REPORT ON HOUSE PRICE PREDICTION

1. Introduction

The Real Estate sector is booming in different places, so to make the most out of it both property buyers and sellers need to make appropriate decisions which will require them to do some analysis and visualization so they can understand which types of trends have followed in the past years concerning prices and other factors if someone is new in such type of investments they will get benefited from this analysis.

Business Problems We are Trying to Solve

- Problems we generally face during buying a house like:
    - We are not aware of general factors that influence the house prices
    - Brokerage Fees increases the price.
    - Tremendous Paper work
    - Here, we are making predictions of the selling price of the house on the basis of the previous sold houses.
    - We are also making conclusion what are the factors affecting the prices of the house.

Main Goal

- Create an analytical framework to understand
    - Key factors impacting house price.
- Develop a modelling framework
    - To estimate the price of a house that is up for sale

About Dataset

The dataset we've consisted of records of different apartments from the United States. It has various columns which describe the apartment such as the area covered, number of

bedrooms and bathrooms, price, year in which it was built, etc. Such a dataset is very useful for the property-dealer firms on which different analysis can be done to get insights apartments across the state/country, they can also predict price with the help of Machine Learning models.

This is what the dataset looks like,

```
1 data.head()
```

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight
0	3876100940	20150427T000000	600000	4.0	1.75	3050.0	9440.0	1	0	0.0
1	3145600250	20150317T000000	190000	2.0	1.00	670.0	3101.0	1	0	0.0
2	7129303070	20140820T000000	735000	4.0	2.75	3040.0	2415.0	2	1	4.0
3	7338220280	20141010T000000	257000	3.0	2.50	1740.0	3721.0	2	0	0.0
4	7950300670	20150218T000000	450000	2.0	1.00	1120.0	4590.0	1	0	0.0

basement	yr_built	yr_renovated	zipcode	lat	long	living_measure15	lot_measure15	furnished	total_area
1250.0	1966	0	98034	47.7228	-122.183	2020.0	8660.0	0.0	12490
0.0	1948	0	98118	47.5546	-122.274	1660.0	4100.0	0.0	3771
0.0	1966	0	98118	47.5188	-122.256	2620.0	2433.0	0.0	5455
0.0	2009	0	98002	47.3363	-122.213	2030.0	3794.0	0.0	5461
0.0	1924	0	98118	47.5663	-122.285	1120.0	5100.0	0.0	5710

Table1- DataSet

It has lots of rows and columns, which is 21613 rows spread across 23 columns. Here is the list of columns this dataset has,

```
1 data.columns
```

```
Index(['cid', 'dayhours', 'price', 'room_bed', 'room_bath', 'living_measure',
       'lot_measure', 'ceil', 'coast', 'sight', 'condition', 'quality',
       'ceil_measure', 'basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'living_measure15', 'lot_measure15', 'furnished',
       'total_area'],
      dtype='object')
```

Table 2 - Columns in

2. EDA & Business Implication

EDA stands for exploratory data analysis where we explore our data and grab insights from it. EDA helps us in getting knowledge in form of various plots and diagrams where we can easily understand the data and its features.

Analysis Of the Data

	count	mean	std	min	25%	50%	75%	max
cid	21613.0	4.580302e+09	2.876566e+09	1.000102e+06	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	21613.0	5.401822e+05	3.673622e+05	7.500000e+04	3.219500e+05	4.500000e+05	6.450000e+05	7.700000e+06
room_bed	21613.0	3.369500e+00	9.283307e-01	0.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
room_bath	21613.0	2.115845e+00	7.683799e-01	0.000000e+00	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
living_measure	21613.0	2.079727e+03	9.181472e+02	2.900000e+02	1.430000e+03	1.910000e+03	2.550000e+03	1.354000e+04
lot_measure	21613.0	1.509003e+04	4.138466e+04	5.200000e+02	5.043000e+03	7.618000e+03	1.066000e+04	1.651359e+06
ceil	21613.0	1.492481e+00	5.397630e-01	1.000000e+00	1.000000e+00	1.500000e+00	2.000000e+00	3.500000e+00
coast	21613.0	7.449220e-03	8.598879e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
sight	21613.0	2.337482e-01	7.655206e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
condition	21613.0	3.407718e+00	6.499332e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	5.000000e+00
quality	21613.0	7.656827e+00	1.175465e+00	1.000000e+00	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
ceil_measure	21613.0	1.788356e+03	8.280848e+02	2.900000e+02	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03
basement	21613.0	2.915090e+02	4.425750e+02	0.000000e+00	0.000000e+00	0.000000e+00	5.600000e+02	4.820000e+03
yr_built	21613.0	1.971012e+03	2.936343e+01	1.900000e+03	1.951000e+03	1.975000e+03	1.997000e+03	2.015000e+03
yr_renovated	21613.0	8.440226e+01	4.016792e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.015000e+03
zipcode	21613.0	9.807794e+04	5.350503e+01	9.800100e+04	9.803300e+04	9.806500e+04	9.811800e+04	9.819900e+04
lat	21613.0	4.756005e+01	1.385637e-01	4.715590e+01	4.747100e+01	4.757180e+01	4.767800e+01	4.777760e+01
long	21613.0	-1.222139e+02	1.407590e-01	-1.225190e+02	-1.223280e+02	-1.222300e+02	-1.221250e+02	-1.213150e+02
living_measure15	21613.0	1.985936e+03	6.830025e+02	3.990000e+02	1.490000e+03	1.840000e+03	2.360000e+03	6.210000e+03
lot_measure15	21613.0	1.275964e+04	2.726932e+04	6.510000e+02	5.100000e+03	7.620000e+03	1.008000e+04	8.712000e+05
furnished	21613.0	1.964558e-01	3.973264e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
total_area	21613.0	1.716808e+04	4.156534e+04	1.423000e+03	7.040000e+03	9.575000e+03	1.297000e+04	1.652659e+06

Table 3- Checking of the Skewness

Observation:

price: Our target column value is in 75k - 7700k range. As Mean is greater than Median data is rightly skewed.

room_bed: Number of bedrooms ranges from 0 - 33. As Mean is greater than Median data is rightly skewed.

room_bath: Number of bathrooms range from 0 - 8. As Mean slightly less than Median, it's slightly leftly skewed.

living_measure: Square footage of house range from 290 - 13,540. As Mean is greater than Median data is rightly skewed.

lot_measure: Square footage of lot range from 520 - 16,51,359. As Mean almost double of Median, it's Highly rightly skewed.

ceil: Number of floors range from 1 - 3.5 As Mean ~ Median, it's almost Normal Distributed.

coast: As this value represent whether house has waterfront view or not. From above analysis we got know, very few houses has waterfront view.

sight: Value ranges from 0 - 4. As Mean > Median, it's rightly skewed

condition: Represents rating of house which ranges from 1 - 5. As Mean is greater than Median data is rightly skewed.

quality: Representing grade given to house which range from 1 - 13. As Mean is greater than Median data is rightly skewed.

ceil_measure: Square footage of house apart from basement ranges in 290 - 9,410 As Mean is greater than Median data is rightly skewed.

basement: Square footage house basement ranges in 0 - 4,820. As Mean highly > Median, it's Highly rightly skewed.

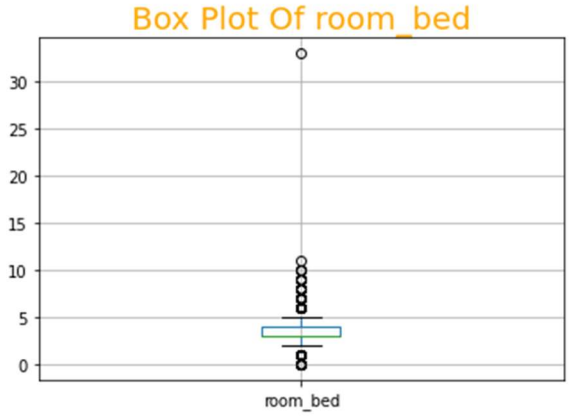
yr_built: House built year ranges from 1900 - 2015. As Mean is less than Median data is left skewed.

