fa57cc0>,
  <matplotlib.axis.XTick at 0x1fd9fa5e1d0>,
  <matplotlib.axis.XTick at 0x1fd9fa5e710>,
  <matplotlib.axis.XTick at 0x1fd9fa57470>,
  <matplotlib.axis.XTick at 0x1fd9fa51a20>,
  <matplotlib.axis.XTick at 0x1fd9fa5e8d0>,
  <matplotlib.axis.XTick at 0x1fd9fa65198>,
  <matplotlib.axis.XTick at 0x1fd9fa656d8>,
  <matplotlib.axis.XTick at 0x1fd9fa65c50>,
  <matplotlib.axis.XTick at 0x1fd9fa6a208>,
  <matplotlib.axis.XTick at 0x1fd9fa6a780>,
  <matplotlib.axis.XTick at 0x1fd9fa6acf8>,
  <matplotlib.axis.XTick at 0x1fd9fa722b0>,
  <matplotlib.axis.XTick at 0x1fd9fa6add8>,
  <matplotlib.axis.XTick at 0x1fd9fa650b8>,
  <matplotlib.axis.XTick at 0x1fd9fa72208>],
 <a list of 25 Text xticklabel objects>)
```

Out[160]:
```
[<matplotlib.lines.Line2D at 0x1fd9fa47a20>]
```

Out[160]:
```
[<matplotlib.lines.Line2D at 0x1fd9fa78dd8>]
```

Out[160]:
```
<matplotlib.legend.Legend at 0x1fd9fa47898>
```

Out[160]:
```
Text(0.5, 0, 'Alpha')
```



Linear (Simple) SVR Result:

Best parameters: {'C': 100, 'epsilon': 1}

Average Cross validation score: 0.5548

Test score: 0.5714

SVR with kernel 'Linear'

In [161]:
```
grid_parms_linear = {'C': [0.01,0.1, 1, 10, 100]}
```

In [162]:
```
svr_linear = SVR(kernel='linear')
grid_svr_linear = GridSearchCV(estimator = svr_linear,param_grid = grid_parms_li
```

In [163]:
```
grid_svr_linear.fit(X_train_scale,y_train)
```

Out[163]:
```
GridSearchCV(cv=6, error_score='raise-deprecating',
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='auto_deprecated',
                           kernel='linear', max_iter=-1, shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=-1, param_grid={'C': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [164]:
```
print("Best parameters: {}".format(grid_svr_linear.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svr_linear.best_score_))
```

```
Best parameters: {'C': 100}
Best cross-validation score: 0.6314
```

In [165]:
```
svr = SVR(kernel = 'linear',C = 100)

        #train the model
svr.fit(X_train_scale, y_train)

        #evaluate the model
print(svr.score(X_train_scale, y_train))
print(svr.score(X_test_scale, y_test))
```

Out[165]:
```
SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
    gamma='auto_deprecated', kernel='linear', max_iter=-1, shrinking=True,
    tol=0.001, verbose=False)

0.6348349244684914
0.6353862043822313
```

