INTRODUCTION TO THE DATASET

This dataset contains property listings for Kings County (includes Seattle) in Washington, USA. We have picked this dataset from Kaggle.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('Regression_dataset.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	7129300520	20141013T000000	221900.0	3	1.0	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	NaN	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.0	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.0	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.0	1680	8080	1.0	

5 rows × 21 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id               21613 non-null int64
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21135 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21601 non-null float64
waterfront       21576 non-null float64
view             21613 non-null int64
condition        21588 non-null float64
grade            21569 non-null float64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(8), int64(12), object(1)
memory usage: 3.5+ MB
```

About the features

1. id: unique id for every listing
2. date: date on which the listing was posted
3. price: price for the property listing in dollars
4. bedrooms: number of bedrooms in the property
5. bathrooms: number of bathrooms in the property
6. sqft_living: constructed area of the property in sqaure feet
7. sqft_lot: lot size of the property in sqaure feet
8. floors: number of floors
9. waterfront: 0: property does not face a water body like a lake or pond, 1: property faces a water body like lake or pond
10. view: number of times a property has been viewed
11. condition: condition of the property on a scale of 1 to 5
12. grade: score given to the property from King's county council on a scale of 1 to 13
13. sqft_above: constructed area other than the basement in sqaure feet
14. sqft_basement: constructed area of the basement in square feet
15. yr_built: year in which the property was built
16. yr_renovated: year in which the property was renovated
17. zipcode: zipcode in which the property is located
18. lat: latitude for the location of the property

19. long: longitude for the location of the property
20. sqft_living15: constructed area for the property in square feet (updated in 2015)
21. sqft_lot15: lot size of the property in square feet (updated in 2015)

In [4]:
```
df.set_index('id', inplace=True)
# id is unique for every listing. Use it as an index to identify every listing
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21613 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21135 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21601 non-null float64
waterfront       21576 non-null float64
view             21613 non-null int64
condition        21588 non-null float64
grade            21569 non-null float64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(8), int64(11), object(1)
memory usage: 3.5+ MB
```

FEATURE: DATE

In [5]:
```
df.drop(columns = ['date'], inplace = True)
# dropping date because it is not useful for our analysis
```

FEATURE: PRICE

In [6]:
```
df['price'].describe()
```

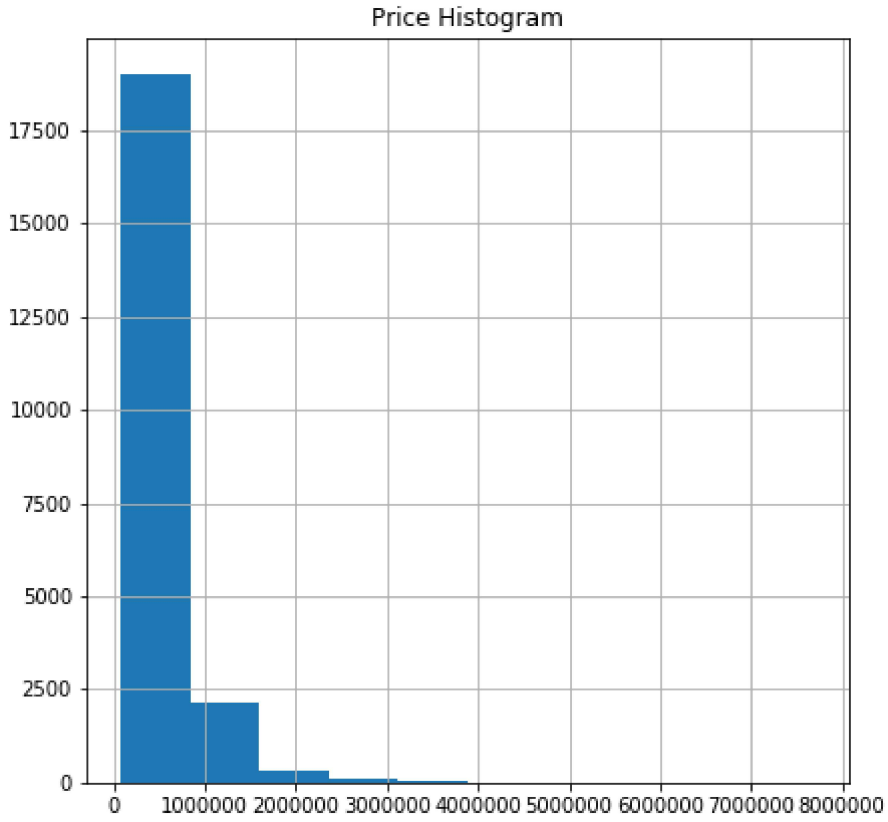
Out[6]:
```
count    2.161300e+04
mean     5.401822e+05
std      3.673622e+05
min      7.500000e+04
25%      3.219500e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

75% of the data has price between 75,000 and 645,000 dollars.

In [9]:
```
# Set the global default size of matplotlib figures
plt.rc('figure', figsize=(15, 10))

# Size of matplotlib figures that contain subplots
fizsize_with_subplots = (15, 15)

# Size of matplotlib histogram bins
bin_size = 10
```

In [10]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['price'].hist()
plt.title('Price Histogram')
```

Out[10]: Text(0.5, 1.0, 'Price Histogram')



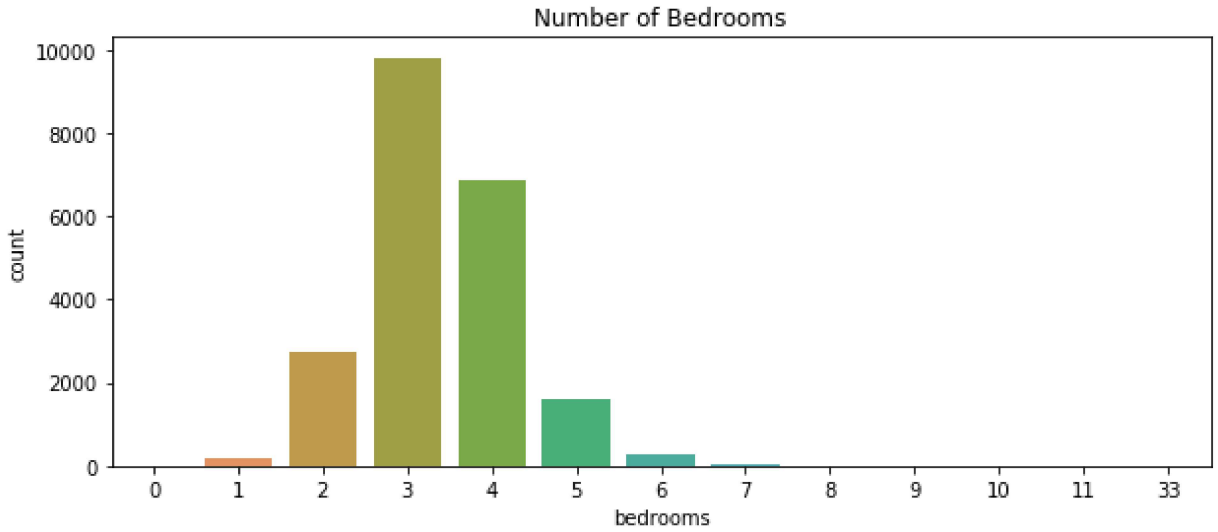
FEATURE: BEDROOMS

In [11]:
```
df['bedrooms'].value_counts()
```

Out[11]:
```
3     9824
4     6882
2     2760
5     1601
6      272
1      199
7       38
8       13
0       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

In [12]:
```
fig_dims = (10, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['bedrooms'], ax = ax).set_title('Number of Bedrooms')
```

Out[12]: Text(0.5, 1.0, 'Number of Bedrooms')



FEATURE: BATHROOMS

In [13]:
```
df[df['bathrooms'].isnull()]
```

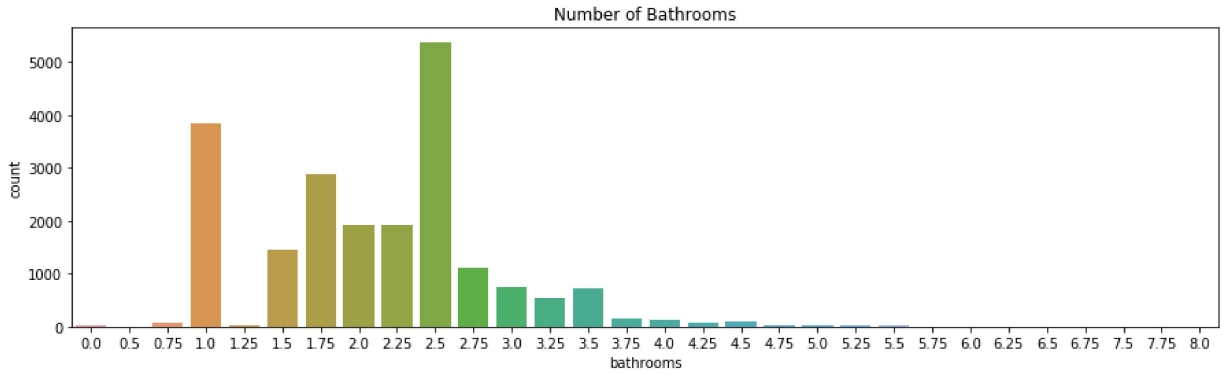
id								
6414100192	538000.0	3	NaN	2570	7242	2.0	0.0	0
1321400060	257500.0	3	NaN	1715	6819	2.0	0.0	0
6054650070	400000.0	3	NaN	1370	9680	1.0	0.0	0
6300500875	385000.0	4	NaN	1620	4980	1.0	0.0	0
2524049179	2000000.0	3	NaN	3050	44867	1.0	0.0	4
...	...	...	...	...	...	...	...	...
191100405	1580000.0	4	NaN	3410	10125	2.0	0.0	0
249000205	1540000.0	5	NaN	4470	8088	2.0	0.0	0
844000965	224000.0	3	NaN	1500	11968	1.0	0.0	0
1523300141	402101.0	2	NaN	1020	1350	2.0	0.0	0
1523300157	325000.0	2	NaN	1020	1076	2.0	0.0	0

In [14]:
```
df['bathrooms'].value_counts()
```

Out[14]:
```
2.50    5380
1.00    3852
1.75    2869
2.00    1930
2.25    1909
1.50    1446
2.75    1109
3.00     753
3.50     731
3.25     535
3.75     142
4.00     136
4.50     100
4.25      77
0.75      63
5.00      21
4.75      20
5.25      12
5.50      10
```

In [15]:
```
import seaborn as sns

fig_dims = (15, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['bathrooms'], ax = ax).set_title('Number of Bathrooms')
```

Out[15]:
```
Text(0.5, 1.0, 'Number of Bathrooms')
```



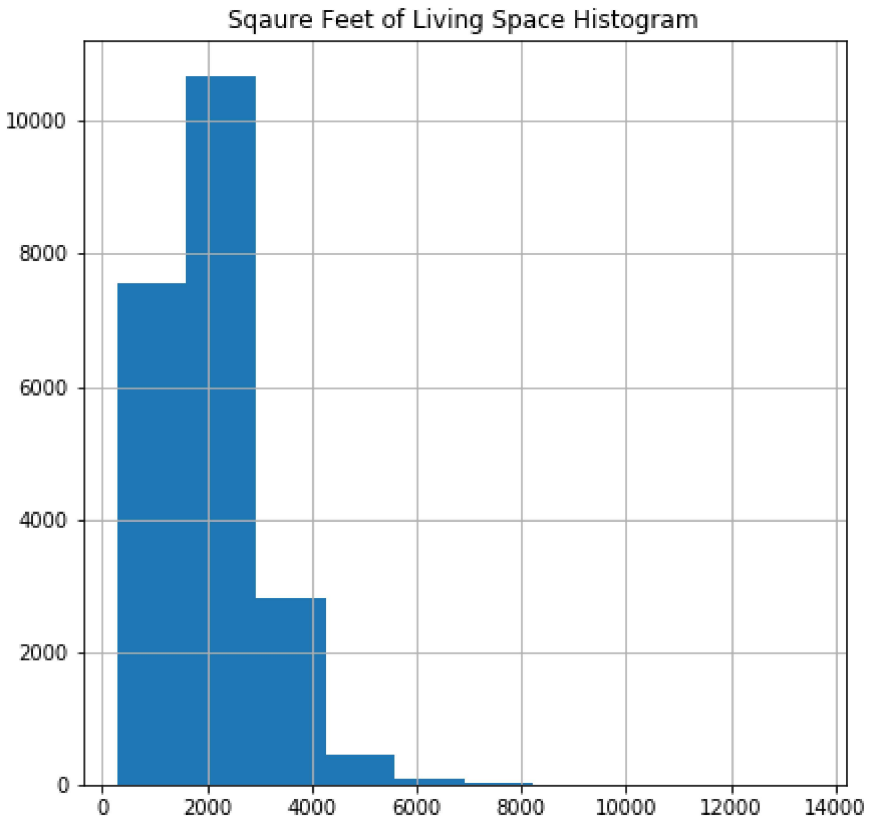
FEATURE: SQFT_LIVING

In [16]:
```
df['sqft_living'].describe()
```

Out[16]:
```
count    21613.000000
mean      2079.899736
std        918.440897
min        290.000000
25%       1427.000000
50%       1910.000000
75%       2550.000000
max      13540.000000
Name: sqft_living, dtype: float64
```

In [17]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['sqft_living'].hist()
plt.title('Sqaure Feet of Living Space Histogram')
```

Out[17]:
```
Text(0.5, 1.0, 'Sqaure Feet of Living Space Histogram')
```



75% of the data has sqaure feet of living spave between 290 and 2550.

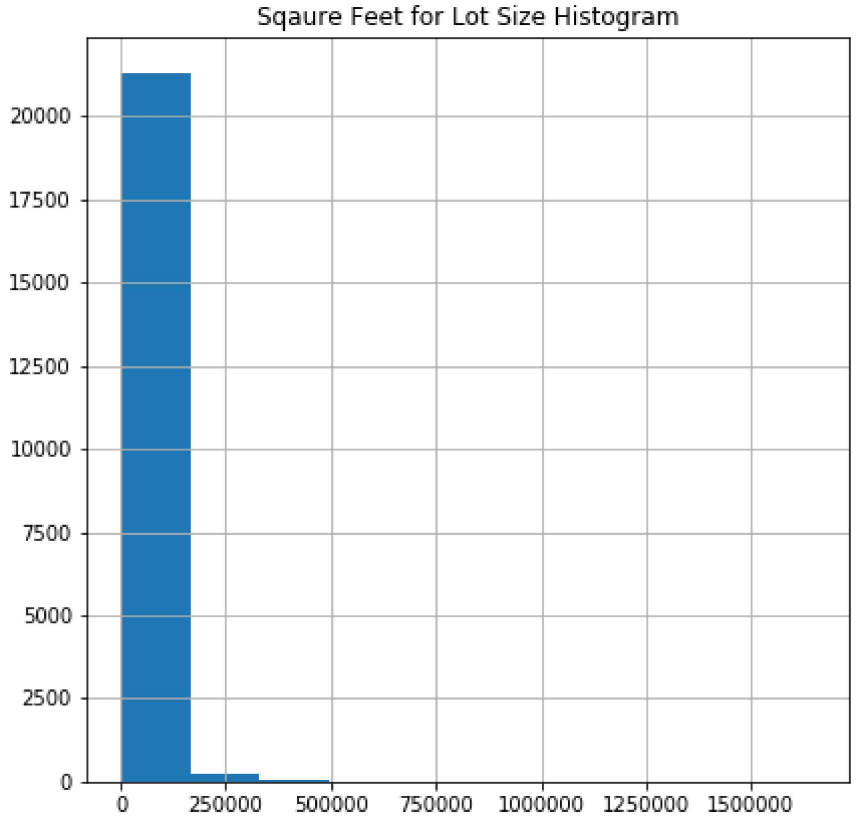
FEATURE: SQFT_LOT

In [18]:
```
df['sqft_lot'].describe()
```

Out[18]:
```
count    2.161300e+04
mean     1.510697e+04
std      4.142051e+04
min      5.200000e+02
25%      5.040000e+03
50%      7.618000e+03
75%      1.068800e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```

In [19]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['sqft_lot'].hist()
plt.title('Sqaure Feet for Lot Size Histogram')
```

Out[19]:
```
Text(0.5, 1.0, 'Sqaure Feet for Lot Size Histogram')
```



75% of the data has sqaure feet for the lot size between 520 and 10,688.

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21613 entries, 7129300520 to 1523300157
Data columns (total 19 columns):
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21135 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21601 non-null float64
waterfront       21576 non-null float64
view             21613 non-null int64
condition        21588 non-null float64
grade            21569 non-null float64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(8), int64(11)
memory usage: 3.3 MB
```

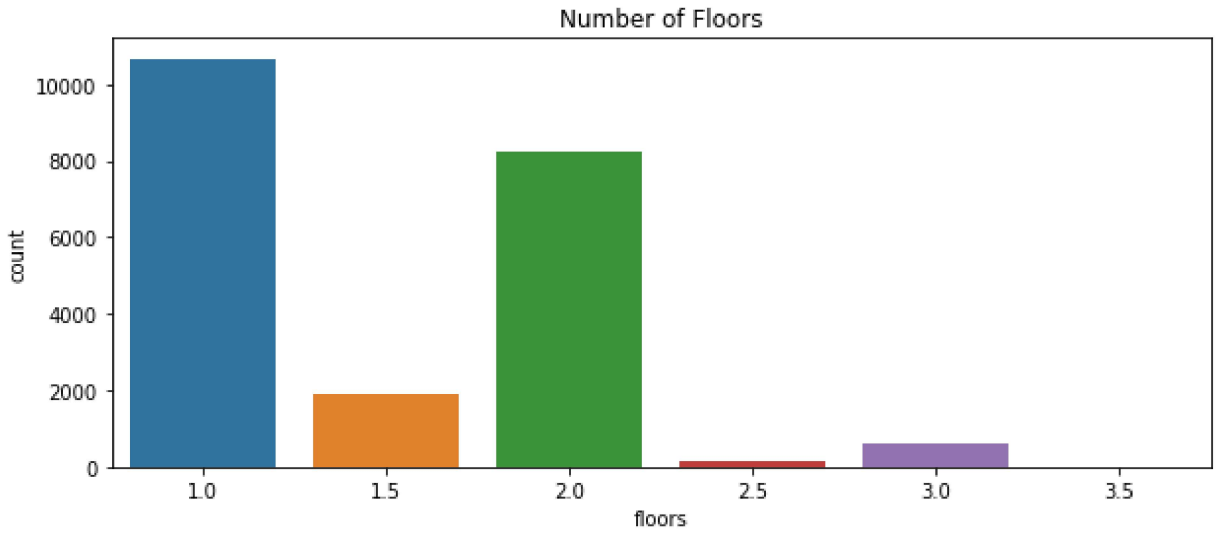
FEATURE: FLOORS

In [23]:
```
df['floors'].unique()
```

Out[23]:
```
array([1. , 2. , 1.5, 3. , 2.5, nan, 3.5])
```

In [24]:
```
fig_dims = (10, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['floors'], ax = ax).set_title('Number of Floors')
```

