Capstone project final report

House price prediction

Batch details	PGP-ML(Mar-23)
Team members	Sandeep upadyaya, Supreeth T Gowda, Thejoroop Reddy T
Group Number	2
Project title	House price prediction
Mentor	Mr. Sandeep

Index

Project summary

The objective of the project is the predict the house price by using regression models. The dataset consist of 23 features and 21613 records. Among the 23 features, there are 18 integer type, 4 float type and 1 objective type. There are no missing values in the dataset.

Regression models used are linear regression, KNN, SVR, decision tree, random forest, Gradient boost. The different matrices like ‘R squared’, ‘Mean absolute error’ and ‘Mean squared error’ are used to evaluate the different models. Benchmark laid in compare with the final model is linear regression. But, mainly considering the R squared value the final model could get almost 10% better results compared to the benchmark model hence concluded to be the final model

Among all the models, Gradient boosting has performed well with trainset accuracy of 81.9 and testset with 75.5.

House price prediction

1.INTRODUCTION

1.1 Problem statement

A house value is simply more than location and square footage. Like the features that make up a person, an educated party would want to know all aspects that give a house its value. For example, you want to sell a house and you don't know the price which you can take — it can't be too low or too high. To find house price you usually try to find similar properties in your neighborhood and based on gathered data you will try to assess your house price.

1.2 Objective:

To predict the price of house which is not too high or too low using regression models. The project challenges to predict the final price of each home.

1.3 About the Dataset:

Dataset consist of 21613 records and 23 attributes with zero missing values. 23 attributes are:

1. Cid: a notation for a house
2. Day_hours: Date house was sold
3. Price: Price is prediction target
4. Room_bed: Number of Bedrooms/House
5. Room_bath: Number of bathrooms/bedrooms
6. Living_measure: square footage of the home
7. Lot_measure: Square footage of the lot
8. Ceil: Total floors (levels) in house
9. Coast: House which has a view to a waterfront
10. Sight: Has been viewed
11. Condition: How good the condition is (Overall)
12. Quality: grade given to the housing unit, based on grading system
13. Ceil_measure: square footage of house apart from basement
14. Basement_measure: square footage of the basement
15. Yr_built: Built Year
16. Yr_renovated: Year when house was renovated
17. Zipcode: zip

18. Lat: Latitude coordinate
19. Long: Longitude coordinate
20. Living_measure15: Living room area in 2015(implies-- some renovations) This might or might not have affected the lot size area
21. Lot_measure15: lot size area in 2015(implies-- some renovations)
22. Furnished: Based on the quality of room
23. Total_area: Measure of both living and lot

1.4 Null values, data types and unique values of dataset

Sl.no	Variables	Null_values	D_type	Unique
1	cid	0	int64	21436
2	Day hours	0	object	372
3	Price	0	int64	3625
4	Room_bed	0	int64	13
5	Room_bath	0	float64	30
6	Living_measure	0	int64	1038
7	Lot_measure	0	int64	9782
8	Ceil	0	float64	6
9	Coast	0	int64	2
10	Sight	0	int64	5
11	Condition	0	int64	5
12	Quality	0	int64	12
13	Ceil_measure	0	int64	946
14	Basement	0	int64	306
15	yr_built	0	int64	116
16	yr_renovated	0	int64	70
17	Zipcode	0	int64	70

18	Latitude	0	float64	5034
19	longitude	0	float64	752
20	Living_measure15	0	int64	777
21	Lot_measure15	0	int64	8689
22	Furnished	0	int64	2
23	Total_area	0	int64	11163

- There are no missing values in the dataset
- There are no duplicated values
- There are 18 integer type, 4 float type and 1 object type

2. Exploratory Data Analysis(EDA)

Exploratory Data Analysis is an approach in analyzing dataset to summarize their main characteristics, often using statistical graphics and other data visualization methods.

The primary motive of EDA is to:

i. Better understanding of the data

ii. Identifying various data patterns

iii. Handling the outliers

iv. Handling the missing value of dataset

v. Encoding the categorical variables

vi. Knowing duplication of record

vii. Finding the correlation between the attributes

2.1 Dropping attributes: There are attributes like cid, zip code, longitude and latitude which are irrelevant in predicting the target variables. Hence after dropping these attributes the dataset is

	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition
0	20141107T000000	808100	4	3.25	3020	13457	1.0	0	0	5
1	20141204T000000	277500	4	2.50	2550	7500	1.0	0	0	3
2	20150420T000000	404000	3	2.50	2370	4324	2.0	0	0	3
3	20140529T000000	300000	2	1.00	820	3844	1.0	0	0	4
4	20150424T000000	699000	2	1.50	1400	4050	1.0	0	0	4

quality	ceil_measure	basement	yr_built	yr_renovated	living_measure15	lot_measure15	furnished	total_area
9	3020	0	1956	0	2120	7553	1	16477
8	1750	800	1976	0	2260	8800	0	10050
8	2370	0	2006	0	2370	4348	0	6694
6	820	0	1916	0	1520	3844	0	4664
8	1400	0	1954	0	1900	5940	0	5450

2.2 Summary statistics:

Graphs show the form of the distribution of the data and are a very useful tool in exploring a dataset. Besides graphs, statistics that summarize the distribution of the data, are used to transform data into information. The five-number summary, which forms the basis for a boxplot, is a good example of summarizing data. The below table is summary statistics of the dataset

	count	mean	std	min	25%	50%	75%	max
price	21613.0	540182.158793	367362.231718	75000.0	321950.00	450000.00	645000.0	7700000.0
room_bed	21613.0	3.370842	0.930062	0.0	3.00	3.00	4.0	33.0
room_bath	21613.0	2.114757	0.770163	0.0	1.75	2.25	2.5	8.0
living_measure	21613.0	2079.899736	918.440897	290.0	1427.00	1910.00	2550.0	13540.0
lot_measure	21613.0	15106.967566	41420.511515	520.0	5040.00	7618.00	10688.0	1651359.0
ceil	21613.0	1.494309	0.539989	1.0	1.00	1.50	2.0	3.5
coast	21613.0	0.007542	0.086517	0.0	0.00	0.00	0.0	1.0
sight	21613.0	0.234303	0.766318	0.0	0.00	0.00	0.0	4.0
condition	21613.0	3.409430	0.650743	1.0	3.00	3.00	4.0	5.0
quality	21613.0	7.656873	1.175459	1.0	7.00	7.00	8.0	13.0
ceil_measure	21613.0	1788.390691	828.090978	290.0	1190.00	1560.00	2210.0	9410.0
basement	21613.0	291.509045	442.575043	0.0	0.00	0.00	560.0	4820.0
yr_built	21613.0	1971.005136	29.373411	1900.0	1951.00	1975.00	1997.0	2015.0
yr_renovated	21613.0	84.402258	401.679240	0.0	0.00	0.00	0.0	2015.0
living_measure15	21613.0	1986.552492	685.391304	399.0	1490.00	1840.00	2360.0	6210.0
lot_measure15	21613.0	12768.455652	27304.179631	651.0	5100.00	7620.00	10083.0	871200.0

furnished	21613.0	0.196687	0.397503	0.0	0.00	0.00	0.0	1.0
total_area	21613.0	17186.867302	41589.081215	1423.0	7035.00	9575.00	13000.0	1652659.0

Dayhours: 5 factor analysis is reflecting for this column

price: Our taget column value is in 75k - 7700k range. As Mean > Median, it 's Right-Skewed.

Room_bed: Number of bedrooms range from 0 - 33. As Mean slightly > Me dian, it's slightly Right-Skewed.

Room_bath: Number of bathrooms range from 0 - 8. As Mean slightly < Me dian, it's slightly Left-Skewed.

Living_measure: Square footage of house range from 290 - 13,540. As Mea n > Median, it's Right-Skewed.

Lot_measure: Square footage of lot range from 520 - 16,51,359. As Mean al most double of Median, it's highly Right-Skewed.

Ceil: Number of floors range from 1 - 3.5 As Mean ~ Median, it's almost No rmal Distributed.

Coast: As this value represent whether house has waterfront view or not. It's categorical column. From above analysis we got know, very few houses has waterfront view.

Sight: Value ranges from 0 - 4. As Mean > Median, it's Right-Skewed

Condition: Represents rating of house which ranges from 1 - 5. As Mean > Median, it's Right-Skewed

Quality: Representign grade given to house which range from 1 - 13. As Me an > Median, it's Right-Skewed.

Ceil_measure: Square footage of house apart from basement ranges in 290 - 9,410. As Mean > Median, it's Right-Skewed.

Basement: Square footage house basement ranges in 0 - 4,820. As Mean hig hlty > Median, it's Highly Right-Skewed.

yr_built: House built year ranges from 1900 - 2015. As Mean < Median, it's Left-Skewed.

yr_renovated: House renovation year only 2015. So this column can be used as Categorical Variable for knowing whether house is renovated or not.

Lat: Latitude ranges from 47.1559 - 47.7776 As Mean < Median, it's Left-Sk ewed.

Long: Longitude ranges from -122.5190 to -121.315 As Mean > Median, it's Right-Skewed.

Living_measure15: Value ranges from 399 to 6,210. As Mean > Median, it' s Right-Skewed.

Lot_measure15: Value ranges from 651 to 8,71,200. As Mean highly > Med ian, it's Highly Right-Skewed.

Furnished: Representing whether house is furnished or not. It's a Categorica l Variable

Total_area: Total area of house ranges from 1,423 to 16,52,659. As Mean is almost double of Median, it's Highly Right-Skewed

- New attribute 'house_age is created by considering 'sold year' & 'yr_built'

```
# Calculate age of property based on built year and sale year
hdf["house_age"] = hdf['sold_year']-hdf['yr_built']
hdf.head(2)
```

2.3 Types of EDA

There are 3 types of EDA:

i) Univariate analysis

ii) Bivariate analysis

iii) Multivariate analysis

2.3.1 Univariate analysis:

This is simplest form of data analysis, where the data being analyzed consi sts of just one variable. Since it’s a single variable, it doesn’t deal with cau ses or relationships. The main purpose of univariate analysis is to describe the data and find patterns that exist within it.

Analyzing the Variables one by one:

2.3.1.1: Dayhours

Changing the 'Dayhours' to 'month year'

```
# changing 'dayhours' to 'month year'
hdfr = hdf.copy()
hdf.dayhours = hdf.dayhours.str.replace('T000000',"")
hdf.dayhours = pd.to_datetime(hdf.dayhours)
hdf.dayhours = hdf.dayhours.apply(lambda x: x.strftime('%m-%Y'))

hdf.rename(columns={'dayhours':'month_year'}, inplace=True)
```

Number of values for 'month year' is

```
hdf['month_year'].value_counts()
```

```
04-2015    2231
07-2014    2211
06-2014    2180
08-2014    1940
10-2014    1878
03-2015    1875
09-2014    1774
05-2014    1768
12-2014    1471
11-2014    1411
02-2015    1250
01-2015     978
05-2015     646
```

Groupby the 'month year' and 'price' and find the mean of price

```
hdf.groupby(['month_year'])['price'].agg('mean')
```

```
month_year
01-2015    525963.251534
02-2015    507919.603200
03-2015    544057.683200
04-2015    561933.463021
05-2014    548166.600113
05-2015    558193.095975
06-2014    558123.736239
07-2014    544892.161013
08-2014    536527.039691
09-2014    529315.868095
10-2014    539127.477636
11-2014    522058.861800
12-2014    524602.893270
```

So the time line of the sale data of the properties is from May-2014 to May-2015 and April month have the highest mean price

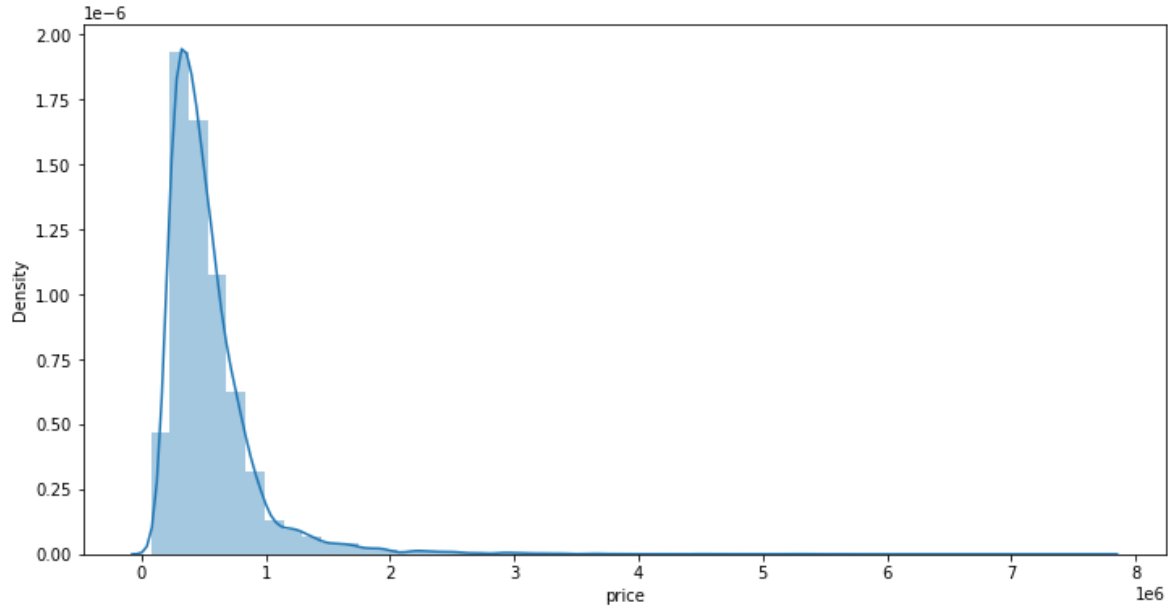
2.3.1.2: Price (Target Variable)

- Summary statistics of Price

```
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
```

- Distribution of price variable

```
#will check for the dstribution of the price variable
plt.figure(figsize=(12,6))
sns.distplot(hdf.price);
```



price of the house is not symmetrical it is rightly skewed with outliers. Price range for most of the house is under 2000000

2.3.1.3: Room_bed

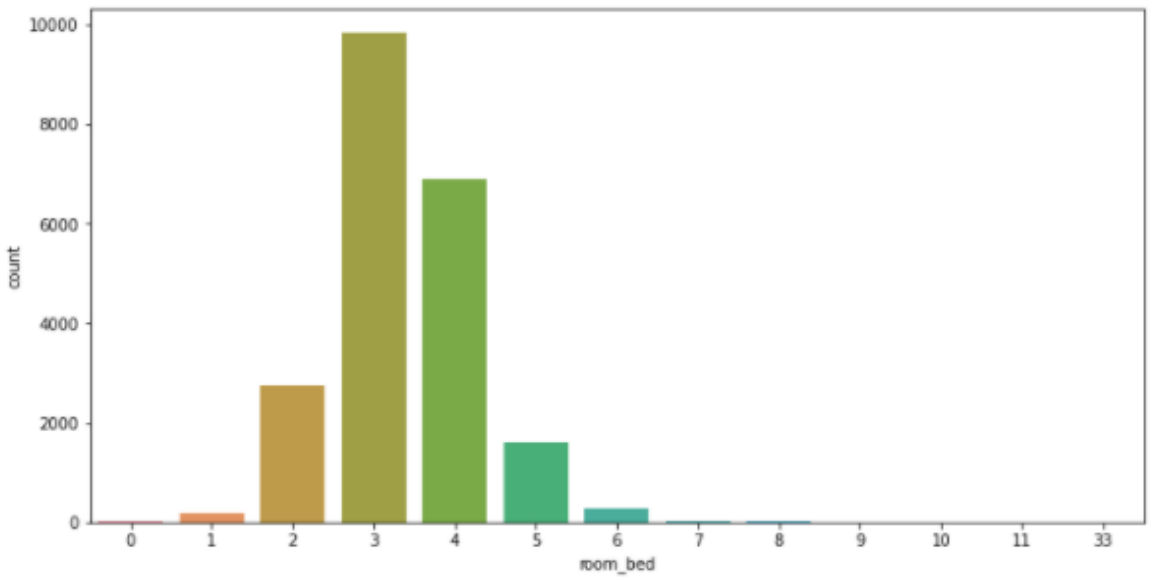
- Number of bed rooms by using python function value_counts

```
hdf['room_bed'].value_counts()
```

```
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: room_bed, dtype: int64
```

- **Graphical representation of room_bed**

```
plt.figure(figsize=(12,6))
sns.countplot(hdf.room_bed);
```



Majority of the house has 3 bedrooms followed by 4 bedrooms, 2 bedrooms,5 bedrooms,6 bedrooms,1 bedroom,7,8 and 0 bedrooms. The 33 bedroom what we got in the above analysis is definitely an outlier it needs to be deleted.

2.3.1.4: Room_bath

- Summary statistics of Price

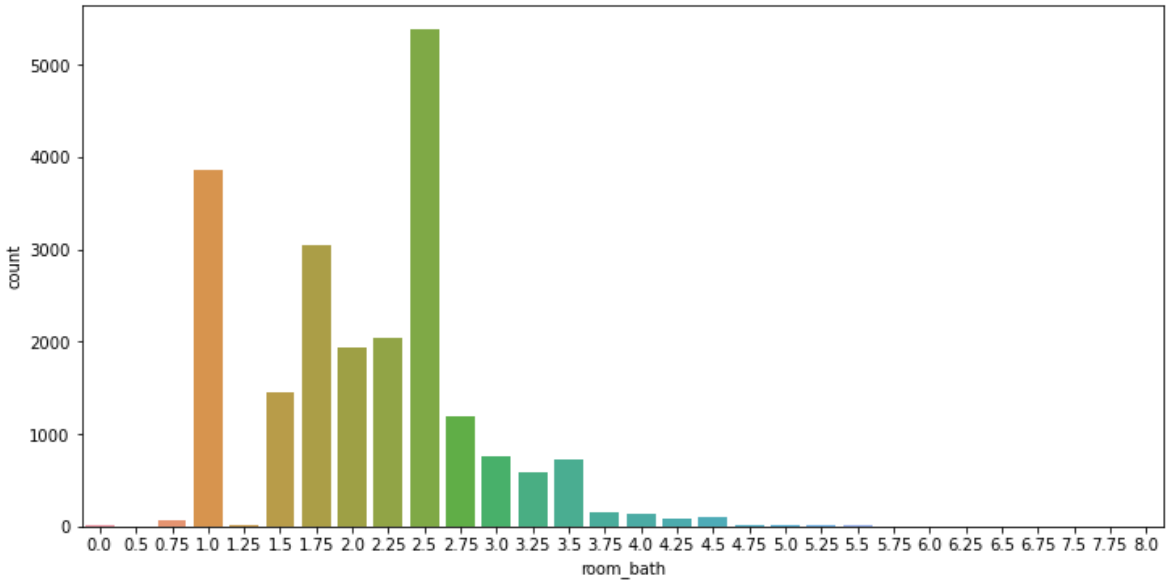
```
hdf.room_bath.describe()
```

```
count    21613.000000
mean         2.114757
std          0.770163
min          0.000000
25%          1.750000
50%          2.250000
75%          2.500000
max          8.000000
Name: room_bath, dtype: float64
```

Number of bathrooms vary from 0 to 8

Graphical representation of bathrooms:

```
plt.figure(figsize=(12,6))
sns.countplot(hdf.room_bath);
```



2.5 number of bathrooms found in majority of the houses followed by 1 bathroom.