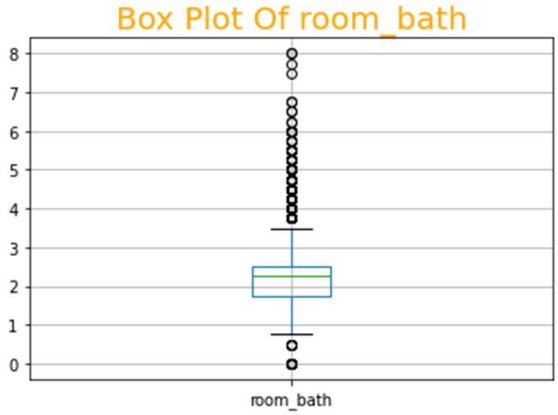
yr_renovated: House renovation year only 2015. So this column can be used as Categorical Variable for knowing whether house is renovated or not.

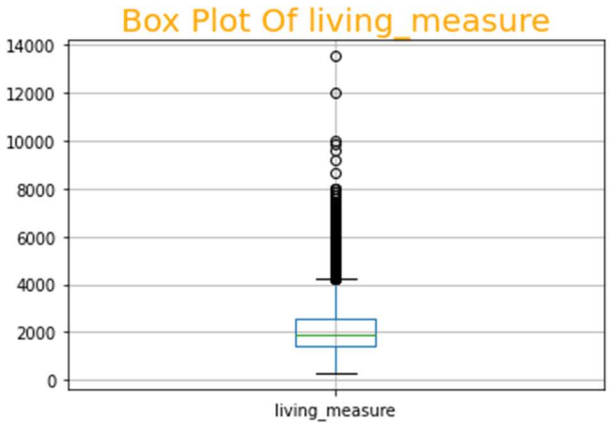
living_measure15: Value ranges from 399 to 6,210. As Mean is greater than Median data is rightly skewed.

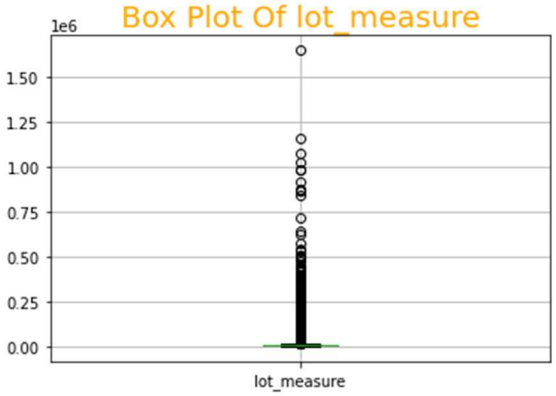
lot_measure15: Value ranges from 651 to 8,71,200 As Mean is highly greater than Median data is highly right skewed.