Out[24]:
```
Text(0.5, 1.0, 'Number of Floors')
```



In [25]:
```
df['floors'].value_counts()
```

Out[25]:
```
1.0    10672
2.0     8238
1.5     1910
3.0      612
2.5      161
3.5        8
Name: floors, dtype: int64
```

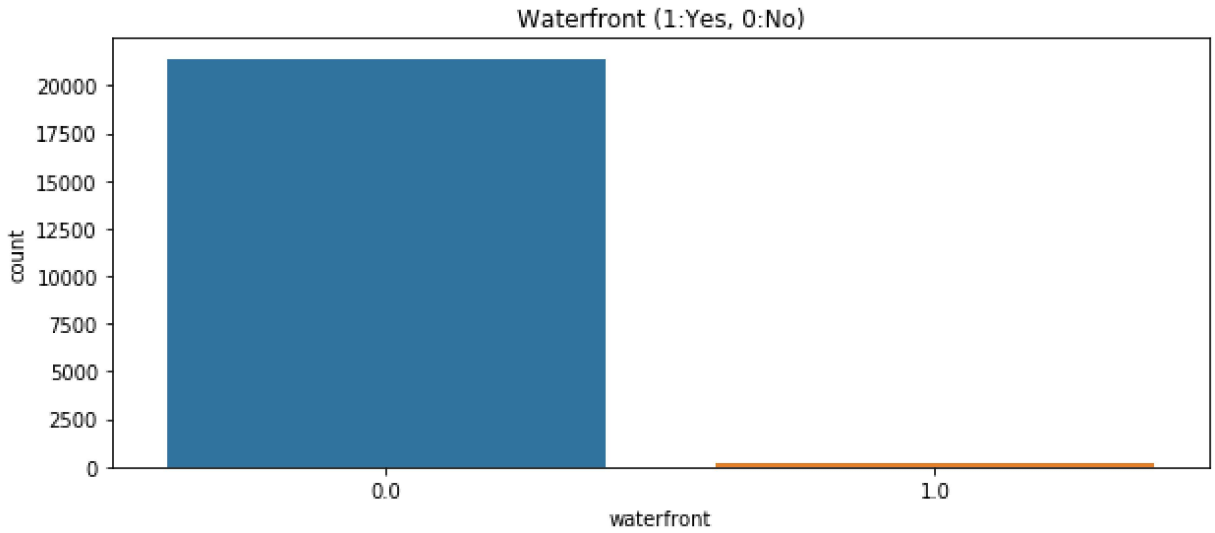
FEATURE: WATERFRONT

In [26]:
```
df['waterfront'].unique()
```

Out[26]:
```
array([ 0.,  1., nan])
```

In [27]:
```
fig_dims = (10, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['waterfront'], ax = ax).set_title('Waterfront (1:Yes, 0:No)')
```

Out[27]:
```
Text(0.5, 1.0, 'Waterfront (1:Yes, 0:No)')
```



In [28]:
```
df['waterfront'].value_counts()
```

Out[28]:
```
0.0    21413
1.0      163
Name: waterfront, dtype: int64
```

163 properties with a waterfront

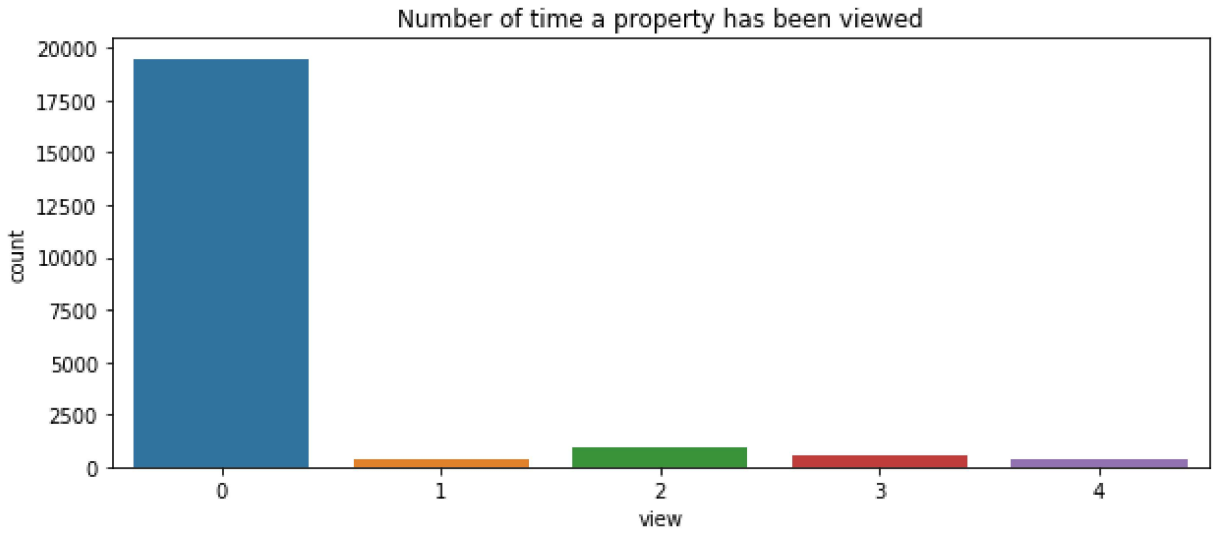
FEATURE: VIEW

In [29]:
```
df['view'].unique()
```

Out[29]:
```
array([0, 3, 4, 2, 1], dtype=int64)
```

In [30]:
```
fig_dims = (10, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['view'], ax = ax).set_title('Number of time a property has been
```

Out[30]:
```
Text(0.5, 1.0, 'Number of time a property has been viewed')
```



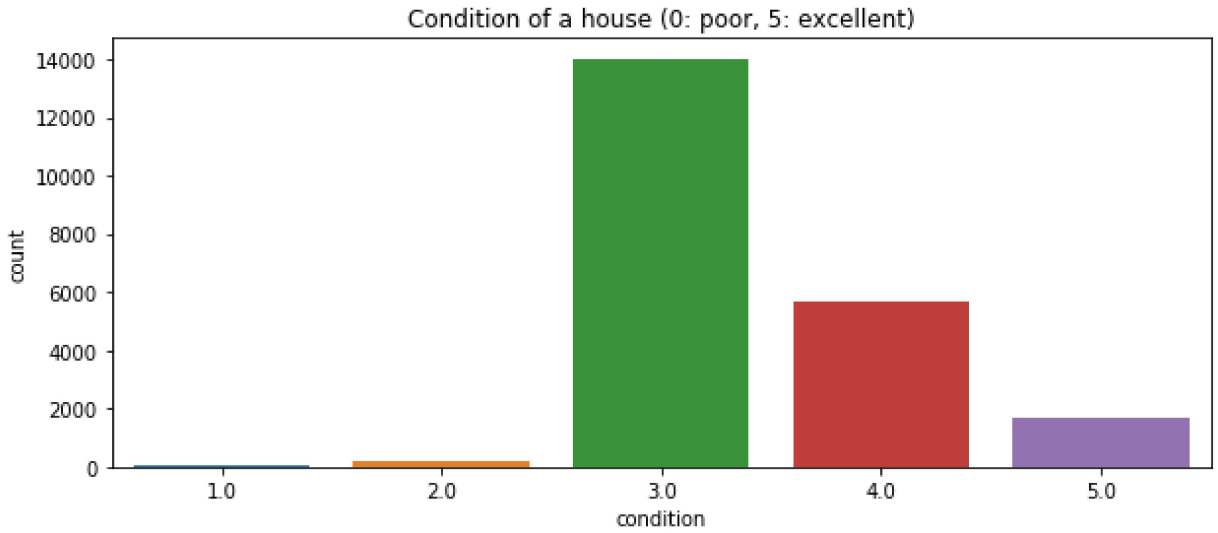
In [31]:
```
df['view'].value_counts()
```

Out[31]:
```
0    19489
2      963
3      510
1      332
4      319
Name: view, dtype: int64
```

FEATURE: CONDITION

In [32]:
```
fig_dims = (10, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['condition'], ax = ax).set_title('Condition of a house (0: poor
```

Out[32]:
```
Text(0.5, 1.0, 'Condition of a house (0: poor, 5: excellent)')
```



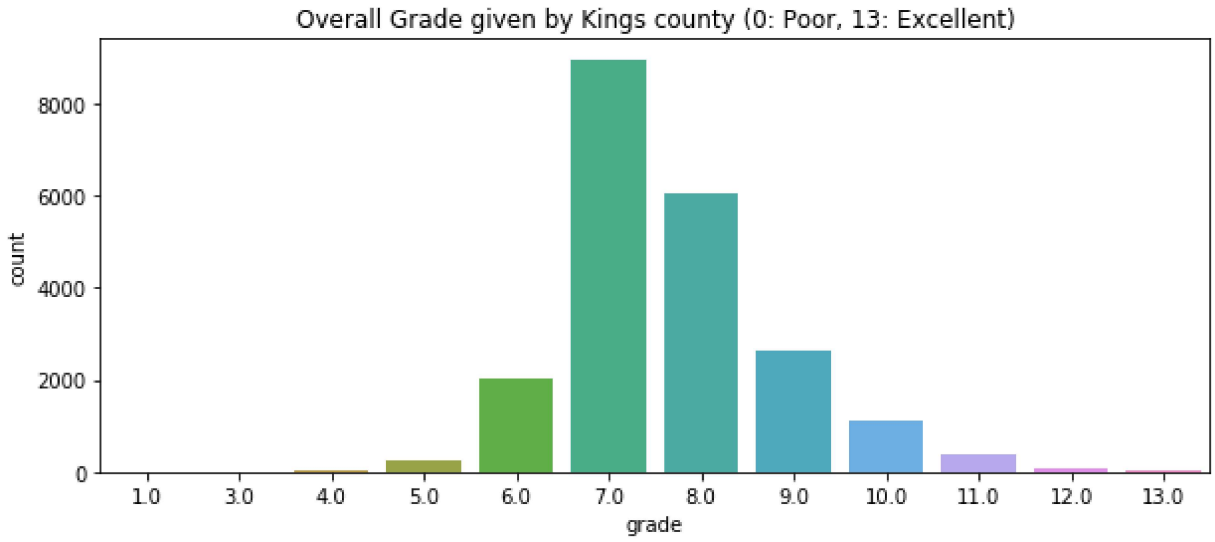
In [33]:
```
df['condition'].value_counts()
```

Out[33]:
```
3.0    14017
4.0     5672
5.0     1697
2.0      172
1.0       30
Name: condition, dtype: int64
```

FEATURE: GRADE

In [34]:
```
fig_dims = (10, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(df['grade'], ax = ax).set_title('Overall Grade given by Kings coun
```

Out[34]:
```
Text(0.5, 1.0, 'Overall Grade given by Kings county (0: Poor, 13: Excellent)')
```



In [35]:
```
df['grade'].value_counts()
```

Out[35]:
```
7.0     8964
8.0     6055
9.0     2612
6.0     2029
10.0    1133
11.0     398
5.0      242
12.0      90
4.0       29
13.0      13
3.0        3
1.0        1
Name: grade, dtype: int64
```

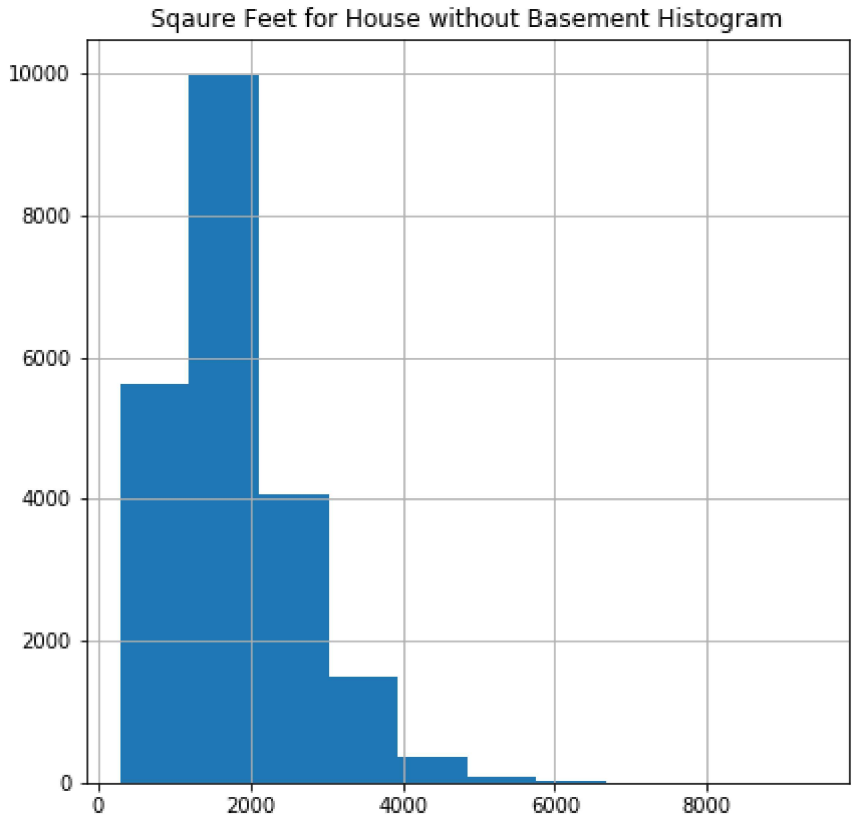
FEATURE: SQFT_ABOVE

In [36]:
```
df['sqft_above'].describe()
```

Out[36]:
```
count    21613.000000
mean      1788.390691
std        828.090978
min        290.000000
25%       1190.000000
50%       1560.000000
75%       2210.000000
max       9410.000000
Name: sqft_above, dtype: float64
```

In [37]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['sqft_above'].hist()
plt.title('Sqaure Feet for House without Basement Histogram')
```

Out[37]:
```
Text(0.5, 1.0, 'Sqaure Feet for House without Basement Histogram')
```



75% of the dataset has square feet for house without basement between 290 and 2210.

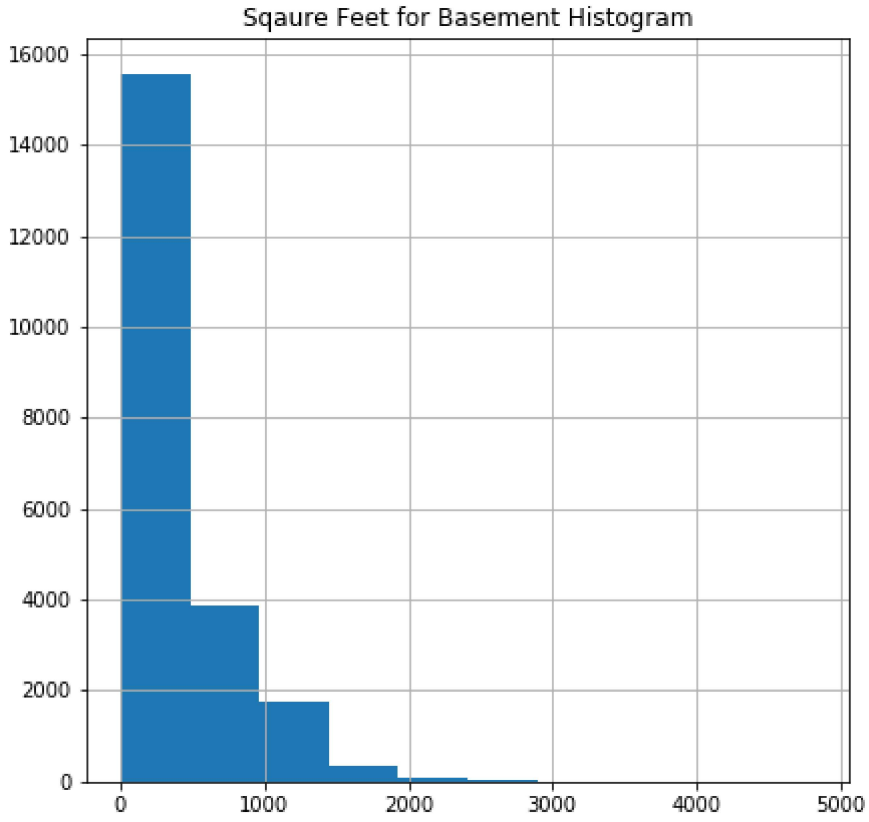
FEATURE: SQFT_BASEMENT

In [38]:
```
df['sqft_basement'].describe()
```

Out[38]:
```
count    21613.000000
mean       291.509045
std        442.575043
min          0.000000
25%          0.000000
50%          0.000000
75%        560.000000
max       4820.000000
Name: sqft_basement, dtype: float64
```

In [39]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['sqft_basement'].hist()
plt.title('Sqaure Feet for Basement Histogram')
```

Out[39]: Text(0.5, 1.0, 'Sqaure Feet for Basement Histogram')



75% of the dataset has square feet for the basement between 0 and 560.

FEATURE: YR_BUILT

In [40]:
```
df['yr_built'].value_counts()
```

Out[40]:
```
2014    559
2006    454
2005    450
2004    433
2003    422
       ...
1933     30
1901     29
1902     27
1935     24
1934     21
Name: yr_built, Length: 116, dtype: int64
```

Listings from the year 1934 to 2014.

FEATURE: YR_RENOVATED

In [41]:
```
df['yr_renovated'].value_counts()
```

Out[41]:
```
0       20699
2014       91
2013       37
2003       36
2000       35
          ...
1934        1
1959        1
1951        1
1948        1
1944        1
Name: yr_renovated, Length: 70, dtype: int64
```

20699 properties have not been renovated since they were built.

FEATURE: ZIP CODE

In [42]:
```
len(df['zipcode'].unique())
```

Out[42]:
```
70
```

70 unique zip codes in the data set.

FEATURES: LAT, LONG

In [7]:
```
df.drop(columns = ['lat', 'long'], inplace = True)
```

Dropping Latitude, Longitude as they are not much useful for our analysis.