2.3.1.5 Living measure (square footage of the home)

Summary statistics and amount of skewness for living measure

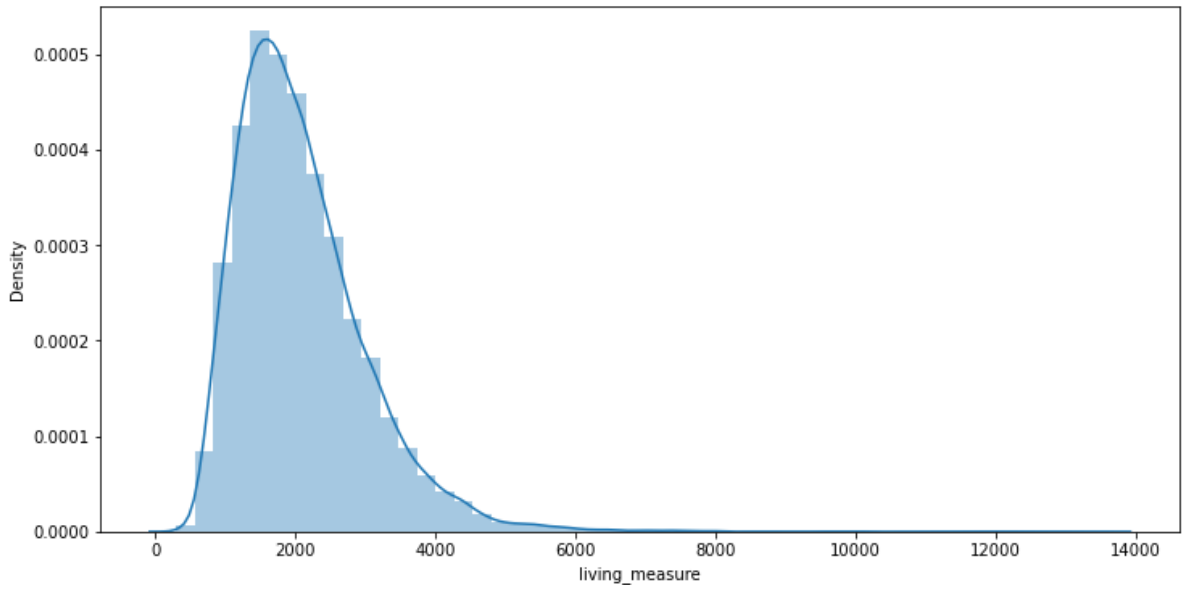
```
plt.figure(figsize=(12,6))
print("Skewness is :",hdf.living_measure.skew())
sns.distplot(hdf.living_measure);
hdf.living_measure.describe()
```

```
Skewness is : 1.471555426802092
```

Living measure is the square footage of the house. Living measure varied from 290 sq.ft to 13540 sq.ft. There is no much deviation of mode from mean. Almost

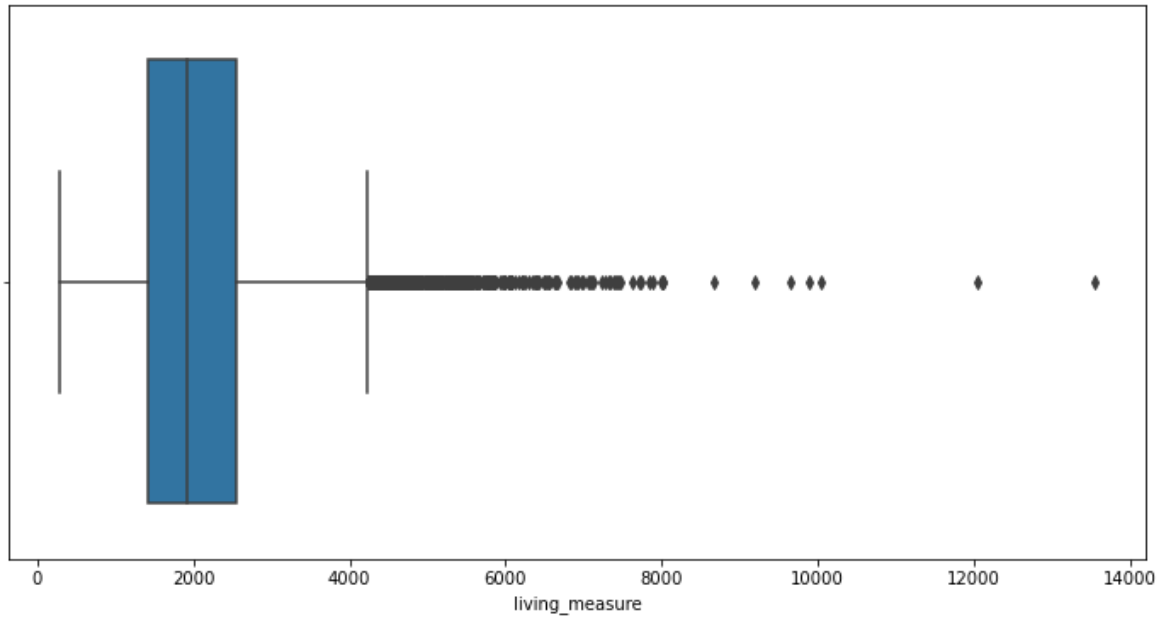
normal distribution with slightly right skewed. Skew factor is 1.47. From the above graph it is observed that majority of the house's living measure is 2000 sq.ft

Distribution of living measure from graph



Box plot for living measure

```
#Let's plot the boxplot for living_measure
plt.figure(figsize=(12,6))
sns.boxplot(hdf.living_measure);
```



Many outliers are observed. It needs to be taken care in the future analysis.

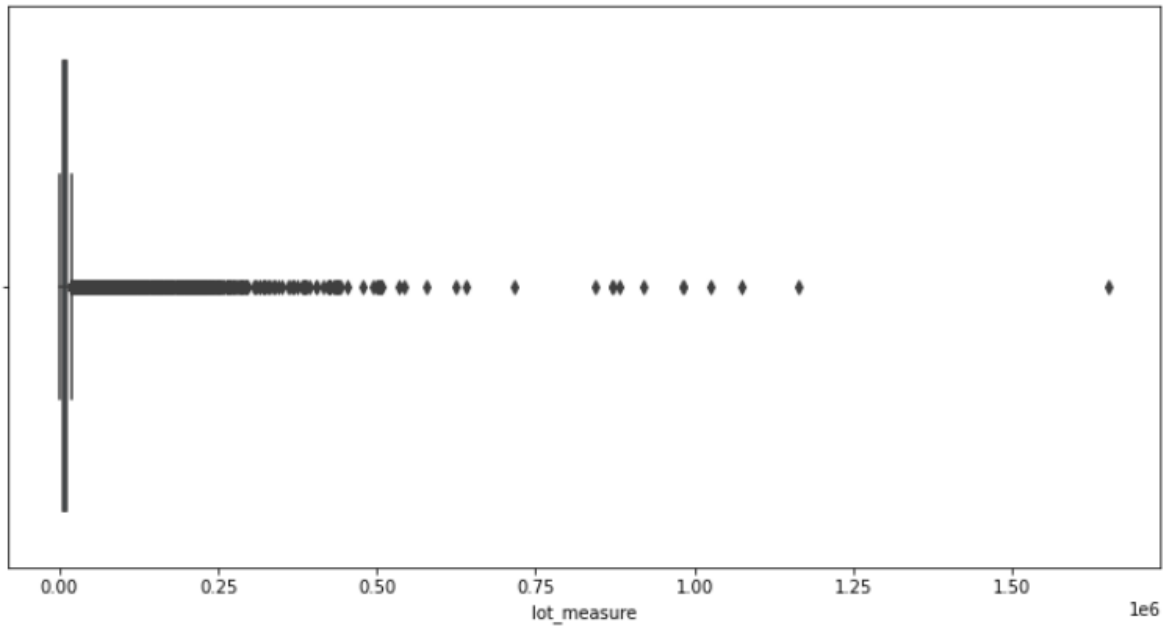
2.3.1.6 lot_measure (square footage of the lot)

Summary statistics and amount of skewness for lot measure

```
plt.figure(figsize=(12,6))
print("Skewness is :",hdf.lot_measure.skew())
sns.boxplot(hdf.lot_measure)
hdf.lot_measure.describe()
```

```
Skewness is : 13.06001895903175
```

Box plot for lot measure



Lot measure varies from 520 to 16.5 Lakh square feet. Distribution is highly skewed with outliers

2.3.1.7 ceil (Total floors in house)

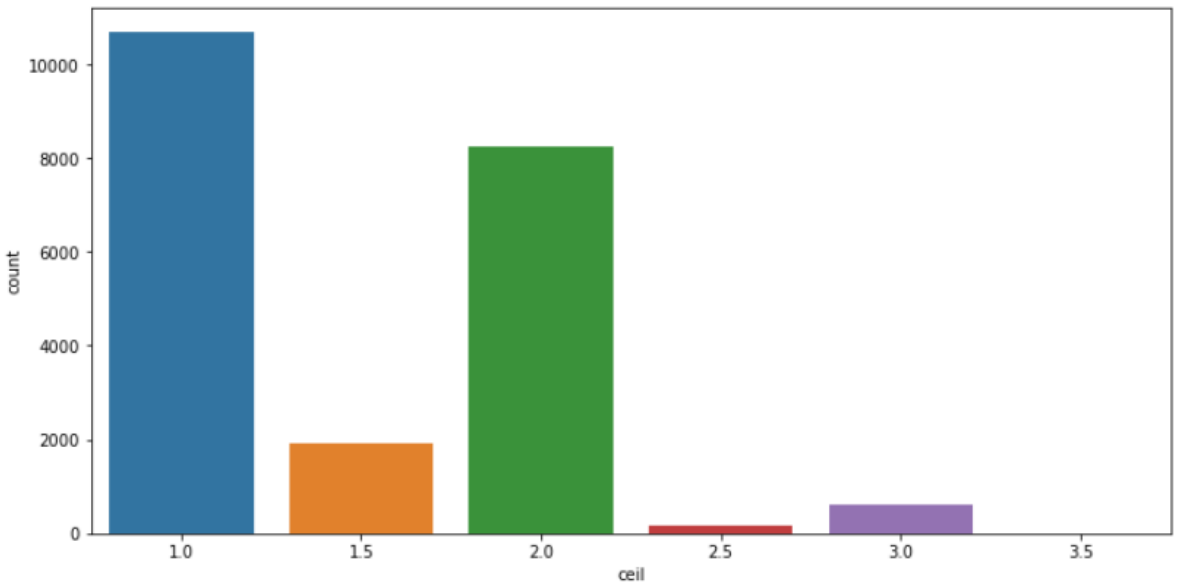
Total number of floors in house by using value_counts

```
hdf.ceil.value_counts()
```

```
1.0    10680
2.0     8241
1.5     1910
3.0      613
2.5      161
3.5        8
Name: ceil, dtype: int64
```

Total floors count by using graph

```
plt.figure(figsize=(12,6))
sns.countplot(hdf.ceil);
```



Most of the house has 1 floor followed by 2 floors,1.5 and 3 floors

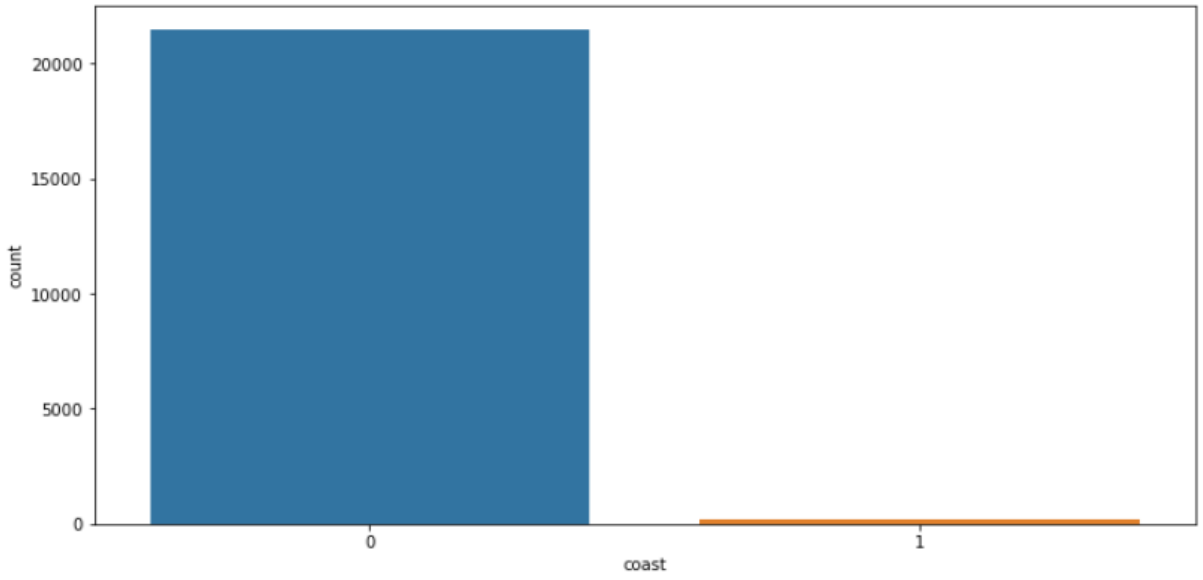
2.3.1.8 coast (House which has a view to a waterfront)

Value count of coast is

```
hdf.coast.value_counts()
```

```
0    21450
1      163
Name: coast, dtype: int64
```

No coast and coast region count from graph is



Coast is found binary type of data. House which have waterfront view represents 1 and house without waterfront view represents 0. Only 163 houses has waterfront view.

2.3.1.9 Sight (represents how many times sight has been viewed)

Value counts for sight is

```
hdf.sight.value_counts()
```

```
0    19489
2      963
3      510
1      332
4      319
Name: sight, dtype: int64
```

Range of the sight varies from 0 to 4. Most of the house have not been viewed. And the respective values of how many times its been viewed is as shown

2.3.1.10: Condition (Overall condition of the house)

Value count for condition of house is

```
hdf.condition.value_counts()
```

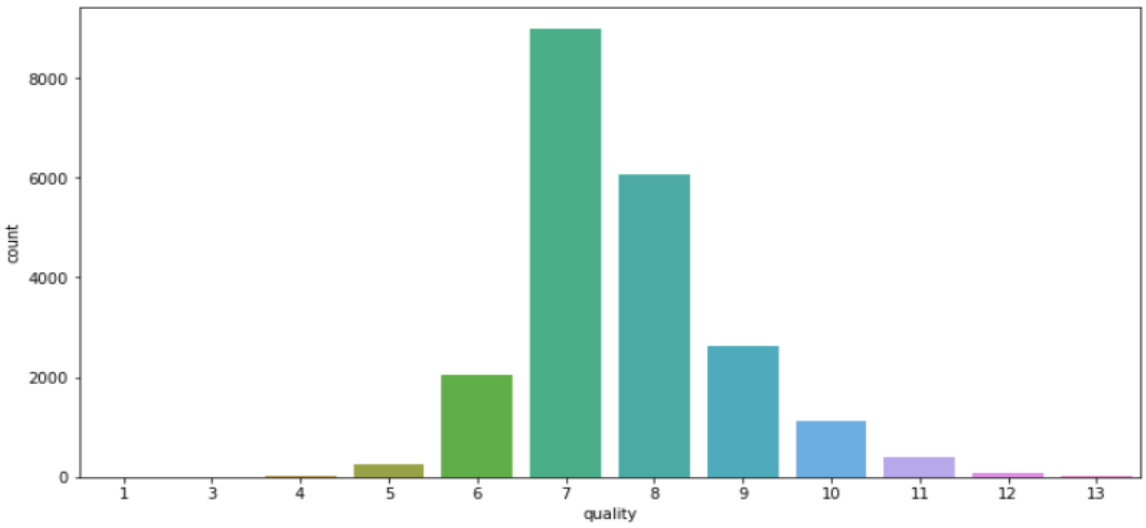
```
3    14031
4     5679
5     1701
2      172
1       30
Name: condition, dtype: int64
```

Overall condition of the house is represented in the scale of 1 to 5. Majority of the house falls in the category of 3

2.3.1.11: Quality (grade given to the housing unit, based on grading system)

Value counts for quality which is based on grading system

```
hdf.quality.value_counts()
plt.figure(figsize=(12,6))
sns.countplot(hdf.quality);
```

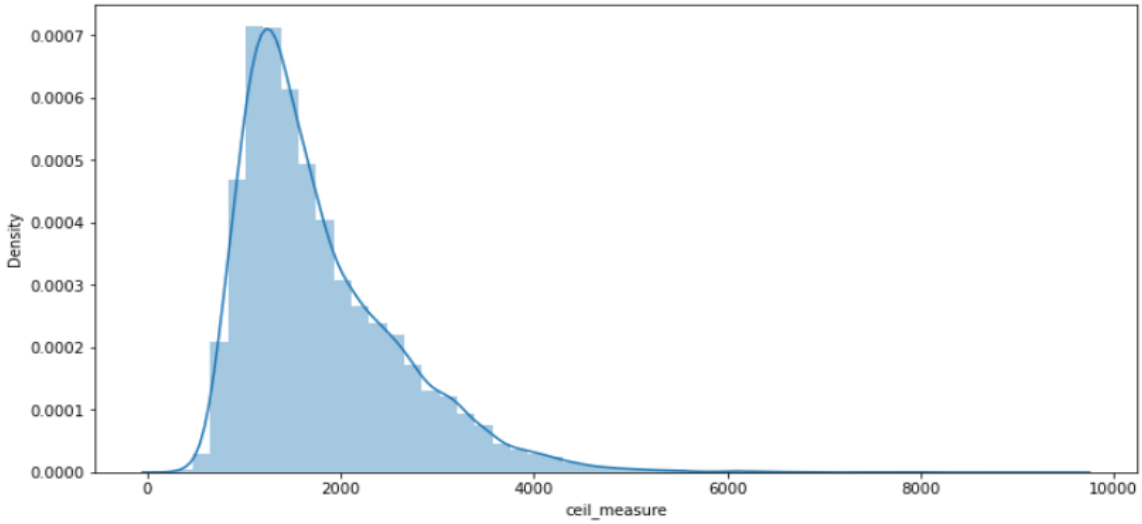


Quality - most properties have quality rating between 6 to 10

There are only 13 properties which have the highest quality rating

2.3.1.12: Ceil_measure (square footage of house apart from basement)

- Graphical representation of ceil measure



- Value_count and skewness for Ceil_measure is

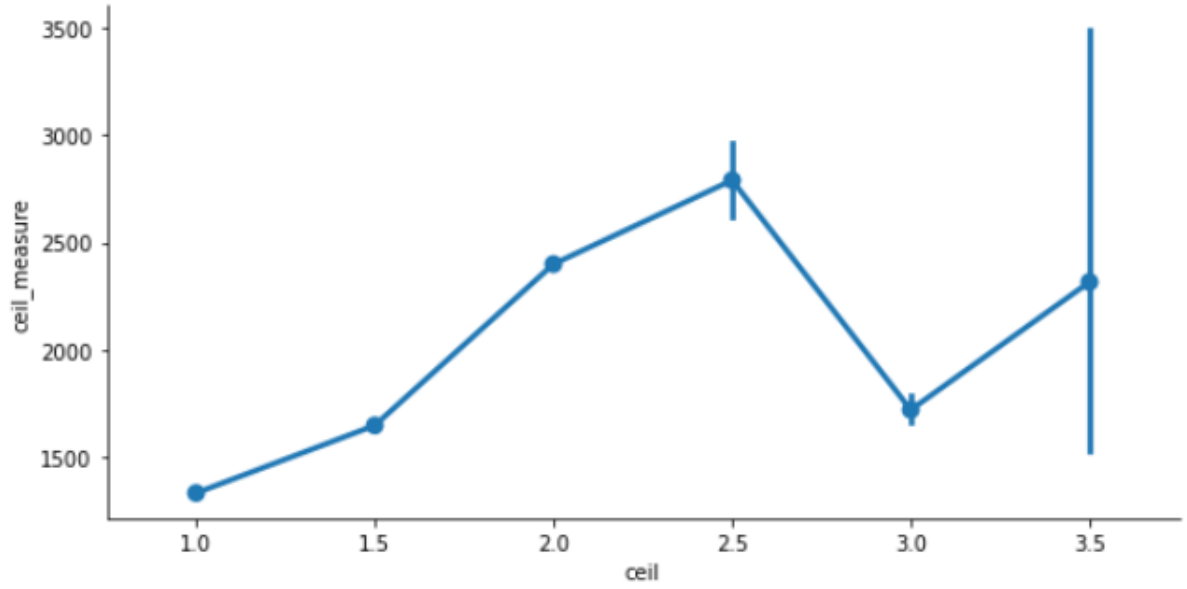
```
print("Skewness is :", hdf.ceil_measure.skew())
plt.figure(figsize=(12,6))
sns.distplot(hdf.ceil_measure)
hdf.ceil_measure.describe()
```

```
Skewness is : 1.4466644733818372
```

```
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: ceil_measure, dtype: float64
```

Relation between ceil and ceil_measure

```
sns.factorplot(x='ceil',y='ceil_measure',data=hdf,size=4,aspect=2);
```



Found there are no direct or linear relation between ceil and ceil_measure

2.3.1.13: Basement_measure (square footage of the basement)

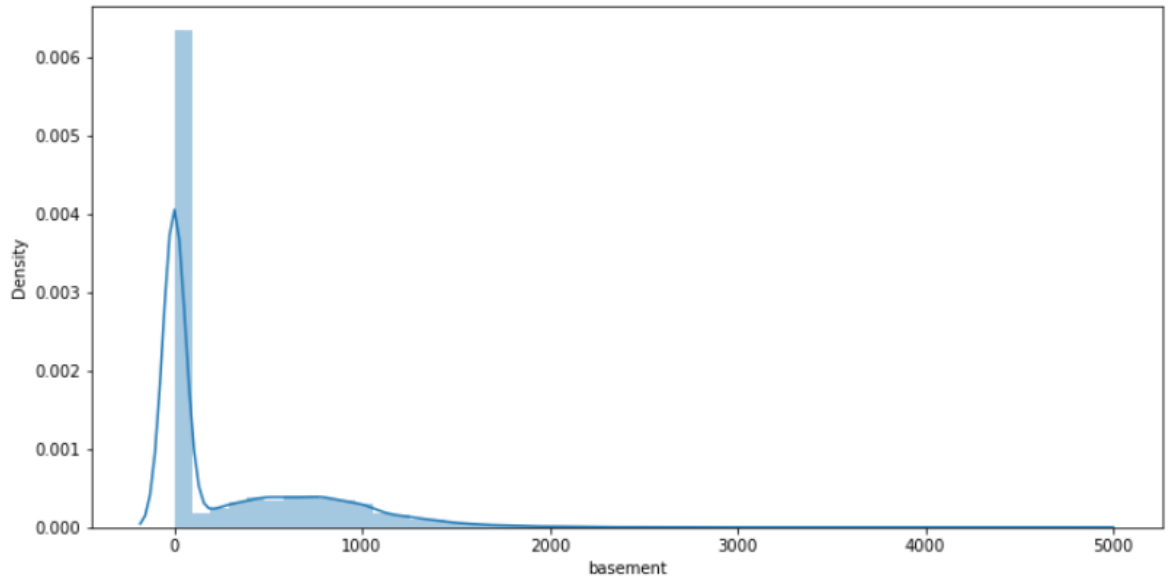
- Skewness and distribution for basement_measure is

```
print("Skewness is :", hdf.basement.skew())
plt.figure(figsize=(12,6))
sns.distplot(hdf.basement);
```

```
Skewness is : 1.5779650555996247
```

If skewness is between -0.5 and 0.5, the distribution is approximately symmetric

As the skewness is greater than 1, the distribution is extremely skewed



2 distribution can be seen. That means there are some properties without basement and some are with basement.