In [166]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(svr , X_train_scale,
scores = cross_val_score(svr , X_train_scale, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.63422537 0.63669211 0.64669841 0.64797949 0.5816611  0.64105024]
0.6313844530856692
```

In [167]:
```
result_svr_linear = pd.DataFrame(grid_svr_linear.cv_results_)
result_svr_linear
```

Out[167]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_sco
0	27.320662	0.404486	3.332883	0.072339	0.01	{'C': 0.01}	-0.0547
1	27.429692	0.486008	3.346784	0.068195	0.1	{'C': 0.1}	-0.0414
2	26.220203	0.255432	3.288037	0.050573	1	{'C': 1}	0.0701
3	27.259319	1.066409	3.357370	0.044849	10	{'C': 10}	0.4578
4	15.662856	0.237806	1.837903	0.035668	100	{'C': 100}	0.6342

5 rows × 23 columns

In [168]:
```
plt.plot(range(result_svr_linear.shape[0]), result_svr_linear['mean_train_score'
plt.plot(range(result_svr_linear.shape[0]), result_svr_linear['mean_test_score']
plt.xticks(range(result_svr_linear.shape[0]), result_svr_linear['param_C'], rota
plt.plot([grid_svr_linear.best_index_], result_svr_linear['mean_train_score'][gr
plt.plot([grid_svr_linear.best_index_], result_svr_linear['mean_test_score'][gri
plt.grid()
plt.legend()
plt.xlabel('C')
```

Out[168]:
```
[<matplotlib.lines.Line2D at 0x1fd9ff6fe48>]
```

Out[168]:
```
[<matplotlib.lines.Line2D at 0x1fd9ff77240>]
```

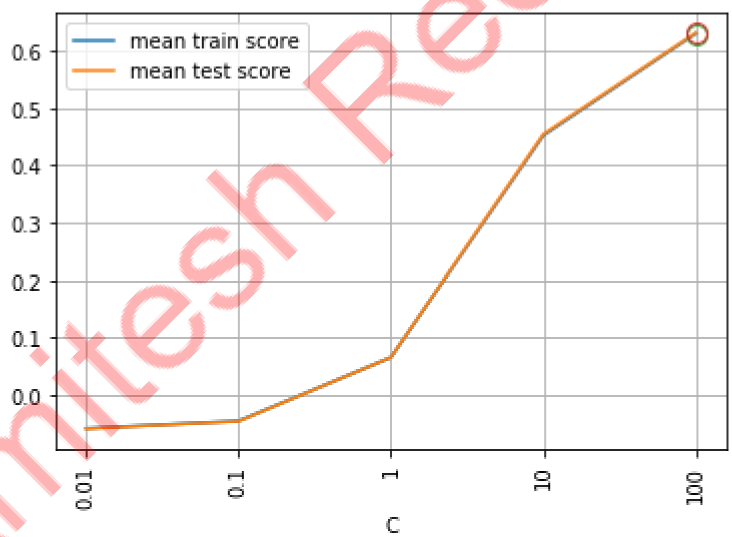
Out[168]:
```
([<matplotlib.axis.XTick at 0x1fd9ff56470>,
  <matplotlib.axis.XTick at 0x1fd9f9e8a58>,
  <matplotlib.axis.XTick at 0x1fd9f9f0da0>,
  <matplotlib.axis.XTick at 0x1fd9ff77898>,
  <matplotlib.axis.XTick at 0x1fd9ff77d68>],
 <a list of 5 Text xticklabel objects>)
```

Out[168]:
```
[<matplotlib.lines.Line2D at 0x1fd9ff77550>]
```

Out[168]:
```
[<matplotlib.lines.Line2D at 0x1fd9f9f0940>]
```

Out[168]:
```
<matplotlib.legend.Legend at 0x1fd9ff7ba90>
```

Out[168]:
```
Text(0.5, 0, 'C')
```



SVR with Kernel as 'Linear' Result:

Best parameters: {'C': 100}

Average Cross validation score: 0.6313

Test score: 0.6353

SVR with kernel 'Poly'

In [169]:
```
grid_parms_svrp = {'C': [1, 10, 100],'degree':[1,3]}
```

In [170]:
```
svr_poly = SVR(kernel='poly')
grid_svr_poly = GridSearchCV(estimator = svr_poly,param_grid = grid_parms_svrp,r
```

In [171]:
```
grid_svr_poly.fit(X_train_scale,y_train)
```

Out[171]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='auto_deprecated', kernel='poly',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='warn', n_jobs=-1,
             param_grid={'C': [1, 10, 100], 'degree': [1, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [172]:
```
print("Best parameters: {}".format(grid_svr_poly.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svr_poly.best_score_))
pd.DataFrame(grid_svr_poly.cv_results_)
```

```
Best parameters: {'C': 100, 'degree': 1}
Best cross-validation score: 0.2062
```

Out[172]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_degree	params
0	14.498907	0.041193	4.631024	0.126628	1	1	{'C': 1, 'degree': 1}
1	14.305654	0.086732	4.967052	0.100506	1	3	{'C': 1, 'degree': 3}
2	15.982145	0.304590	4.600349	0.302120	10	1	{'C': 10, 'degree': 1}
3	15.743284	0.281034	4.793856	0.259102	10	3	{'C': 10, 'degree': 3}
4	12.637298	0.144064	3.299894	0.060482	100	1	{'C': 100, 'degree': 1}
5	12.474434	0.247785	3.334822	0.039191	100	3	{'C': 100, 'degree': 3}

In [174]:
```
svr_p = SVR(kernel='poly',C=100,degree = 1)
svr_p.fit(X_train_scale, y_train)
svr_p.score(X_train_scale, y_train)
svr_p.score(X_test_scale, y_test)
```

Out[174]:
```
SVR(C=100, cache_size=200, coef0=0.0, degree=1, epsilon=0.1,
    gamma='auto_deprecated', kernel='poly', max_iter=-1, shrinking=True,
    tol=0.001, verbose=False)
```

Out[174]:
```
0.2837454148754319
```

Out[174]:
```
0.29004952647656823
```

In [175]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
#scores = cross_val_score(logreg, iris.data, iris.target)
kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(svr_p, X_train_scale
scores = cross_val_score(svr_p, X_train_scale, y_train, cv=kfold)
print(np.mean(scores))
```

```
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.
22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.
22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.
22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.
22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.
22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.
22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Cross-validation scores:
[0.25259198 0.25069119 0.25842774 0.26232613 0.21306053 0.25761964]

C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