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21613 entries, 7129300520 to 1523300157
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21613 non-null  float64
 1   bedrooms       21613 non-null  int64  
 2   bathrooms      21135 non-null  float64
 3   sqft_living    21613 non-null  int64  
 4   sqft_lot       21613 non-null  int64  
 5   floors         21601 non-null  float64
 6   waterfront     21576 non-null  float64
 7   view           21613 non-null  int64  
 8   condition      21588 non-null  float64
 9   grade          21569 non-null  float64
 10  sqft_above     21613 non-null  int64  
 11  sqft_basement  21613 non-null  int64  
 12  yr_built       21613 non-null  int64  
 13  yr_renovated   21613 non-null  int64  
 14  zipcode        21613 non-null  int64  
 15  sqft_living15  21613 non-null  int64  
 16  sqft_lot15     21613 non-null  int64  
dtypes: float64(6), int64(11)
memory usage: 3.0 MB
```

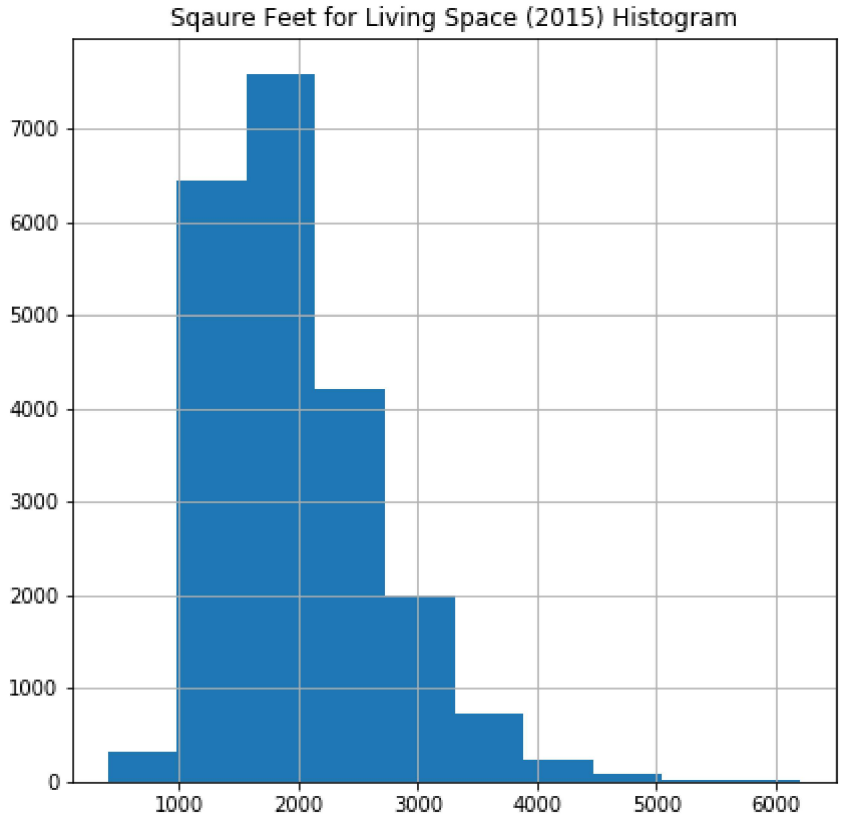
FEATURE: SQFT_LIVING15

In [45]:
```
df['sqft_living15'].describe()
```

Out[45]:
```
count    21613.000000
mean      1986.552492
std        685.391304
min        399.000000
25%       1490.000000
50%       1840.000000
75%       2360.000000
max       6210.000000
Name: sqft_living15, dtype: float64
```

In [46]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['sqft_living15'].hist()
plt.title('Sqaure Feet for Living Space (2015) Histogram')
```

Out[46]: Text(0.5, 1.0, 'Sqaure Feet for Living Space (2015) Histogram')



75% of the dataset has sqaure feet for living space when updated in 2015 between 399 and 2360.

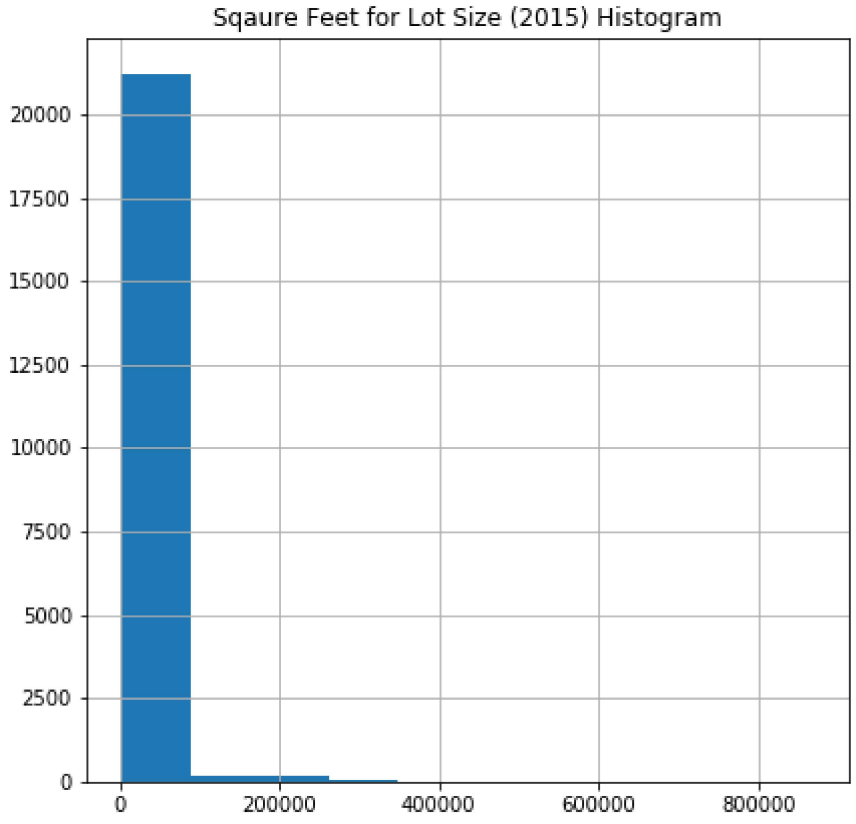
FEATURE: SQFT_LOT15

In [47]:
```
df['sqft_lot15'].describe()
```

Out[47]:
```
count     21613.000000
mean      12768.455652
std       27304.179631
min         651.000000
25%        5100.000000
50%        7620.000000
75%       10083.000000
max      871200.000000
Name: sqft_lot15, dtype: float64
```

In [48]:
```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (2, 2)

plt.subplot2grid(fig_dims, (0, 0))
df['sqft_lot15'].hist()
plt.title('Sqaure Feet for Lot Size (2015) Histogram')
```

Out[48]:
```
Text(0.5, 1.0, 'Sqaure Feet for Lot Size (2015) Histogram')
```



75% of the dataset has square feet for lot size when updated in 2015 between 651 and 10,083.

IMPUTING NULL VALUES AND FEATURE TRANSFORMATION

In [51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21613 entries, 7129300520 to 1523300157
Data columns (total 17 columns):
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21135 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21601 non-null float64
waterfront       21576 non-null float64
view             21613 non-null int64
condition        21588 non-null float64
grade            21569 non-null float64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(6), int64(11)
memory usage: 3.6 MB
```

In [52]:
```
df.isna().sum()
```

Out[52]:
```
price             0
bedrooms          0
bathrooms       478
sqft_living       0
sqft_lot          0
floors           12
waterfront       37
view              0
condition        25
grade            44
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated      0
zipcode           0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

IMPUTING VALUES FOR NUMBER OF FLOORS

In [9]:
```
train_grouped=df.groupby(['bedrooms', 'sqft_living15'])
df['floors'] = df['floors'].fillna(train_grouped['floors'].transform('median'))
```

In [10]:
```
df['floors'].value_counts()
```

Out[10]:
```
1.00    10679
2.00     8240
1.50     1912
3.00      612
2.50      161
3.50        8
1.75        1
Name: floors, dtype: int64
```

IMPUTING VALUES FOR NUMBER OF BATHROOMS

In [11]:
```
train_grouped=df.groupby(['bedrooms', 'floors'])
df['bathrooms'] = df['bathrooms'].fillna(train_grouped['bathrooms'].transform('m
```

In [9]:
```
df['bathrooms'].value_counts()
```

Out[9]:
```
2.50    5570
1.00    3867
1.75    3008
2.25    1972
2.00    1962
1.50    1447
2.75    1109
3.00     757
3.50     733
3.25     567
3.75     142
4.00     136
4.50     100
4.25      77
0.75      63
5.00      21
4.75      20
5.25      12
5.50      10
```

IMPUTING VALUES FOR WATERFRONT

In [12]:
```
train_grouped=df.groupby(['zipcode', 'bedrooms', 'bathrooms'])
df['waterfront'] = df['waterfront'].fillna(train_grouped['waterfront'].transform
```

In [11]:
```
df['waterfront'].value_counts()
```

Out[11]:
```
0.0    21449
1.0      163
Name: waterfront, dtype: int64
```

ADDING A NEW FEATURE 'AGE OF THE PROPERTY'

Age of the property can be computed by subtracting the year in which the property was built from 2015. We subtract from 2015 because the listings are from that year.

In [13]:
```
df['property_age'] = ''
```

In [14]:
```
for i in range(0,len(df)):
    df.iloc[i,17] = 2015 - df.iloc[i,12]
```

ADDING A NEW FEATURE 'NUMBERS OF YEARS SINCE RENOVATION'

In [15]:
```
df['yr_last_renovated'] = ''
```

In [16]:
```
for i in range(0,len(df)):
    if df.iloc[i,13] == 0:
        df.iloc[i,18] = 0
    else:
        df.iloc[i,18] = 2015 - df.iloc[i,13]
```

In [17]:
```
df.head()
```

Out[17]:

id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditi
7129300520	221900.0	3	1.0	1180	5650	1.0	0.0	0	3
6414100192	538000.0	3	2.5	2570	7242	2.0	0.0	0	3
5631500400	180000.0	2	1.0	770	10000	1.0	0.0	0	3
2487200875	604000.0	4	3.0	1960	5000	1.0	0.0	0	5
1954400510	510000.0	3	2.0	1680	8080	1.0	0.0	0	3

Now, we can drop yr_built and yr_renovated. Instead, use property_age and yr_last_renovated for our analysis.

In [18]:
```
df.drop(columns = ['yr_built', 'yr_renovated'], inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21613 entries, 7129300520 to 1523300157
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   price              21613 non-null  float64
 1   bedrooms           21613 non-null  int64  
 2   bathrooms          21613 non-null  float64
 3   sqft_living        21613 non-null  int64  
 4   sqft_lot           21613 non-null  int64  
 5   floors             21613 non-null  float64
 6   waterfront         21612 non-null  float64
 7   view               21613 non-null  int64  
 8   condition          21588 non-null  float64
 9   grade              21569 non-null  float64
 10  sqft_above         21613 non-null  int64  
 11  sqft_basement      21613 non-null  int64  
 12  zipcode            21613 non-null  int64  
 13  sqft_living15      21613 non-null  int64  
```

IMPUTING VALUES FOR CONDITION

In [19]:
```
train_grouped=df.groupby(['property_age', 'bedrooms', 'bathrooms'])
df['condition'] = df['condition'].fillna(train_grouped['condition'].transform('m
```

In [20]:
```
df['condition'].value_counts()
```

Out[20]:
```
3.0    14031
4.0     5680
5.0     1697
2.0      172
1.0       30
3.5        1
Name: condition, dtype: int64
```

IMPUTING VALUES FOR GRADE

In [21]:
```
train_grouped=df.groupby(['property_age', 'bedrooms', 'bathrooms'])
df['grade'] = df['grade'].fillna(train_grouped['grade'].transform('median'))
```

In [20]:
```
df['grade'].value_counts()
```

Out[20]:
```
7.0     8986
8.0     6064
9.0     2616
6.0     2033
10.0    1134
11.0     398
5.0      242
12.0      90
4.0       29
13.0      13
3.0        3
6.5        2
1.0        1
Name: grade, dtype: int64
```

In [22]:
```
df.drop(df[df['grade'].isnull()].index , inplace=True)
df.drop(df[df['condition'].isnull()].index , inplace=True)
df.drop(df[df['waterfront'].isnull()].index , inplace=True)
```

In [22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21608 entries, 7129300520 to 1523300157
Data columns (total 17 columns):
price                21608 non-null float64
bedrooms             21608 non-null int64
bathrooms            21608 non-null float64
sqft_living          21608 non-null int64
sqft_lot             21608 non-null int64
floors               21608 non-null float64
waterfront           21608 non-null float64
view                 21608 non-null int64
condition            21608 non-null float64
grade                21608 non-null float64
sqft_above           21608 non-null int64
sqft_basement        21608 non-null int64
zipcode              21608 non-null int64
sqft_living15        21608 non-null int64
sqft_lot15           21608 non-null int64
property_age         21608 non-null int64
```

ONE HOT ENCODING FOR ZIP CODE

In [23]:
```
cols = pd.get_dummies(df['zipcode'], prefix= 'zip_code')
df[cols.columns] = cols
df.drop('zipcode', axis = 1, inplace = True)
# CREATED ONE HOT FOR ZIP CODE
```

In [24]:
```
df.columns
```

Out[24]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'sqft_living15', 'sqft_lot15', 'property_age',
       'yr_last_renovated', 'zip_code_98001', 'zip_code_98002',
       'zip_code_98003', 'zip_code_98004', 'zip_code_98005', 'zip_code_9800
6',
       'zip_code_98007', 'zip_code_98008', 'zip_code_98010', 'zip_code_9801
1',
       'zip_code_98014', 'zip_code_98019', 'zip_code_98022', 'zip_code_9802
3',
       'zip_code_98024', 'zip_code_98027', 'zip_code_98028', 'zip_code_9802
9',
       'zip_code_98030', 'zip_code_98031', 'zip_code_98032', 'zip_code_9803
3',
       'zip_code_98034', 'zip_code_98038', 'zip_code_98039', 'zip_code_9804
0',
       'zip_code_98042', 'zip_code_98045', 'zip_code_98052', 'zip_code_9805
3',
       'zip_code_98055', 'zip_code_98056', 'zip_code_98058', 'zip_code_9805
9',
```

DROPPING SQFT_LIVING AND SQFT_LOT, INSTEAD USE THE UPDATED VALUES FOR THESE FEATURES : SQFT_LIVING15, SQFT_LOT15.

The value for these features were updated in 2015.

In [25]:
```
df.drop(columns = ['sqft_living', 'sqft_lot'], inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21608 entries, 7129300520 to 1523300157
Data columns (total 84 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   price              21608 non-null  float64
 1   bedrooms           21608 non-null  int64  
 2   bathrooms          21608 non-null  float64
 3   floors             21608 non-null  float64
 4   waterfront         21608 non-null  float64
 5   view               21608 non-null  int64  
 6   condition          21608 non-null  float64
 7   grade              21608 non-null  float64
 8   sqft_above         21608 non-null  int64  
 9   sqft_basement      21608 non-null  int64  
 10  sqft_living15      21608 non-null  int64  
 11  sqft_lot15         21608 non-null  int64  
 12  property_age       21608 non-null  object 
 13  yr_last_renovated  21608 non-null  object 
```

REGRESSION MODELS AND ANALYSIS

PRICE IS TARGET VARIABLE

In [26]:
```
X = df.drop(['price'], axis = 1)
y = df['price']
```

In [30]:
```
len(X.columns)
```

Out[30]:
```
83
```

In [27]:
```
%matplotlib inline
from pandas.plotting import scatter_matrix

attributes = X.columns[:5]
scatter_matrix(X[attributes], figsize = (15,15), c = y, alpha = 0.8, marker = 'O
```

Out[27]:
```
array([[<AxesSubplot:xlabel='bedrooms', ylabel='bedrooms'>,
        <AxesSubplot:xlabel='bathrooms', ylabel='bedrooms'>,
        <AxesSubplot:xlabel='floors', ylabel='bedrooms'>,
        <AxesSubplot:xlabel='waterfront', ylabel='bedrooms'>,
        <AxesSubplot:xlabel='view', ylabel='bedrooms'>],
       [<AxesSubplot:xlabel='bedrooms', ylabel='bathrooms'>,
        <AxesSubplot:xlabel='bathrooms', ylabel='bathrooms'>,
        <AxesSubplot:xlabel='floors', ylabel='bathrooms'>,
        <AxesSubplot:xlabel='waterfront', ylabel='bathrooms'>,
        <AxesSubplot:xlabel='view', ylabel='bathrooms'>],
       [<AxesSubplot:xlabel='bedrooms', ylabel='floors'>,
        <AxesSubplot:xlabel='bathrooms', ylabel='floors'>,
        <AxesSubplot:xlabel='floors', ylabel='floors'>,
        <AxesSubplot:xlabel='waterfront', ylabel='floors'>,
        <AxesSubplot:xlabel='view', ylabel='floors'>],
       [<AxesSubplot:xlabel='bedrooms', ylabel='waterfront'>,
        <AxesSubplot:xlabel='bathrooms', ylabel='waterfront'>,
        <AxesSubplot:xlabel='floors', ylabel='waterfront'>,
        <AxesSubplot:xlabel='waterfront', ylabel='waterfront'>,
```

In [79]:
```
%matplotlib inline
from pandas.plotting import scatter_matrix

attributes = X.columns[5:10]
scatter_matrix(X[attributes], figsize = (15,15), c = y, alpha = 0.8, marker = 'O
```

Out[79]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001E63FF7DE48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641458A08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E63FFA4F48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E63FFE1988>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E64001A3C8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E640050D88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E640089C08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E6400C2D48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E6400CC948>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E640105B08
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E6401710C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E6401A9148>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E6401E1248>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E64021B388>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E640253488
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E64028A588>,
```

In [80]:
```
%matplotlib inline
from pandas.plotting import scatter_matrix

attributes = X.columns[10:15]
scatter_matrix(X[attributes], figsize = (15,15), c = y, alpha = 0.8, marker = 'O
```

Out[80]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001E641AC3B08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E642FA7448>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641AE1A88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641B19488>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641B50E88
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E641B89888>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641BC4248>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641BF9E08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641C04A08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641C3CC08
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E641CAA1C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641CE0208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641D18308>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641D51408>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E641D8A548
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E641DCAE48>,
```