- Value counts for basement measure:

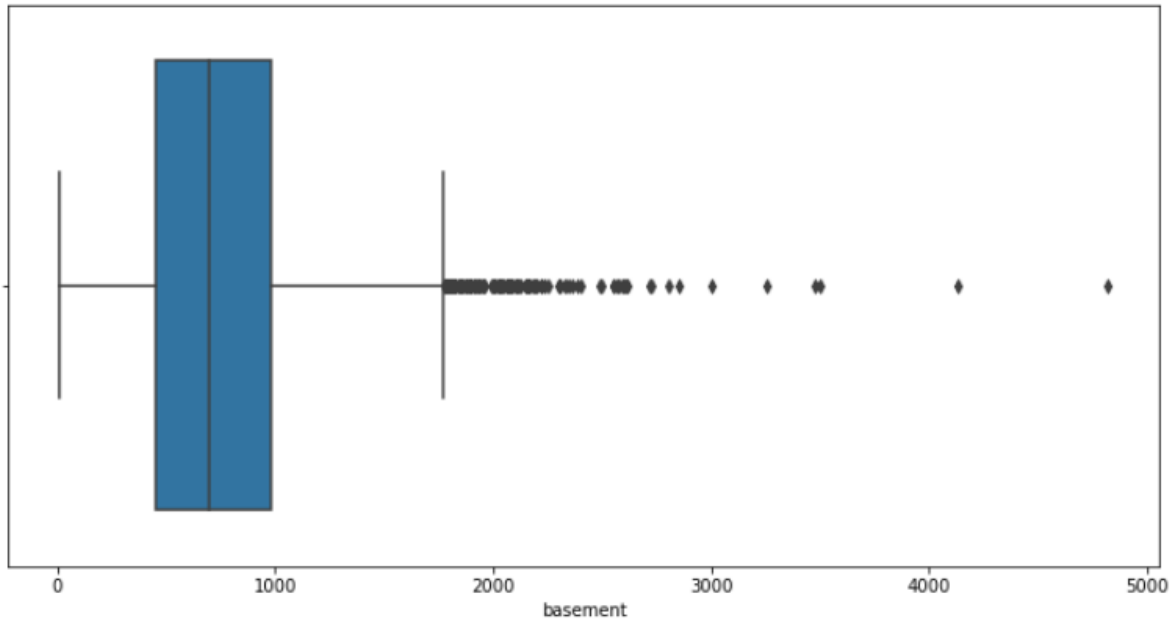
```
hdf.basement.value_counts()
```

```
0       13126
600       221
700       218
500       214
800       206
         ...
792         1
2590        1
935         1
2390        1
248         1
Name: basement, Length: 306, dtype: int64
```

More than 50% of the property doesn't have basement

Boxplot for basement_measure

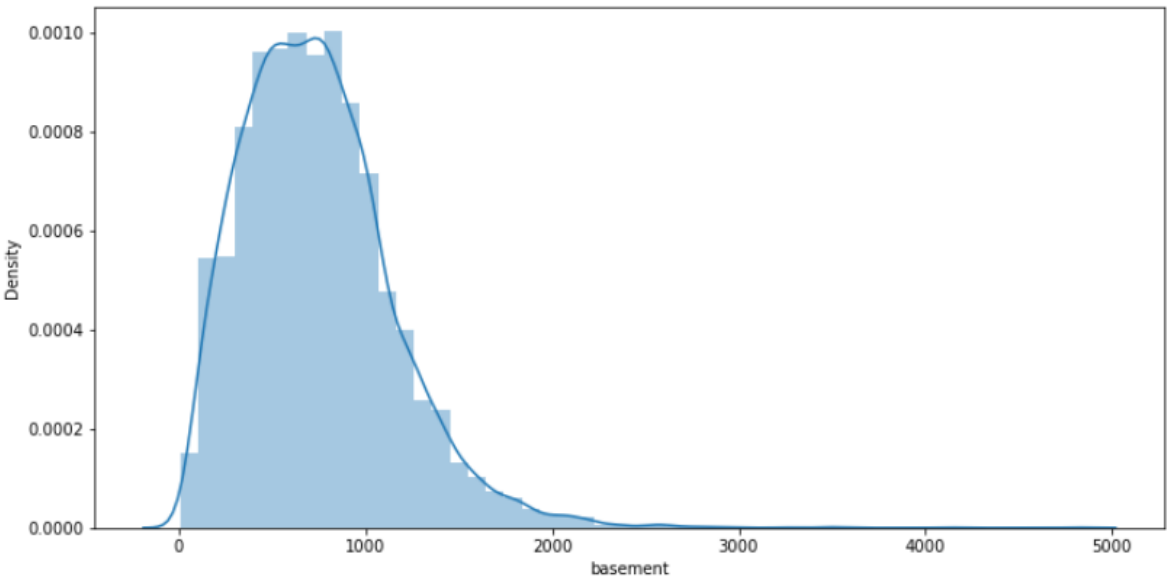
```
#let's plot boxplot for properties which have basements only
hdf_base=hdf[hdf['basement']>0]
plt.figure(figsize=(12,6));
sns.boxplot(hdf_base['basement']);
```



Boxplot for property which has basement only. We can see that there are many outliers that needs to addressed.

Distribution of houses with basement only

```
#Distribution of houses having basement
plt.figure(figsize=(12,6))
sns.distplot(hdf_base.basement);
```

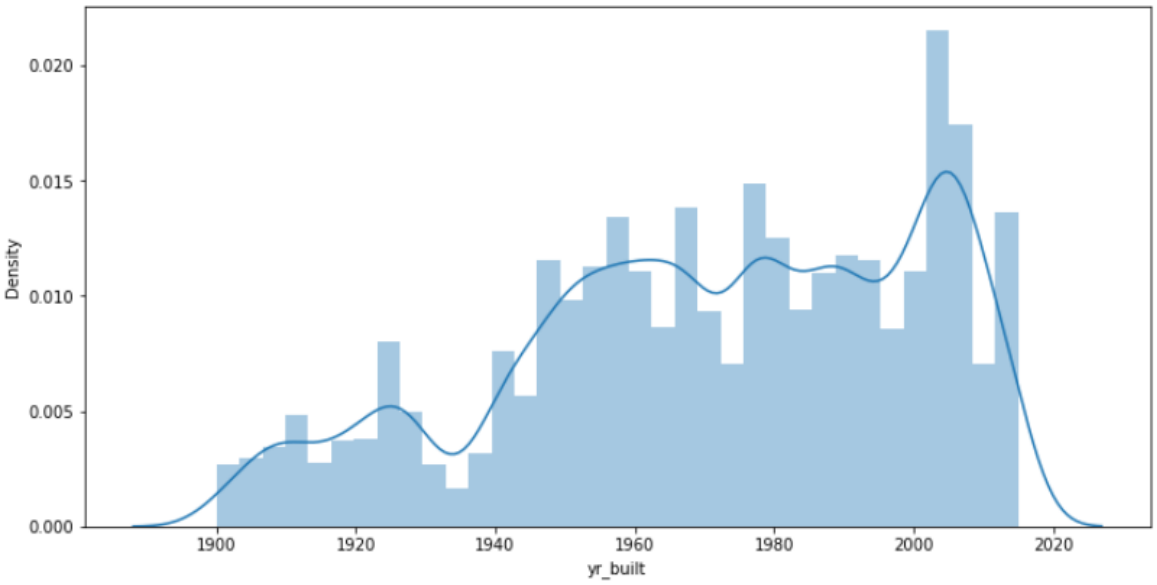


Distribution having basement is right-skewed

2.3.1.14: Yr_built (Built Year)

- The below graph is the distribution of yr_built

```
plt.figure(figsize=(12,6))
sns.distplot(hdf.yr_built);
```



Value counts for yr_built is

```
hdf.yr_built.value_counts(ascending=False)
```

```
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
```

variation of yr_built can be observed from 1900 to 2014, With more properties built in the year 2014

2.3.1.15: Yr_renovated (Year when house was renovated)

Summary statistic of yr_renovated

```
hdf.yr_renovated.describe()
```

```
count    21613.000000
mean        84.402258
std        401.679240
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       2015.000000
Name: yr_renovated, dtype: float64
```

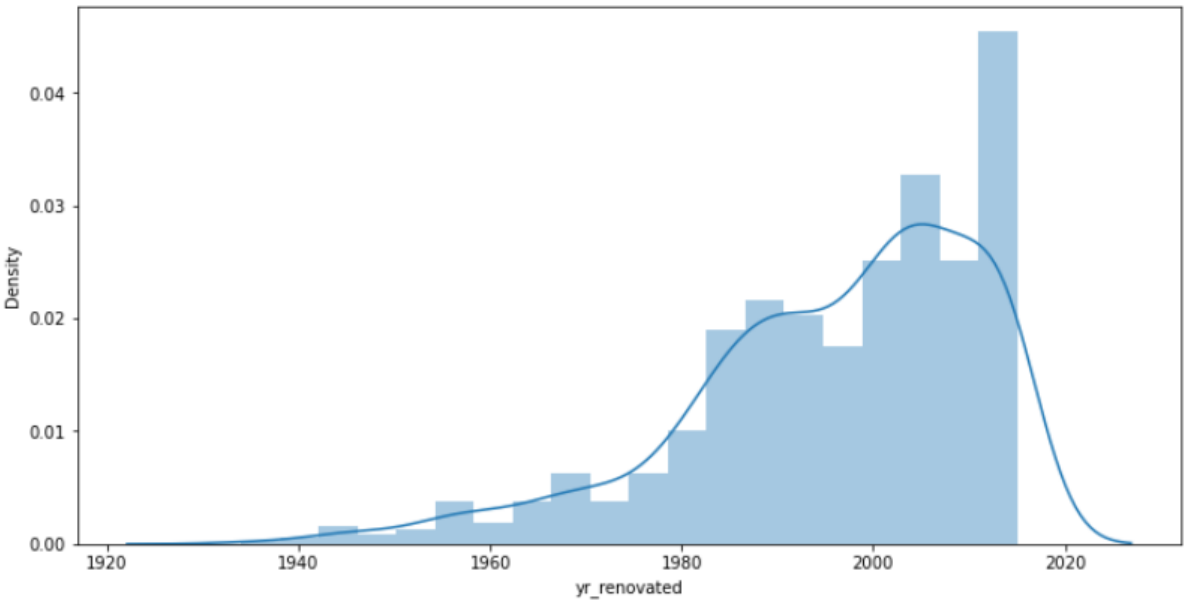
```
hdf[hdf.yr_renovated>0].shape
```

```
(914, 25)
```

914 houses were renovated

Distribution of renovated houses:

```
hdf_renovated = hdf[hdf.yr_renovated>0]
plt.figure(figsize=(12,6))
sns.distplot(hdf_renovated.yr_renovated);
```

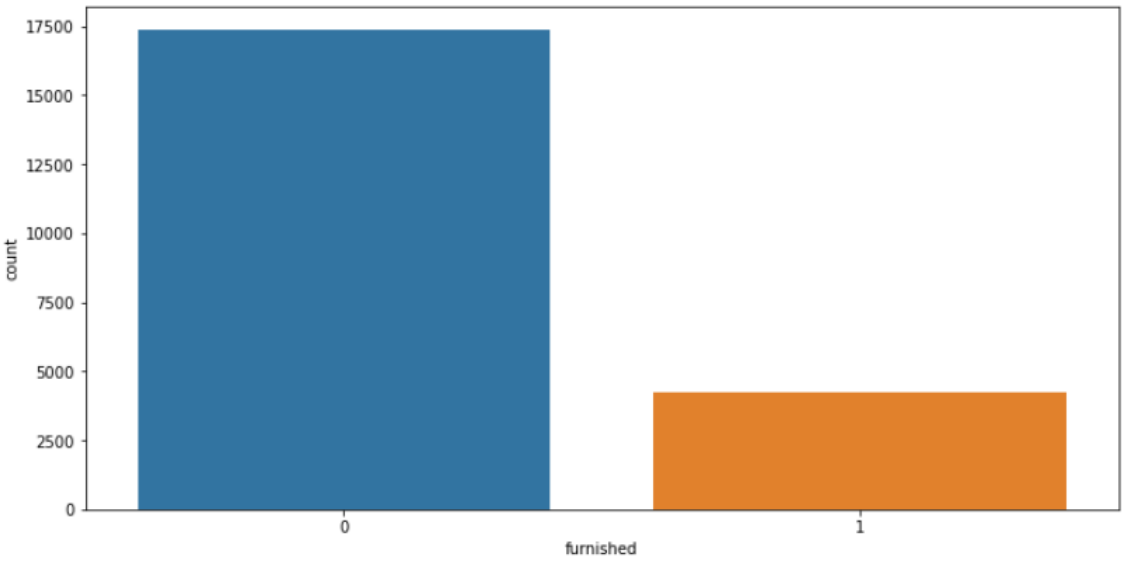


2.3.1.16: Furnished (Based on the quality of room)

Value_counts of house in which 0 as furnished and 1 as not furnished

```
fig=plt.figure(figsize=(12,6))
sns.countplot(hdf.furnished)
hdf.furnished.value_counts()
```

```
0    17362
1     4251
Name: furnished, dtype: int64
```

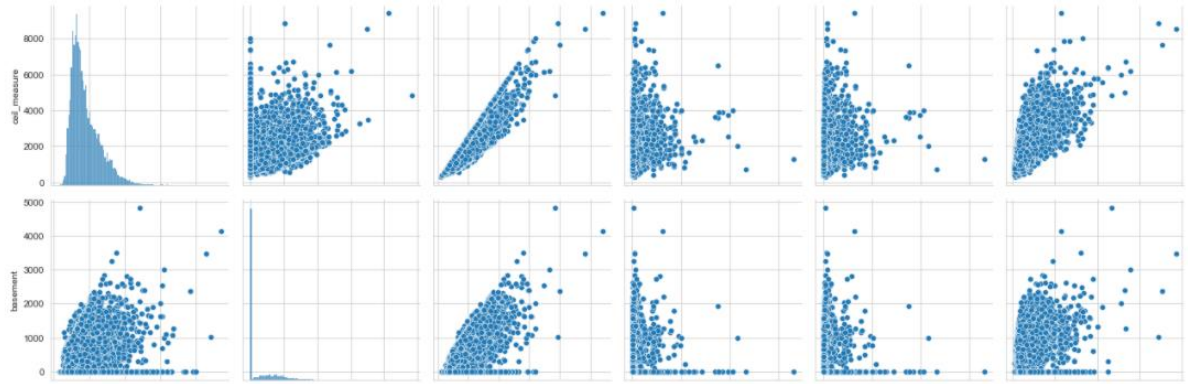


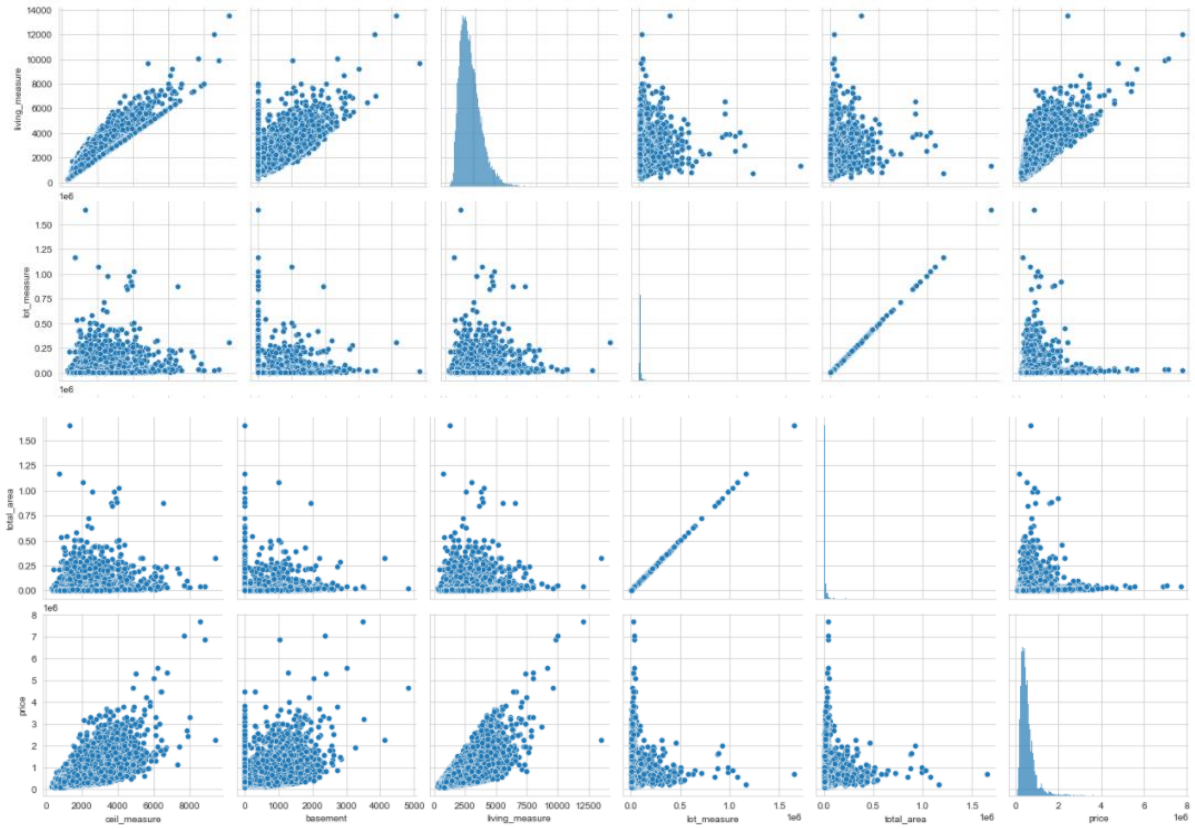
2.3.2 Bivariate analysis

Pairplot:

To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function. This shows the relationship for (n, 2) combination of variable in a Dataframe as a matrix of plots and the diagonal plots are the univariate plots

```
#Pairplot for data visualization
plt.close();
sns.set_style('whitegrid');
sns.pairplot(hdf,vars = ['ceil_measure','basement','living_measure','lot_measure','total_area','price'],size=3);
plt.show()
```





From above pair plot, we observed/deduced below

1.price: price distribution is Right-Skewed as we deduced earlier from our 5-factor analysis

2.room_bed: our target variable (price) and room_bed plot is not linear. Its distribution have lot of gaussians

3.room_bath: It's plot with price has somewhat linear relationship. Distribution has number of gaussians.

4.living_measure: Plot against price has strong linear relationship. It also have linear relationship with room_bath variable. So might remove one of these 2. Distribution is Right-Skewed.

5.lot_measure: No clear relationship with price.

6.ceil: No clear relationship with price. We can see, it's have 6 unique values only. Therefore, we can convert this column into categorical column for values.

7.coast: No clear relationship with price. Clearly it's categorical variable with 2 unique values.

8.sight: No clear relationship with price. This has 5 unique values. Can be converted to Categorical variable.

9.condition: No clear relationship with price. This has 5 unique values. Can be converted to Categorical variable

10.quality: Somewhat linear relationship with price. Has discrete values from 1 - 13. Can be converted to Categorical variable.

11.ceil_measure: Strong linear relationship with price. Also with room_bath and living_measure features. Distribution is Right-Skewed.

12.basement: No clear relationship with price.

13.yr_built: No clear relationship with price.

14.yr_renovated: No clear relationship with price. Have 2 unique values. Can be converted to Categorical Variable which tells whether house is renovated or not.

15.zipcode, lat, long: No clear relationship with price or any other feature.

16.living_measure15: Somewhat linear relationship with target feature. It's same as living_measure. Therefore we can drop this variable.

17.lot_measure15: No clear relationship with price or any other feature.

18.furnished: No clear relationship with price or any other feature. 2 unique values so can be converted to Categorical Variable

19.total_area: No clear relationship with price. But it has Very Strong linear relationship with lot_measure. So one of it can be dropped.

- In brief, below features should be converted to Categorical Variable

  ceil, coast, sight, condition, quality, yr_renovated, furnished

- Below columns can be dropped after checking pearson factor

  zipcode, lat, long, living_measure15, lot_measure15, total_area, sold_year

Correlation between independent variables and target variable

```
# let's check correlation of different features with target variable
hdf.corr()['price']
```

```
price                1.000000
room_bed             0.308338
room_bath            0.525134
living_measure       0.702044
lot_measure          0.089655
ceil                 0.256786
coast                0.266331
sight                0.397346
condition            0.036392
quality              0.667463
ceil_measure         0.605566
basement             0.323837
yr_built             0.053982
yr_renovated         0.126442
living_measure15     0.585374
lot_measure15        0.082456
furnished            0.565991
total_area           0.104796
sold_year            0.003554
house_age           -0.053921
Name: price, dtype: float64
```

We have linear relationships in below features as we got to know from above matrix

1. price: room_bath, living_measure, quality, living_measure15, furnished

2.living_measure: price, room_bath. So we can consider dropping 'room_bath' variable

3.quality: price, room_bath, living_measure.

4.ceil_measure: price, room_bath, living_measure, quality.