```
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\sures\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

0.24911953373197618
```

In [176]:

```
result_svr_poly= pd.DataFrame(grid_svr_poly.cv_results_)
result_svr_poly
```

Out[176]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_degree	para
0	14.498907	0.041193	4.631024	0.126628	1	1	{'C 'degr
1	14.305654	0.086732	4.967052	0.100506	1	3	{'C 'degr
2	15.982145	0.304590	4.600349	0.302120	10	1	{'C': 'degr
3	15.743284	0.281034	4.793856	0.259102	10	3	{'C': 'degr
4	12.637298	0.144064	3.299894	0.060482	100	1	1 'degr

In [177]:
```
plt.plot(range(result_svr_poly.shape[0]), result_svr_poly['mean_train_score'], la
plt.plot(range(result_svr_poly.shape[0]), result_svr_poly['mean_test_score'], lal
plt.xticks(range(result_svr_poly.shape[0]), result_svr_poly['param_C'], rotation
plt.plot([grid_svr_poly.best_index_], result_svr_poly['mean_train_score'][grid_s
plt.plot([grid_svr_poly.best_index_], result_svr_poly['mean_test_score'][grid_svr
plt.grid()
plt.legend()
```

Out[177]:
```
[<matplotlib.lines.Line2D at 0x1fd9fff49b0>]
```

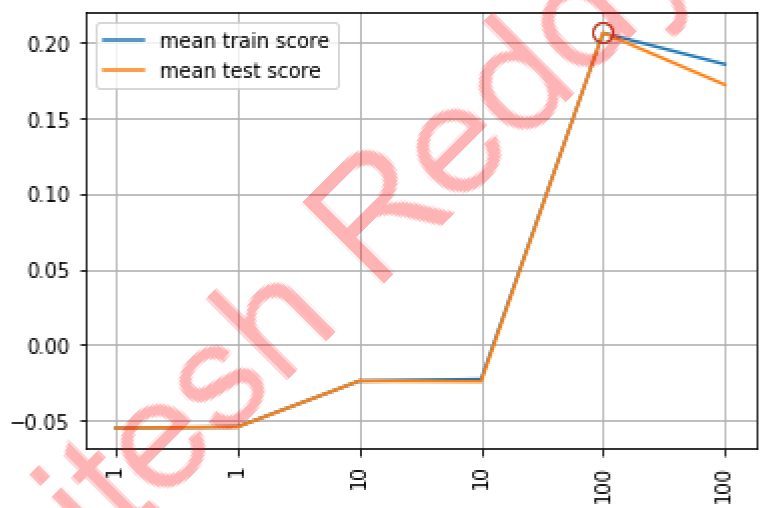
Out[177]:
```
[<matplotlib.lines.Line2D at 0x1fd9fff4d68>]
```

Out[177]:
```
([<matplotlib.axis.XTick at 0x1fd9ffd6048>,
  <matplotlib.axis.XTick at 0x1fd9ffd1908>,
  <matplotlib.axis.XTick at 0x1fd9ffd14e0>,
  <matplotlib.axis.XTick at 0x1fda0000400>,
  <matplotlib.axis.XTick at 0x1fda00008d0>,
  <matplotlib.axis.XTick at 0x1fda0000da0>],
 <a list of 6 Text xticklabel objects>)
```

Out[177]:
```
[<matplotlib.lines.Line2D at 0x1fd9fff4e80>]
```

Out[177]:
```
[<matplotlib.lines.Line2D at 0x1fd9fa5ebe0>]
```

Out[177]:
```
<matplotlib.legend.Legend at 0x1fda000aba8>
```



SVR with Kernel as 'Poly' Result:

Best parameters: {'C': 100, 'degree': 1}

Average Cross validation score: 0.2491

Test score: 0.2900

SVR with kernel 'rbf'

In [178]:
```
grid_parms_rbf = {'C': [0.1, 1, 10, 100],'gamma':[0.1, 1, 10, 100]}
```

In [179]:
```
svr_rbf = SVR(kernel='rbf')
grid_svr_rbf = GridSearchCV(estimator = svr_rbf,param_grid = grid_parms_rbf,retu
```

In [180]:
```
grid_svr_rbf.fit(X_train_scale,y_train)
```

Out[180]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='auto_deprecated', kernel='rbf',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='warn', n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': [0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [181]:
```
print("Best parameters: {}".format(grid_svr_rbf.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svr_rbf.best_score_))
```