CHECKING FOR DATA LEAKAGE

In [27]:
```
columns = X
target = y

corr_list = [];
for i in range(len(X.columns)):
    corr_list.append(np.corrcoef(columns.iloc[:,i],target)[0,1])

for i in range(len(corr_list)):
    if corr_list[i] >= 0.5:
        print(X.columns[i], ' ', round(corr_list[i], 4))
```

```
bathrooms   0.5187
grade   0.6673
sqft_above   0.6059
sqft_living15   0.585
```

In [82]:
```
print(corr_list)
```

```
[0.3083594589361765, 0.5186585714110926, 0.25744987011382675, 0.263761948625549
2, 0.396430361308104, 0.036859222941225185, 0.6673107783936755, 0.6059118235859
404, 0.32306996824893724, 0.5849646206177209, 0.0824685929622751, -0.0536782735
6056376, 0.06958936539082772, -0.0922310812174531, -0.08034442892916133, -0.076
8043772143547, 0.27146704506125086, 0.06517516833310152, 0.13388452751735627,
0.017039572717025223, 0.033127728761368665, -0.021634365531309648, -0.012929831
681778492, -0.017333867453217258, -0.029588584153442683, -0.06397885793498992,
-0.10614916110034336, 0.006776333201475536, 0.029236739222066443, -0.0243632180
33473747, 0.024275374752955877, -0.07277809981626318, -0.07398103547850786, -0.
06003127207516818, 0.10271062400139602, -0.008047285007119157, -0.0788603721380
9603, 0.2127584331449846, 0.20517808014444688, -0.10042756413176263, -0.0278645
04495761996, 0.04733199246568763, 0.0520395602695864, -0.0720195666147662, -0.0
4495021513786569, -0.0745281322928754, -0.01884274197100275, -0.003976559041253
352, -0.010624983812139988, 0.009239751754767421, 0.05505586447987311, 0.088774
95691527543, 0.03741819334842722, -0.07184585665753927, 0.06882136997804096, 0.
020736041942116715, 0.09113195768879115, -0.0754084195930616, 0.011868076849258
126, -0.046827832072182045, 0.06596678590893489, 0.17015414760764264, 0.0362314
22596516044, 0.02667246861268013, 0.016223586677179897, -0.05177625304522625,
0.07819000206336492, 0.030022034693069712, -0.026761006537272453, -0.0405715061
33490524, -0.0638155111739213, 0.0035298978441152973, 0.018898975550206743, -0.
057199916444069285, -0.03576322402202353, -0.046032933808231334, -0.02214266234
1346923, -0.09171637179495684, 0.040594894924441675, -0.06928155221638244, -0.0
54439141117664167, -0.0740613029285442, 0.0838213847926988]
```

SPLITTING AND SCALING THE DATASET

In [28]:
```
#using min max scaler for now
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

X_train_org, X_test_org, y_train, y_test = train_test_split(X,y, random_state = 

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train_org)
X_test = scaler.transform(X_test_org)
print('Training data shape: '+str(X_train.shape))
print('Test data shape: '+ str(X_test.shape))
```

```
Training data shape: (16206, 83)
Test data shape: (5402, 83)
```

We used Min Max Scaler becuase from the scatter plots shown above, it is observed that very few variables have a normal distribution.

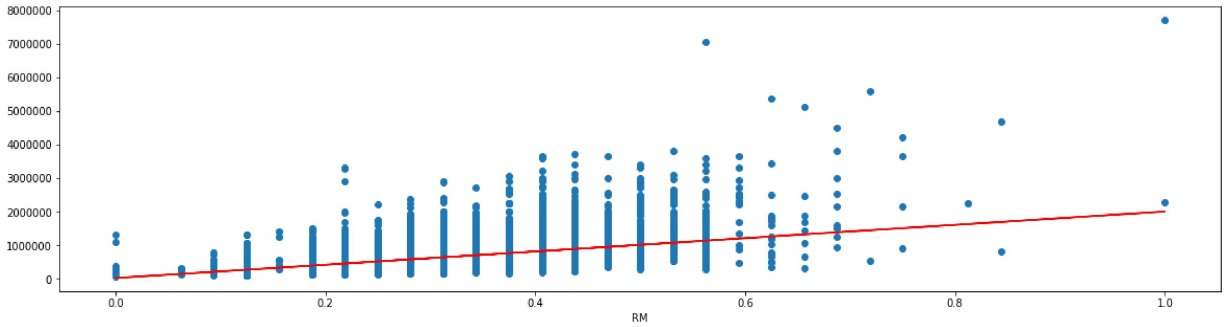
Linear Regression

In [84]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lreg = LinearRegression()

lreg.fit(X_train, y_train)
print('Train Score: ',lreg.score(X_train, y_train))
print('Test Score: ',lreg.score(X_test, y_test))

y_pred = lreg.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
Train Score:  0.8060535086321381
Test Score:  0.8093820423935696
MSE:  27248083255.02425
```

In [85]:
```
%matplotlib inline
import matplotlib.pyplot as plt

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

plt.subplots(figsize = (20,5))
plt.plot(X_train_rm, y_predict, c = 'r')
plt.scatter(X_train_rm,y_train)
plt.xlabel('RM')
```

Out[85]:
```
Text(0.5, 0, 'RM')
```



Polynomial Regression

Degree:2

In [86]:
```
from  sklearn.preprocessing  import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
lreg.fit(X_train_poly, y_train)

train_score = lreg.score(X_train_poly, y_train)
test_score = lreg.score(X_test_poly, y_test)

y_pred = lreg.predict(X_test_poly)
mse = mean_squared_error(y_test, y_pred)

print('Train Score: ', train_score)
print('Test Score: ', test_score)
print('MSE: ', mse)
```

```
Train Score:  0.9145076330256813
Test Score:  -8.755274022457158e+16
MSE:  1.2515317994174954e+28
```

Degree: 3

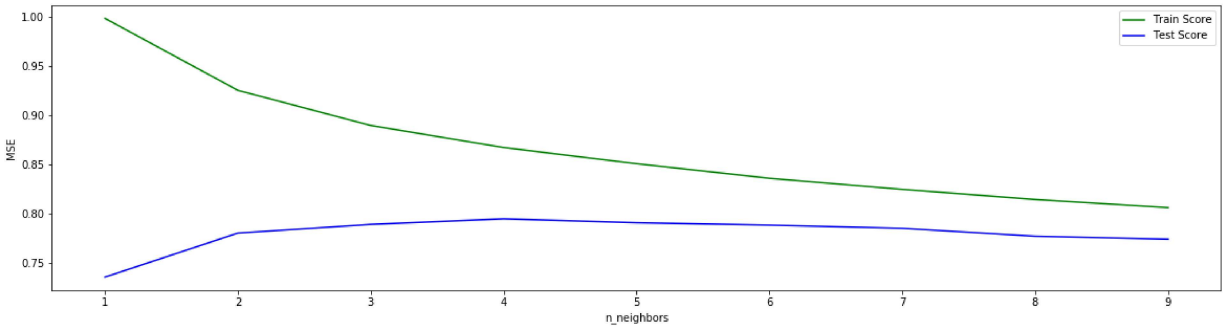
In [87]:
```
from  sklearn.preprocessing  import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
lreg.fit(X_train_poly, y_train)

train_score = lreg.score(X_train_poly, y_train)
test_score = lreg.score(X_test_poly, y_test)

y_pred = lreg.predict(X_test_poly)
mse = mean_squared_error(y_test, y_pred)

print('Train Score: ', train_score)
print('Test Score: ', test_score)
print('MSE: ', mse)
```

```
Train Score:  0.969159584067397
Test Score:  -1.0515784377112036e+18
MSE:  1.5031897928055753e+29
```

K-Neighbors Regressor

In [88]:
```
from sklearn.neighbors import KNeighborsRegressor

%matplotlib inline
train_score_array = []
test_score_array = []

for k in range(1,10):
    knn_reg = KNeighborsRegressor(k)
    knn_reg.fit(X_train, y_train)
    train_score_array.append(knn_reg.score(X_train, y_train))
    test_score_array.append(knn_reg.score(X_test, y_test))

x_axis = range(1,10)

plt.subplots(figsize = (20,5))
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('MSE')
```

Out[88]: Text(0, 0.5, 'MSE')

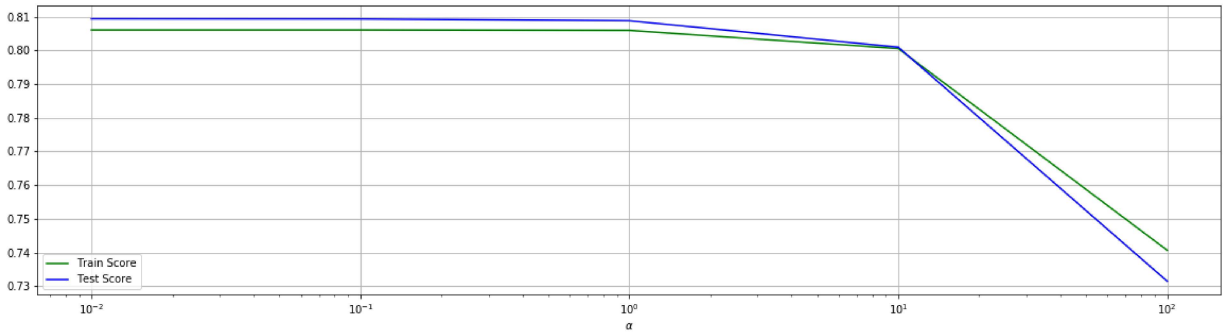


In [91]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

tuned_parameters=dict(n_neighbors=x_axis)

knn_reg = KNeighborsRegressor()
gri1 = GridSearchCV(knn_reg, tuned_parameters, cv = 3, return_train_score = True
gri_model1 = gri1.fit(X_train, y_train)

print(gri_model1.best_params_)
print('validation score: ', gri_model1.best_score_)

print('Train Score: ',gri_model1.score(X_train, y_train))
print('Test Score: ',gri_model1.score(X_test, y_test))

y_pred = gri_model1.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'n_neighbors': 3}
validation score:  0.7470688771457796
Train Score:  0.8895655611417164
Test Score:  0.7889576318301038
MSE:  30167671978.229298
```

Ridge Regression

In [90]:
```
from sklearn.linear_model import Ridge
%matplotlib inline
import matplotlib.pyplot as plt

x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,  y_train)
    train_score_list.append(ridge.score(X_train, y_train))
    test_score_list.append(ridge.score(X_test, y_test))


plt.subplots(figsize = (20,5))
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
plt.grid()
```

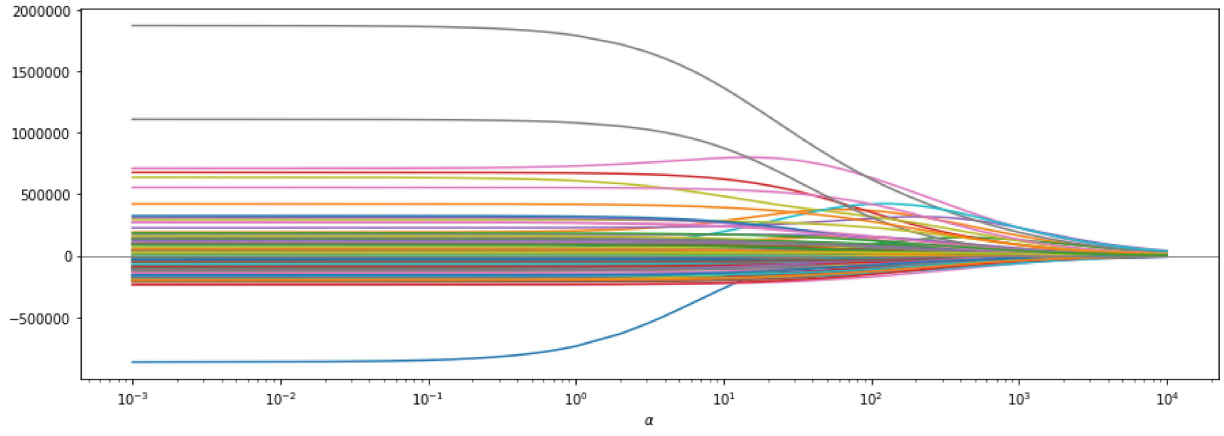


In [92]:
```
np.random.seed(0)

tuned_parameters = [{'alpha':x_range}]

ridge = Ridge(max_iter = 1000, tol = 0.1, random_state = 0)
gri2 = GridSearchCV(ridge, tuned_parameters, cv = 5, return_train_score = True, 
gri_model2 = gri2.fit(X_train, y_train)

print(gri2.best_params_)
print('validation score: ', gri2.best_score_)

print('Train Score: ',gri_model2.score(X_train, y_train))
print('Test Score: ',gri_model2.score(X_test, y_test))

y_pred = gri_model2.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'alpha': 0.1}
validation score:  0.8022422771524041
Train Score:  0.8060518957690112
Test Score:  0.8093329406040315
MSE:  27255102161.669266
```

In [93]:
```
%matplotlib inline
import numpy as np

x_range1 = np.linspace(0.001, 1, 100).reshape(-1,1)
x_range2 = np.linspace(1, 10000, 10000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    coeff.append(ridge.coef_ )

coeff = np.array(coeff)

plt.subplots(figsize = (15,5))
for i in range(len(X.columns)):
    plt.plot(x_range, coeff[:,i])#, label = X.columns[i])

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
           ncol=3, fancybox=True, shadow=True)
plt.show()
```

```
No handles with labels found to put in legend.
```

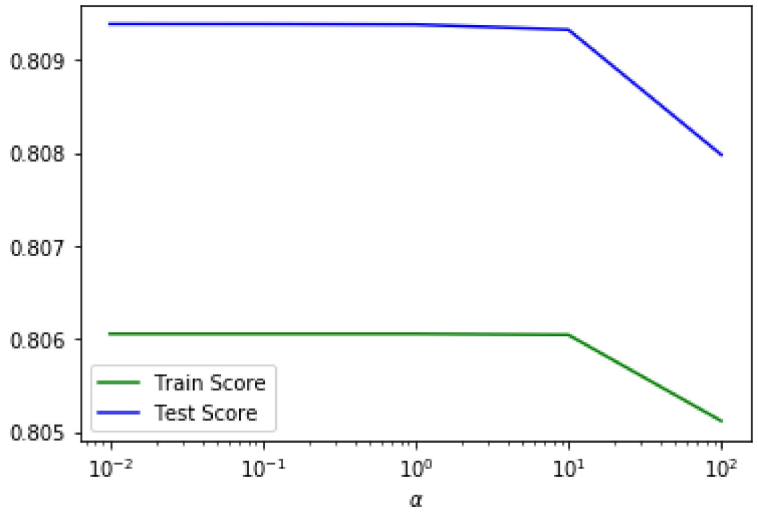


This figure is just to check at what value of alpha does the coefficients converge to zero.

Lasso Regression

In [94]:
```
from sklearn.linear_model import Lasso
x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    lasso = Lasso(alpha, max_iter=100, random_state=0)
    lasso.fit(X_train,y_train)
    train_score_list.append(lasso.score(X_train,y_train))
    test_score_list.append(lasso.score(X_test, y_test))


plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

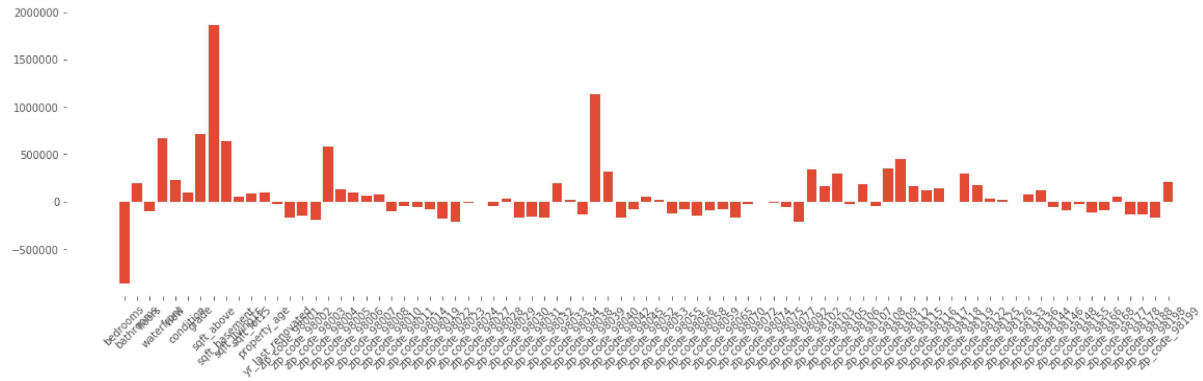
Out[94]:
```
Text(0.5, 0, '$\\alpha$')
```



In [95]:
```
np.random.seed(0)

tuned_parameters = [{'alpha':x_range}]

lasso = Lasso(max_iter = 1000, tol=0.1, random_state=0)
gri3 = GridSearchCV(lasso, tuned_parameters, cv = 5, return_train_score = True, 
gri_model3 = gri3.fit(X_train,y_train)

print(gri_model3.best_params_)
print('validation score: ', gri_model3.best_score_)

print('Train Score: ',gri_model3.score(X_train, y_train))
print('Test Score: ',gri_model3.score(X_test, y_test))

y_pred = gri_model3.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'alpha': 1}
validation score:  0.7981869642507479
Train Score:  0.8002178535977403
Test Score:  0.8036311094553663
MSE:  28070156376.895428
```

In [100]:
```
lasso = Lasso(alpha = 1)
lasso.fit(X_train, y_train)
#dict(zip(X.columns, lasso.coef_))
plt.subplots(figsize = (20,5))
plt.style.use('ggplot')
plt.bar(range(len(X.columns)), lasso.coef_)
plt.box(False)
plt.xticks(range(len(X.columns)), X.columns, rotation = 45)
plt.grid()
```



sqft_above is the most important feature in this dataset based on the Lasso model with alpha = 1.

SGD Regressor

In [102]:
```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(random_state= 0, max_iter = 10000, penalty = 'l2', learni
                       eta0 = 0.05, shuffle = False)
sgd_reg.fit(X_train, y_train)
print('Train Score: ',sgd_reg.score(X_train, y_train))
print('Test Score: ',sgd_reg.score(X_test, y_test))

print("SGD regression coefficient: " + str(sgd_reg.coef_))
print("SGD regression intercept: " + str(sgd_reg.intercept_))
print("SGD regression iterations: " + str(sgd_reg.n_iter_))

y_pred = sgd_reg.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
Train Score:  0.7943173918495281
Test Score:  0.7959419967553129
SGD regression coefficient: [-6.63597242e+05  2.02707016e+05 -8.87675466e+04
5.88547849e+05
  2.53800559e+05  8.55080971e+04  7.41995494e+05  1.70775363e+06
  5.36113576e+05  1.10896871e+05  1.14726457e+05  6.88020166e+04
 -3.82713028e+04 -2.22172190e+05 -1.81478257e+05 -2.15590298e+05
  5.26867249e+05  1.38384691e+05  1.55768674e+05  3.50096035e+04
  4.80777970e+04 -1.41339872e+05 -9.43097914e+04 -9.96789783e+04
 -1.00718361e+05 -1.88729773e+05 -2.34378071e+05  3.26553009e+03
 -1.67260743e+04 -8.80786185e+04 -1.78360381e+03 -2.09786031e+05
 -1.81510758e+05 -2.04899683e+05  2.00210461e+05 -9.98303271e+03
 -1.83778770e+05  1.05051324e+06  2.65520890e+05 -1.99548004e+05
 -1.15489155e+05  2.09506630e+04  1.35168078e+03 -1.52560910e+05
 -1.08571650e+05 -1.69765778e+05 -7.66812113e+04 -1.09200282e+05
 -1.87428054e+05 -4.96997791e+04 -1.56810288e+04 -5.08765794e+04
 -7.57386507e+04 -2.26959042e+05  2.40365327e+05  1.19744457e+05
  2.30522187e+05 -1.02082352e+05  1.66943529e+05 -6.35996020e+04
  3.11810583e+05  4.09701068e+05  1.18419447e+05  9.95168806e+04
```

SVM Regressor

Linear SVR

In [112]:
```
from sklearn.svm import LinearSVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for e in [0.01, 0.1, 1, 10]:
        linear = LinearSVR(C = C, epsilon = e)
        linear.fit(X_train, y_train)
        print('Train Score: ',linear.score(X_train,y_train),'C: ' ,C,'e: ' ,e)
        print('Test Score: ',linear.score(X_test, y_test),'C: ' ,C,'e: ' ,e)
```