5.living_measure15: price, living_measure, quality. So we can consider dropping living_measure15 as well. As it's giving same info as living_measure.

6.lot_measure15: lot_measure. Therefore, we can consider dropping lot_measure15, as it's giving same info.

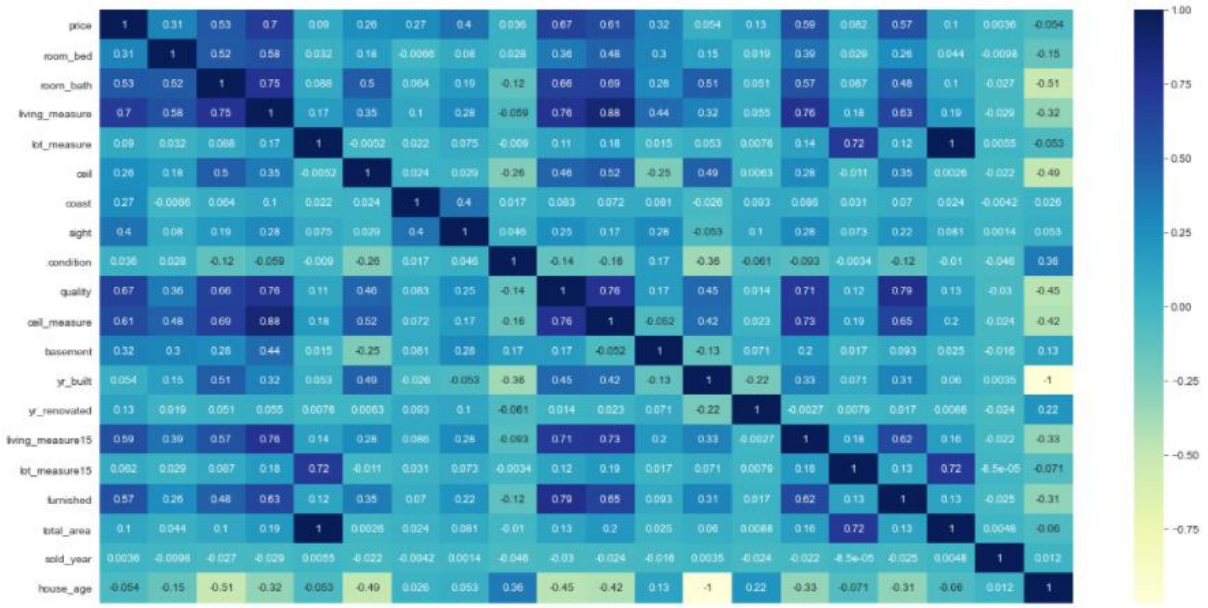
7.furnished: quality

8.total_area: lot_measure, lot_measure15. Therefore, we can consider dropping total_area feature as well. As it's giving same info as lot_measure.

Heatmap

A heatmap is a graphical representation where individual values of a matrix are represented as colors. A heatmap is very useful in visualizing the concentration of values between two dimensions of a matrix. This helps in finding patterns and gives a perspective of depth.

```
# Plotting heatmap
plt.subplots(figsize =(20, 10))
sns.heatmap(hdf_corr,cmap="YlGnBu",annot=True);
```



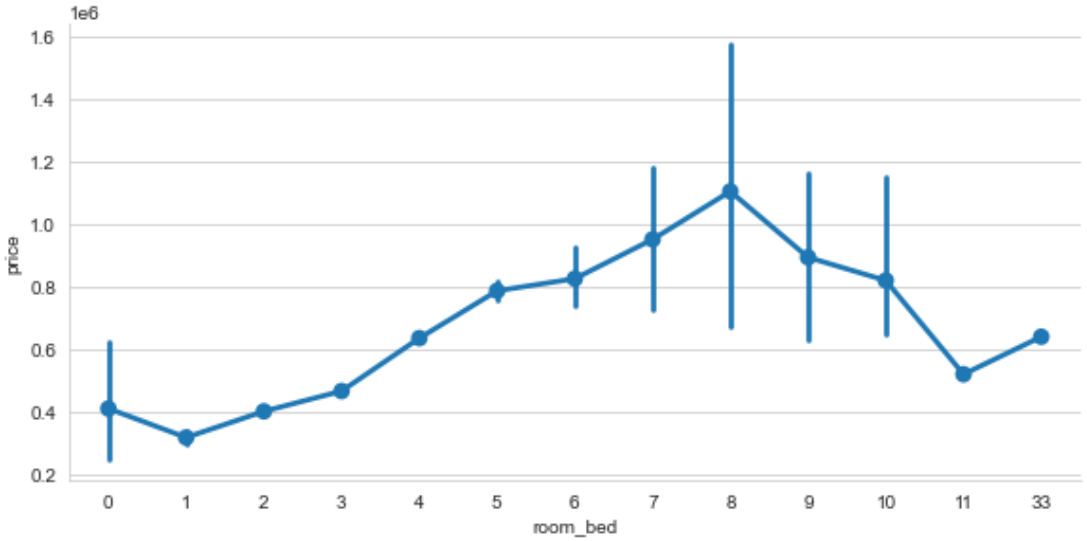
Room_bed v/s price

```
#Room_bed - outliers can be seen easily. Mean and median of price increases with number bedrooms/house uptill a point
#and then drops
sns.factorplot(x='room_bed',y='price',data=hdf, size=4, aspect=2);

#groupby
hdf.groupby('room_bed')['price'].agg(['mean','median','size'])
```

room_bed	mean	median	size
0	4.102231e+05	288000.0	13
1	3.176580e+05	299000.0	199
2	4.013877e+05	374000.0	2760

3	4.662766e+05	413000.0	9824
4	6.355647e+05	549997.5	6882
5	7.868741e+05	620000.0	1601
6	8.258535e+05	650000.0	272
7	9.514478e+05	728580.0	38
8	1.105077e+06	700000.0	13
9	8.939998e+05	817000.0	6
10	8.200000e+05	660000.0	3
11	5.200000e+05	520000.0	1
33	6.400000e+05	640000.0	1



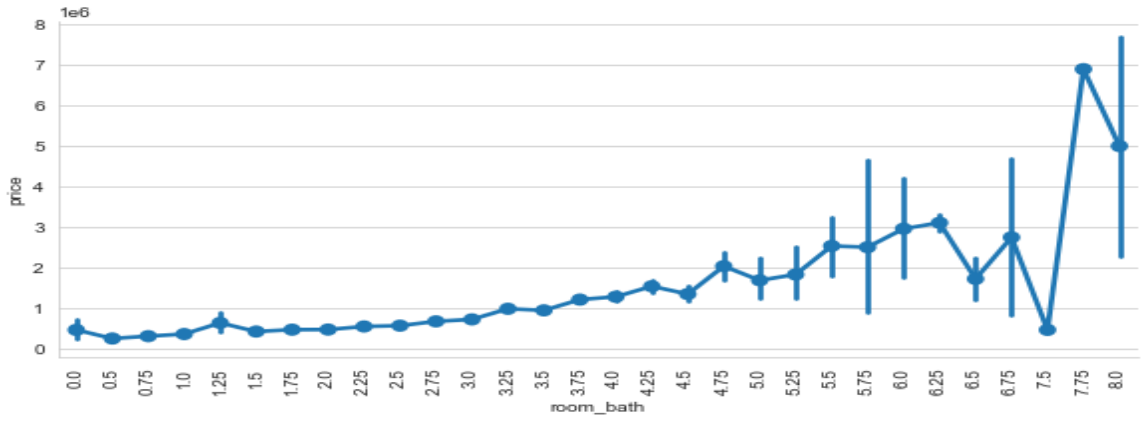
With increasing in room bed 'Price' is increasing. After 8 room_bed the graph is following decreasing trend

Room_bath v/s price

```
#room_bath - outliers can be seen easily. Overall mean and median price increares with increasing room_bath
sns.factorplot(x='room_bath',y='price',data=hdf,size=4, aspect=2);
plt.xticks(rotation=90)
#groupby
hdf.groupby('room_bath')['price'].agg(['mean','median','size'])
```

	mean	median	size
room_bath			
0.00	4.490950e+05	317500	10
0.50	2.373750e+05	264000	4
0.75	2.945209e+05	273500	72
1.00	3.470412e+05	320000	3852

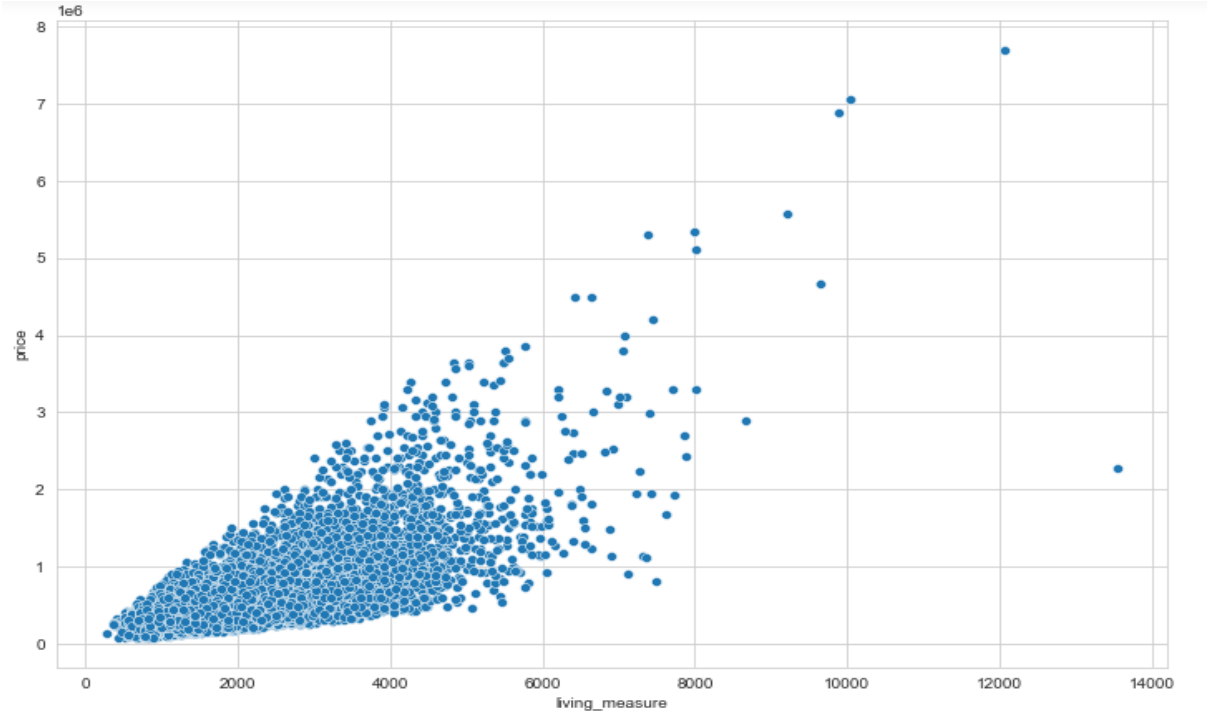
1.25	6.217722e+05	516500	9
1.50	4.093457e+05	370000	1446
1.75	4.549158e+05	422900	3048
2.00	4.579050e+05	423250	1930
2.25	5.337688e+05	472500	2047
2.50	5.536618e+05	499950	5380
2.75	6.603505e+05	605000	1185
3.00	7.086619e+05	600000	753
3.25	9.707532e+05	835000	589
3.50	9.324017e+05	820000	731
3.75	1.198179e+06	1070000	155
4.00	1.268405e+06	1055000	136
4.25	1.526653e+06	1380000	79
4.50	1.334211e+06	1060000	100
4.75	2.022300e+06	2300000	23
5.00	1.674167e+06	1430000	21
5.25	1.817962e+06	1420000	13
5.50	2.522500e+06	2340000	10
5.75	2.492500e+06	1930000	4
6.00	2.948333e+06	2895000	6
6.25	3.095000e+06	3095000	2
6.50	1.710000e+06	1710000	2
6.75	2.735000e+06	2735000	2
7.50	4.500000e+05	450000	1
7.75	6.890000e+06	6890000	1
8.00	4.990000e+06	4990000	2



Price is following upward trend with respect to room_bath

Living_measure v/s Price (Square foot of home)

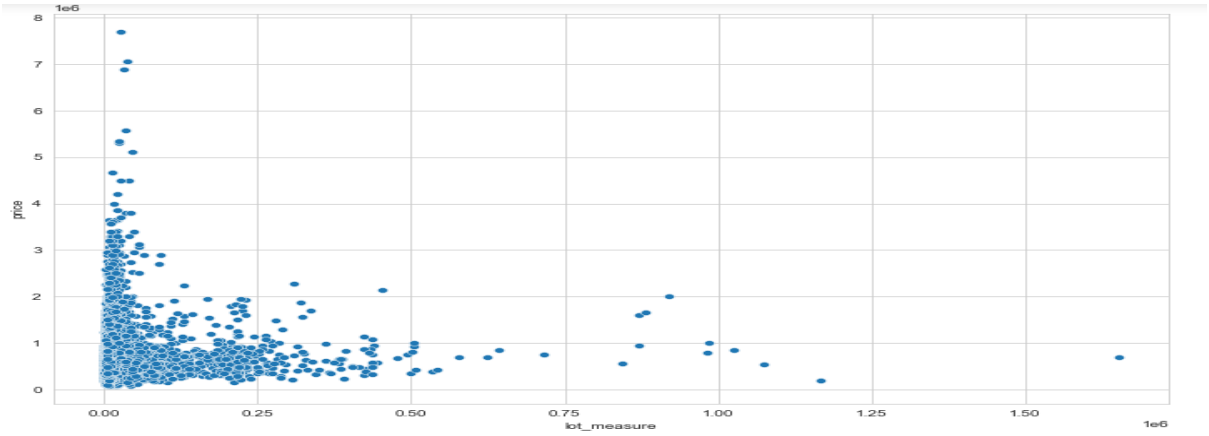
```
#Living_measure - price increases with increase in living measure
plt.figure(figsize=(12, 8))
print(sns.scatterplot(hdf['living_measure'],hdf['price']));
```



There is a clear linear relationship can be observed with respect to price and living_measure. Also few outliers can be visible

Lot_measure v/s Price(Square foot of lot)

```
plt.figure(figsize=(12, 8))
print(sns.scatterplot(hdf['lot_measure'],hdf['price']));
```



lot_measure-there seems to be no relation between lot_measure and price

lot_measure - data value range is very large so breaking it get better view.

No clear trend is observed between price and lot_measure. Also the correlation between price and lot_measure is 0.082456. We could drop this feature

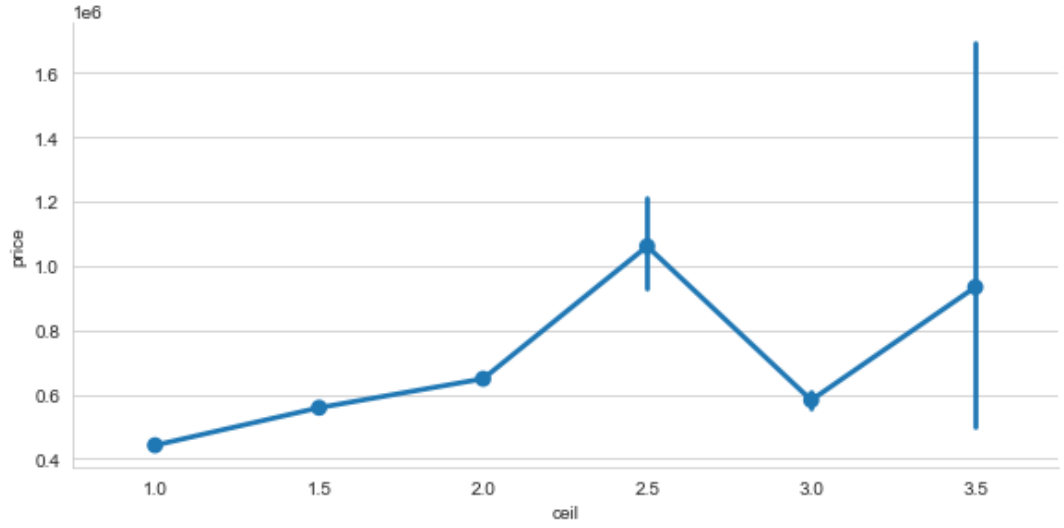
Ceil v/s Price (Number of floors)

Mean, median and size for price with respect to ceil

```
print(sns.factorplot(x='ceil',y='price',data=hdf, size = 4, aspect = 2))
#groupby
hdf.groupby('ceil')['price'].agg(['mean','median','size'])
```

```
<seaborn.axisgrid.FacetGrid object at 0x000001D10A378BB0>
```

ceil	mean	median	size
1.0	4.422196e+05	390000	10680
1.5	5.590449e+05	524475	1910
2.0	6.490515e+05	542950	8241
2.5	1.061021e+06	799200	161
3.0	5.826201e+05	490000	613
3.5	9.339375e+05	534500	8



ceil - median price increases initially and then falls. There is some slight upward trend in price with the ceil

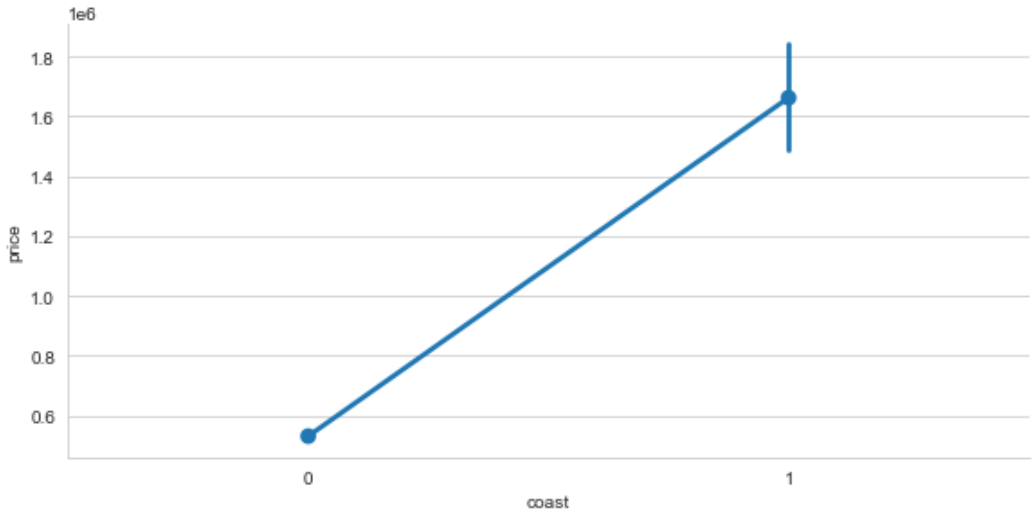
Coast v/s Price

Mean and median for price with respect to coast

```
print(sns.factorplot(x='coast',y='price',data=hdf, size = 4, aspect = 2));
#groupby
hdf.groupby('coast')['living_measure','price'].agg(['median','mean'])
```

```
<seaborn.axisgrid.FacetGrid object at 0x000001D1116D5F70>
```

	living_measure		price	
	median	mean	median	mean
coast				
0	1910	2071.587972	450000	5.316534e+05
1	2850	3173.687117	1400000	1.662524e+06



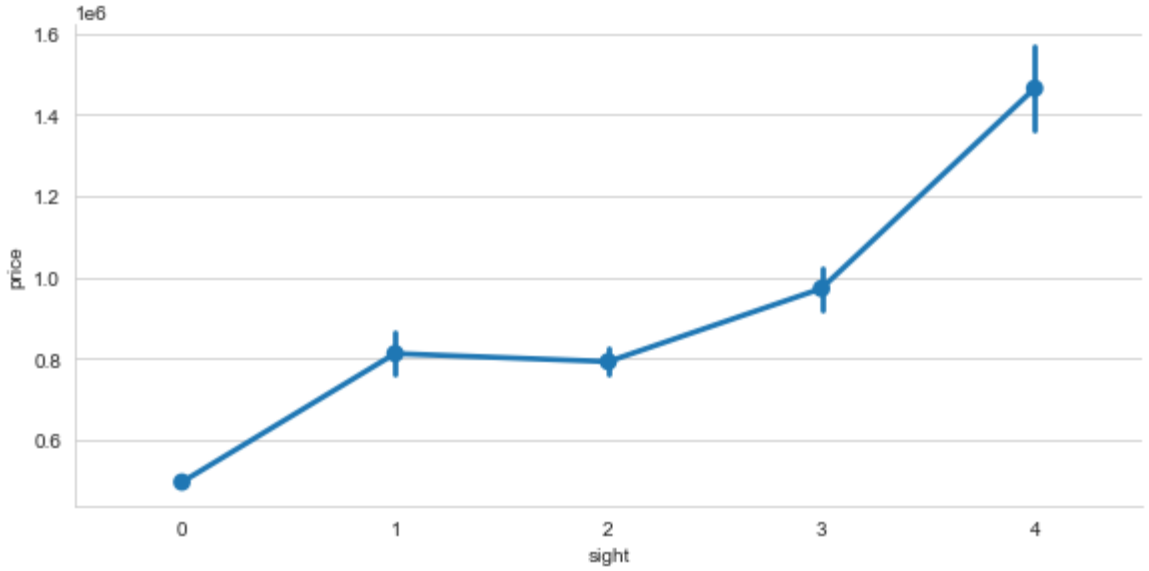
coast - mean and median of waterfront view is high however such houses are very small in compare to non-waterfront view. Also, living_measure mean and median is greater for waterfront house. The house properties with water_front tend to have higher price compared to that of non-water_front properties.