```
Best parameters: {'C': 100, 'gamma': 0.1}
Best cross-validation score: -0.0412
```

In [193]:
```
svr_rbf = SVR(kernel='rbf',C=100,gamma=0.1)
svr_rbf.fit(X_train_scale, y_train)
svr_rbf.score(X_train_scale, y_train)
svr_rbf.score(X_test_scale, y_test)
```

Out[193]:
```
SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.1,
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Out[193]:
```
-0.032844858673138466
```

Out[193]:
```
-0.03450470646512982
```

In [183]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(svr_rbf, X_train_sca
scores = cross_val_score(svr_rbf, X_train_scale, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[-0.03381093 -0.04146735 -0.03190031 -0.03328871 -0.03862546 -0.04433172]
-0.03723741473126533
```

In [184]:
```
result_rbf = pd.DataFrame(grid_svr_rbf.cv_results_)
result_rbf
```

Out[184]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	param
0	11.330334	0.125268	3.097708	0.139693	0.1	0.1	{'C': 0. 'gamma 0.
1	12.243555	0.381897	3.180818	0.358904	0.1	1	{'C': 0. 'gamma
2	12.032121	0.478521	4.028549	0.353016	0.1	10	{'C': 0. 'gamma 1
3	13.576654	0.615906	5.750605	0.626988	0.1	100	{'C': 0. 'gamma 10
4	10.657468	0.084945	3.655548	0.062237	1	0.1	{'C': 'gamma 0.
5	11.766167	0.103654	3.543514	0.137770	1	1	{'C': 'gamma
6	11.654134	0.049942	4.158866	0.026475	1	10	{'C': 'gamma 1
7	13.023799	0.016312	5.219028	0.035598	1	100	{'C': 'gamma 10
8	10.410131	0.081175	3.290524	0.038172	10	0.1	{'C': 1 'gamma 0.
9	11.356603	0.081094	3.384267	0.019435	10	1	{'C': 1 'gamma
10	11.243234	0.070768	4.266910	0.031012	10	10	{'C': 1 'gamma 1
11	13.335964	0.142453	5.280530	0.037150	10	100	{'C': 1 'gamma 10
12	10.820697	0.037753	3.319779	0.130962	100	0.1	{'C': 10 'gamma 0.
13	11.597286	0.150764	3.535534	0.116806	100	1	{'C': 10 'gamma
14	11.307395	0.328424	3.757275	0.191096	100	10	{'C': 10 'gamma 1
15	9.973311	0.485107	3.137954	0.016946	100	100	{'C': 10 'gamma 10

In [185]:
```
plt.plot(range(result_rbf.shape[0]), result_rbf['mean_train_score'], label = 'mea
plt.plot(range(result_rbf.shape[0]), result_rbf['mean_test_score'], label = 'mean
plt.xticks(range(result_svr_poly.shape[0]), result_rbf['param_C'], rotation = 90
plt.plot([grid_svr_rbf.best_index_], result_rbf['mean_train_score'][grid_svr_rbf
plt.plot([grid_svr_rbf.best_index_], result_rbf['mean_test_score'][grid_svr_rbf.
plt.grid()
plt.legend()
```

Out[185]:
```
[<matplotlib.lines.Line2D at 0x1fda007e588>]
```

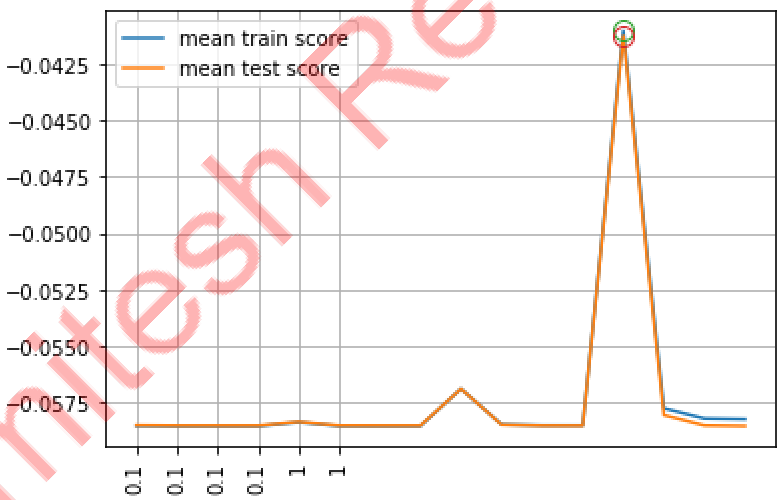
Out[185]:
```
[<matplotlib.lines.Line2D at 0x1fda007e940>]
```

Out[185]:
```
([<matplotlib.axis.XTick at 0x1fda0046518>,
  <matplotlib.axis.XTick at 0x1fda0046e10>,
  <matplotlib.axis.XTick at 0x1fd9ffab278>,
  <matplotlib.axis.XTick at 0x1fda007ee80>,
  <matplotlib.axis.XTick at 0x1fda008e4a8>,
  <matplotlib.axis.XTick at 0x1fda008e9b0>],
 <a list of 6 Text xticklabel objects>)
```

Out[185]:
```
[<matplotlib.lines.Line2D at 0x1fda0027588>]
```

Out[185]:
```
[<matplotlib.lines.Line2D at 0x1fd9f9e8da0>]
```

Out[185]:
```
<matplotlib.legend.Legend at 0x1fda008e320>
```



SVR with Kernel as 'rbf' Result:

Best parameters: {'C': 100, 'gamma': 0.1}

Average Cross validation score: -0.0372

Test score: -0.0345

Best Model for the prediction

Based on the cross validation score and the test score for above models, it is inferred that the polynominal regression is the best model to predict the house prices.

In [201]:
```
d = {'Model': ['Linear Regression', 'KNN Regression','Ridge Regression','Lasso Re
     'Cross-Validation Score': [grid_search_lr.best_score_, grid_search_knn.best_
```

In [204]:
```
result = pd.DataFrame(data=d)
result
```

Out[204]:

	Model	Cross-Validation Score
0	Linear Regression	0.703833
1	KNN Regression	0.705060
2	Ridge Regression	0.704406
3	Lasso Regression	0.704398
4	Polynominal Regression	0.791462
5	Simple SVR	0.554950
6	SVR with Linear kernel	0.631385
7	SVR with Poly kernel	0.206212
8	SVR with rbf kernel	-0.041249

In [187]:
```
pol = PolynomialFeatures(degree = 2)
X_pol = pol.fit_transform(X_train_org)
Xt_pol = pol.fit_transform(X_test_org)
pol_reg = LinearRegression()
pol_reg.fit(X_pol,y_train)
ypred = pol_reg.predict(Xt_pol)
```

Out[187]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [195]:
```python
with np.printoptions(threshold=np.inf):
    print(ypred)
```