```
Train Score:  -2.1926333202778827 C:  0.01 e:  0.01
Test Score:  -2.0833949991511838 C:  0.01 e:  0.01
Train Score:  -2.1926333202778827 C:  0.01 e:  0.1
Test Score:  -2.0833949991511838 C:  0.01 e:  0.1
Train Score:  -2.1926333202778827 C:  0.01 e:  1
Test Score:  -2.0833949991511838 C:  0.01 e:  1
Train Score:  -2.1926333202778827 C:  0.01 e:  10
Test Score:  -2.0833949991511838 C:  0.01 e:  10
Train Score:  -2.167391590011791 C:  0.1 e:  0.01
Test Score:  -2.0597238907378035 C:  0.1 e:  0.01
Train Score:  -2.167391590011791 C:  0.1 e:  0.1
Test Score:  -2.0597238907378026 C:  0.1 e:  0.1
Train Score:  -2.167391590011791 C:  0.1 e:  1
Test Score:  -2.0597238907378026 C:  0.1 e:  1
Train Score:  -2.167391590011791 C:  0.1 e:  10
Test Score:  -2.0597238907378026 C:  0.1 e:  10
Train Score:  -1.9225847664848539 C:  1 e:  0.01
Test Score:  -1.830041578218387 C:  1 e:  0.01
Train Score:  -1.9225847664848539 C:  1 e:  0.1
```

In [107]:
```
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.01, 0.1, 1, 10, 100, 1000, 10000],
              'epsilon': [0.01, 0.1, 1, 10]}

grid_search_linear = GridSearchCV(LinearSVR(), param_grid, cv=5, return_train_sc
svr_model_linear = grid_search_linear.fit(X_train, y_train)

print(svr_model_linear.best_params_)
print('Validation score:', svr_model_linear.best_score_)

print('Train Score: ',svr_model_linear.score(X_train,y_train))
print('Test Score: ',svr_model_linear.score(X_test,y_test))

y_pred = svr_model_linear.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'epsilon': 0.01}
Validation score: 0.7205225606617492
Train Score:  0.7300956240194971
Test Score:  0.7224602225273767
MSE:  39673213678.897514
```

The value for C is quite big here through GridSearchCV but the values for train, validation and test scores are very close and it is interpreted that the model is able to generalize the complex

structure of the data well.

Linear Kernel

In [32]:
```
from sklearn.svm import SVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for e in [0.01, 0.1, 1, 10]:
        linear = SVR(kernel = 'linear', C = C, epsilon = e)
        linear.fit(X_train, y_train)
        print('Train Score: ',linear.score(X_train,y_train),'C: ' ,C,'epsilon: '
        print('Test Score: ',linear.score(X_test, y_test),'C: ' ,C,'epsilon: ' ,
```

```
Train Score:  -0.058841744623332026 C:  0.01 epsilon:  0.01
Test Score:  -0.06454541200678832 C:  0.01 epsilon:  0.01
Train Score:  -0.058841744623332026 C:  0.01 epsilon:  0.1
Test Score:  -0.0645454120067881 C:  0.01 epsilon:  0.1
Train Score:  -0.058841675146019456 C:  0.01 epsilon:  1
Test Score:  -0.06454534120655309 C:  0.01 epsilon:  1
Train Score:  -0.058851664088799716 C:  0.01 epsilon:  10
Test Score:  -0.0645553912717558 C:  0.01 epsilon:  10
Train Score:  -0.058672552752522256 C:  0.1 epsilon:  0.01
Test Score:  -0.0643841723114904 C:  0.1 epsilon:  0.01
Train Score:  -0.058672552752522256 C:  0.1 epsilon:  0.1
Test Score:  -0.06438417231149063 C:  0.1 epsilon:  0.1
Train Score:  -0.05867327117750798 C:  0.1 epsilon:  1
Test Score:  -0.06438489503528255 C:  0.1 epsilon:  1
Train Score:  -0.05867361825437167 C:  0.1 epsilon:  10
Test Score:  -0.06438522795430246 C:  0.1 epsilon:  10
Train Score:  -0.05691687299537618 C:  1 epsilon:  0.01
Test Score:  -0.06270729398271335 C:  1 epsilon:  0.01
Train Score:  -0.05691687299537618 C:  1 epsilon:  0.1
```

In [34]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

param_grid = {'C': [0.1, 1, 100, 1000, 10000],
              'epsilon': [0.01, 0.1, 1, 10]}

grid_search_linear = GridSearchCV(SVR(kernel = 'linear'), param_grid, cv=5, retu
svr_model_linear = grid_search_linear.fit(X_train, y_train)

print(svr_model_linear.best_params_)
print('Validation score:', svr_model_linear.best_score_)

print('Train Score: ',svr_model_linear.score(X_train,y_train))
print('Test Score: ',svr_model_linear.score(X_test,y_test))

y_pred = svr_model_linear.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'epsilon': 1}
Validation score: 0.721220625895948
Train Score:  0.7304706943571697
Test Score:  0.7228324973608125
MSE:  39619998463.59157
```

The value for C is quite big here through GridSearchCV but the values for train, validation and test scores are very close and it is interpreted that the model is able to generalize the complex structure of the data well.

RBF Kernel

In [114]:
```
from sklearn.svm import SVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for g in [0.01, 0.1, 1, 10]:
        rbf = SVR(kernel = 'rbf', C = C, gamma = g)
        rbf.fit(X_train, y_train)
        print('Train Score: ',rbf.score(X_train,y_train),'C: ' ,C,'gamma: ' ,g)
        print('Test Score: ',rbf.score(X_test, y_test),'C: ' ,C,'gamma: ' ,g)
```

```
Train Score:  -0.058860057562585055 C:  0.01 gamma:  0.01
Test Score:  -0.06456285850311949 C:  0.01 gamma:  0.01
Train Score:  -0.058857429434937325 C:  0.01 gamma:  0.1
Test Score:  -0.06456035305767349 C:  0.01 gamma:  0.1
Train Score:  -0.05885604196977412 C:  0.01 gamma:  1
Test Score:  -0.06455908072399641 C:  0.01 gamma:  1
Train Score:  -0.05885972299779674 C:  0.01 gamma:  10
Test Score:  -0.06456258089405731 C:  0.01 gamma:  10
Train Score:  -0.058856707572268574 C:  0.1 gamma:  0.01
Test Score:  -0.06455966333318397 C:  0.1 gamma:  0.01
Train Score:  -0.05883135342105783 C:  0.1 gamma:  0.1
Test Score:  -0.06453554345799772 C:  0.1 gamma:  0.1
Train Score:  -0.058817219472000026 C:  0.1 gamma:  1
Test Score:  -0.06452255784418348 C:  0.1 gamma:  1
Train Score:  -0.05885391155641395 C:  0.1 gamma:  10
Test Score:  -0.06455744101339 C:  0.1 gamma:  10
Train Score:  -0.058824102927453215 C:  1 gamma:  0.01
Test Score:  -0.06452861401207266 C:  1 gamma:  0.01
Train Score:  -0.05856586357273619 C:  1 gamma:  0.1
```

In [115]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = {'C': [0.1, 1, 100, 1000, 10000],
              'gamma': [0.01, 0.1, 1, 10]}

grid_search_rbf = GridSearchCV(SVR(kernel = 'rbf'), param_grid, cv=5, return_tra
svr_model_rbf = grid_search_rbf.fit(X_train, y_train)

print(svr_model_rbf.best_params_)
print('Validation score:', svr_model_rbf.best_score_)

print('Train Score: ',svr_model_rbf.score(X_train,y_train))
print('Test Score: ',svr_model_rbf.score(X_test,y_test))

y_pred = svr_model_rbf.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'gamma': 0.1}
Validation score: 0.5975357687075278
Train Score:  0.6192422886360438
Test Score:  0.6035700481078412
MSE:  56668093969.64391
```

The value for C is quite big here through GridSearchCV but the values for train, validation and test scores are very close and it is interpreted that the model is able to generalize the complex

structure of the data well.

Poly Kernel

In [116]:
```
from sklearn.svm import SVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for coef in [0.01, 0.1, 1, 10]:
        poly = SVR(kernel = 'poly', degree = 3, C = C, coef0 = coef)
        poly.fit(X_train, y_train)
        print('Train Score: ',poly.score(X_train,y_train),'C: ' ,C,'coef0: ' ,co
        print('Test Score: ',poly.score(X_test, y_test),'C: ' ,C,'coef0: ' ,coef
```

```
Train Score:  -0.058860483302784816 C:  0.01 coef0:  0.01
Test Score:  -0.0645632673791603 C:  0.01 coef0:  0.01
Train Score:  -0.0588604724393591 C:  0.01 coef0:  0.1
Test Score:  -0.06456325693199982 C:  0.01 coef0:  0.1
Train Score:  -0.058859668645011576 C:  0.01 coef0:  1
Test Score:  -0.06456248516857932 C:  0.01 coef0:  1
Train Score:  -0.05879235044505071 C:  0.01 coef0:  10
Test Score:  -0.0644983387767355 C:  0.01 coef0:  10
Train Score:  -0.0588604763492695 C:  0.1 coef0:  0.01
Test Score:  -0.06456326066470086 C:  0.1 coef0:  0.01
Train Score:  -0.058860367715022566 C:  0.1 coef0:  0.1
Test Score:  -0.06456315619310238 C:  0.1 coef0:  0.1
Train Score:  -0.058853284884907264 C:  0.1 coef0:  1
Test Score:  -0.06455639943678926 C:  0.1 coef0:  1
Train Score:  -0.05816797829911625 C:  0.1 coef0:  10
Test Score:  -0.06390265777647652 C:  0.1 coef0:  10
Train Score:  -0.05886040681415361 C:  1 coef0:  0.01
Test Score:  -0.06456319352014295 C:  1 coef0:  0.01
Train Score:  -0.05885935414610022 C:  1 coef0:  0.1
```

In [35]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = {'C': [0.1, 1, 10, 100, 1000, 10000],
              'coef0': [0.01, 0.1, 1, 10]}

grid_search_poly = GridSearchCV(SVR(kernel = 'poly'), param_grid, cv=5, return_t
svr_model_poly = grid_search_poly.fit(X_train, y_train)

print(svr_model_poly.best_params_)
print('Validation score:', svr_model_poly.best_score_)

print('Train Score: ',svr_model_poly.score(X_train,y_train))
print('Test Score: ',svr_model_poly.score(X_test,y_test))

y_pred = svr_model_poly.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'coef0': 10}
Validation score: 0.7549443634453178
Train Score:  0.7593349014269299
Test Score:  0.7552182592565814
MSE:  34990581867.725494
```

The value for C is quite big here through GridSearchCV but the values for train, validation and test scores are very close and it is interpreted that the model is able to generalize the complex structure of the data well.

Decision Tree Regressor

In [29]:
```
from sklearn.tree import DecisionTreeRegressor

for leaf in [2, 5, 7, 10, 15, 20]:
    dtr = DecisionTreeRegressor(min_samples_leaf = leaf)
    dtr.fit(X_train, y_train)
    print('Train Score: ',dtr.score(X_train,y_train),'min_samples_leaf: ' ,leaf)
    print('Test Score: ',dtr.score(X_test, y_test),'min_samples_leaf: ' ,leaf)
```

```
Train Score:  0.9717911718349778 min_samples_leaf:  2
Test Score:  0.6525985818550648 min_samples_leaf:  2
Train Score:  0.8891731888650789 min_samples_leaf:  5
Test Score:  0.6739662240410413 min_samples_leaf:  5
Train Score:  0.8520510595015419 min_samples_leaf:  7
Test Score:  0.7219845920713572 min_samples_leaf:  7
Train Score:  0.8243602213067842 min_samples_leaf:  10
Test Score:  0.7004812966528191 min_samples_leaf:  10
Train Score:  0.7792949543439533 min_samples_leaf:  15
Test Score:  0.6930623920550905 min_samples_leaf:  15
Train Score:  0.7571364547437054 min_samples_leaf:  20
Test Score:  0.6889514577171962 min_samples_leaf:  20
```

In [30]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {'min_samples_leaf': range(3,20)}

dtr = GridSearchCV(DecisionTreeRegressor(), param_grid, cv = 5, return_train_sco
decision_tree = dtr.fit(X_train, y_train)

## tree_model = clf.best_estimator_
print(decision_tree.best_params_)
print('Validation score:', decision_tree.best_score_)

print('Train Score: ',decision_tree.score(X_train,y_train))
print('Test Score: ',decision_tree.score(X_test,y_test))

y_pred = decision_tree.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'min_samples_leaf': 9}
Validation score: 0.6800405443270736
Train Score:  0.8315005319288606
Test Score:  0.7006875337562508
MSE:  42785533440.21972
```

CONCLUSION

After running all the models and comparing the performance on training and testing data sets, it is observed that Linear Regression and Ridge Regression (alpha = 0.1) have the highest training and testing scores among all the models.
The top 4 models are mentioned here in decreasing order of training and testing scores: Linear Regression > Ridge Regression > Lasso Regression > SGD Regressor

Best Parameters for every model

Model	Parameters
Polynomial Regression	Degree = 2
KNN Regressor	k = 3
Ridge Regression	alpha = 0.1
Lasso Regression	alpha = 1
SGD Regressor	number of iterations = 36
Linear SVR	C = 10000, epsilon = 0.01
SVR Linear Kernel	C = 10000, epsilon = 1
SVR RBF Kernel	C = 10000, gamma = 0.1
SVR Poly Kernel	C = 10000, coef0 = 10
Decision Tree Regressor	min_samples_leaf = 9

Performance on Dataset

Model	Linear Regression	Polynomial Regression	KNN Regressor	Ridge Regression	Lasso Regression
Params	default	degree = 2	k = 3	alpha = 0.1	alpha = 1
Training Score	0.8060	0.9145	0.8896	0.8061	0.8002
Validation Score	-	-	0.7471	0.8022	0.7982
Testing score	0.8094	-8.7553e+16	0.7890	0.8093	0.8036
MSE	27248083255.0241	1.2515e+28	30167671978.2293	27255102161.6693	28070156376.8954

Linear Regression with default parameters

In [33]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lreg = LinearRegression()

lreg.fit(X_train, y_train)
print('Train Score: ',lreg.score(X_train, y_train))
print('Test Score: ',lreg.score(X_test, y_test))

y_pred = lreg.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
Train Score:  0.8060535086321381
Test Score:  0.8093820423935696
MSE:  27248083255.02425
```

Ridge Regression with alpha = 0.1

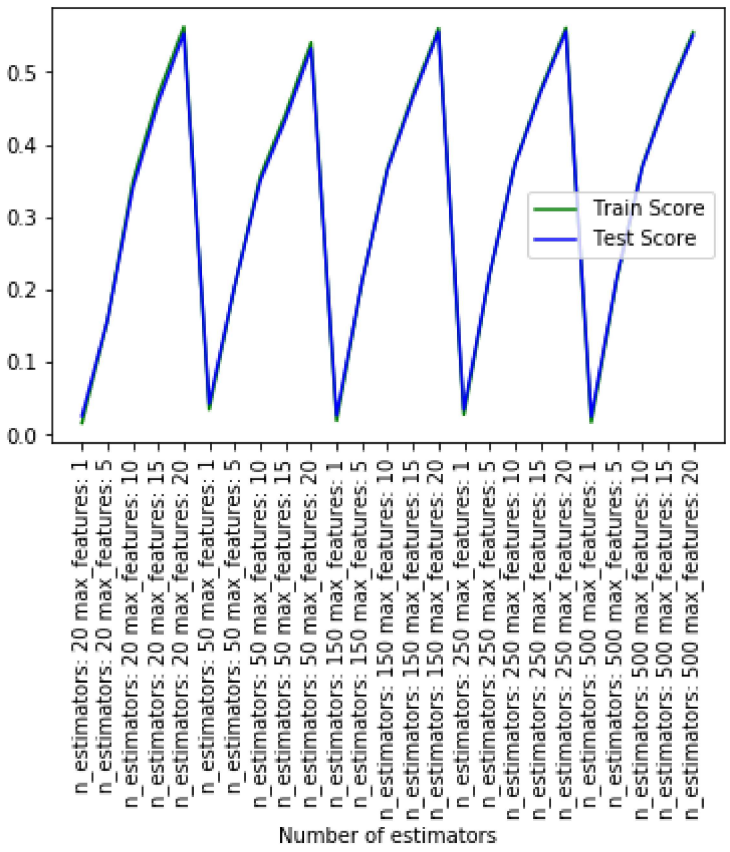
In [35]:
```
from  sklearn.linear_model import Ridge

ridge = Ridge(alpha = 0.1)
ridge.fit(X_train,y_train)
print('Train score: ',ridge.score(X_train,y_train))
print('Test score: ', ridge.score(X_test, y_test))
y_pred = ridge.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
Train score:  0.8060518957690112
Test score:  0.8093329406040315
MSE:  27255102161.669266
```