Sight v/s Price

Mean and median for price with respect to sight

```
print(sns.factorplot(x='sight',y='price',data=hdf, size = 4, aspect = 2));
#groupby
hdf.groupby('sight')['price','living_measure'].agg(['mean','median','size'])
```

	price			living_measure		
	mean	median	size	mean	median	size
sight						
0	4.966235e+05	432500	19489	1997.761660	1850	19489
1	8.125186e+05	690944	332	2568.960843	2420	332
2	7.927462e+05	675000	963	2655.257529	2470	963
3	9.724684e+05	802500	510	3018.564706	2840	510
4	1.464363e+06	1190000	319	3351.473354	3050	319



sight - have outliers. The house sighted more have high price (mean and median) and have large living area as well. Properties with higher price have more no of sights compared to that of houses with lower price.

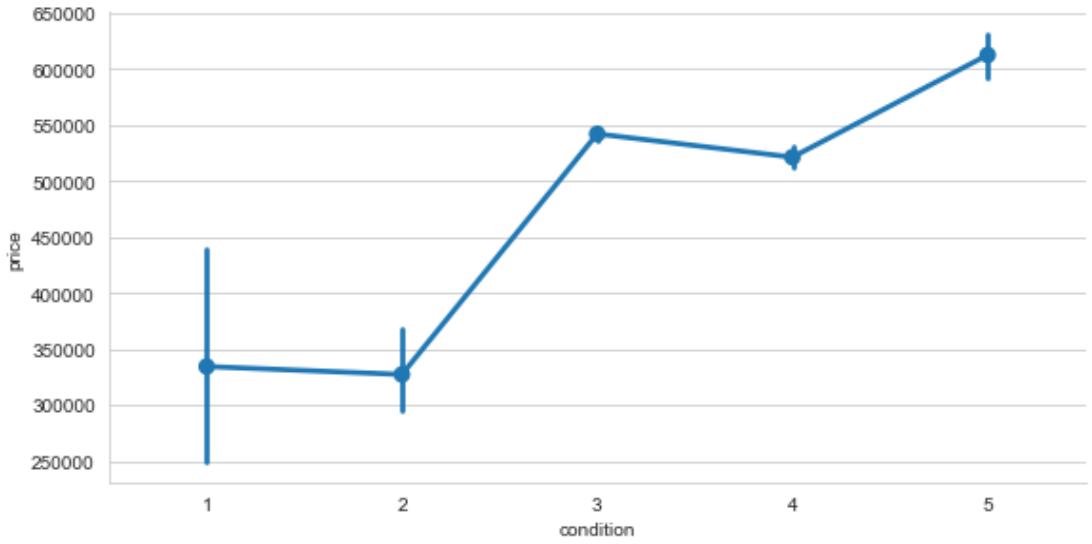
condition v/s Price

Mean and median for price with respect to condition

```
print(sns.factorplot(x='condition',y='price',data=hdf, size = 4, aspect = 2))
#groupby
hdf.groupby('condition')['price','living_measure'].agg(['mean','median','size'])
```

	price			living_measure		
	mean	median	size	mean	median	size
condition						
1	334431.666667	262500	30	1216.000000	1000	30

2	327316.215116	279000	172	1410.058140	1320	172
3	542097.086024	450000	14031	2149.042050	1970	14031
4	521300.705230	440000	5679	1950.991724	1820	5679
5	612577.742504	526000	1701	2022.911229	1880	1701



condition - as the condition rating increases its price and living measure mean and median also increases. The price of the house increases with condition rating of the house

Quality v/s Price

Mean and median for price with respect to quality

```
print(sns.factorplot(x='quality',y='price',data=hdf, size = 4, aspect = 2))
#groupby
hdf.groupby('quality')['price','living_measure'].agg(['mean','median','size'])
```

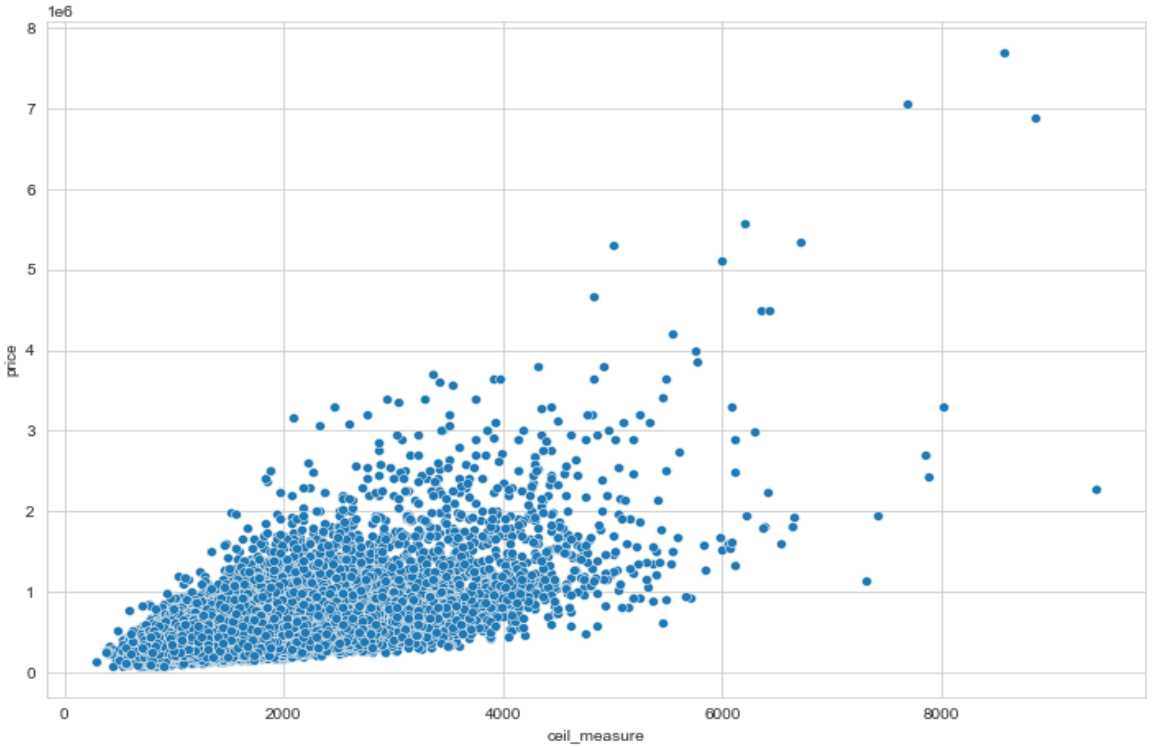
	price			living_measure		
	mean	median	size	mean	median	size
quality						
1	1.420000e+05	142000.0	1	290.000000	290	1
3	2.056667e+05	262000.0	3	596.666667	600	3
4	2.143810e+05	205000.0	29	660.482759	660	29
5	2.485240e+05	228700.0	242	983.326446	905	242
6	3.019166e+05	275276.5	2038	1191.561335	1120	2038

7	4.025933e+05	375000.0	8981	1689.400401	1630	8981
8	5.428955e+05	510000.0	6068	2184.748517	2150	6068
9	7.737382e+05	720000.0	2615	2868.139962	2820	2615
10	1.072347e+06	914327.0	1134	3520.299824	3450	1134
11	1.497792e+06	1280000.0	399	4395.448622	4260	399
12	2.192500e+06	1820000.0	90	5471.588889	4965	90
13	3.710769e+06	2980000.0	13	7483.076923	7100	13

quality - with grade increase price and living_measure increase (mean and median). There is clear increase in price of the house with higher rating on quality

Ceil_measure v/s Price

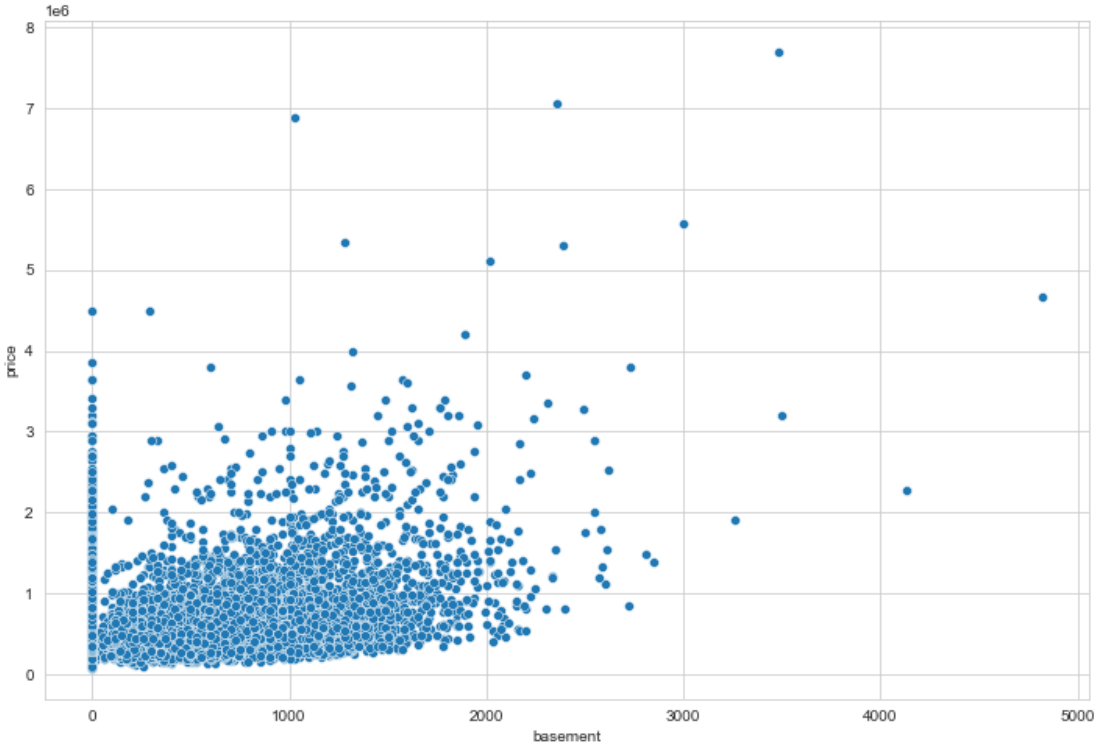
```
plt.figure(figsize=(12, 8))
print(sns.scatterplot(hdf['ceil_measure'],hdf['price']));
```



ceil_measure - price increases with increase in ceil measure. There is upward trend in price with ceil_measure.

Basement v/s price

```
plt.figure(figsize=(12, 8))
print(sns.scatterplot(hdf['basement'],hdf['price']));
```



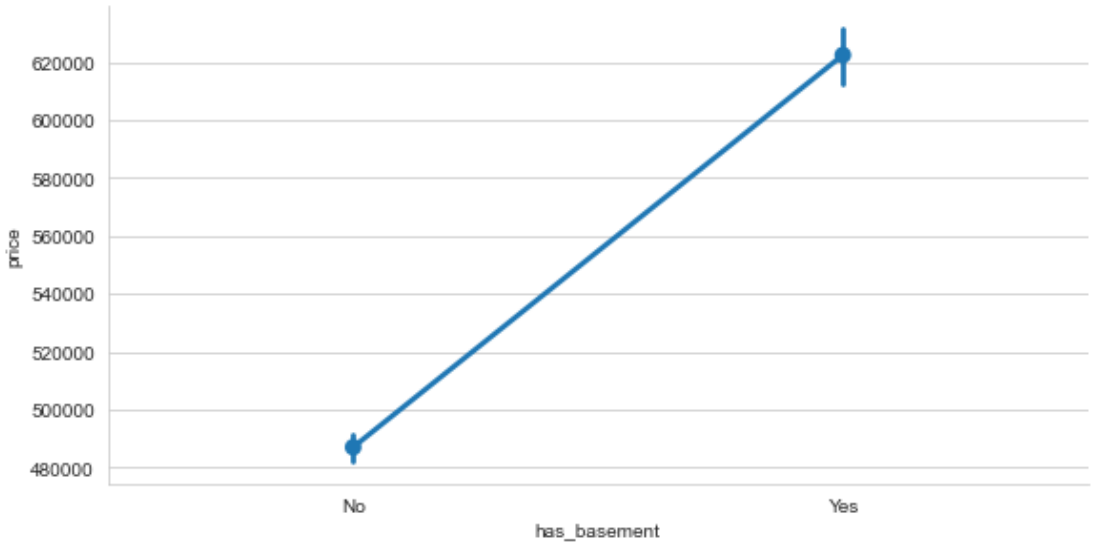
We will create the categorical variable for basement 'has_basement' for houses with basement and no basement. This categorical variable will be used for further analysis

```
#Binning Basement to analyse data
def create_basement_group(series):
    if series == 0:
        return "No"
    elif series > 0:
        return "Yes"

hdf['has_basement'] = hdf['basement'].apply(create_basement_group)
```

```
#basement - after binning we data shows with basement houses are costlier and have higher
#living measure (mean & median)
print(sns.factorplot(x='has_basement',y='price',data=hdf, size = 4, aspect = 2));
hdf.groupby('has_basement')['price','living_measure'].agg(['mean','median','size'])
```

	price			living_measure		
	mean	median	size	mean	median	size
has_basement						
No	486945.394789	411500	13126	1928.879628	1740	13126
Yes	622518.174384	515000	8487	2313.467539	2100	8487



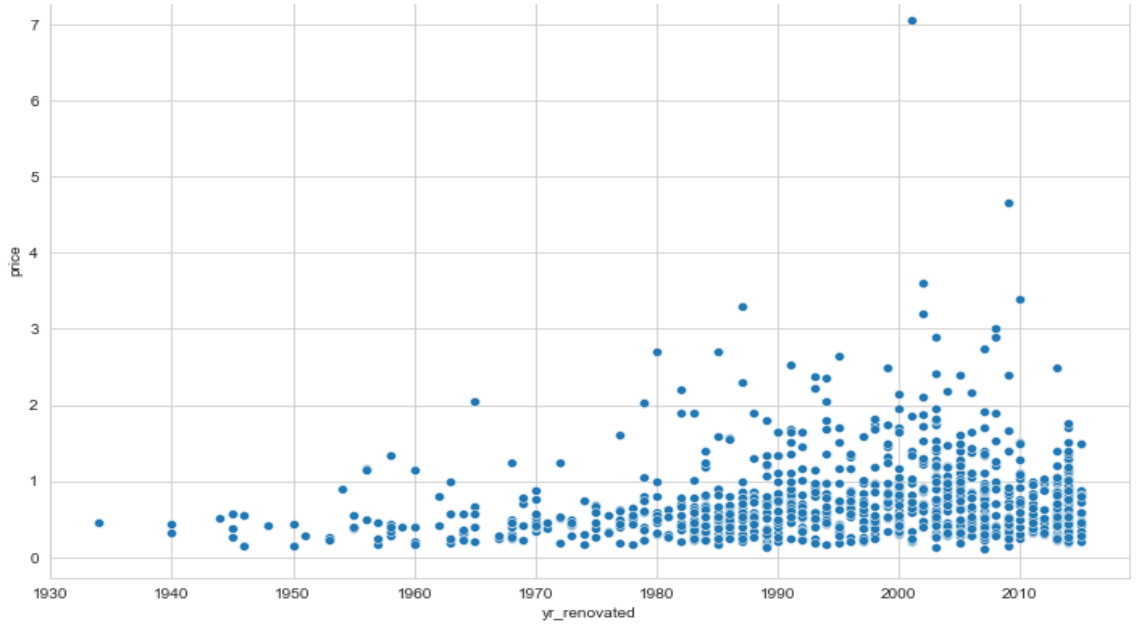
The houses with basement has better price compared to that of houses without basement.

yr_renovated v/s Price

mean and median for price with respect to year renovated

```
#yr_renovated -
plt.figure(figsize=(12,8))
x=hdf[hdf['yr_renovated']>0]
print(sns.scatterplot(x['yr_renovated'],x['price']))
#groupby
x.groupby('yr_renovated')['price'].agg(['mean','median','size'])
```

yr_renovated	mean	median	size
1934	459950.000000	459950.0	1
1940	378400.000000	378400.0	2
1944	521000.000000	521000.0	1
1945	398666.666667	375000.0	3
1946	351137.500000	351137.5	2
...	...	...	...
2011	607496.153846	577000.0	13
2012	625181.818182	515000.0	11
2013	664960.810811	560000.0	37
2014	655030.098901	575000.0	91
2015	659156.250000	651000.0	16

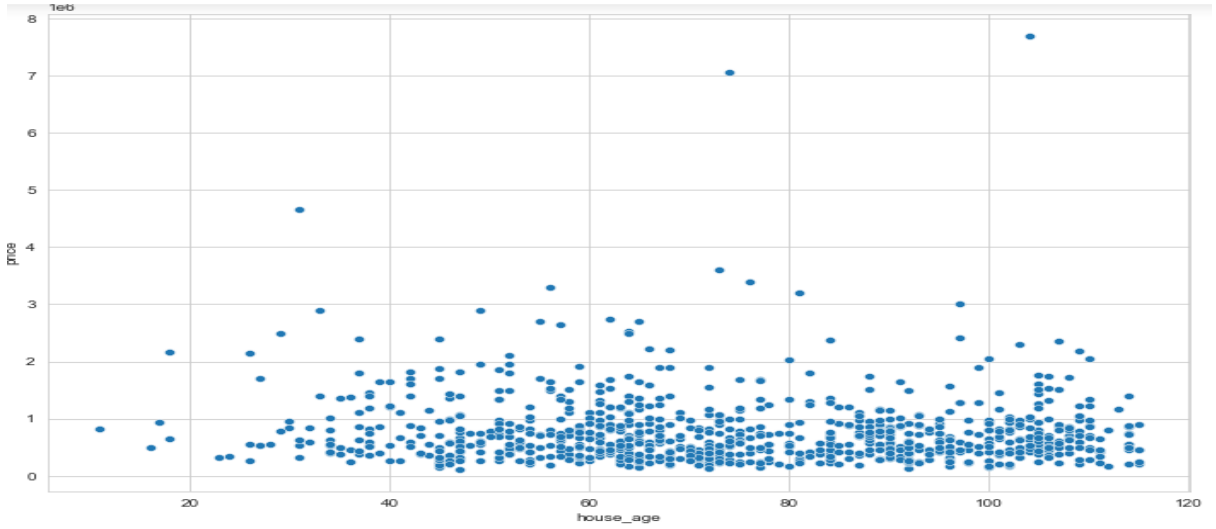


So most houses are renovated after 1980's. We will create new categorical variable 'has_renovated' to categorize the property as renovated and non-renovated. For further analysis we will use this categorical variable.

House age vs price

Analyzing house age with respect to price

```
# We have created new feature 'Houseage'. Will analyze Houseage with respect to Price
#yr_renovated -
plt.figure(figsize=(12,8))
# x=hdf[hdf['yr_renovated']>0]
print(sns.scatterplot(x['house_age'],x['price']))
#groupby
x.groupby('house_age')['price'].agg(['mean','median','size'])
```



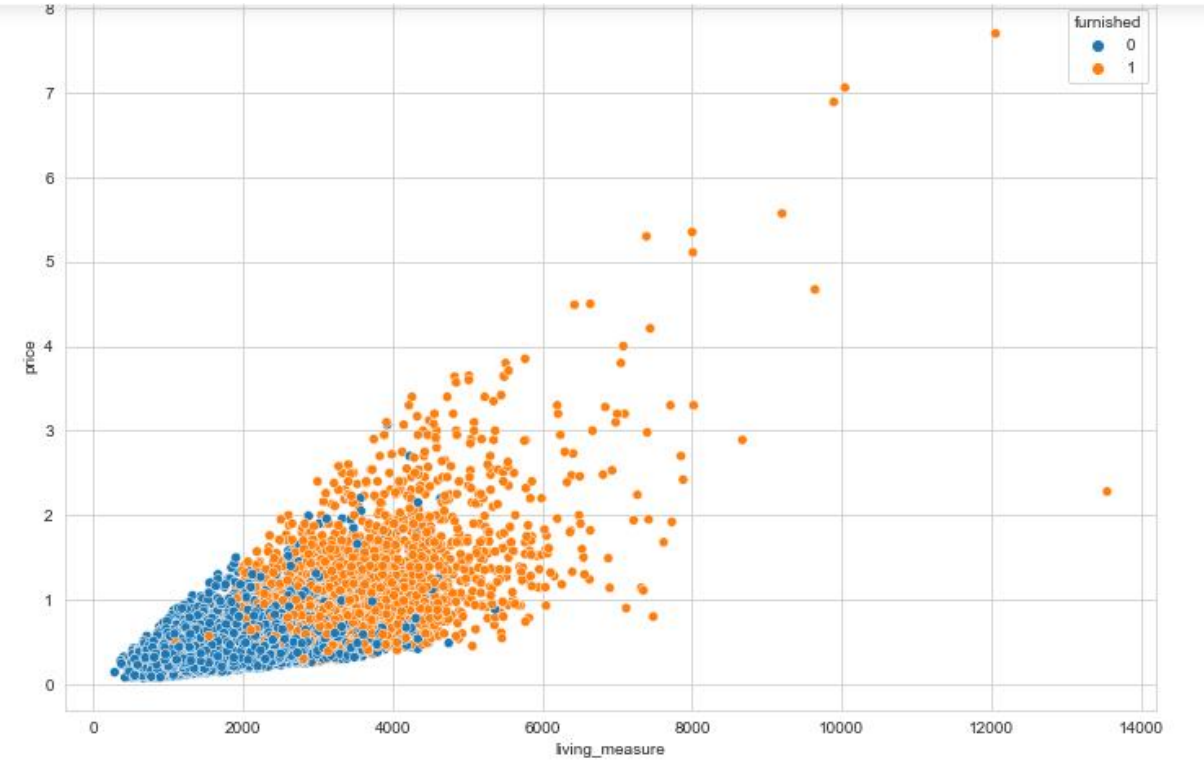
Trend is not clear with respect to age of the house and Price. Correlation between them is also very low.

furnished v/s Price

Mean and median of price with respect to furnished

```
plt.figure(figsize=(12, 8))
print(sns.scatterplot(hdf['living_measure'],hdf['price'],hue=hdf['furnished']))
#groupby
hdf.groupby('furnished')['price','living_measure'].agg(['mean','median','size'])
```

	price			living_measure		
	mean	median	size	mean	median	size
furnished						
0	437300.158968	401000	17362	1792.256652	1720	17362
1	960374.414961	810000	4251	3254.696072	3110	4251



furnished - Furnished has higher price value and has greater living_measure. Furnished houses have higher price than that of the Non-furnished houses

3. Data Pre-processing

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

3.1 Outliers handling

We have seen outliers in many of the attributes like columns room_bath, living_measure, lot_measure, ceil_measure and Basement

```
# Let us write a function for outliers treatment
def outlier_treatment(datacolumn):
    sorted(datacolumn)
    Q1,Q3 = np.percentile(datacolumn , [25,75])
    IQR = Q3-Q1
    lower_range = Q1-(1.5 * IQR)
    upper_range = Q3+(1.5 * IQR)
    return lower_range,upper_range
```

```
hdf_with_outlier = hdf.copy()      #--->  dataframe with outliers
```

```
hdf_log_trans  = hdf.copy()        #----> dataframe after Log Transformation
```

3.2 Treating outliers for column - ceil_measure

```
lowerbound,upperbound = outlier_treatment(hdf.ceil_measure)
print(lowerbound,upperbound)
```

Lower bound and upper bound values ranges from -34 to 3740

Let us have 3 dataframes

1. hdf: without outliers (Its outliers are found and dropped)

2. hdf_log_trans: The columns where outliers are found that particular column is log transformed

3. hdf_with_outlier: The dataframe with outliers.