```
[ 341597.40040728  702660.62726908  484603.9052442   429292.390924
  378652.15614089  291040.97503573  910605.41031334  921098.78258406
  460615.36765849  586550.24702009  731334.69904633  394656.6315059
  475202.15136696  426664.17566913  822988.67983069  758345.93563601
  654118.67215681  666314.05665389  298052.69265396  590541.03637684
  559036.23678609  385056.74812627  221439.20806613  547969.07705364
  337678.51230448  753764.27594396  429842.28582262  224481.99298393
  478859.63070237  431387.78808331  489926.17724677  414933.22694872
  292005.74726543  332530.19191765  852038.64820287  265889.67016193
  316979.95360975  262764.64894809  548419.99999253  384401.00567746
  457103.39279147  750351.72843836 1187277.64062186  794662.97705005
  426535.9462004   397197.19972121  689789.19742275  474511.31364627
  379114.87023796  883924.68498636 1041520.18617112  751244.89115981
  513184.7653153   339289.05826937  333057.43847961  479922.42780261
  436867.24486438  231423.59747122  268730.40000508  720082.15212843
  448312.81380294  651346.79590288  554844.33883619  762787.29889828
  962364.68571667  534996.54897123  345524.12955321  600983.06320147
  257289.54212081  288191.6280486   921666.82460959  233585.0619591
 1260677.75732788  338789.54883157  274001.08604076  540877.58997174
```