PROJECT 2

Bagging for SGD Regressor

In [39]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.linear_model import SGDRegressor

n_estimators = [20,50,150,250,500]
max_features = [1,5,10,15,20]
param_list = []
train_score_list = []
test_score_list = []

for estimator in n_estimators:
    for features in max_features:
        sgd_reg = SGDRegressor(random_state = 0, max_iter = 10000, penalty = 'l2
                        eta0 = 0.05, shuffle = False)
        bag_reg = BaggingRegressor(sgd_reg, max_samples = 0.5, max_features = fe
                                   bootstrap = True, random_state = 0, oob_score = T

        bag_reg.fit(X_train, y_train)
        train_score_list.append(bag_reg.score(X_train, y_train))
        test_score_list.append(bag_reg.score(X_test, y_test))
        param_list.append("n_estimators: " + str(estimator) + ' ' + "max_feature

#f, axes = plt.subplots(1,2)
plt.plot(param_list, train_score_list, c = 'g', label = 'Train Score')
plt.plot(param_list, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel("Number of estimators")
plt.legend()
plt.xticks(rotation = 90)
#plt.ylabel("Test Score")
plt.show()
```

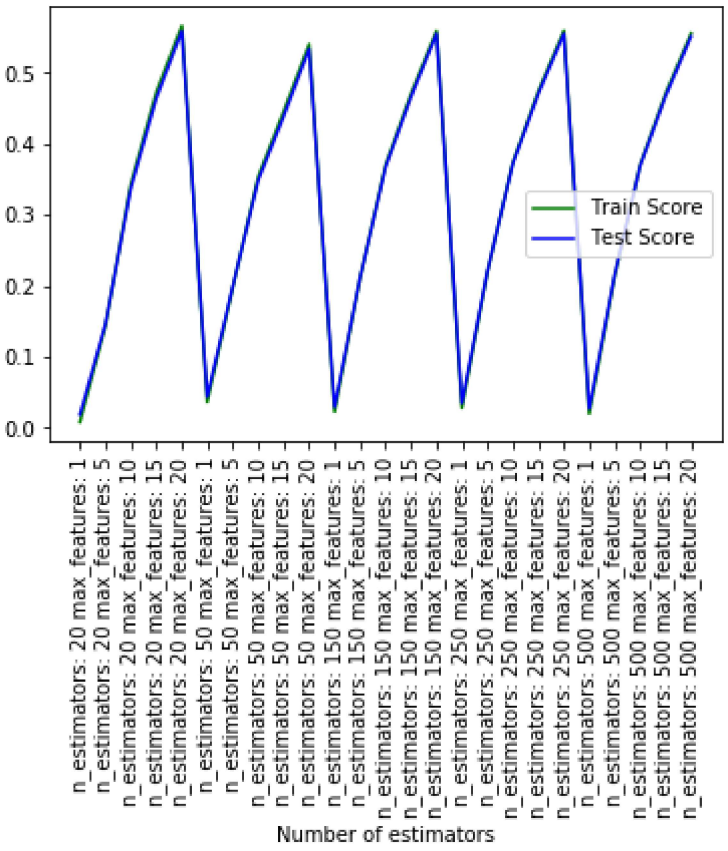


In [40]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [20,50,150,250,500],
              "max_features":[1,5,10,15,20],
              "max_samples": [0.1,0.3,0.5]}

sgd_reg = SGDRegressor(random_state = 0, max_iter = 10000, penalty = 'l2', learn
                       eta0 = 0.05, shuffle = False)

sgd_bag_grid = GridSearchCV(BaggingRegressor(sgd_reg, bootstrap = True, oob_scor
sgd_bag_grid_model = sgd_bag_grid.fit(X_train, y_train)

print(sgd_bag_grid_model.best_params_)
print('Validation score: ',sgd_bag_grid_model.best_score_)

print('Train Score: ',sgd_bag_grid_model.score(X_train, y_train))
print('Test Score: ',sgd_bag_grid_model.score(X_test, y_test))

y_pred = sgd_bag_grid_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'max_features': 20, 'max_samples': 0.3, 'n_estimators': 20}
Validation score:  0.5677115662754662
Train Score:  0.5589871547840561
Test Score:  0.5509060711366776
MSE:  64196201221.81469
```

Pasting for SGD Regressor

In [42]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.linear_model import SGDRegressor

n_estimators = [20,50,150,250,500]
max_features = [1,5,10,15,20]
param_list = []
train_score_list = []
test_score_list = []

for estimator in n_estimators:
    for features in max_features:
        sgd_reg = SGDRegressor(random_state = 0, max_iter = 10000, penalty = 'l2
                        eta0 = 0.05, shuffle = False)
        bag_reg = BaggingRegressor(sgd_reg, max_samples = 0.5, max_features = fe
                                   bootstrap = False, random_state = 0)

        bag_reg.fit(X_train, y_train)
        train_score_list.append(bag_reg.score(X_train, y_train))
        test_score_list.append(bag_reg.score(X_test, y_test))
        param_list.append("n_estimators: " + str(estimator) + ' ' + "max_feature

#f, axes = plt.subplots(1,2)
plt.plot(param_list, train_score_list, c = 'g', label = 'Train Score')
plt.plot(param_list, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel("Number of estimators")
plt.legend()
plt.xticks(rotation = 90)
#plt.ylabel("Test Score")
plt.show()
```



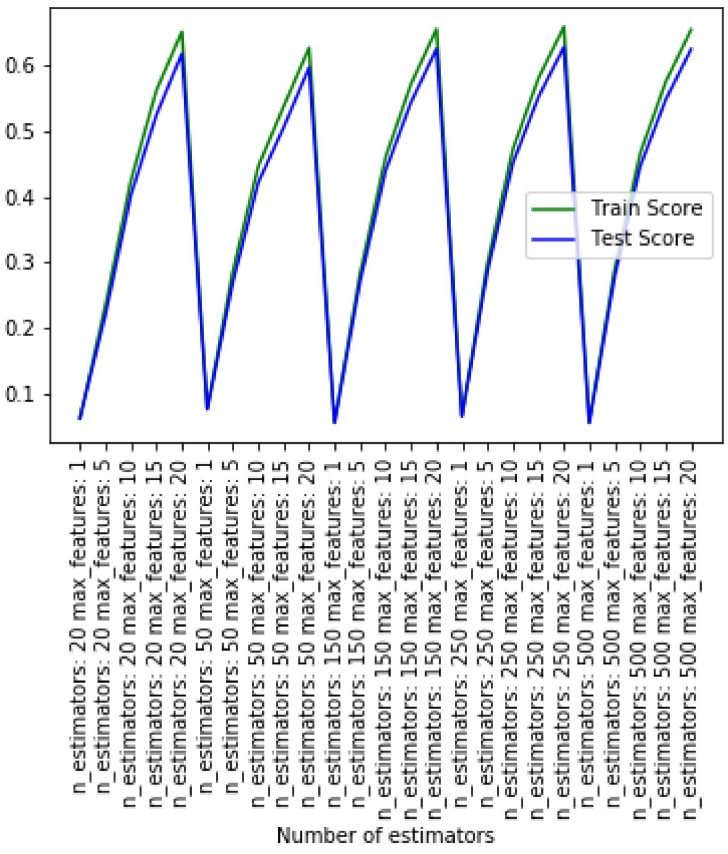
In [44]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [20,50,150,250,500],
              "max_features":[1,5,10,15,20],
              "max_samples": [0.1,0.3,0.5]}

sgd_reg = SGDRegressor(random_state = 0, max_iter = 10000, penalty = 'l2', learn
                       eta0 = 0.05, shuffle = False)
#sgd_reg = SGDRegressor(random_state = 0)

sgd_bag_grid = GridSearchCV(BaggingRegressor(sgd_reg, bootstrap = False, random_
                            cv = 5, return_train_score = True)
sgd_bag_grid_model = sgd_bag_grid.fit(X_train, y_train)

print(sgd_bag_grid_model.best_params_)
print('Validation score: ',sgd_bag_grid_model.best_score_)

print('Train Score: ',sgd_bag_grid_model.score(X_train, y_train))
print('Test Score: ',sgd_bag_grid_model.score(X_test, y_test))

y_pred = sgd_bag_grid_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'max_features': 20, 'max_samples': 0.5, 'n_estimators': 20}
Validation score:  0.5693555465285673
Train Score:  0.5662121617324214
Test Score:  0.5587716348344909
MSE:  63071849995.010414
```

For SGD Regressor, the best parameters obtained through Porject 1 are used for both Bagging and Pasting with and without Grid Search.

Bagging for Decision Tree Regressor

In [45]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor

n_estimators = [20,50,150,250,500]
max_features = [1,5,10,15,20]
param_list = []
train_score_list = []
test_score_list = []

dtr = DecisionTreeRegressor(min_samples_leaf = 9)

for estimator in n_estimators:
    for features in max_features:
        bag_reg = BaggingRegressor(dtr, max_samples = 0.5, max_features = featur
                                   bootstrap = True, random_state = 0, oob_score = T

        bag_reg.fit(X_train, y_train)
        train_score_list.append(bag_reg.score(X_train, y_train))
        test_score_list.append(bag_reg.score(X_test, y_test))
        param_list.append("n_estimators: " + str(estimator) + ' ' + "max_feature


plt.plot(param_list, train_score_list, c = 'g', label = 'Train Score')
plt.plot(param_list, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel("Number of estimators")
plt.legend()
plt.xticks(rotation = 90)
plt.show()
```

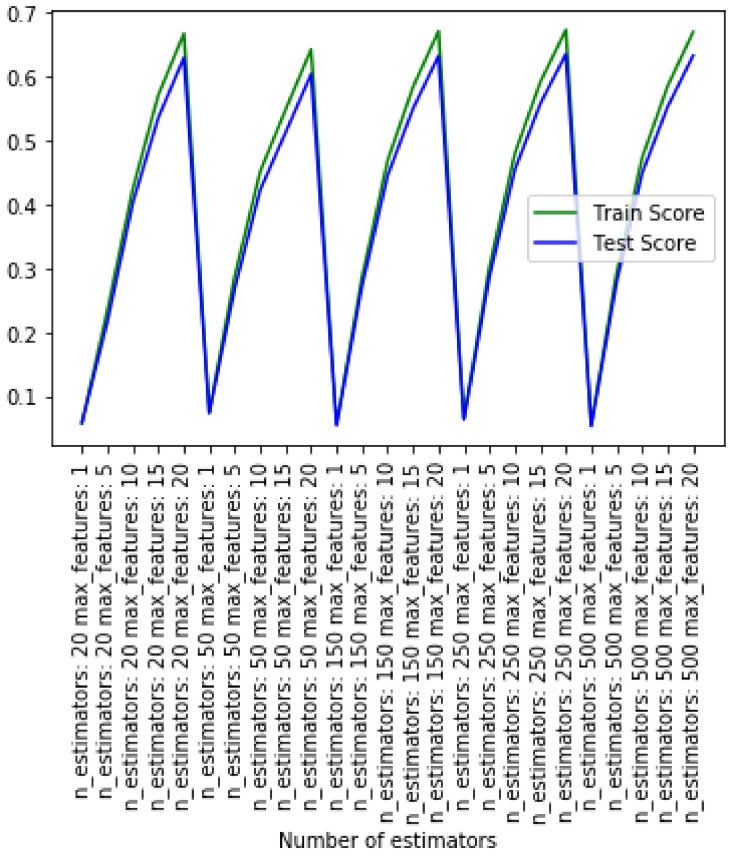


In [46]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [20,50,150,250,500],
              "max_features":[1,5,10,15,20],
              "max_samples": [0.1,0.3,0.5]}

dtr = DecisionTreeRegressor(min_samples_leaf = 9)

dtr_bag_grid = GridSearchCV(BaggingRegressor(dtr, bootstrap = True, oob_score = 
                            cv = 5, return_train_score = True)
dtr_bag_grid_model = dtr_bag_grid.fit(X_train, y_train)

print(dtr_bag_grid_model.best_params_)
print('Validation score: ',dtr_bag_grid_model.best_score_)

print('Train Score: ',dtr_bag_grid_model.score(X_train, y_train))
print('Test Score: ',dtr_bag_grid_model.score(X_test, y_test))

y_pred = dtr_bag_grid_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'max_features': 20, 'max_samples': 0.5, 'n_estimators': 250}
Validation score:  0.6113193325992352
Train Score:  0.658800640471781
Test Score:  0.6278667258154307
MSE:  53194979970.78448
```

Pasting for Decision Tree Regressor

In [47]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor

n_estimators = [20,50,150,250,500]
max_features = [1,5,10,15,20]
param_list = []
train_score_list = []
test_score_list = []

dtr = DecisionTreeRegressor(min_samples_leaf = 9)

for estimator in n_estimators:
    for features in max_features:
        bag_reg = BaggingRegressor(dtr, max_samples = 0.5, max_features = featur
                                   bootstrap = False, random_state = 0)

        bag_reg.fit(X_train, y_train)
        train_score_list.append(bag_reg.score(X_train, y_train))
        test_score_list.append(bag_reg.score(X_test, y_test))
        param_list.append("n_estimators: " + str(estimator) + ' ' + "max_feature


plt.plot(param_list, train_score_list, c = 'g', label = 'Train Score')
plt.plot(param_list, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel("Number of estimators")
plt.legend()
plt.xticks(rotation = 90)
plt.show()
```



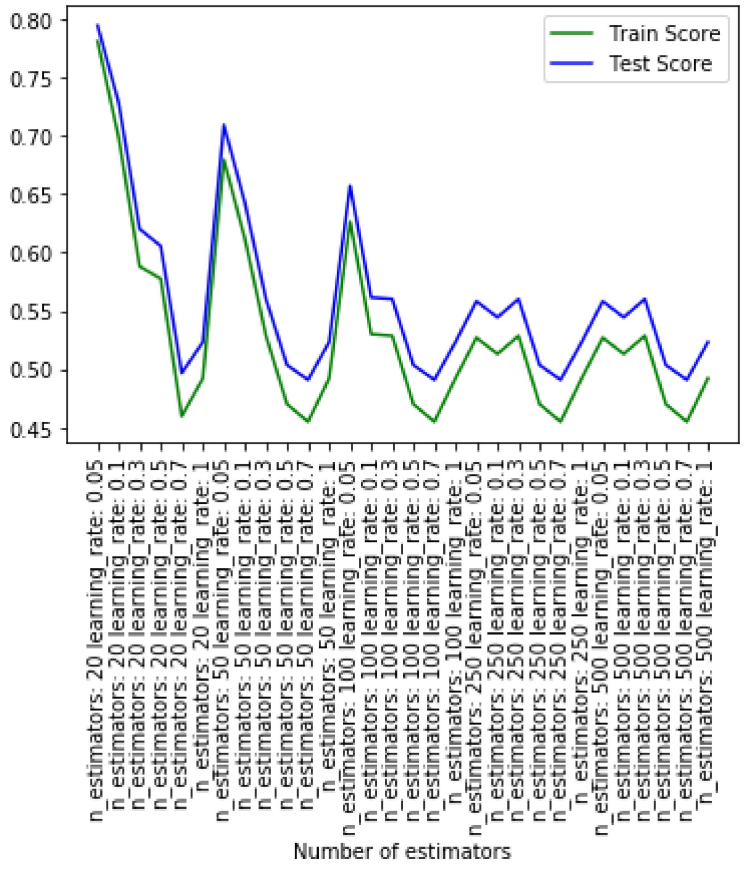
In [48]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [20,50,150,250,500],
              "max_features":[1,5,10,15,20],
              "max_samples": [0.1,0.3,0.5]}

dtr = DecisionTreeRegressor(min_samples_leaf = 9)

dtr_bag_grid = GridSearchCV(BaggingRegressor(dtr, bootstrap = False, random_stat
                            cv = 5, return_train_score = True)
dtr_bag_grid_model = dtr_bag_grid.fit(X_train, y_train)

print(dtr_bag_grid_model.best_params_)
print('Validation score: ',dtr_bag_grid_model.best_score_)

print('Train Score: ',dtr_bag_grid_model.score(X_train, y_train))
print('Test Score: ',dtr_bag_grid_model.score(X_test, y_test))

y_pred = dtr_bag_grid_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'max_features': 20, 'max_samples': 0.5, 'n_estimators': 250}
Validation score:  0.618597061404007
Train Score:  0.6734508635945906
Test Score:  0.6354711358183972
MSE:  52107959631.94532
```

For Decision Tree Regressor, the best parameters obtained through Porject 1 are used for both Bagging and Pasting with and without Grid Search.