All the three dataframe have been considered for model building seperately

Dropping the outliers records from dataset

```
#dropping the outliers record from the dataset
hdf.drop(hdf[(hdf.ceil_measure > upperbound) | (hdf.ceil_measure < lowerbound) ].index, inplace=True)
```

All the outliers values in both upper and lower limit have been removed

Let us analyse by log transform

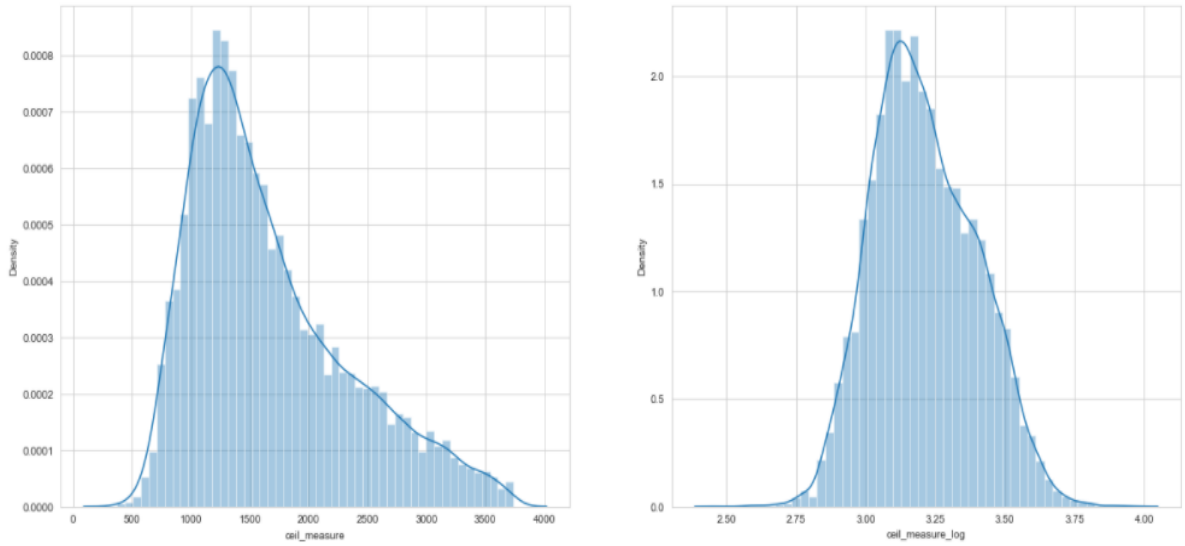
```
# Let us anlyse also by considering log transormation
hdf_log_trans['ceil_measure_log'] = np.log10(hdf_log_trans.ceil_measure)
```

```
hdf_log_trans.drop(['ceil_measure'],axis=1,inplace=True)
hdf_log_trans.head(2)
```

Only 'ceil_measure_log' is considered and the original column have been dropped

```
plt.figure(figsize=(20, 8))
plt.subplot(1,2,1)
sns.distplot(hdf.ceil_measure);

plt.subplot(1,2,2)
sns.distplot(hdf_log_trans.ceil_measure_log);
```



After log transformation distribution found almost normal

```
#ceil_measure
print("Skewness after droping outlier is :", hdf.ceil_measure.skew())
print("Skewness after log transformation is :", hdf_log_trans.ceil_measure_log.skew())
```

```
Skewness after droping outlier is : 0.8927058787474383
Skewness after log transformation is : 0.253384126301554
```

Treating outliers for column - living_measure

```
lowerbound_lim,upperbound_lim = outlier_treatment(hdf.living_measure)
print(lowerbound_lim,upperbound_lim)
```

```
-160.0 4000.0
```

```
hdf[(hdf.living_measure < lowerbound_lim) | (hdf.living_measure > upperbound_lim)]
```

```
hdf.drop(hdf[(hdf.living_measure>upperbound_lim) | (hdf.living_measure<lowerbound_lim)].index,inplace=True)
```

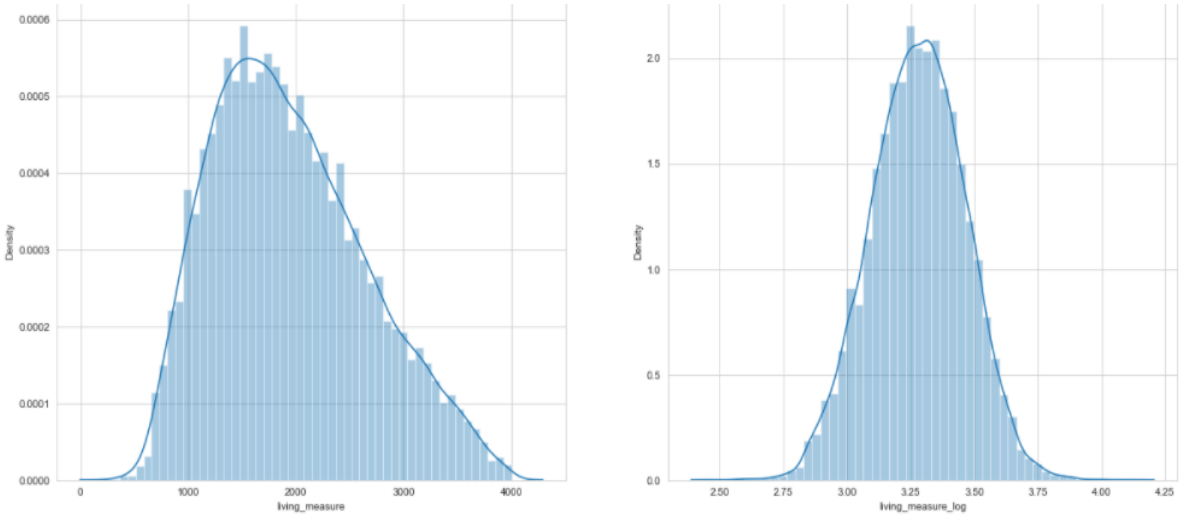
Now let us consider the log transformation:

```
hdf_log_trans['living_measure_log'] = np.log10(hdf_log_trans.living_measure)
hdf_log_trans.drop(['living_measure'],axis=1,inplace=True)
hdf_log_trans.head(2)
```

	month_year	price	room_bed	room_bath	lot_measure	ceil	coast	sight	condition	quality
0	11-2014	808100	4	3.25	13457	1.0	0	0	5	9
1	12-2014	277500	4	2.50	7500	1.0	0	0	3	8

yr_renovated	living_measure15	lot_measure15	furnished	total_area	sold_year	house_age	has_basement	ceil_measure_log	living_measure_log
0	2120	7553	1	16477	2014	58	No	3.480007	3.480007
0	2260	8800	0	10050	2014	38	Yes	3.243038	3.406540

Plot a distribution plot for living measure and living measure log



Skewness after dropping outlier is

```
#Living measure skewness
print("Skewness after droping outlier is :", hdf.living_measure.skew())
print("Skewness after log transformation is :", hdf_log_trans.living_measure_log.skew())
```

```
Skewness after droping outlier is : 0.47196155430940123
Skewness after log transformation is : -0.0354376929956991
```

Treating outliers with column-lot_measure

Lower bound_lom and upper bound_lom outlier treatment

```
lowerbound_lom,upperbound_lom = outlier_treatment(hdf.lot_measure)
print(lowerbound_lom,upperbound_lom)
```

```
-2774.875 17958.125
```

Dataset with lot_measure less than lowerbound_lom and greater than upperbound_lom

```
hdf[(hdf.lot_measure < lowerbound_lom) | (hdf.lot_measure > upperbound_lom)]
```

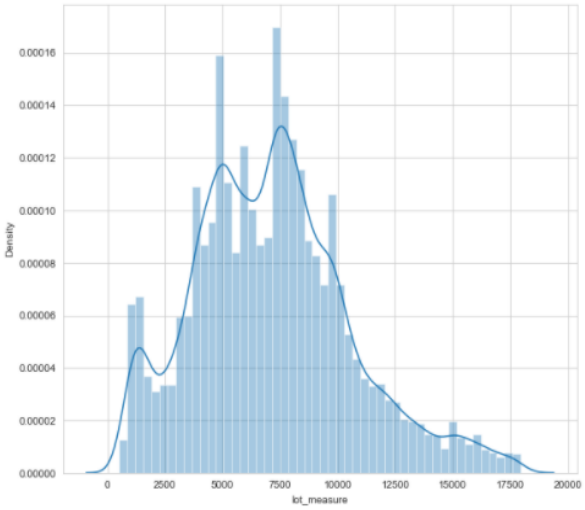
We got 2128 records which are outliers. Drop these records from dataset

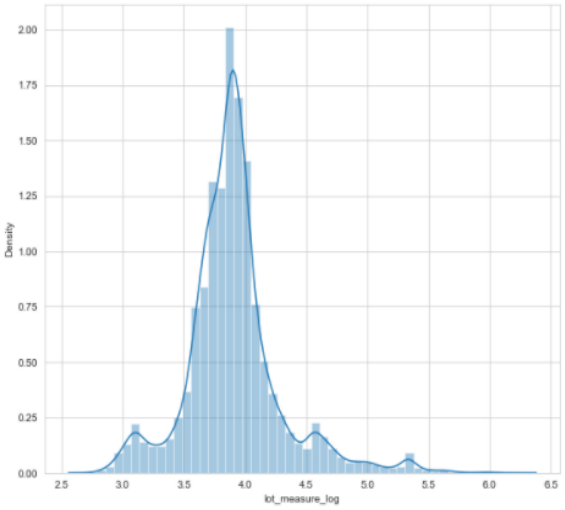
```
#dropping the record from the dataset
hdf.drop(hdf[(hdf.lot_measure > upperbound_lom) | (hdf.lot_measure < lowerbound_lom) ].index, inplace=True)
```

Log transformation

```
# Let us consider log transformation
hdf_log_trans['lot_measure_log'] = np.log10(hdf_log_trans.lot_measure)
hdf_log_trans.drop(['lot_measure'],axis=1,inplace=True)
hdf_log_trans.head(2)
```

Plot for lot_measure and lot_measure_log





Dropping the unwanted columns

From the data analysis it is found that ‘cid’, ‘month_year’, ‘year renovated’, ‘zip code’, ‘latitude’ and ‘longitude’ are irrelevant in predicting the target variable(Price), so need to drop them

```
df = hdf.drop(['cid','month_year','yr_renovated','zipcode','lat','long'],axis=1)
```

Final dataset

	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition	quality
0	808100	4	3.25	3020	13457	1.0	0	0	5	9
1	277500	4	2.50	2550	7500	1.0	0	0	3	8
2	404000	3	2.50	2370	4324	2.0	0	0	3	8
3	300000	2	1.00	820	3844	1.0	0	0	4	6
4	699000	2	1.50	1400	4050	1.0	0	0	4	8
...	...	...	...	...	...	...	...	...	...	...
21608	300000	4	2.50	2303	3826	2.0	0	0	3	8
21609	320000	4	2.50	3490	5000	2.0	0	0	3	8
21610	483453	4	2.75	2790	5527	2.0	0	0	3	8
21611	365000	2	2.00	1440	15000	1.0	0	0	3	7
21612	354950	3	1.00	970	5922	1.5	0	0	3	7

ceil_measure	basement	yr_built	living_measure15	lot_measure15	furnished	total_area	sold_year	house_age	has_basement
3020	0	1956	2120	7553	1	16477	2014	58	No
1750	800	1976	2260	8800	0	10050	2014	38	Yes
2370	0	2006	2370	4348	0	6694	2015	9	No
820	0	1916	1520	3844	0	4664	2014	98	No
1400	0	1954	1900	5940	0	5450	2015	61	No
...	...	...	...	...	...	...	...	...	...
2303	0	2006	2516	4500	0	6129	2014	8	No
3490	0	2003	2910	5025	0	8490	2014	11	No
2790	0	2014	2620	5509	0	8317	2014	0	No
1440	0	1985	1780	15000	0	16440	2014	29	No
970	0	1949	1730	6128	0	6892	2015	66	No

For the convenience lets rename dataframes as

df0 ------> with outliers

df ------> without outliers (Outliers have been dropped)

df1-------> Log transformed (Some of the features only)

One hot encoding:

It is a technique to treat categorical variables. It simply creates additional features based on number unique values in categorical features.

Every unique value in the category will be added as a feature

One hot encoding is the process of creating dummy variables

Creating dummies for categorical variables: 'room_bed', 'room_bath', 'ceil', 'coast', 'sight', 'condition', 'quality', 'furnished', 'has_basement'.

```
# Getting dummies for columns ceil, coast, sight, condition, quality, yr_renovated, furnished
df = pd.get_dummies(df, columns=['room_bed', 'room_bath', 'ceil', 'coast', 'sight', 'condition', 'quality', 'furnished',
                                 'has_basement'],drop_first=True)
```

Final dataframes for modelling

Dataframe shape without outliers is

```
df.shape
```

```
(18287, 20)
```

Dataframe shape with outliers is

```
df0.shape
```

```
(21613, 79)
```

Dataframe shape with log transformation

```
df1.shape
```

```
(21613, 20)
```

4. Data modelling

Machine learning model is the product of training a machine learning algorithm with training data.

While algorithms are simply general approaches to solve an objective, ML models can evaluate future unknown data and make predictions.

One can create many models from same algorithm, as long as different training data are available.

Types of ML algorithms

1. Supervised learning:
2. Unsupervised learning:

Supervised learning: Supervised learning can be further grouped into regression and classification:

- **Regression**: Regression problem is when the output variable is real value, such as number of students in classroom, weight of person

  Ex: Linear regression, KNN regressor, support vector regressor, Decision tree, random forest

- **Classification:** A classification problem is when the output variable is a category, such as ‘red’ or ‘blue’ color, has ‘disease’ or ‘no disease’

  Ex: Logistic regression, Decision tree, random forest

The target variable ‘Price’ is a real value and is an example of supervised regression problem

Important libraries to be used in the project

```
# Data wrangling libraries
import pandas as pd
import numpy as np

# Data visualisation libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
# Machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn import linear_model
```

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from scipy.stats import zscore

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Four different dataframes has been made for modelling.

1. Original dataframe with outliers
2. Dataframe without outliers
3. Dataframe with log transformed columns
4. Scaled dataframe

Function is defined to calculate model performance of given model and dataset

```
# Function to calculate model performance of given model and data set

def model_performance(model,method,data_type, X_train, X_test, y_train, y_test):
    """Input: Takes Machine leaning model, method used, training and test data set as input
        * model : Machine learning used
        * method : Method used of machine learning model (eg: SVM, NB, KNN, DT, RF)
        * data_type : Type of data used to train and test model (eg:original data, balanced, standardized,
                      important features only, with or without outliers)
        * X_train, X_test : Training and testing independent features
        * y_train, y_test : Training and testing target variable
       Output: Prints out model performance for given model
       and adds difference performance measures to the result dataframe"""

    global ResultsDF
    model.fit(X_train, y_train)

    print('Method: ', str(method))
    print('--'*10)
    print("Accuracy score for training data: ", model.score(X_train, y_train))
    print("Accuracy score for test data: ", model.score(X_test, y_test))
    print()
    y_pred = model.predict(X_test)

    model_r2_score = "%.3f" %float(r2_score(y_test, y_pred))
    model_mse = "%.3f" %float(mean_squared_error(y_test, y_pred))
    model_mae = round(mean_absolute_error(y_test, y_pred),3)

    TempResultsDF = pd.DataFrame({'METHOD':[str(method)],'Data type':[str(data_type)], 'r2_score':model_r2_score,
                                  'Mean_squared_error':model_mse, 'Mean_absolute_error':model_mae})
    ResultsDF = pd.concat([ResultsDF, TempResultsDF], ignore_index=True)
    return ResultsDF
```

Three performance metrics is considered to evaluate the model.

1. R squared
2. Mean squared error
3. Mean absolute error

4.1 Linear regression (Benchmark model)

Performance of the model with outlier, without outlier, and log transformation

```
# Apply best parameters on data with outliers
lr_model_final = LinearRegression(n_jobs=1)
model_performance(lr_model_final, 'Linear regression', 'With outliers', X_train0, X_test0, y_train0, y_test0)
```

```
Method:  Linear regression
--------------------
Accuracy score for training data:  0.7060769805016307
Accuracy score for test data:  0.6639796698273294
```

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	132536.83

```
# Model for data without outliers
model_performance(lr_model_final, 'Linear regression', 'Without outliers', X_train, X_test, y_train, y_test)
```

```
Method:  Linear regression
--------------------
Accuracy score for training data:  0.62268595023945
Accuracy score for test data:  -3676.8455087474017
```

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	132536.83
1	Linear regression	Without outliers	-3676.846	257742418262298.562	466444.96

```
# Model for data with log transformation
model_performance(lr_model_final, 'Linear regression', 'With log transformation', X_train1, X_test1, y_train1, y_test1)
```

```
Method:  Linear regression
--------------------
Accuracy score for training data:  0.7089582363870846
Accuracy score for test data:  0.6673062091484205
```

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	132536.830
1	Linear regression	Without outliers	-3676.846	257742418262298.562	466444.960
2	Linear regression	With log transformation	0.667	44677948394.429	131227.421

Model performance for with outliers and log transformation for training data is more and it is 0.706 and 0.708

Model performance for with outliers and log transformation for test data is more and it is 0.664 and 0.667

Model for data with scaling:

Scaling is a method used to normalize the range of independent variables or features of data

```
# Model for data with scaling
model_performance(lr_model_final, 'Linear regression', 'Scaled', X0_train_scale, X0_test_scale, y0_train_scale, y0_test_scale)
```

```
Method:  Linear regression
--------------------
Accuracy score for training data:  0.7060756883040656
Accuracy score for test data:  -1.120552529983192e+23
```

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	1.325368e+05
1	Linear regression	Without outliers	-3676.846	257742418262298.562	4.664450e+05
2	Linear regression	With log transformation	0.667	44677948394.429	1.312274e+05
3	Linear regression	Scaled	-1120552529983191971266560.000	150480680687457488596305559284613120.000	2.983731e+15

- Linear regression model gave consistent result without overfitting or underfitting for overall data and data with log transformed columns
- Linear regression model overfits and performed very poor for data with outliers and for scaled data

4.2 KNN Regressor

Model performance for with outliers, without outliers and for log transformation

```
# Data with outliers
knn_model_final = KNeighborsRegressor(metric='manhattan', n_neighbors=15, weights='distance')
model_performance(knn_model_final, 'KNN Regressor', 'With outliers', X_train0, X_test0, y_train0, y_test0)
```

```
Method:  KNN Regressor
--------------------
Accuracy score for training data:  0.9999192307343191
Accuracy score for test data:  0.526331595465001
```

```
# Model for data without outliers
model_performance(knn_model_final, 'KNN Regressor', 'Without outliers', X_train, X_test, y_train, y_test)
```

```
Method:  KNN Regressor
--------------------
Accuracy score for training data:  0.9998257944485558
Accuracy score for test data:  0.466888058585884
```

```
# Model for data with log transformation
model_performance(knn_model_final, 'KNN Regressor', 'With log transformation', X_train1, X_test1, y_train1, y_test1)
```

```
Method:  KNN Regressor
--------------------
Accuracy score for training data:  0.9999192307343191
Accuracy score for test data:  0.41019167457168515
```

```
# Model for data with scaling
model_performance(knn_model_final, 'KNN Regressor', 'Scaled', X0_train_scale, X0_test_scale, y0_train_scale, y0_test_scale)
```

```
Method:  KNN Regressor
--------------------
Accuracy score for training data:  0.9999192307343191
Accuracy score for test data:  0.6201179942724948
```

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	1.325368e+05
1	Linear regression	Without outliers	-3676.846	257742418262298.562	4.664450e+05

2	Linear regression	With log transformation	0.667	44677948394.429	1.312274e+05
3	Linear regression	Scaled	-1120552529983191971266560.000	150480680687457488596305559284613120.000	2.983731e+15
4	KNN Regressor	With outliers	0.526	63609640804.289	1.484478e+05
5	KNN Regressor	Without outliers	0.467	37360340628.169	1.303636e+05
6	KNN Regressor	With log transformation	0.410	79206245053.870	1.618641e+05
7	KNN Regressor	Scaled	0.620	51014924578.011	1.300346e+05

- KNN regressor overfits for all the dataframes
- Best model with KNN regressor is achieved for scaled data

4.3 Support vector regressor

Model performance for with outliers, without outliers and for log transformation

```
# Data with outliers
svr_model_final = SVR(C=10, gamma=0.001)
model_performance(svr_model_final, 'SVR', 'With outliers', X_train0, X_test0, y_train0, y_test0)
```

```
Method:  SVR
--------------------
Accuracy score for training data:  -0.06003636250169819
Accuracy score for test data:  -0.060735228969050814
```

```
# Model for data without outliers
model_performance(svr_model_final, 'SVR', 'Without outliers', X_train, X_test, y_train, y_test)
```

```
Method:  SVR
--------------------
Accuracy score for training data:  -0.04815102322849718
Accuracy score for test data:  -0.052142976190769685
```

```
# Model for data with log transformation
model_performance(svr_model_final, 'SVR', 'With log transformation', X_train1, X_test1, y_train1, y_test1)
```

```
Method:  SVR
--------------------
Accuracy score for training data:  -0.0600360900204
Accuracy score for test data:  -0.06073494826251613
```

```
# Model for data with scaling
model_performance(svr_model_final, 'SVR', 'Scaled', X0_train_scale, X0_test_scale, y0_train_scale, y0_test_scale)
```

```
Method:  SVR
--------------------
Accuracy score for training data:  -0.05692497531549412
Accuracy score for test data:  -0.057498105512244546
```

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	1.325368e+05
1	Linear regression	Without outliers	-3676.846	257742418262298.562	4.664450e+05
2	Linear regression	With log transformation	0.667	44677948394.429	1.312274e+05
3	Linear regression	Scaled	-1120552529983191971266560.000	150480680687457488596305559284613120.000	2.983731e+15
4	KNN Regressor	With outliers	0.526	63609640804.289	1.484478e+05
5	KNN Regressor	Without outliers	0.467	37360340628.169	1.303636e+05
6	KNN Regressor	With log transformation	0.410	79206245053.870	1.618641e+05
7	KNN Regressor	Scaled	0.620	51014924578.011	1.300346e+05

8	SVR	With outliers	-0.061	142447725575.902	2.213682e+05
9	SVR	Without outliers	-0.052	73733895128.580	1.814342e+05
10	SVR	With log transformation	-0.061	142447687879.401	2.213681e+05
11	SVR	Scaled	-0.057	142013007409.448	2.209149e+05

Support vector machine regressor is fitting worse than horizontal line for all the dataframes. Hence SVM model is not suitable for the present data.

4.4 Decision tree

Model performance for with outliers, without outliers and for log transformation

```
dtr_model_final = DecisionTreeRegressor(criterion='mse', max_depth=7, max_features=6, min_samples_leaf=3)
model_performance(dtr_model_final, 'Decision tree', 'With outliers', X_train0, X_test0, y_train0, y_test0)
```

```
Method:  Decision tree
--------------------
Accuracy score for training data:  0.5002197782539484
Accuracy score for test data:  0.39905555843433593
```

```
# Model for data without outliers
model_performance(dtr_model_final, 'Decision tree', 'Without outliers', X_train, X_test, y_train, y_test)
```

```
Method:  Decision tree
--------------------
Accuracy score for training data:  0.5292637931656831
Accuracy score for test data:  0.45074567520017716
```

```
# Model for data with log transformation
model_performance(dtr_model_final, 'Decision tree', 'With log transformation', X_train1, X_test1, y_train1, y_test1)
```

```
Method:  Decision tree
--------------------
Accuracy score for training data:  0.4237716847161761
Accuracy score for test data:  0.4040315664010662
```

```
# Model for data with scaling
model_performance(dtr_model_final, 'Decision tree', 'Scaled', X0_train_scale, X0_test_scale, y0_train_scale, y0_test_scale)
```

```
Method:  Decision tree
--------------------
Accuracy score for training data:  0.41746813124461535
Accuracy score for test data:  0.3870741657581972
```

- Decision tree regressor worked consistently with all dataframes without any overfitting or underfitting.
- Decision tree gave better result for overall data and bad result for scaled data.

4.5 Random forest

Model performance for with outliers, without outliers and for log transformation

```
rfr_model_final = RandomForestRegressor(n_estimators=100, criterion='mse', max_depth=7, max_features=6, min_samples_leaf=3)
model_performance(rfr_model_final, 'Random Forest', 'With outliers', X_train0, X_test0, y_train0, y_test0)
```

```
Method:  Random Forest
--------------------
Accuracy score for training data:  0.6643784767517316
Accuracy score for test data:  0.6447314604052443
```

```
# Model for data without outliers
model_performance(rfr_model_final, 'Random Forest', 'Without outliers', X_train, X_test, y_train, y_test)
```

```
Method:  Random Forest
--------------------
Accuracy score for training data:  0.617722211846204
Accuracy score for test data:  0.5659388889652308
```

```
# Model for data with log transformation
model_performance(rfr_model_final, 'Random Forest', 'With log transformation', X_train1, X_test1, y_train1, y_test1)
```

```
Method:  Random Forest
--------------------
Accuracy score for training data:  0.6694024165001397
Accuracy score for test data:  0.6474902166840218
```

```
# Model for data with scaling
model_performance(rfr_model_final, 'Random Forest', 'Scaled', X0_train_scale, X0_test_scale, y0_train_scale, y0_test_scale)
```

```
Method:  Random Forest
--------------------
Accuracy score for training data:  0.6697098844149403
Accuracy score for test data:  0.6478158821553217
```

- Random forest regressor gave better and consistant result without overfitting or underfitting.
- Model performance is better for overall data without dropping out outliers.
- Out of all the algorithms, Linear regression for overall original data gave a better performance hence that can be considered to be a better and final model.

Performance evaluation metrics of all algorithms

	METHOD	Data type	r2_score	Mean_squared_error	Mean_absolute_error
0	Linear regression	With outliers	0.664	45124674351.470	1.325368e+05
1	Linear regression	Without outliers	-3676.846	257742418262298.562	4.664450e+05
2	Linear regression	With log transformation	0.667	44677948394.429	1.312274e+05
3	Linear regression	Scaled	-1120552529983191971266560.000	1504806806874574885963055928461312.000	2.983731e+15
4	KNN Regressor	With outliers	0.526	63609640804.289	1.484478e+05
5	KNN Regressor	Without outliers	0.467	37360340628.169	1.303636e+05
6	KNN Regressor	With log transformation	0.410	79206245053.870	1.618641e+05
7	KNN Regressor	Scaled	0.620	51014924578.011	1.300346e+05
8	SVR	With outliers	-0.061	142447725575.902	2.213682e+05
9	SVR	Without outliers	-0.052	73733895128.580	1.814342e+05
10	SVR	With log transformation	-0.061	142447687879.401	2.213681e+05
11	SVR	Scaled	-0.057	142013007409.448	2.209149e+05
12	Decision tree	With outliers	0.399	80701730800.163	1.760350e+05
13	Decision tree	Without outliers	0.451	38491594488.738	1.363225e+05
14	Decision tree	With log transformation	0.404	80033495223.602	1.849118e+05
15	Decision tree	Scaled	0.387	82310730001.223	1.793482e+05
16	Random Forest	With outliers	0.645	47709545277.507	1.403591e+05
17	Random Forest	Without outliers	0.566	30418885232.029	1.208910e+05
18	Random Forest	With log transformation	0.647	47339067757.200	1.380667e+05

19	Random Forest	Scaled	0.648	47295333652.385	1.387160e+05

Improving Performance of all these models will be done by fine tuning the models by using cross validation, random searchCV and hyperparameters.

4.6 Gradient boosting

Model performance with outliers, without outliers, Log transformation and with scaling

```
#with outliers
gbr_model_final = GradientBoostingRegressor(n_estimators = 200, learning_rate = 0.1)
model_performance(gbr_model_final, 'Gradient Boost', 'With outliers', X_train0, X_test0, y_train0, y_test0)
```

```
Method:  Gradient Boost
--------------------
Accuracy score for training data:  0.8191147659823541
Accuracy score for test data:  0.7555687183076282
```

```
# Model for data without outliers
model_performance(gbr_model_final, 'Gradient Boost', 'Without outliers', X_train, X_test, y_train, y_test)
```

```
Method:  Gradient Boost
--------------------
Accuracy score for training data:  0.7316346257004744
Accuracy score for test data:  0.6703631208864331
```

```
# Model for data with log transformation
model_performance(gbr_model_final, 'Gradient Boost', 'With log transformation', X_train1, X_test1, y_train1, y_test1)
```

```
Method:  Gradient Boost
--------------------
Accuracy score for training data:  0.8191147659823541
Accuracy score for test data:  0.7564834982795052
```

```
# Model for data with scaling
model_performance(gbr_model_final, 'Gradient Boost', 'Scaled', X0_train_scale, X0_test_scale, y0_train_scale, y0_test_scale)
```

```
Method:  Gradient Boost
--------------------
Accuracy score for training data:  0.8191147659823541
Accuracy score for test data:  0.7585772657396522
```

Gradient Boost Regressor algorithm is giving a consistent and better result compared to all other models. Hence, here after Gradient Boost Regressor with log transformation data will be considered for further analysis

Performance evaluation metrics of all algorithms

METHOD	Data type	Training_accuracy	Testing_accuracy	r2_score	Mean_squared_error	Mean_absolute_error
Linear regression	With outliers	0.706077	6.520010e-01	0.652	46733309175.500	1.330160e+05
Linear regression	Without outliers	0.622670	-2.150636e+03	-2150.636	150786045077654.938	3.999795e+05
Linear regression	With log transformation	0.708958	6.673062e-01	0.667	44677948394.446	1.312274e+05
Linear regression	Scaled	0.706025	-2.531550e+24	-2531549966033982748360704.000	339656436354820110107433982178099 20.000	1.573133e+16
Lasso regression	With outliers	0.701991	6.791048e-01	0.679	43093499618.842	1.323014e+05
Lasso regression	Without outliers	0.620398	6.133616e-01	0.613	27095513613.212	1.142645e+05
Lasso regression	With log transformation	0.705170	6.768164e-01	0.677	43400809386.308	1.308605e+05
Lasso regression	Scaled	0.706046	6.683454e-01	0.668	44538396008.374	1.322423e+05

8	Ridge regression	With outliers	0.706043	6.693214e-01	0.669	44407331592.926	1.321866e+05
9	Ridge regression	Without outliers	0.622582	6.167161e-01	0.617	26860430991.836	1.139426e+05
10	Ridge regression	With log transformation	0.708930	6.684827e-01	0.668	44519951447.477	1.311323e+05
11	Ridge regression	Scaled	0.706072	6.679393e-01	0.668	44592924952.121	1.322804e+05
12	KNN Regressor	With outliers	0.999919	5.263350e-01	0.526	63609183649.854	1.484578e+05
13	KNN Regressor	Without outliers	0.999826	4.670024e-01	0.467	37352327000.878	1.303406e+05
14	KNN Regressor	With log transformation	0.999919	4.101985e-01	0.410	79205331788.063	1.618633e+05
15	KNN Regressor	Scaled	0.999919	6.201179e-01	0.620	51014938853.768	1.300347e+05
16	SVR	With outliers	-0.060036	-6.073523e-02	-0.061	142447725575.902	2.213682e+05
17	SVR	Without outliers	-0.048151	-5.214298e-02	-0.052	73733895128.580	1.814342e+05
18	SVR	With log transformation	-0.060036	-6.073495e-02	-0.061	142447687879.401	2.213681e+05
19	SVR	Scaled	-0.056925	-5.749811e-02	-0.057	142013007409.448	2.209149e+05
20	Decision tree	With outliers	0.585350	5.622286e-01	0.562	58788982264.562	1.557410e+05
21	Decision tree	Without outliers	0.497994	4.517121e-01	0.452	38423867121.829	1.367441e+05
22	Decision tree	With log transformation	0.602433	5.383827e-01	0.538	61991278586.669	1.508232e+05
23	Decision tree	Scaled	0.570265	5.114167e-01	0.511	65612582725.783	1.577361e+05
24	Random Forest	With outliers	0.670464	6.463961e-01	0.646	47485994132.667	1.391672e+05
25	Random Forest	Without outliers	0.617572	5.673673e-01	0.567	30318779221.299	1.209974e+05
26	Random Forest	With log transformation	0.667452	6.489873e-01	0.649	47138021439.001	1.385901e+05
27	Random Forest	Scaled	0.665775	6.435271e-01	0.644	47871280278.982	1.396453e+05
28	Gradient Boost	With outliers	0.819115	7.555687e-01	0.756	32825043597.847	1.186247e+05
29	Gradient Boost	Without outliers	0.731635	6.703631e-01	0.670	23100863309.537	1.070734e+05
30	Gradient Boost	With log transformation	0.819115	7.564835e-01	0.756	32702196422.758	1.185778e+05
31	Gradient Boost	Scaled	0.819115	7.585773e-01	0.759	32421021248.749	1.184498e+05

4.7 Feature importance

It refers to the technique that assign a score to input features based on how useful they are at predicting a target variable.

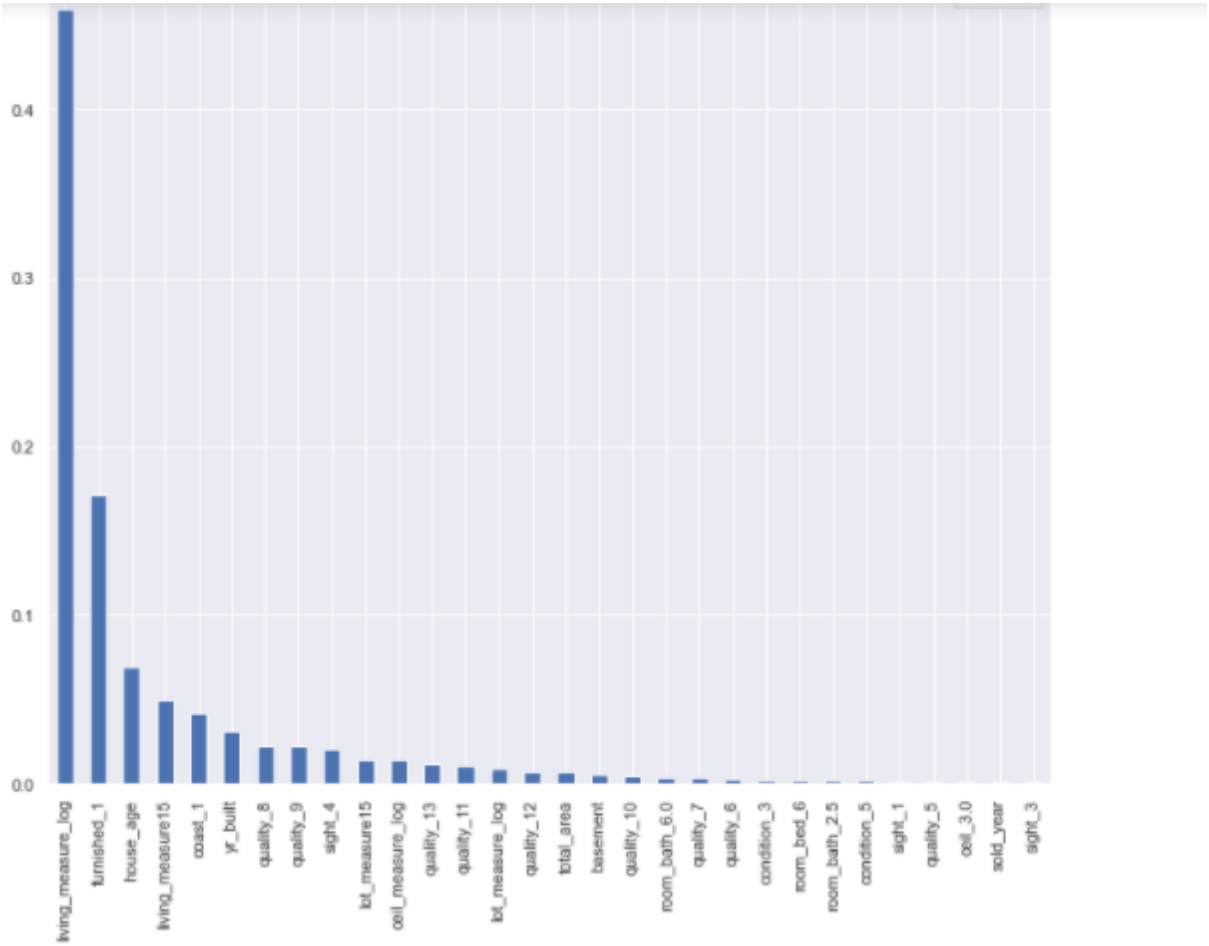
There are many types and sources of feature importance scores, although popular examples include statistical correlation scores, coefficients calculated as part of linear models, decision trees, and permutation importance scores.

```
#feature importance
gbr_imp_feature_1= pd.DataFrame(gbr.feature_importances_, columns = ["Imp"], index = X1.columns)
gbr_imp_feature_1.sort_values(by="Imp",ascending=False)
gbr_imp_feature_1['Imp'] = gbr_imp_feature_1['Imp'].map('{0:.5f}'.format)
gbr_imp_feature_1=gbr_imp_feature_1.sort_values(by="Imp",ascending=False)
gbr_imp_feature_1.Imp=gbr_imp_feature_1.Imp.astype("float")

gbr_imp_feature_1[:30].plot.bar(figsize=(12, 10))

print("First 20 feature importance:\t",(gbr_imp_feature_1[:20].sum())*100)
print("First 30 feature importance:\t",(gbr_imp_feature_1[:30].sum())*100)
```

```
First 20 feature importance:     Imp    98.048
dtype: float64
First 30 feature importance:     Imp    99.465
dtype: float64
```



First 20 features have an importance of 98.037 and first 30 have importance of 99.445

1.The ensemble models (Random Forrest and Gradient Boost Regressor) have performed well compared to that of linear, KNN, SVR models.

2.The best performance is given by Gradient boosting model with training score-82%, Testing score 76%

3.The top key features that drive the price of the property are:'living_measure_log','furnished_1','house_age','living_measure15','coast_1','yr_built' etc...