Adaboosting for Linear Regression

In [41]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.linear_model import LinearRegression
%matplotlib inline

n_estimators = [20,50,100,250,500]
learning_rate = [0.05,0.1,0.3,0.5,0.7,1]
param_list = []
train_score_list = []
test_score_list = []

lreg = LinearRegression()

for estimator in n_estimators:
    for rate in learning_rate:
        adb_reg = AdaBoostRegressor(lreg, random_state=0, n_estimators = estimat

        adb_reg.fit(X_train, y_train)
        train_score_list.append(adb_reg.score(X_train, y_train))
        test_score_list.append(adb_reg.score(X_test, y_test))
        param_list.append("n_estimators: " + str(estimator) + ' ' + "learning_ra

plt.plot(param_list, train_score_list, c = 'g', label = 'Train Score')
plt.plot(param_list, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel("Number of estimators")
plt.legend()
plt.xticks(rotation = 90)
plt.show()
```



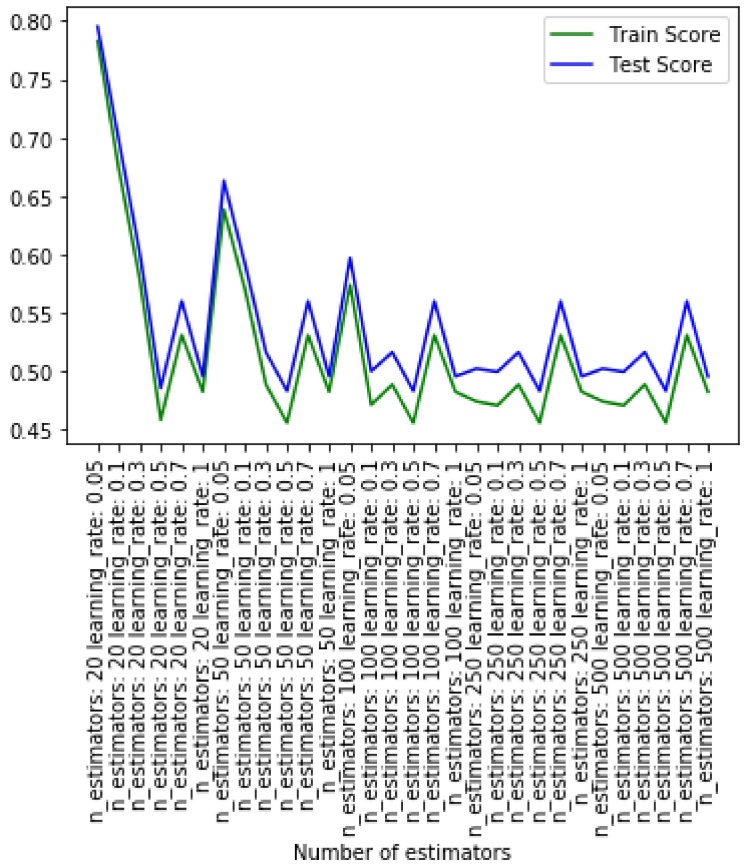
In [42]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [20,50,100,250,500],
              "learning_rate":[0.05,0.1,0.3,0.5,0.7,1],
              "loss" : ["exponential", "linear", "square"]}
lreg = LinearRegression()

grid_adb_lreg = GridSearchCV(AdaBoostRegressor(lreg, random_state = 0), param_gr
grid_adb_lreg_model = grid_adb_lreg.fit(X_train, y_train)

print(grid_adb_lreg_model.best_params_)
print('Validation score: ',grid_adb_lreg_model.best_score_)

print('Train Score: ',grid_adb_lreg_model.score(X_train, y_train))
print('Test Score: ',grid_adb_lreg_model.score(X_test, y_test))

y_pred = grid_adb_lreg_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'learning_rate': 0.05, 'loss': 'exponential', 'n_estimators': 20}
Validation score:  0.7941098699117426
Train Score:  0.8028448384554943
Test Score:  0.8081187716196271
MSE:  27428662816.647167
```

Default parameters for linear regression are used in adaboosting.

Adaboosting for Lasso Regression

In [31]:
```
from sklearn.linear_model import Lasso
from sklearn.ensemble import AdaBoostRegressor
%matplotlib inline

n_estimators = [20,50,100,250,500]
learning_rate = [0.05,0.1,0.3,0.5,0.7,1]
param_list = []
train_score_list = []
test_score_list = []

lasso = Lasso(alpha = 1, max_iter = 1000, tol = 0.1, random_state = 0)

for estimator in n_estimators:
    for rate in learning_rate:
        adb_reg = AdaBoostRegressor(lasso, random_state=0, n_estimators = estima

        adb_reg.fit(X_train, y_train)
        train_score_list.append(adb_reg.score(X_train, y_train))
        test_score_list.append(adb_reg.score(X_test, y_test))
        param_list.append("n_estimators: " + str(estimator) + ' ' + "learning_ra

plt.plot(param_list, train_score_list, c = 'g', label = 'Train Score')
plt.plot(param_list, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel("Number of estimators")
plt.legend()
plt.xticks(rotation = 90)
plt.show()
```



In [32]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [20,50,100,250,500],
              "learning_rate":[0.05,0.1,0.3,0.5,0.7,1],
              "loss" : ["exponential", "linear", "square"]}
lasso = Lasso(alpha = 1, max_iter = 1000, tol = 0.1, random_state = 0)

grid_adb_lasso = GridSearchCV(AdaBoostRegressor(lasso, random_state = 0), param_
grid_adb_lasso_model = grid_adb_lasso.fit(X_train, y_train)

print(grid_adb_lasso_model.best_params_)
print('Validation score: ',grid_adb_lasso_model.best_score_)

print('Train Score: ',grid_adb_lasso_model.score(X_train, y_train))
print('Test Score: ',grid_adb_lasso_model.score(X_test, y_test))

y_pred = grid_adb_lasso_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'learning_rate': 0.05, 'loss': 'exponential', 'n_estimators': 20}
Validation score:  0.7943769838727383
Train Score:  0.8015160513474398
Test Score:  0.8072484926447483
MSE:  27553065754.651333
```

For Lasso Regression, the best parameters obtained through Porject 1 are used for adaboosting with and without Grid Search.

Gradient Boosting

In [36]:
```
from  sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {"n_estimators": [50,100,150,250,500],
              "learning_rate":[0.05,0.1,0.3,0.5,0.7,1],
              "max_depth": [3,5,7],
              "max_features":['auto', 'sqrt', 'log2']}

gbrt_reg = GridSearchCV(GradientBoostingRegressor(random_state = 0), param_grid)
gbrt_reg_model = gbrt_reg.fit(X_train,y_train)

print(gbrt_reg_model.best_params_)
print('Validation score: ',gbrt_reg_model.best_score_)

print('Train Score: ',gbrt_reg_model.score(X_train, y_train))
print('Test Score: ',gbrt_reg_model.score(X_test, y_test))

y_pred = gbrt_reg_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'learning_rate': 0.1, 'max_depth': 5, 'max_features': 'log2', 'n_estimators':
500}
Validation score:  0.8667677659666434
Train Score:  0.9513365796588781
Test Score:  0.8905497686035402
MSE:  15645477765.164137
```

In [40]:
```
from  sklearn.ensemble import GradientBoostingRegressor

gbrt_reg = GradientBoostingRegressor(random_state = 0, learning_rate = 0.1, max_
                                     max_features = 'log2', n_estimators = 500)
gbrt_reg_model = gbrt_reg.fit(X_train,y_train)

print('Train Score: ',gbrt_reg_model.score(X_train, y_train))
print('Test Score: ',gbrt_reg_model.score(X_test, y_test))

y_pred = gbrt_reg_model.predict(X_test)
print('MSE: ',mean_squared_error(y_test, y_pred))

%matplotlib inline

plt.subplots(figsize = (40,40))
y_list = gbrt_reg_model.feature_importances_
y_pos = np.arange(len(y_list))
features = X.columns
plt.barh(y_pos, y_list, align='center', alpha=0.4)
plt.yticks(y_pos, features)
```

```
Train Score:  0.9513365796588781
Test Score:  0.8905497686035402
MSE:  15645477765.164137
```

sqft_above is the most important feature followed by grade, bathrooms, sqft_living15 and view in descending order of feature importance. Top 5 features in decreasing order of importance are named here.

Principal Component Analysis

In [33]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 0.95)
X_train_with_pca = pca.fit_transform(X_train)
X_test_with_pca = pca.transform(X_test)
```

In [35]:
```
X_train_with_pca.shape
```

Out[35]:
```
(16206, 66)
```

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

Out[37]:
```
(16206, 83)
```

After applying PCA with 95% explained variance, number of features are reduced to 66 from our earlier analysis with 83 features.

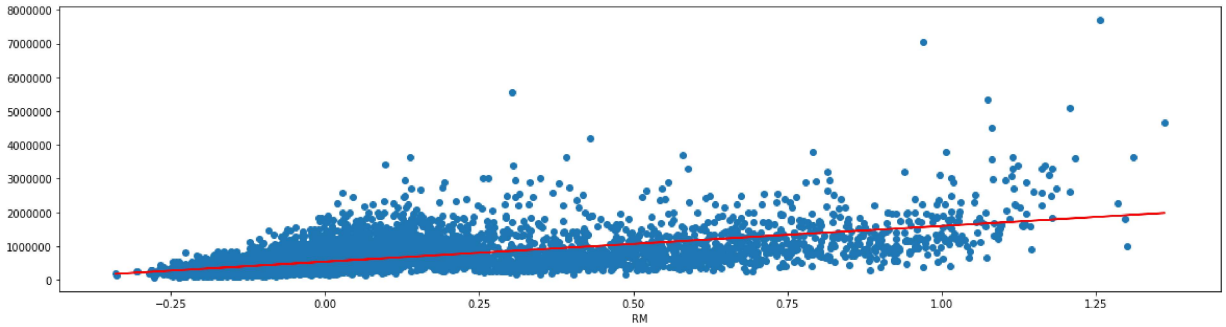
Linear Regression with PCA

In [38]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lreg = LinearRegression()

lreg.fit(X_train_with_pca, y_train)
print('Train Score: ',lreg.score(X_train_with_pca, y_train))
print('Test Score: ',lreg.score(X_test_with_pca, y_test))

y_pred = lreg.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
Train Score:  0.7246522481879739
Test Score:  0.7161870304054059
MSE:  40569941685.852394
```

In [39]:
```
%matplotlib inline
import matplotlib.pyplot as plt

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

plt.subplots(figsize = (20,5))
plt.plot(X_train_rm, y_predict, c = 'r')
plt.scatter(X_train_rm,y_train)
plt.xlabel('RM')
```

Out[39]:
```
Text(0.5, 0, 'RM')
```



Polynomial Regression with PCA

Degree : 2

In [40]:
```
from  sklearn.preprocessing  import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
X_train_poly = poly.fit_transform(X_train_with_pca)
X_test_poly = poly.transform(X_test_with_pca)
lreg.fit(X_train_poly, y_train)

train_score = lreg.score(X_train_poly, y_train)
test_score = lreg.score(X_test_poly, y_test)

y_pred = lreg.predict(X_test_poly)
mse = mean_squared_error(y_test, y_pred)

print('Train Score: ', train_score)
print('Test Score: ', test_score)
print('MSE: ', mse)
```

```
Train Score:  0.895692641179714
Test Score:  -3.997607947179766e+18
MSE:  5.714422478001948e+29
```

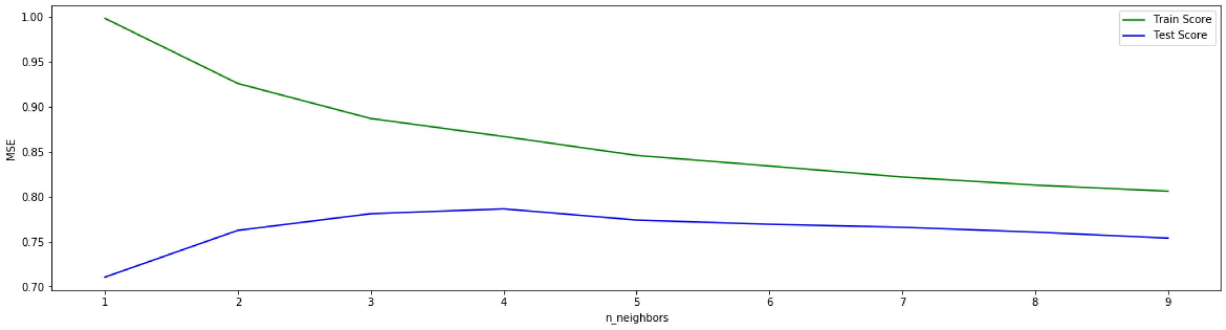
Degree : 3

K-Neighbors Regressor with PCA

K Neighbors Regressor with PCA

In [43]:
```
from sklearn.neighbors import KNeighborsRegressor

%matplotlib inline
train_score_array = []
test_score_array = []

for k in range(1,10):
    knn_reg = KNeighborsRegressor(k)
    knn_reg.fit(X_train_with_pca, y_train)
    train_score_array.append(knn_reg.score(X_train_with_pca, y_train))
    test_score_array.append(knn_reg.score(X_test_with_pca, y_test))

x_axis = range(1,10)

plt.subplots(figsize = (20,5))
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('MSE')
```

Out[43]: Text(0, 0.5, 'MSE')

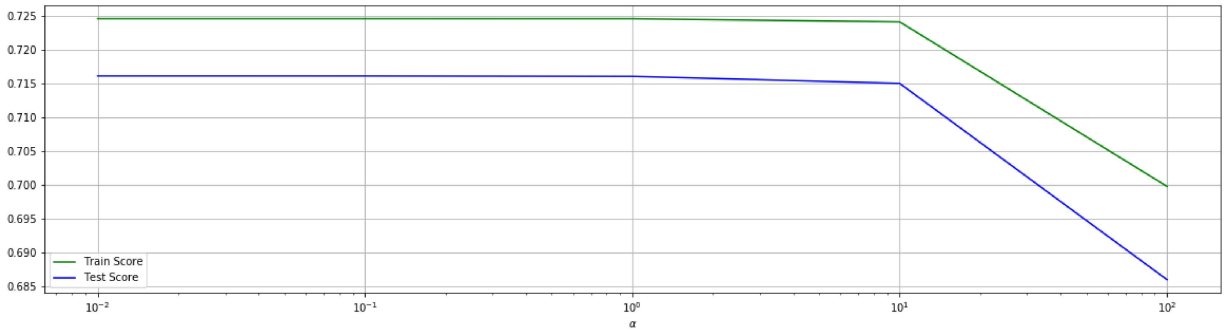


In [44]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

tuned_parameters = dict(n_neighbors = x_axis)

knn_reg = KNeighborsRegressor()
gri1 = GridSearchCV(knn_reg, tuned_parameters, cv = 3, return_train_score = True
gri_model1 = gri1.fit(X_train_with_pca, y_train)

print(gri_model1.best_params_)
print('validation score: ', gri_model1.best_score_)

print('Train Score: ',gri_model1.score(X_train_with_pca, y_train))
print('Test Score: ',gri_model1.score(X_test_with_pca, y_test))

y_pred = gri_model1.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'n_neighbors': 4}
validation score:  0.7496495568029958
Train Score:  0.8669970303714798
Test Score:  0.7863737613950257
MSE:  30537026039.19648
```

Ridge Regression with PCA

In [45]:
```
from sklearn.linear_model import Ridge
%matplotlib inline
import matplotlib.pyplot as plt

x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train_with_pca, y_train)
    train_score_list.append(ridge.score(X_train_with_pca, y_train))
    test_score_list.append(ridge.score(X_test_with_pca, y_test))


plt.subplots(figsize = (20,5))
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
plt.grid()
```

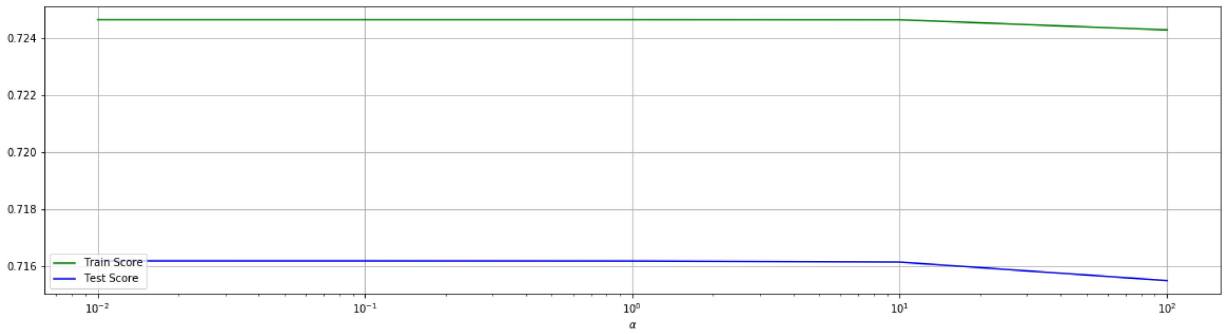


In [46]:
```
np.random.seed(0)

tuned_parameters = [{'alpha':x_range}]

ridge = Ridge(max_iter = 1000, tol = 0.1, random_state = 0)
gri2 = GridSearchCV(ridge, tuned_parameters, cv = 5, return_train_score = True, 
gri_model2 = gri2.fit(X_train_with_pca, y_train)

print(gri2.best_params_)
print('validation score: ', gri2.best_score_)

print('Train Score: ',gri_model2.score(X_train_with_pca, y_train))
print('Test Score: ',gri_model2.score(X_test_with_pca, y_test))

y_pred = gri_model2.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'alpha': 1}
validation score:  0.7221226581211907
Train Score:  0.7246471156552796
Test Score:  0.7161171782320283
MSE:  40579926777.8114
```

Lasso Regression with PCA

In [56]:
```
from sklearn.linear_model import Lasso

x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    lasso = Lasso(alpha, max_iter = 1000, random_state = 0)
    lasso.fit(X_train_with_pca,y_train)
    train_score_list.append(lasso.score(X_train_with_pca,y_train))
    test_score_list.append(lasso.score(X_test_with_pca, y_test))


plt.subplots(figsize = (20,5))
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
plt.grid()
```



In [57]:
```
np.random.seed(0)

tuned_parameters = [{'alpha':x_range}]

lasso = Lasso(max_iter = 1000, tol = 0.1, random_state = 0)
gri3 = GridSearchCV(lasso, tuned_parameters, cv = 5, return_train_score = True,
gri_model3 = gri3.fit(X_train_with_pca,y_train)

print(gri_model3.best_params_)
print('validation score: ', gri_model3.best_score_)

print('Train Score: ',gri_model3.score(X_train_with_pca, y_train))
print('Test Score: ',gri_model3.score(X_test_with_pca, y_test))

y_pred = gri_model3.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'alpha': 10}
validation score:  0.7221112813711809
Train Score:  0.7246486329348281
Test Score:  0.7161478889811224
MSE:  40575536797.671814
```

SGD Regressor with PCA

In [59]:
```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(random_state = 0, max_iter = 10000, penalty = 'l2', learn
                       eta0 = 0.05, shuffle = False)
sgd_reg.fit(X_train_with_pca, y_train)
print('Train Score: ',sgd_reg.score(X_train_with_pca, y_train))
print('Test Score: ',sgd_reg.score(X_test_with_pca, y_test))

print("SGD regression coefficient: " + str(sgd_reg.coef_))
print("SGD regression intercept: " + str(sgd_reg.intercept_))
print("SGD regression iterations: " + str(sgd_reg.n_iter_))

y_pred = sgd_reg.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
Train Score:  0.7099803300192229
Test Score:  0.6983915201491695
SGD regression coefficient: [ 321318.66651668 1087150.33931537  133259.890675
85 -223368.18324173
  385490.18347982   48858.09088123 -295248.4194576  -159126.72695693
  124086.19125057  169513.66947728 -106086.72082595  190691.70376089
   16356.55197333 -123930.64369857 -118261.3917086  -167390.83448448
  -92441.83532095  254828.52835172  401808.94454609  199926.41209645
  200464.14779333   53564.26953231  286978.03319123 -201245.73637702
  214778.28709824   35728.52428234   83791.6605875    55809.56202557
 -182601.01477617 -303273.56744408   50576.33596761  200260.99050931
 -407198.51947493   61645.15413051   62573.6354875   168667.68852537
  -10328.32663695   67003.98441451 -107769.00785339  162261.42839026
  -22040.07728785  -44364.17189543 -113434.85976702  -23369.23265217
   20919.84962793   29314.51861803   81665.33449444   56059.0505672
   -3179.38537381   70900.4282609  -160155.81824191  -64397.14000558
  -41913.91316138  278233.39066784  -97183.13302858  -48378.28519297
 -149465.24646306  -18340.78540291  145272.79110322 -121210.97217428
  -55509.16335998  136069.48634126  142224.77489317  -76607.34454712
```