4.8 Pickling the model

Pickling is a useful python tool that allows you to save your models, to minimize lengthy re-training and allow you to share, commit and reload pre-trained machine learning models

```
# Pickling Gradient Boost Regressor model:
import pickle
GBR = GradientBoostingRegressor(n_estimators = 200, learning_rate = 0.1)
GBR = GBR.fit(X_train1, y_train1)
y_pred = GBR.predict(X_test1)
with open('GB_pickle','wb') as f:
    pickle.dump(GBR,f)
```

Gradient Boost model is stored in drive for future use.

```
# Just checking the pickled model
with open('GB_pickle','rb') as f:
    mp=pickle.load(f)
mp.predict(X_test1)

GBR_r2_score = "%.3f" %float(r2_score(y_test1, y_pred))
GBR_mse = "%.3f" %float(mean_squared_error(y_test1, y_pred))
GBR_mae = round(mean_absolute_error(y_test1, y_pred),3)
train_acc = GBR.score(X_train1, y_train1)
test_acc = GBR.score(X_test1, y_test1)

print(f'''Model Train accuracy : {train_acc}''')
print(f'''Model Test accuracy : {test_acc}''')
print(f'''Model r2_score : {GBR_r2_score}''')
print(f'''Model MSE : {GBR_mse}''')
print(f'''Model MAE : {GBR_mae}''')
```

```
Model Train accuracy : 0.8191147659823541
Model Test accuracy : 0.7553975917390104
Model r2_score : 0.755
Model MSE : 32848024441.529
Model MAE : 118632.44
```

After removing few variables which are not relevant in predicting the target, the final independent variables are

```
df1_final.columns
```

```
Index(['price', 'room_bed', 'room_bath', 'ceil', 'coast', 'sight', 'condition',
       'quality', 'basement', 'yr_built', 'living_measure15', 'lot_measure15',
       'furnished', 'total_area', 'sold_year', 'house_age', 'ceil_measure_log',
       'living_measure_log', 'lot_measure_log', 'has_basement_Yes'],
      dtype='object')
```

4.9 Build Gradient boosting regressor for final dataset

since there are only 20 attributes. Let us build our GBR model for this data:

```
# Seperating dependemt and independent variables and splitting training an testing for "data with log transformation".
X1_new = df1_final.drop(columns='price', axis=1)
y1_new = df1_final['price']

X_train1_new, X_test1_new, y_train1_new, y_test1_new = train_test_split(X1_new, y1_new, test_size=0.3, random_state=42)
```

Since it is learnt that the best model is Gradient Boost for our regression problem. Also, the hyperparameter tuning is done by gridsearch.The best model with best hyperparameter and the

best data is found from our analysis. And the same is considered for model building without one hot encoding.

```
gbr_new=GradientBoostingRegressor(n_estimators = 200, learning_rate = 0.1)
gbr_new.fit(X_train1_new, y_train1_new)
y_pred_new= gbr_new.predict(X_test1_new)
```

```
gbr_r2_score_new = "%.3f" %float(r2_score(y_test1_new, y_pred_new))
gbr_mse_new = "%.3f" %float(mean_squared_error(y_test1_new, y_pred_new))
gbr_mae_new = round(mean_absolute_error(y_test1_new, y_pred_new),3)
train_acc =gbr_new.score(X_train1_new, y_train1_new)
test_acc = gbr_new.score(X_test1_new, y_test1_new)
```

```
gbr_result_df_new = pd.DataFrame({'METHOD':['Gradient Boost Regressor_new'],'Training_accuracy':[train_acc],
                                  'Testing_accuracy':[test_acc], 'r2_score':[gbr_r2_score_new],
                                   'Mean_squared_error':[gbr_mse_new], 'Mean_absolute_error':[gbr_mae_new]})

gbr_result_df_new1 = pd.concat([gbr_result_df,gbr_result_df_new],ignore_index=True)
gbr_result_df_new1
```

	METHOD	Training_accuracy	Testing_accuracy	r2_score	Mean_squared_error	Mean_absolute_error
0	Gradient Boost Regressor	0.819115	0.757270	0.757	32596555977.581	118515.387
1	Gradient Boost Regressor_new	0.822317	0.750626	0.751	33488764496.022	118877.219

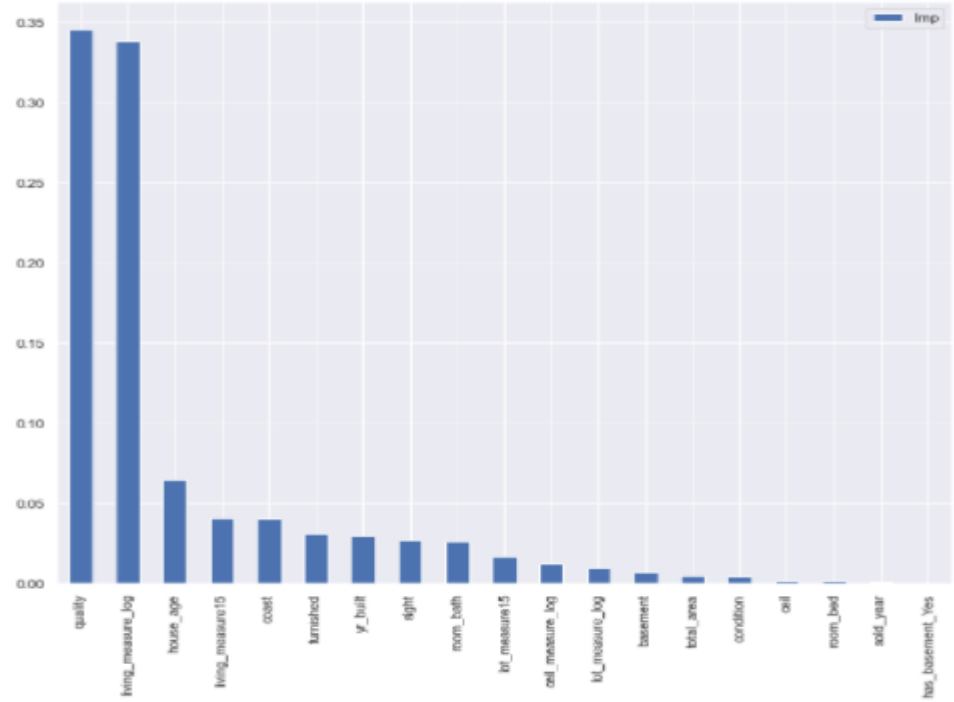
4.11 Feature importance

```
#feature importance
gbr_imp_feature_1= pd.DataFrame(gbr_new.feature_importances_, columns = ["Imp"], index = X_train1_new.columns)
gbr_imp_feature_1.sort_values(by="Imp",ascending=False)
gbr_imp_feature_1['Imp'] = gbr_imp_feature_1['Imp'].map('{0:.5f}'.format)
gbr_imp_feature_1=gbr_imp_feature_1.sort_values(by="Imp",ascending=False)
gbr_imp_feature_1.Imp=gbr_imp_feature_1.Imp.astype("float")

gbr_imp_feature_1[:30].plot.bar(figsize=(12, 10))

print("First 10 feature importance:\t",(gbr_imp_feature_1[:10].sum())*100)
print("First 15 feature importance:\t",(gbr_imp_feature_1[:15].sum())*100)
```

```
First 10 feature importance:     Imp    95.894
dtype: float64
First 15 feature importance:     Imp    99.63
dtype: float64
```



4.12 Pickling the model

Pickling The final Gradient Boost Regressor model without considering one hot encoding:

```
# Pickling The final Gradient Boost Regressor model without considering one hot encoding:
import pickle
gbr_new=GradientBoostingRegressor(n_estimators = 200, learning_rate = 0.1)
gbr_new.fit(X_train1_new, y_train1_new)
y_pred_new= gbr_new.predict(X_test1_new)
with open('GBR_new_pickle','wb') as f:
    pickle.dump(gbr_new,f)
```

```
# Just checking the pickled model
with open('GBR_new_pickle','rb') as f:
    mp=pickle.load(f)
mp.predict(X_test1_new)

gbr_r2_score_new = "%.3f" %float(r2_score(y_test1_new, y_pred_new))
gbr_mse_new = "%.3f" %float(mean_squared_error(y_test1_new, y_pred_new))
gbr_mae_new = round(mean_absolute_error(y_test1_new, y_pred_new),3)
train_acc =gbr_new.score(X_train1_new, y_train1_new)
test_acc = gbr_new.score(X_test1_new, y_test1_new)

print(f'''Model Train accuracy : {train_acc}''')
print(f'''Model Test accuracy : {test_acc}''')
print(f'''Model r2_score : {gbr_r2_score_new}''')
print(f'''Model MSE : {gbr_mse_new}''')
print(f'''Model MAE : {gbr_mae_new}''')
```

```
Model Train accuracy : 0.822317085271464
Model Test accuracy : 0.7510656793474786
Model r2_score : 0.751
Model MSE : 33429763456.804
Model MAE : 118850.05
```

Best model is pickled and crosschecked.

4.13 Checking the price of house by inputing different values for independent attributes

House price prediction for new data point

```
def predict_price(room_bed, room_bath, ceil, coast, sight, condition,
       quality, basement, yr_built, living_measure15, lot_measure15,
       furnished, total_area, sold_year, house_age, ceil_measure_log,
       living_measure_log, lot_measure_log, has_basement_Yes):

    x = np.zeros(len(X_train1_new.columns))
    x[0] = room_bed
    x[1] = room_bath
    x[2] = ceil
    x[3] = coast
    x[4] = sight
    x[5] = condition
    x[6] = quality
    x[7] = basement
    x[8] = yr_built
    x[9] = living_measure15
```

```
x[10] = lot_measure15
x[11] = furnished
x[12] = total_area
x[13] = sold_year
x[14] = house_age
x[15] = ceil_measure_log
x[16] = living_measure_log
x[17] = lot_measure_log
x[18] = has_basement_Yes

return gbr_new.predict([x])[0]
```

```
# cross checking the values
# first row
pre1 = predict_price(4,3.25,1,0,0,5,9,0,1956,2120,7553,1,16477,2014,58,3.480007,3.480007,4.128948,0)
print(f'''The Predicted price of the house for the given attributes from our model is: {pre1}''')
```

```
The Predicted price of the house for the given attributes from our model is: 1027601.2914111025
```

Model evaluation

Many models were built for different data sets before concluding the best model for the given problem as explained in the previous section in detail.
At the end of numerous trials carried out on the different data sets and the models, the gradient boosting regressor is selected as the final model for the current project.

The objective of the model in the present study was to predict the house price given many features or attributes about the house as accurately as possible so that the difference between actual and predicted price is minimized. Hence the problem was regression problem predicting a real valued number as an output. To achieve the above objective various models were tried and at last the gradient boosting algorithm with 200 numbers of estimators and 0.1 as a learning rate gave the best result out of the lot and hence concluded to be the final model for the present case.

The output of the model confirmed that the features like ‘living measure’, ‘furnished’ and ‘house age’ were most prominent features in predicting the house price along with many other parameters.

To decide the best model among the lot, the models were compared with the benchmark model ‘Linear regression’ performance using ‘R squared’, ‘Mean absolute error’ and ‘Mean squared error’ as a evaluation matrices.

Comparison to benchmark

In the present study, instead of keeping the particular value as a benchmark, the basic linear regression model performance was considered to be a benchmark and

further different models were built and trained in order improve the results compared the results of the benchmark model. The benchmark 'Linear regression' model could get us around 65% accurate result in predicting the house prices.

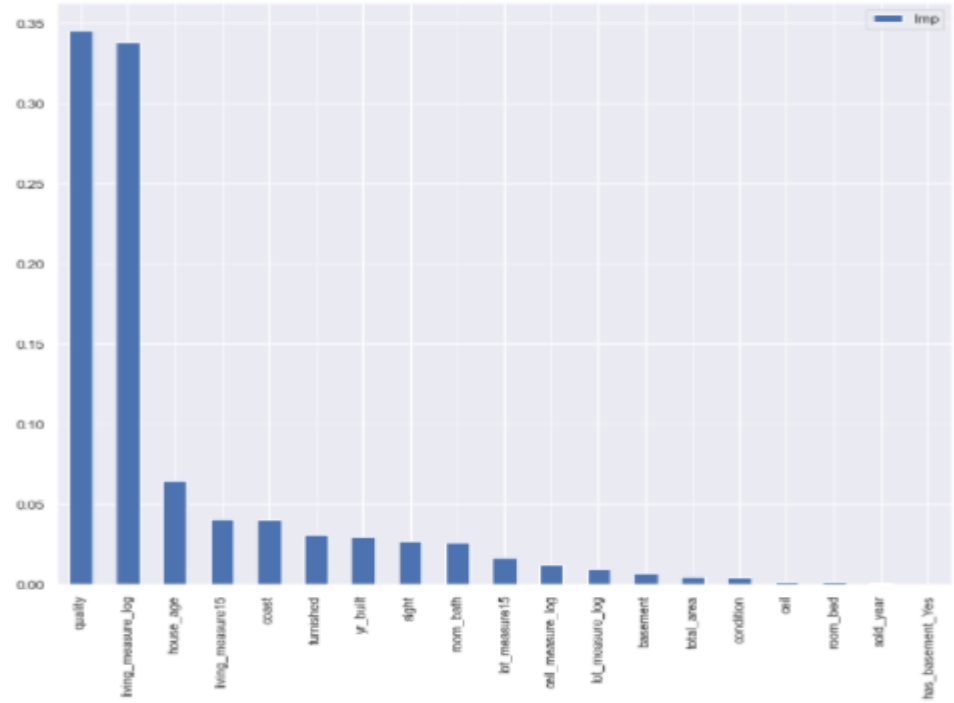
The different matrices like 'R squared', 'Mean absolute error' and 'Mean squared error' are used to evaluate the different models. But, mainly considering the R squared value the final model could get almost 10% better results compared to the benchmark model hence concluded to be the final model.

The different models performed differently on all the datasets but finally the ensembles model with proper hyper parameter tuning could give better results in predicting the house price hence could improve the results compared to benchmark model results. The better performance could be due to accommodating the non-linear behaviour in the model which was not present in the benchmark model and also the generalisation of results of many models due to the use of ensemble technique in the form of gradient boosting has helped reduce the bias and variance without losing maximum feature importance and information from the data. Hence the final model could perform better compared to the benchmark set.

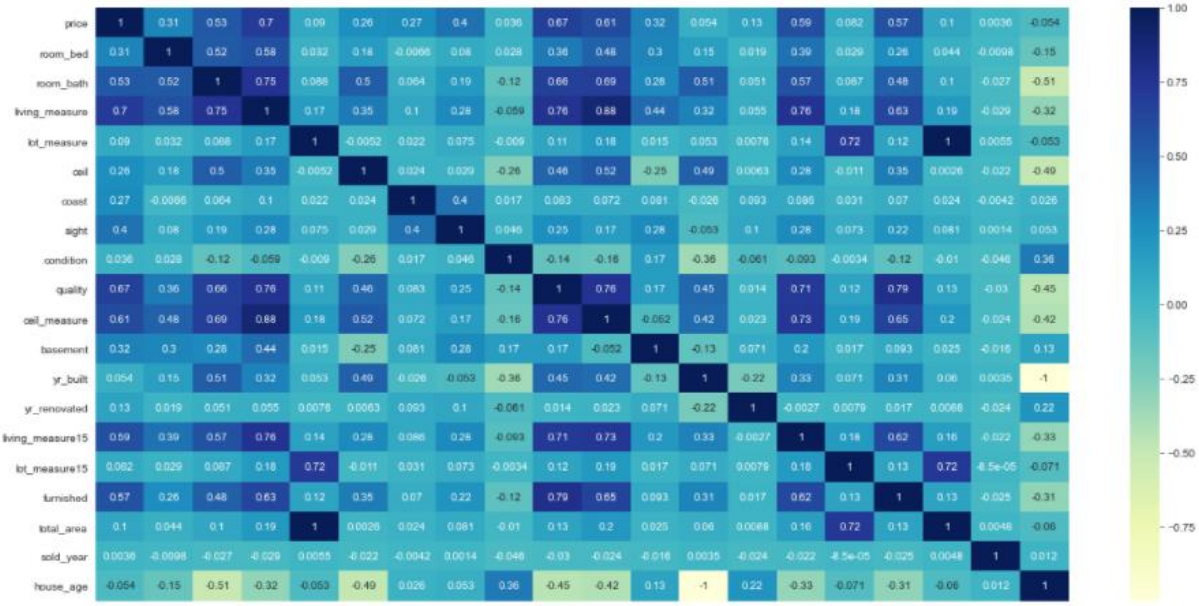
Visualizations

In addition to quantifying the model and the solution, the below are the relevant visualizations that support the ideas/insights that gleaned from the data.

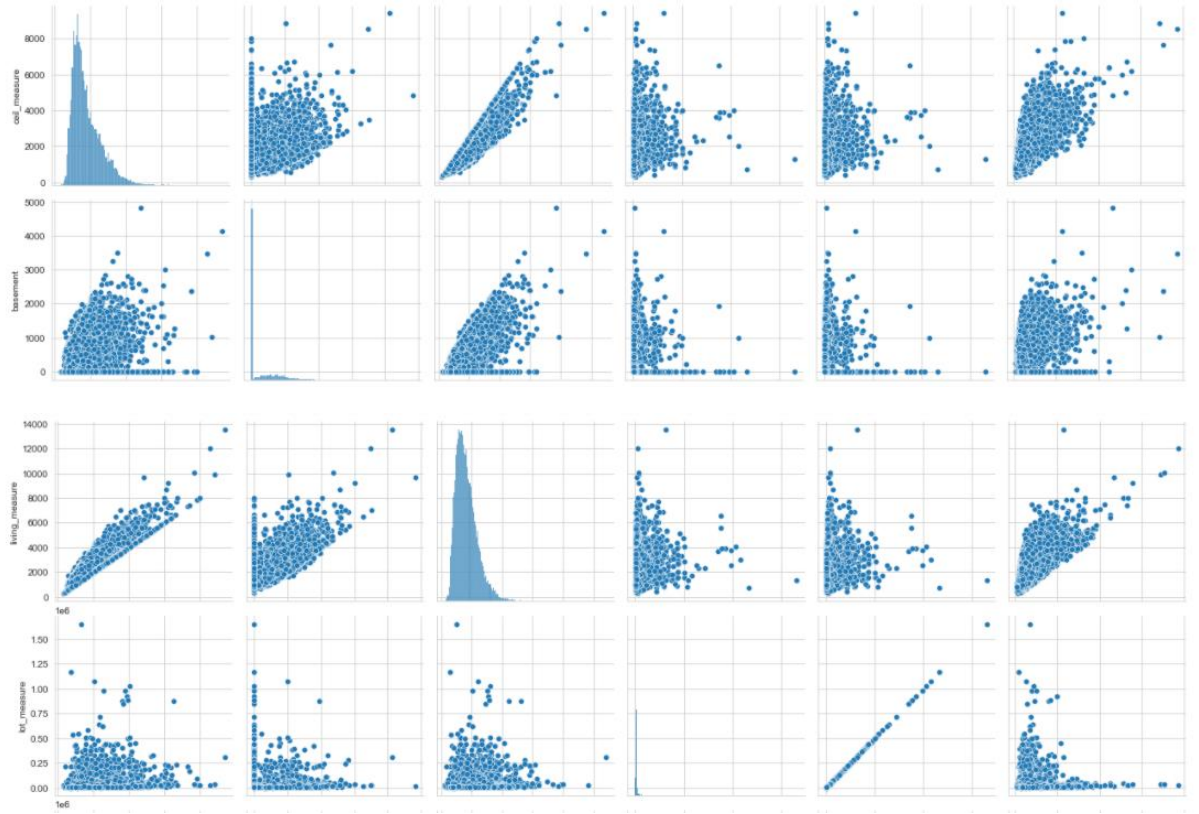
Feature importances:

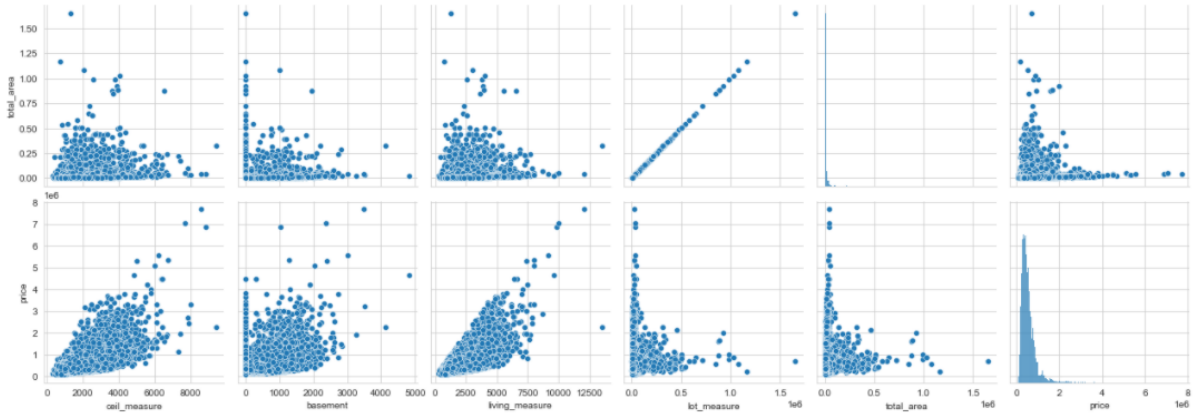


Heatmap



Bivariate analysis





Implications

We have arrived at our final model which is able to predict the price of a house with 95% of confidence interval. Our final model train accuracy is 82% and test accuracy is 76%.

Table showing price of the house (actual v/s predicted)

	Row	Actual Price	Predicted Price
0	1st row	808100.0	1.027601e+06
1	4th row	300000.0	3.739634e+05
2	5th row	699000.0	5.804210e+05

From the above observation it can be seen that a random cross checking is made showing the comparison of our model's predicting price v/s actual price of the house. Our model's prediction can be found in line with the actual price of the house tested for different records.

Limitations

The main limitations of our solution are,

- **Treating outliers:** there can be found quite numerous outliers in many attributes. Removing outliers yielded drop in the model accuracy in almost all the algorithms we tried.

- **Handling categorical variable:** for the same, we tried one hot encoding which results in the increase of number of attributes. That would be very difficult to input the different attributes when it is taken to production environment.

- **Visualization:** Since, the number of independent attributes is high the visualization and drawing the relation between each attribute were not easy.

We have found outliers individually for each attribute and taken care for the same. For the final model we have tested by removing dummy variables and the model behaved smoothly without change in its performance and accuracy. For visualization we tried univariate, bivariate and multivariate plots and tried to extract maximum relation among independent variable and between independent and target variable.

Conclusions and closing reflections

We have built different models on the datasets with all possible hyperparameter tuning. Ensembles models particularly Gradient Boost Regressor is performing better with good accuracy and performance score. The final best model is pickled for further use and exposed as rest API using ‘flask’ framework which is able to predict the price of the property in the local machine.

The top key features to consider for pricing a property are: ’quality’,’living_measure,’house age’, ‘Coast’, ‘furnished', 'yr_built'. So, one needs to thoroughly introspect its property on parameters suggested and list its price accordingly, similarly if one wants buy house - needs to check the features suggested above in house and calculate the predicted price. The same can then be compared to listed price.

For further improvisation, the datasets can be made by treating outliers in different ways. Making polynomial features and improvising the model performance can also be explored further. The scope for future work is model is pickled and exposed as rest API it can be deployed in the external server using AWS or Heroku cloud platform.