SVM Regressor with PCA

Linear SVR

In [60]:
```
from sklearn.svm import LinearSVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for e in [0.01, 0.1, 1, 10]:
        linear = LinearSVR(C = C, epsilon = e)
        linear.fit(X_train_with_pca, y_train)
        print('Train Score: ',linear.score(X_train_with_pca,y_train),'C: ' ,C,'e
        print('Test Score: ',linear.score(X_test_with_pca, y_test),'C: ' ,C,'e:
```

```
Train Score:  -2.194124311013772 C:  0.01 e:  0.01
Test Score:  -2.084795024474868 C:  0.01 e:  0.01
Train Score:  -2.194124311013772 C:  0.01 e:  0.1
Test Score:  -2.084795024474868 C:  0.01 e:  0.1
Train Score:  -2.194124311013772 C:  0.01 e:  1
Test Score:  -2.084795024474868 C:  0.01 e:  1
Train Score:  -2.194124311013772 C:  0.01 e:  10
Test Score:  -2.084795024474868 C:  0.01 e:  10
Train Score:  -2.182242545433953 C:  0.1 e:  0.01
Test Score:  -2.07366968196057 C:  0.1 e:  0.01
Train Score:  -2.182242545433953 C:  0.1 e:  0.1
Test Score:  -2.07366968196057 C:  0.1 e:  0.1
Train Score:  -2.182242545433953 C:  0.1 e:  1
Test Score:  -2.07366968196057 C:  0.1 e:  1
Train Score:  -2.182242545433953 C:  0.1 e:  10
Test Score:  -2.07366968196057 C:  0.1 e:  10
Train Score:  -2.0651991270327734 C:  1 e:  0.01
Test Score:  -1.9640532890386122 C:  1 e:  0.01
Train Score:  -2.0651991270327734 C:  1 e:  0.1
```

In [61]:
```
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.01, 0.1, 1, 10, 100, 1000, 10000],
              'epsilon': [0.01, 0.1, 1, 10]}

grid_search_linear = GridSearchCV(LinearSVR(), param_grid, cv=5, return_train_sc
svr_model_linear = grid_search_linear.fit(X_train_with_pca, y_train)

print(svr_model_linear.best_params_)
print('Validation score:', svr_model_linear.best_score_)

print('Train Score: ',svr_model_linear.score(X_train_with_pca,y_train))
print('Test Score: ',svr_model_linear.score(X_test_with_pca,y_test))

y_pred = svr_model_linear.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'epsilon': 10}
Validation score: 0.654565398975611
Train Score:  0.6573963281596407
Test Score:  0.6465903783939372
MSE:  50518507876.00592
```

Linear Kernel

In [62]:
```
from sklearn.svm import SVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for e in [0.01, 0.1, 1, 10]:
        linear = SVR(kernel = 'linear', C = C, epsilon = e)
        linear.fit(X_train_with_pca, y_train)
        print('Train Score: ',linear.score(X_train_with_pca,y_train),'C: ' ,C,'e
        print('Test Score: ',linear.score(X_test_with_pca, y_test),'C: ' ,C,'eps
```

```
Train Score:  -0.058841829990694094 C:  0.01 epsilon:  0.01
Test Score:  -0.06454550095913802 C:  0.01 epsilon:  0.01
Train Score:  -0.05884182999069254 C:  0.01 epsilon:  0.1
Test Score:  -0.06454550095913647 C:  0.01 epsilon:  0.1
Train Score:  -0.05884176534582575 C:  0.01 epsilon:  1
Test Score:  -0.06454543491795017 C:  0.01 epsilon:  1
Train Score:  -0.05885169202622387 C:  0.01 epsilon:  10
Test Score:  -0.06455542250303736 C:  0.01 epsilon:  10
Train Score:  -0.058673269208896484 C:  0.1 epsilon:  0.01
Test Score:  -0.0643849166299848 C:  0.1 epsilon:  0.01
Train Score:  -0.058673269373305637 C:  0.1 epsilon:  0.1
Test Score:  -0.06438491692653958 C:  0.1 epsilon:  0.1
Train Score:  -0.05867337280312346 C:  0.1 epsilon:  1
Test Score:  -0.06438502427566539 C:  0.1 epsilon:  1
Train Score:  -0.05867444229038887 C:  0.1 epsilon:  10
Test Score:  -0.06438608665119672 C:  0.1 epsilon:  10
Train Score:  -0.05692176200658294 C:  1 epsilon:  0.01
Test Score:  -0.0627125655736398 C:  1 epsilon:  0.01
Train Score:  -0.05692176200658294 C:  1 epsilon:  0.1
```

In [63]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

param_grid = {'C': [0.1, 1, 100, 1000, 10000],
              'epsilon': [0.01, 0.1, 1, 10]}

grid_search_linear = GridSearchCV(SVR(kernel = 'linear'), param_grid, cv=5, retu
svr_model_linear = grid_search_linear.fit(X_train_with_pca, y_train)

print(svr_model_linear.best_params_)
print('Validation score:', svr_model_linear.best_score_)

print('Train Score: ',svr_model_linear.score(X_train_with_pca,y_train))
print('Test Score: ',svr_model_linear.score(X_test_with_pca,y_test))

y_pred = svr_model_linear.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'epsilon': 10}
Validation score: 0.6551977128363198
Train Score:  0.6576385005575971
Test Score:  0.6469002013230347
MSE:  50474219913.463615
```

RBF Kernel

In [64]:
```
from sklearn.svm import SVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for g in [0.01, 0.1, 1, 10]:
        rbf = SVR(kernel = 'rbf', C = C, gamma = g)
        rbf.fit(X_train_with_pca, y_train)
        print('Train Score: ',rbf.score(X_train_with_pca,y_train),'C: ' ,C,'gamm
        print('Test Score: ',rbf.score(X_test_with_pca, y_test),'C: ' ,C,'gamma:
```

```
Train Score:  -0.058860059237787254 C:  0.01 gamma:  0.01
Test Score:  -0.06456286024248414 C:  0.01 gamma:  0.01
Train Score:  -0.05885741041057413 C:  0.01 gamma:  0.1
Test Score:  -0.0645603341908334 C:  0.01 gamma:  0.1
Train Score:  -0.05885561866504507 C:  0.01 gamma:  1
Test Score:  -0.06455867636411639 C:  0.01 gamma:  1
Train Score:  -0.058859573544907784 C:  0.01 gamma:  10
Test Score:  -0.06456243989171129 C:  0.01 gamma:  10
Train Score:  -0.05885670747123428 C:  0.1 gamma:  0.01
Test Score:  -0.0645596637699084 C:  0.1 gamma:  0.01
Train Score:  -0.05883115856097376 C:  0.1 gamma:  0.1
Test Score:  -0.06453534975947983 C:  0.1 gamma:  0.1
Train Score:  -0.058812775264349026 C:  0.1 gamma:  1
Test Score:  -0.06451830209971221 C:  0.1 gamma:  1
Train Score:  -0.05885253813345748 C:  0.1 gamma:  10
Test Score:  -0.06455615193164888 C:  0.1 gamma:  10
Train Score:  -0.0588242054244168 C:  1 gamma:  0.01
Test Score:  -0.06452872255484654 C:  1 gamma:  0.01
Train Score:  -0.05856072572082982 C:  1 gamma:  0.1
```

In [65]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = {'C': [0.1, 1, 100, 1000, 10000],
              'gamma': [0.01, 0.1, 1, 10]}

grid_search_rbf = GridSearchCV(SVR(kernel = 'rbf'), param_grid, cv=5, return_tra
svr_model_rbf = grid_search_rbf.fit(X_train_with_pca, y_train)

print(svr_model_rbf.best_params_)
print('Validation score:', svr_model_rbf.best_score_)

print('Train Score: ',svr_model_rbf.score(X_train_with_pca,y_train))
print('Test Score: ',svr_model_rbf.score(X_test_with_pca,y_test))

y_pred = svr_model_rbf.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'gamma': 0.1}
Validation score: 0.5746923021550296
Train Score:  0.5931989297612039
Test Score:  0.576747235785402
MSE:  60502308922.281
```

Poly Kernel

In [66]:
```
from sklearn.svm import SVR

for C in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    for coef in [0.01, 0.1, 1, 10]:
        poly = SVR(kernel = 'poly', degree = 3, C = C, coef0 = coef)
        poly.fit(X_train_with_pca, y_train)
        print('Train Score: ',poly.score(X_train_with_pca,y_train),'C: ' ,C,'coe
        print('Test Score: ',poly.score(X_test_with_pca, y_test),'C: ' ,C,'coef0
```

```
Train Score:  -0.058860483903156346 C:  0.01 coef0:  0.01
Test Score:  -0.06456326796065692 C:  0.01 coef0:  0.01
Train Score:  -0.05886047368652991 C:  0.01 coef0:  0.1
Test Score:  -0.06456325815620478 C:  0.01 coef0:  0.1
Train Score:  -0.05885951995517269 C:  0.01 coef0:  1
Test Score:  -0.06456234384131343 C:  0.01 coef0:  1
Train Score:  -0.058775620542531604 C:  0.01 coef0:  10
Test Score:  -0.0644824379694604 C:  0.01 coef0:  10
Train Score:  -0.0588604823529999 C:  0.1 coef0:  0.01
Test Score:  -0.06456326647968114 C:  0.1 coef0:  0.01
Train Score:  -0.05886038018674067 C:  0.1 coef0:  0.1
Test Score:  -0.06456316843516197 C:  0.1 coef0:  0.1
Train Score:  -0.05885185583913377 C:  0.1 coef0:  1
Test Score:  -0.06455504481845953 C:  0.1 coef0:  1
Train Score:  -0.05799558004386252 C:  0.1 coef0:  10
Test Score:  -0.06373849888749161 C:  0.1 coef0:  10
Train Score:  -0.05886046685143276 C:  1 coef0:  0.01
Test Score:  -0.06456325166992039 C:  1 coef0:  0.01
Train Score:  -0.05885946791456531 C:  1 coef0:  0.1
```

In [67]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = {'C': [0.1, 1, 10, 100, 1000, 10000],
              'coef0': [0.01, 0.1, 1, 10]}

grid_search_poly = GridSearchCV(SVR(kernel = 'poly'), param_grid, cv=5, return_t
svr_model_poly = grid_search_poly.fit(X_train_with_pca, y_train)

print(svr_model_poly.best_params_)
print('Validation score:', svr_model_poly.best_score_)

print('Train Score: ',svr_model_poly.score(X_train_with_pca,y_train))
print('Test Score: ',svr_model_poly.score(X_test_with_pca,y_test))

y_pred = svr_model_poly.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'C': 10000, 'coef0': 10}
Validation score: 0.6723458446827132
Train Score:  0.6748662320356393
Test Score:  0.6655786620457516
MSE:  47804207815.762276
```

Decision Tree Regressor with PCA

In [68]:
```
from sklearn.tree import DecisionTreeRegressor

for leaf in [2, 5, 7, 10, 15, 20]:
    dtr = DecisionTreeRegressor(min_samples_leaf = leaf)
    dtr.fit(X_train_with_pca, y_train)
    print('Train Score: ',dtr.score(X_train_with_pca,y_train),'min_samples_leaf:
    print('Test Score: ',dtr.score(X_test_with_pca, y_test),'min_samples_leaf: '
```

```
Train Score:  0.9830448873245385 min_samples_leaf:  2
Test Score:  0.7079261757927482 min_samples_leaf:  2
Train Score:  0.9245005886905882 min_samples_leaf:  5
Test Score:  0.7202436219509218 min_samples_leaf:  5
Train Score:  0.8935894650746494 min_samples_leaf:  7
Test Score:  0.7086643407712666 min_samples_leaf:  7
Train Score:  0.8639222180639009 min_samples_leaf:  10
Test Score:  0.7266477501264779 min_samples_leaf:  10
Train Score:  0.8269960842339326 min_samples_leaf:  15
Test Score:  0.7177991681424796 min_samples_leaf:  15
Train Score:  0.8099614268777355 min_samples_leaf:  20
Test Score:  0.716611385920269 min_samples_leaf:  20
```

In [69]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

param_grid = {'min_samples_leaf': range(3,20)}

dtr = GridSearchCV(DecisionTreeRegressor(), param_grid, cv = 5, return_train_sco
decision_tree = dtr.fit(X_train_with_pca, y_train)

print(decision_tree.best_params_)
print('Validation score:', decision_tree.best_score_)

print('Train Score: ',decision_tree.score(X_train_with_pca,y_train))
print('Test Score: ',decision_tree.score(X_test_with_pca,y_test))

y_pred = decision_tree.predict(X_test_with_pca)
print('MSE: ',mean_squared_error(y_test, y_pred))
```

```
{'min_samples_leaf': 13}
Validation score: 0.6769149003279922
Train Score:  0.837159899581537
Test Score:  0.7254812291794326
MSE:  39241372724.33667
```

Deep Learning Model

In [29]:
```
X_train_DLM = np.asarray(X_train)
X_test_DLM = np.asarray(X_test)
y_train_DLM = np.asarray(y_train)
y_test_DLM = np.asarray(y_test)
```

In [31]:
```
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense

#Defining model
dlm = Sequential()
dlm.add(Dense(16, input_dim = X_train_DLM.shape[1], kernel_initializer = 'normal
dlm.add(Dense(8, activation = 'relu', kernel_initializer='normal', name = 'hidde
dlm.add(Dense(1, kernel_initializer='normal'))

#compiling model
dlm.compile(loss = 'mse', optimizer = 'sgd', metrics = ['mse', 'mae'])

#fit model
dlm.fit(X_train_DLM, y_train_DLM, epochs = 20, batch_size = 128)
```

```
Epoch 1/20
127/127 [==============================] - 0s 2ms/step - loss: 194081472512.0
000 - mse: 194081472512.0000 - mae: 281810.7188
Epoch 2/20
127/127 [==============================] - 0s 2ms/step - loss: 132298170368.0
000 - mse: 132298170368.0000 - mae: 229618.0000
Epoch 3/20
127/127 [==============================] - 0s 2ms/step - loss: 131920855040.0
000 - mse: 131920855040.0000 - mae: 232703.9688
Epoch 4/20
127/127 [==============================] - 0s 2ms/step - loss: 131911598080.0
000 - mse: 131911598080.0000 - mae: 232817.0469
Epoch 5/20
127/127 [==============================] - 0s 2ms/step - loss: 131910508544.0
000 - mse: 131910508544.0000 - mae: 233045.9375
Epoch 6/20
127/127 [==============================] - 0s 2ms/step - loss: 131888291840.0
000 - mse: 131888291840.0000 - mae: 233539.3438
Epoch 7/20
```

In [33]:
```
#evaluate model
dlm.evaluate(X_test_DLM, y_test_DLM)
```

```
169/169 [==============================] - 0s 2ms/step - loss: 143058485248.000
0 - mse: 143058485248.0000 - mae: 234001.6562
```

Out[33]:
```
[143058485248.0, 143058485248.0, 234001.65625]
```

In [35]:
```
from sklearn.metrics import mean_squared_error
y_pred_DLM = dlm.predict(X_test_DLM)
print(dlm.evaluate(X_train_DLM,y_train_DLM))
print(dlm.evaluate(X_test_DLM,y_test_DLM))
print(mean_squared_error(y_test_DLM, y_pred_DLM))
```

```
507/507 [==============================] - 1s 1ms/step - loss: 131898712064.000
0 - mse: 131898712064.0000 - mae: 232324.4531
[131898712064.0, 131898712064.0, 232324.453125]
169/169 [==============================] - 0s 2ms/step - loss: 143058485248.000
0 - mse: 143058485248.0000 - mae: 234001.6562
[143058485248.0, 143058485248.0, 234001.65625]
143058489951.65958
```

CONCLUSION

Ensembling

Model	Bagging - SGD Regressor	Pasting - SGD Regressor	Bagging - Decision Tree Regressor	Pasting - Decision Tree Regressor	
max_features	20	20	20	20	
max_samples	0.3	0.5	0.5	0.5	
n_estimators	20	20	250	250	
learning_rate	---	---	---	---	
loss	---	---	---	---	
max_depth	---	---	---	---	
Training Score	0.5589871547840561	0.5662121617324214	0.658800640471781	0.6734508635945906	0.
Validation Score	0.5677115662754662	0.5693555465285673	0.6113193325992352	0.618597061404007	0.
Testing score	0.5509060711366776	0.5587716348344909	0.6278667258154307	0.6354711358183972	0.
MSE	64196201221.81469	63071849995.010414	53194979970.78448	52107959631.94532	27

Project 1 Results

Model	Linear Regression	Polynomial Regression	KNN Regressor	Ridge Regression	Lasso Regression	
Params	default	degree = 2	k = 3	alpha = 0.1	alpha = 1	
Training Score	0.8060	0.9145	0.8896	0.8061	0.8002	
Validation Score	-	-	0.7471	0.8022	0.7982	
Testing score	0.8094	-8.7553e+16	0.7890	0.8093	0.8036	
MSE	27248083255.0241	1.2515e+28	30167671978.2293	27255102161.6693	28070156376.8954	2

Results after PCA

Model	Linear Regression	Polynomial Regression	KNN Regressor	Ridge Regression	Lasso Regressi
Params	default	degree = 2	k = 4	alpha = 1	alpha =
Training Score	0.7247	0.8957	0.8669	0.7246	0.72
Validation Score	-	-	0.7496	0.7221	0.72
Testing score	0.7162	-3.998e+18	0.7864	0.7161	0.71
MSE	40569941685.852394	5.7144e+29	30537026039.19648	40579926777.8114	40575536797.6718

Results from Decision Tree regressor are better after PCA. Results from KNN are similar with or without PCA. For rest of the models, results are better without PCA. Linear regression without PCA and Ridge Regression with alpha = 0.1 wihtout PCA are the best models for prediciting the target variable.