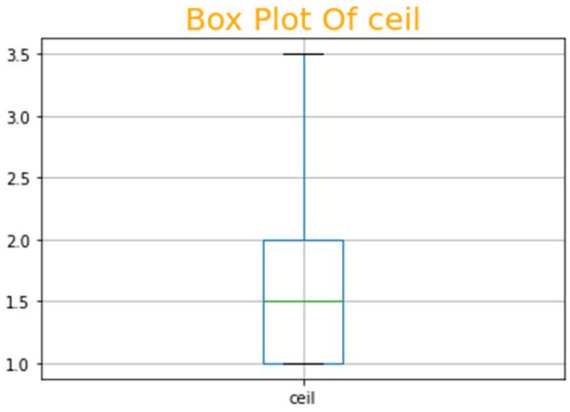
Uni-variate Analysis - By BoxPlot

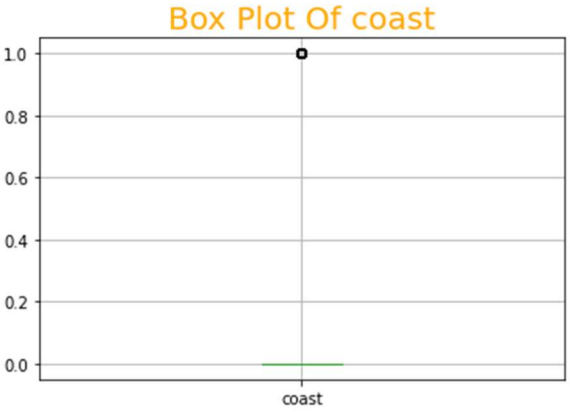




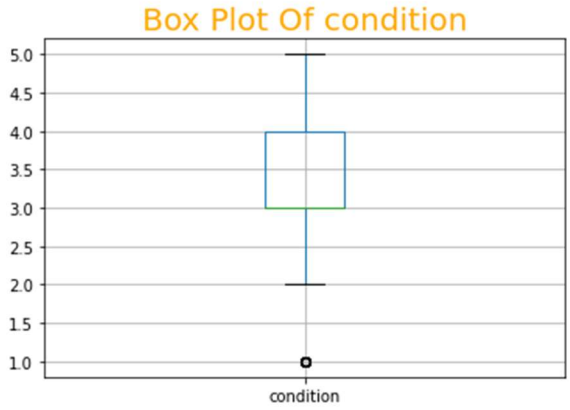


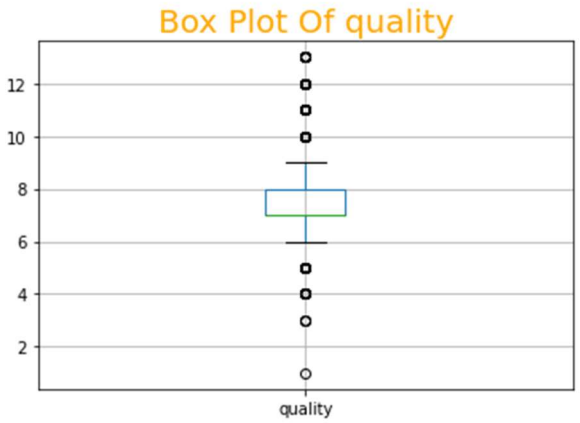


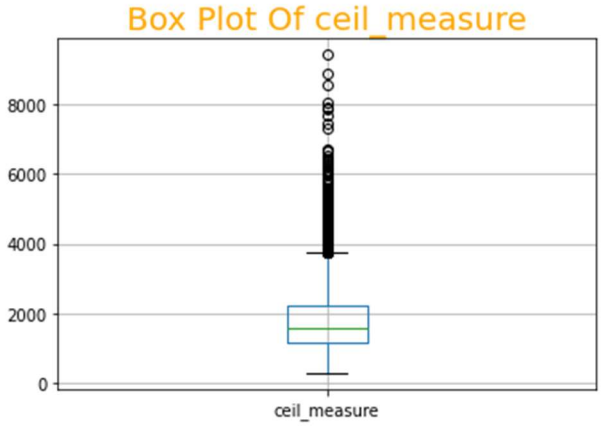


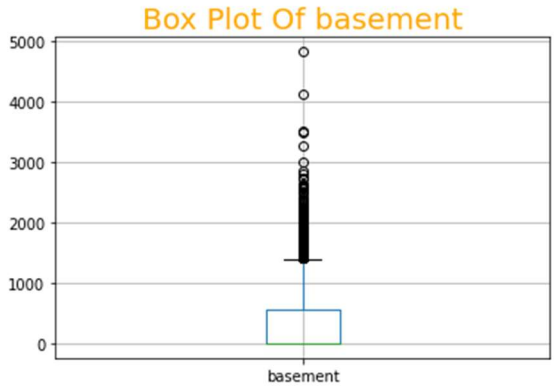


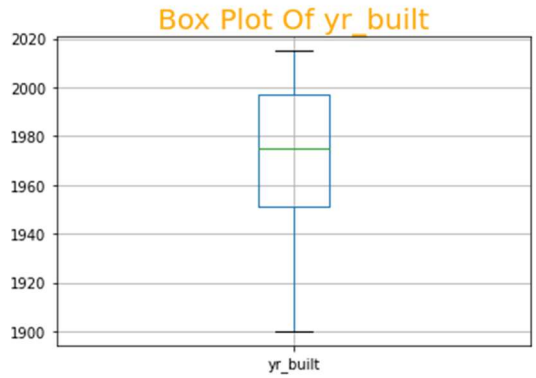

Box Plot Of sight
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
sight

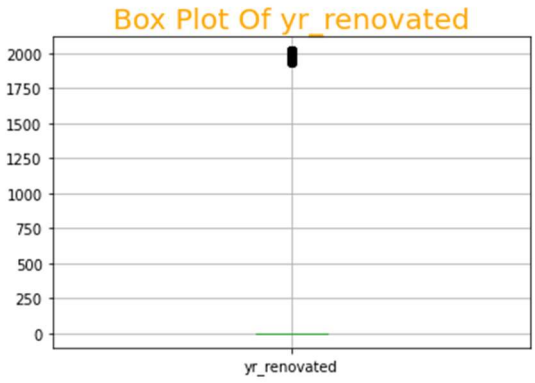

Box Plot Of condition
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
condition

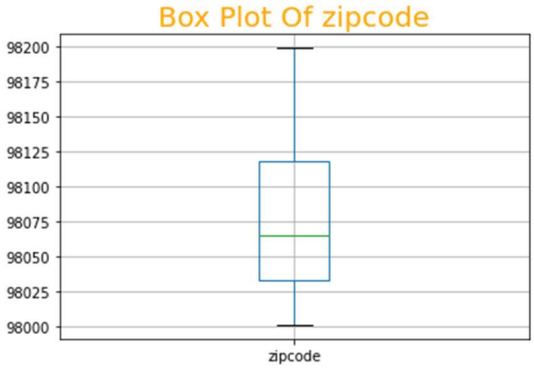

Box Plot Of quality
12
10
8
6
4
2
quality

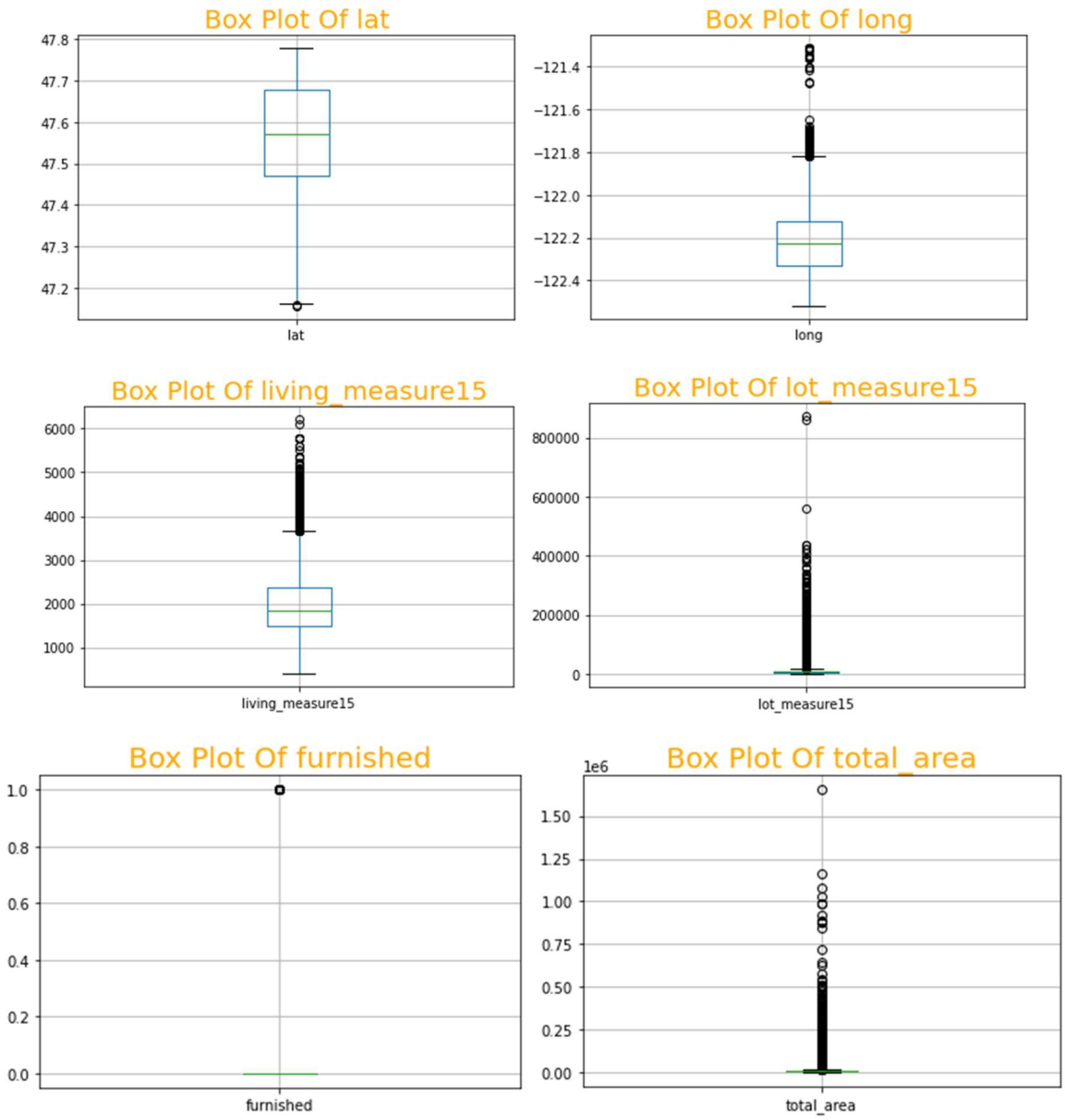

Box Plot Of ceil_measure
8000
6000
4000
2000
0
ceil_measure


Box Plot Of basement
5000
4000
3000
2000
1000
0
basement


Box Plot Of yr_built
2020
2000
1980
1960
1940
1920
1900
yr_built


Box Plot Of yr_renovated
2000
1750
1500
1250
1000
750
500
250
0
yr_renovated


Box Plot Of zipcode
98200
98175
98150
98125
98100
98075
98050
98025
98000
zipcode



Plot 1- Boxplot

Observation:

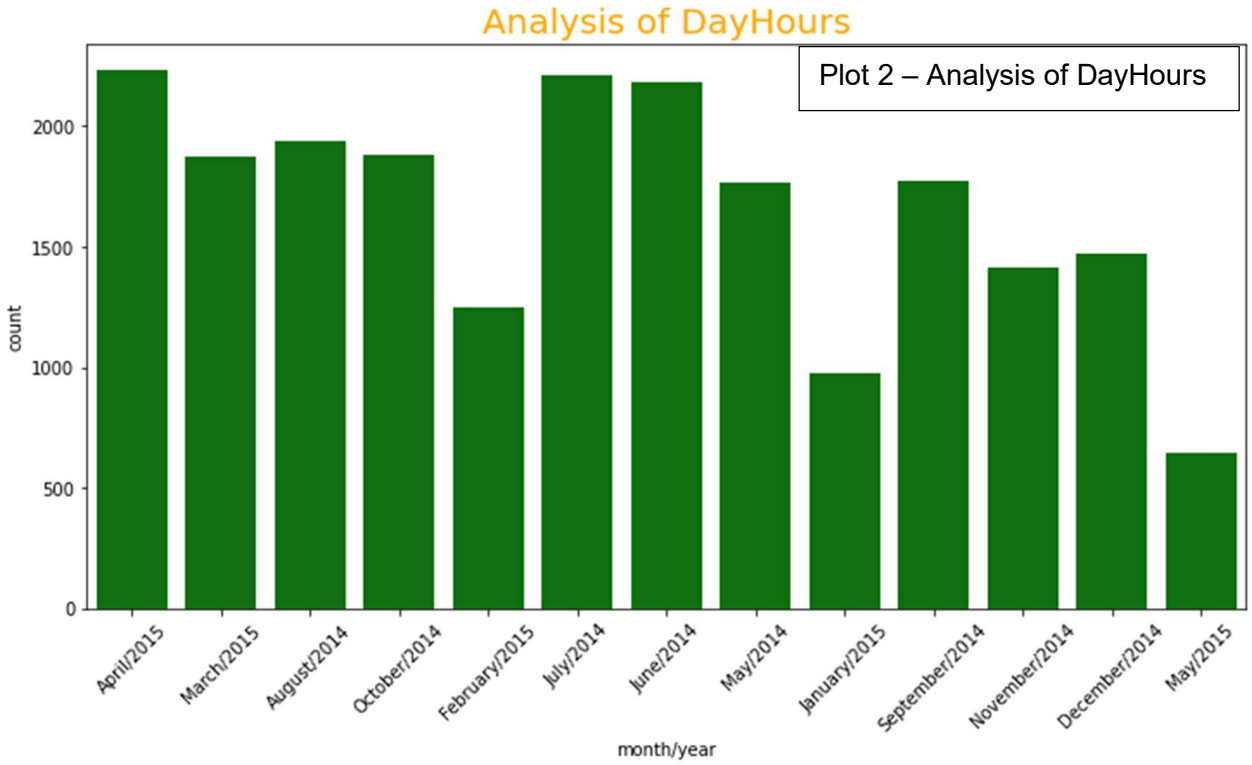
There are lots of feature which are having outliers**.**

Uni-Variate Analysis Of Each Column

Analysis of Dayhours

Observation:

- We can see from the graph that in april 2015 and june and july of 2014 most houses are sold.



Plot 2 – Analysis of DayHours

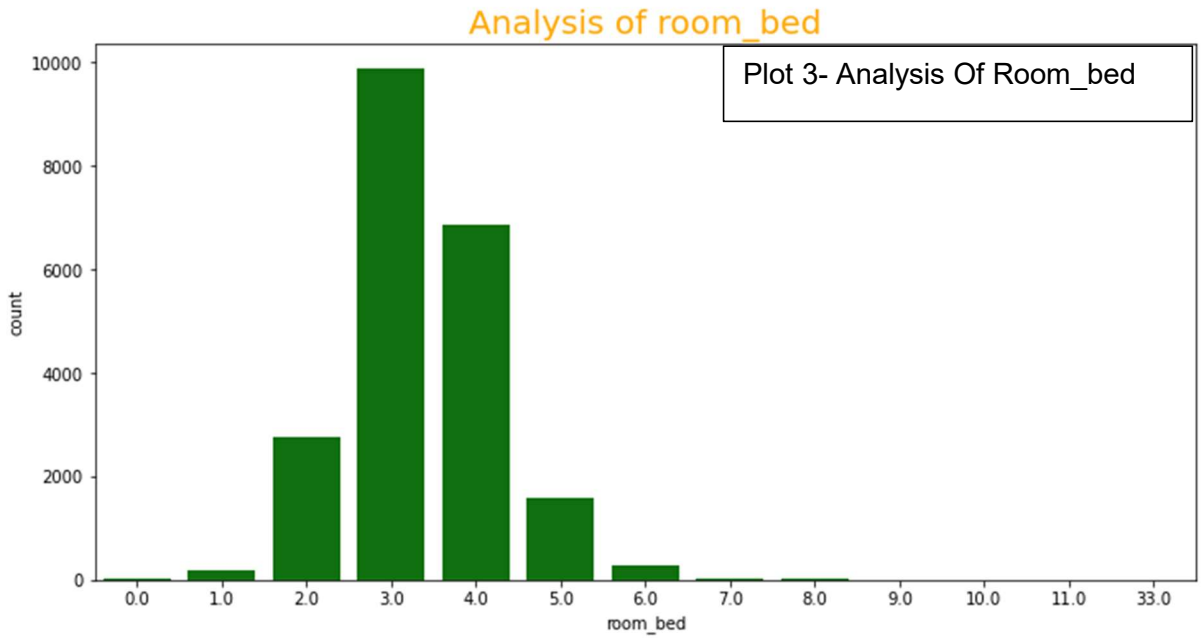
- In the dataset, we can see the sales is high in the month of April/2015.

Analysis of room_bed:

Analysis of room_bed:

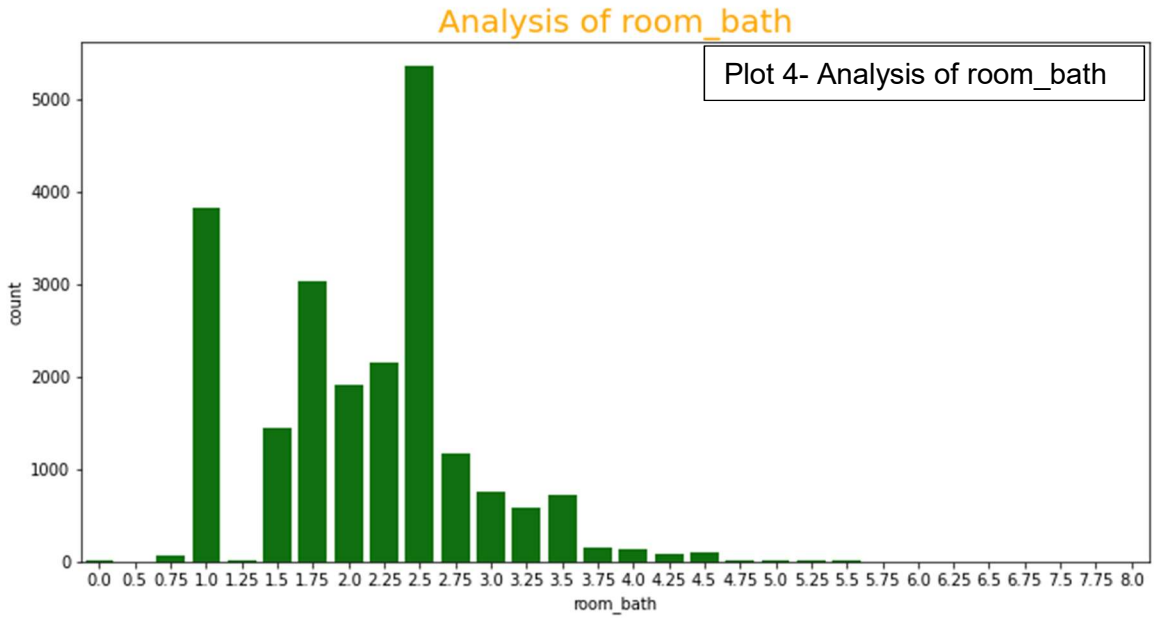
Observation:

- Most of the houses have 3 or 4 bedrooms



Plot 3- Analysis Of Room_bed

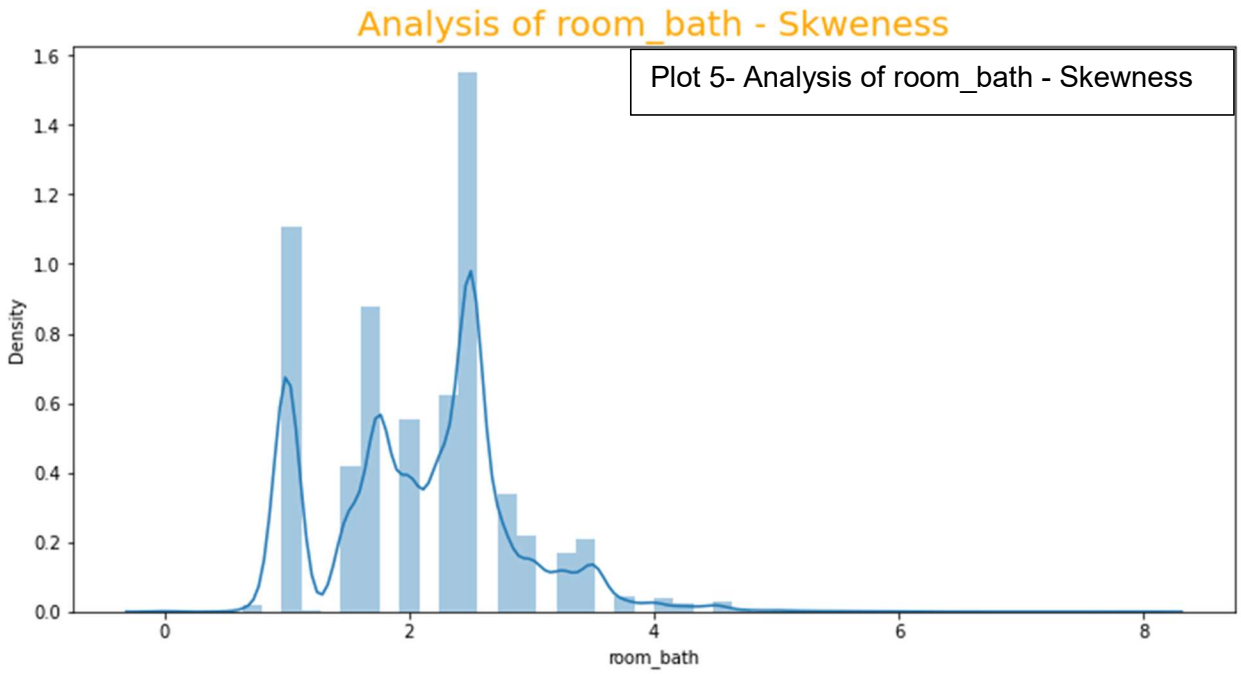
Analysis of room_bath:



Plot 4- Analysis of room_bath

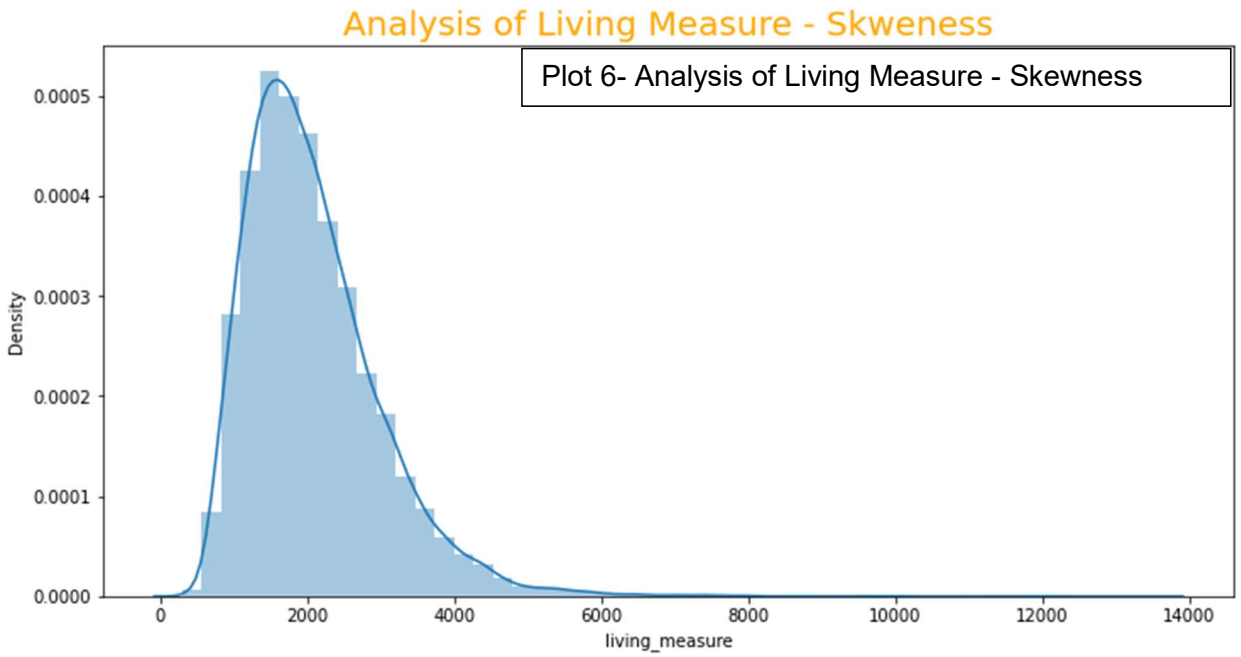
Observation:

- Majority of the houses have bathroom in the range of 1.0 to 2.5



Plot 5- Analysis of room_bath - Skewness

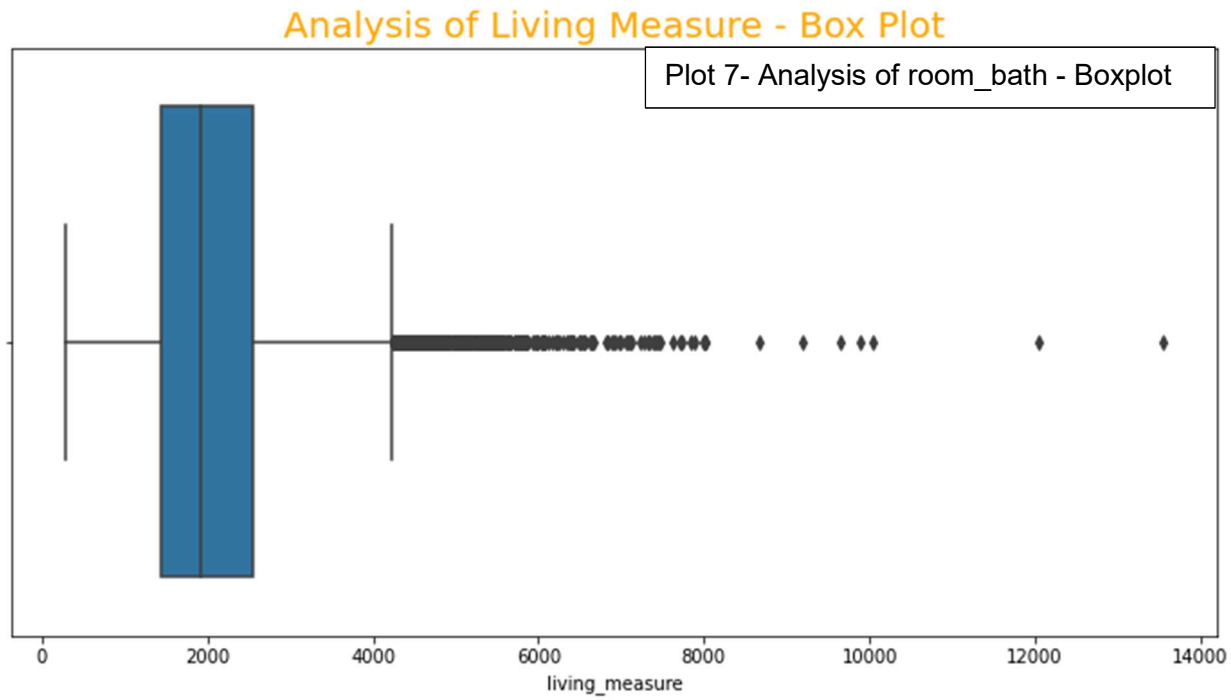
Analysis of Living Measure



Plot 6- Analysis of Living Measure - Skewness

Observation:

- Data is rightly skewed

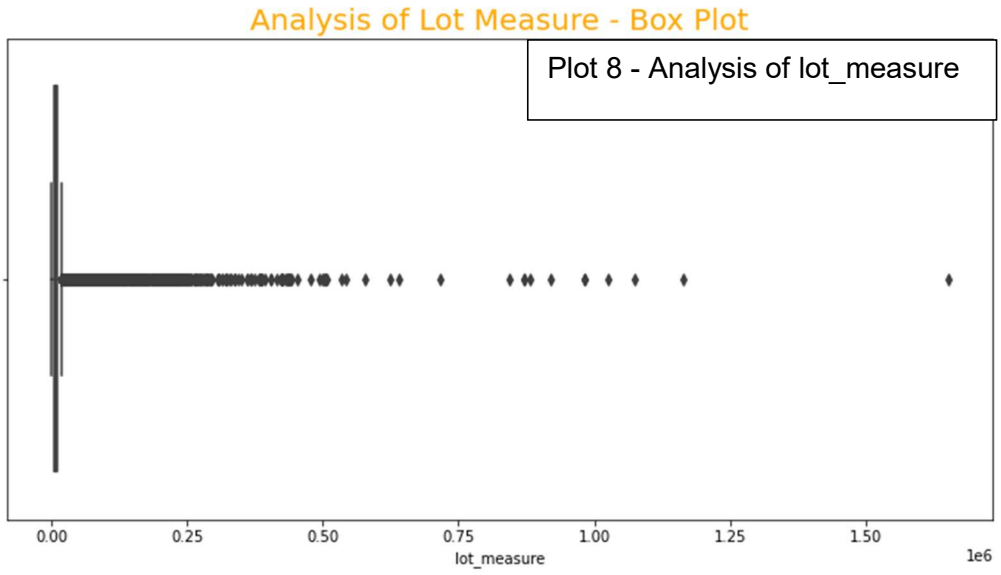


Plot 7- Analysis of room_bath - Boxplot

Observation:

- There are many outliers in living measure. We will handle them later in outlier removal.

Analysis of lot_measure

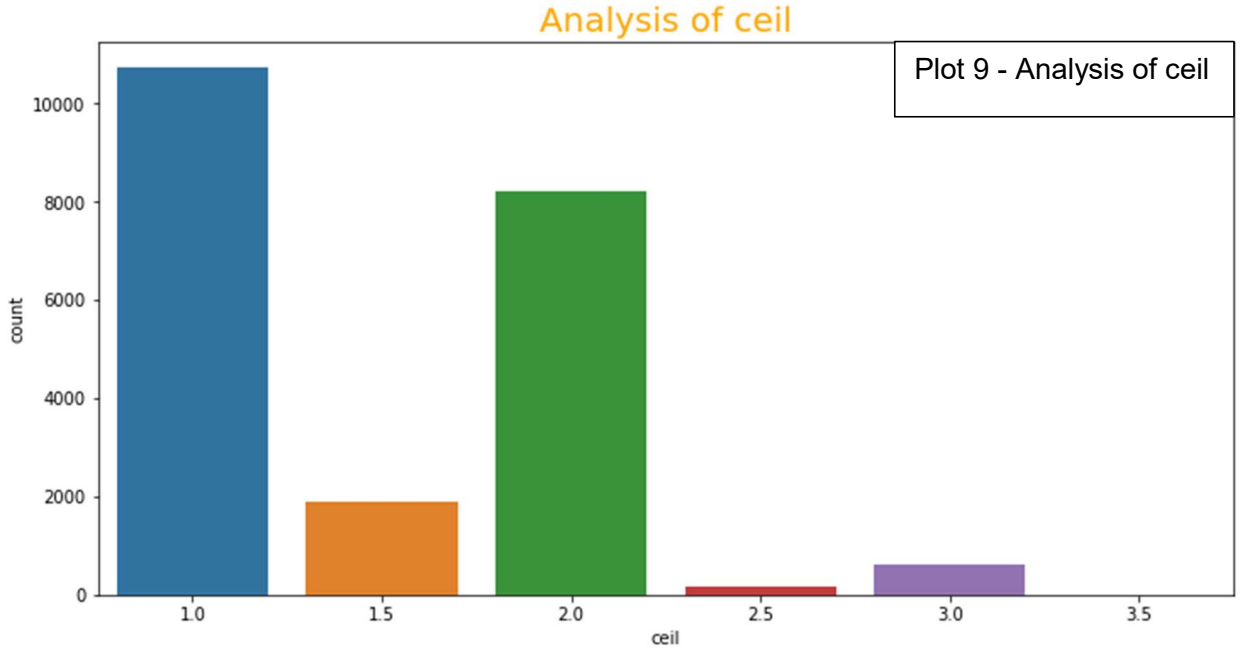


Plot 8 - Analysis of lot_measure

Observation:

- Data is skewed as visible from plot

Analysis of Ceil:

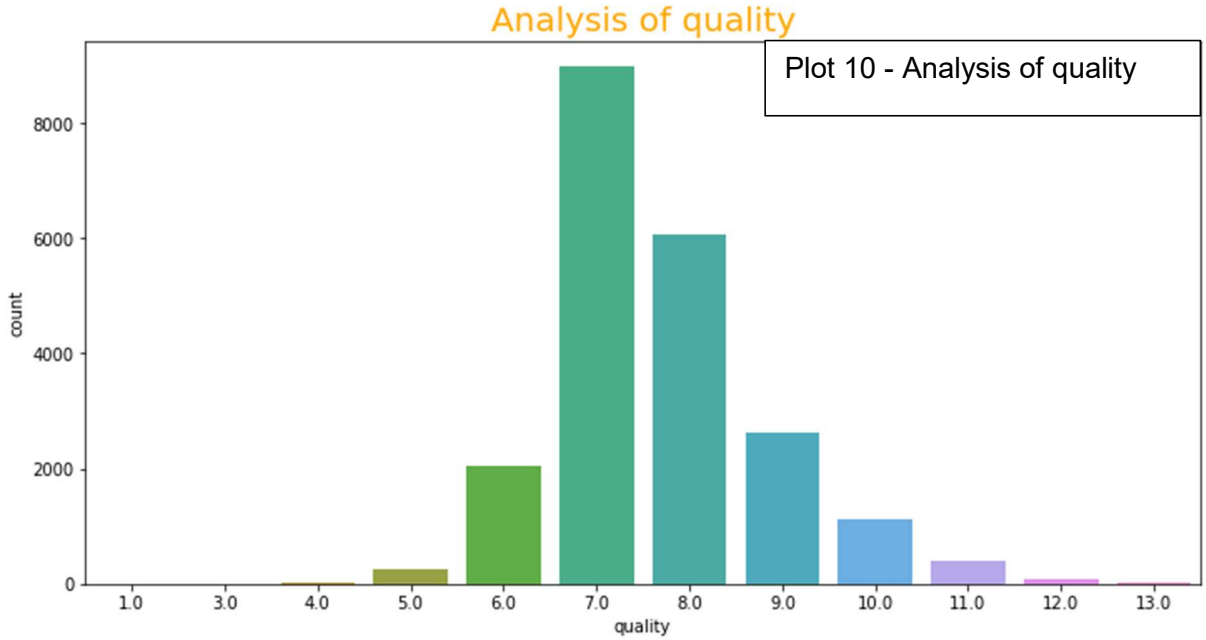


Plot 9 - Analysis of ceil

Observation:

- Most houses have 1 and 2 floors

Analysis of Quality:

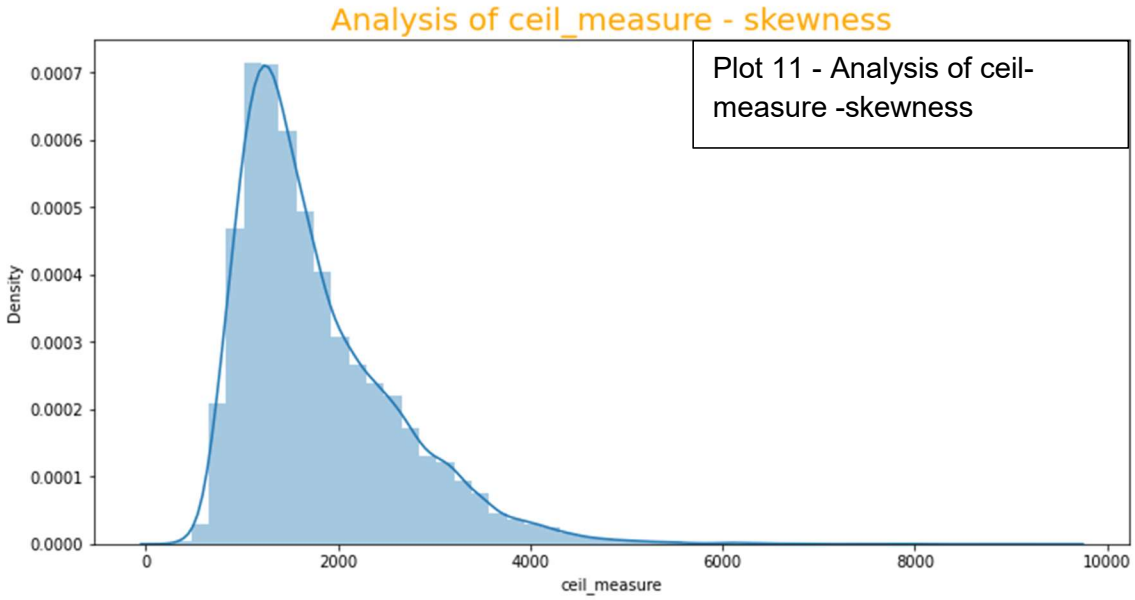


Plot 10 - Analysis of quality

Observation:

- Most properties have quality rating between 6 to 10

Analysis of ceil_measure

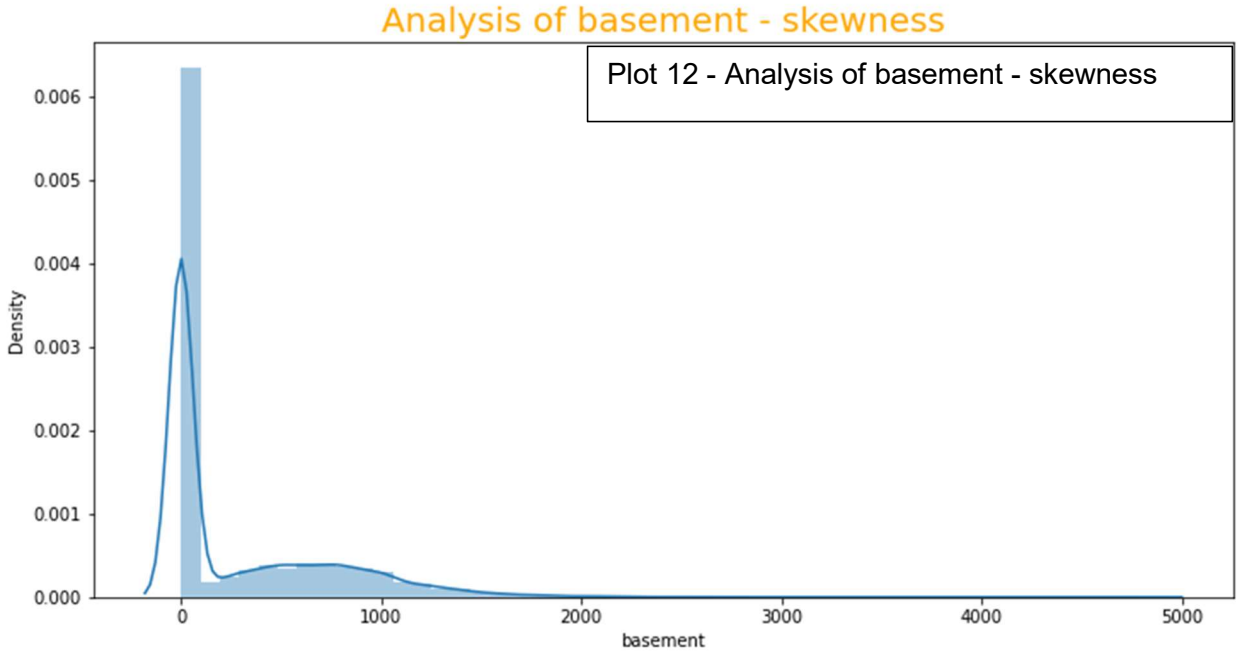


Plot 11 - Analysis of ceil-measure -skewness

Observation:

- There is no pattern in Ceil Vs Ceil_measure and is highly skewed

Analysis of basement



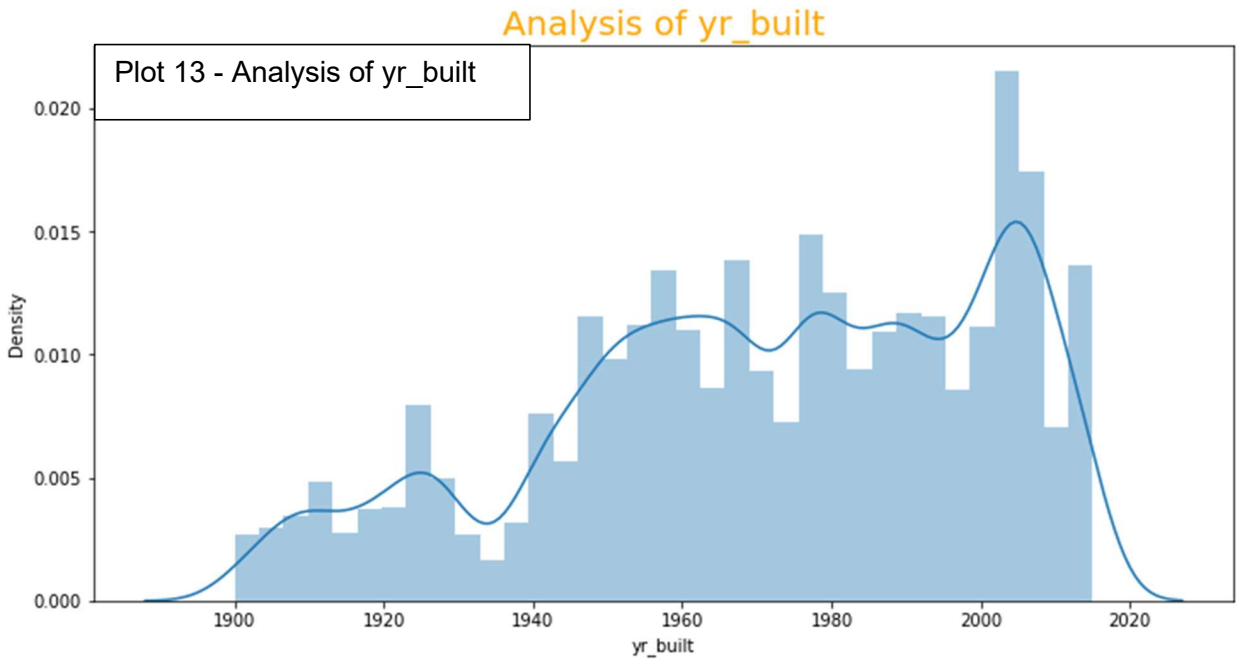
Observation:

- We can see 2 gaussians, which tells us there are propeties which don't have basements and some have the basements

Analysis of yr_built:

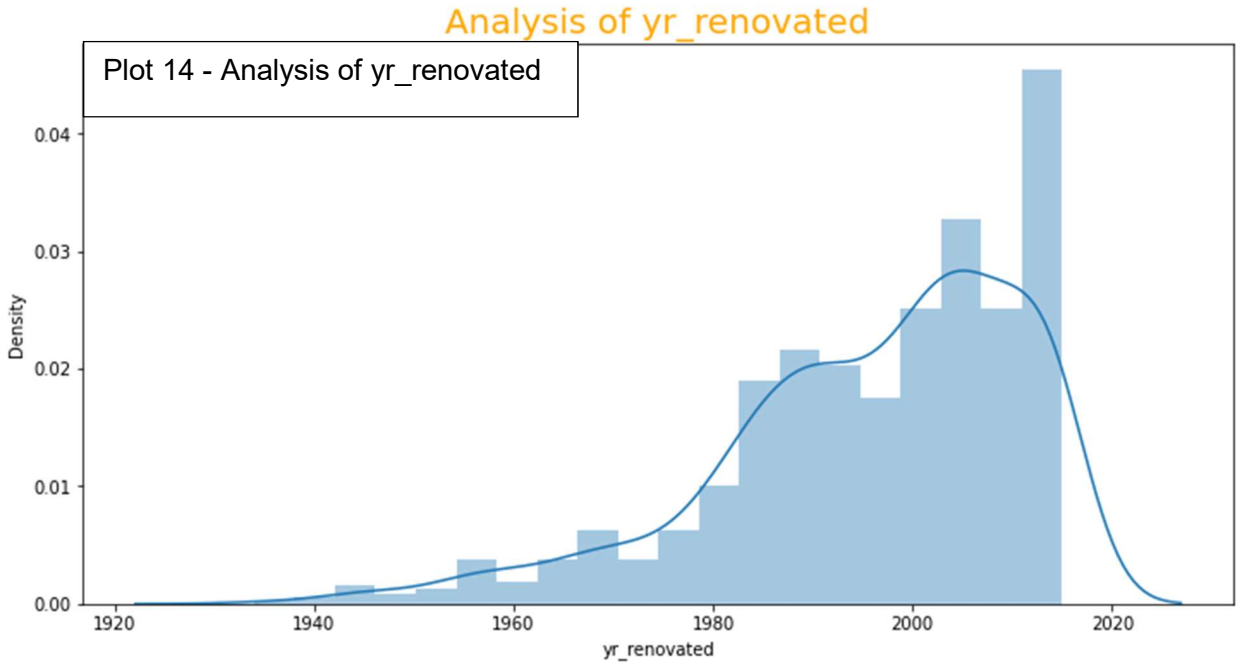
Observation:

The built year of the properties range from 1900 to 2014 and we can see upward trend with time



Plot 13 - Analysis of yr_built

Analysis of yr_renovated

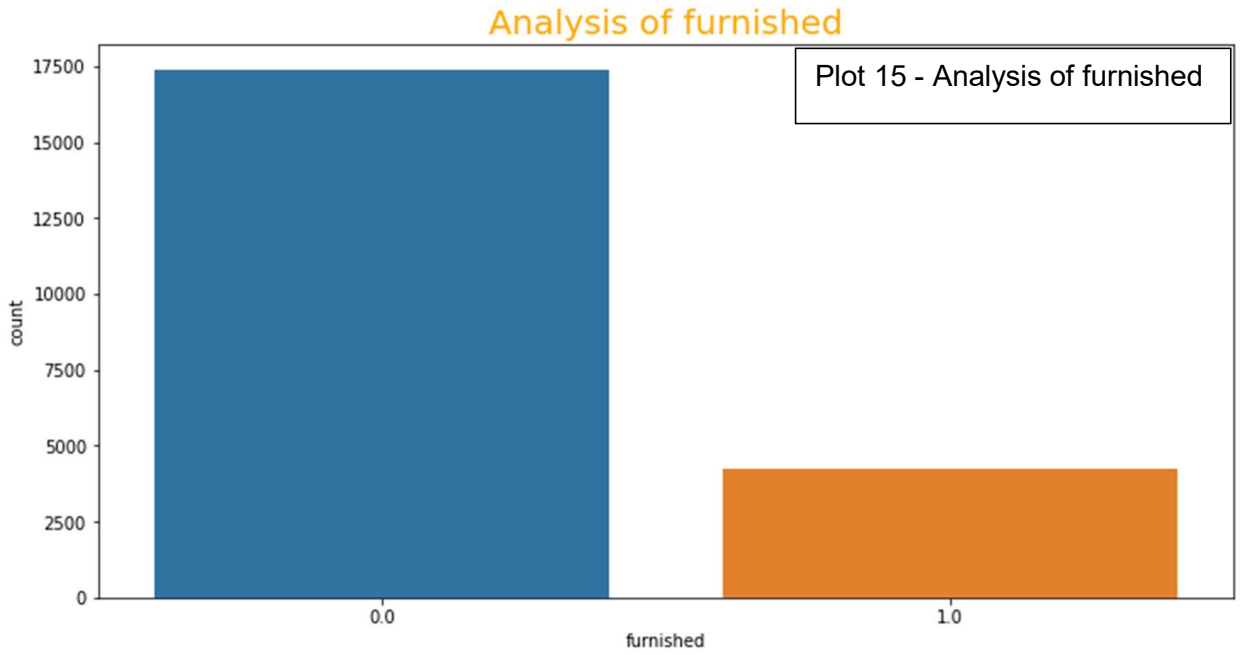


Plot 14 - Analysis of yr_renovated

Observation:

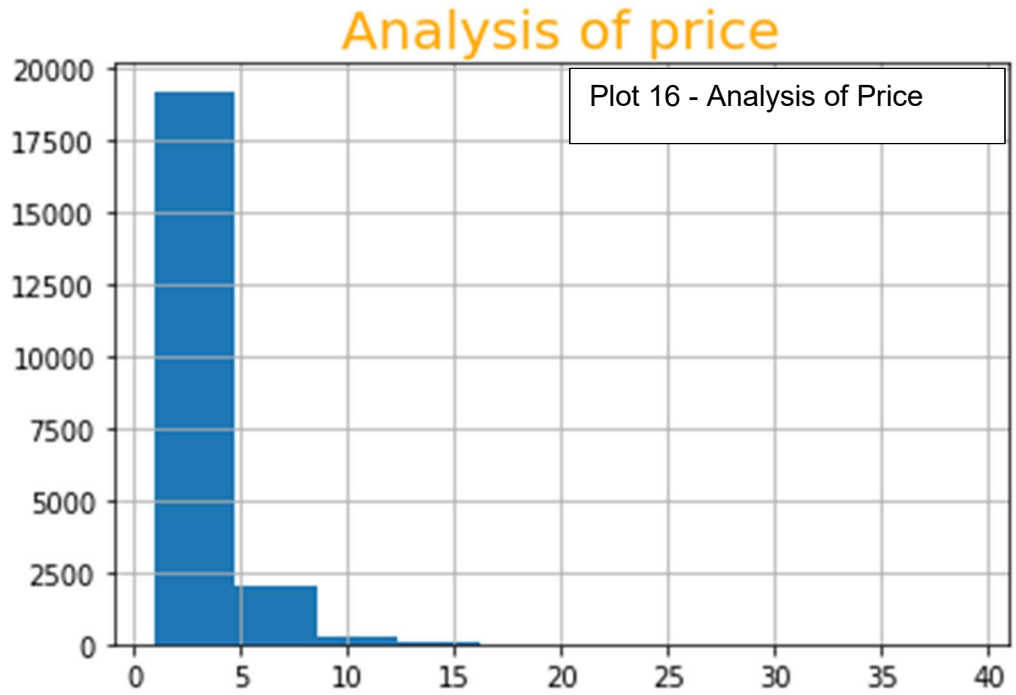
- We can see the data is skewed in left side there must be oulier in it.

Analysis of furnished



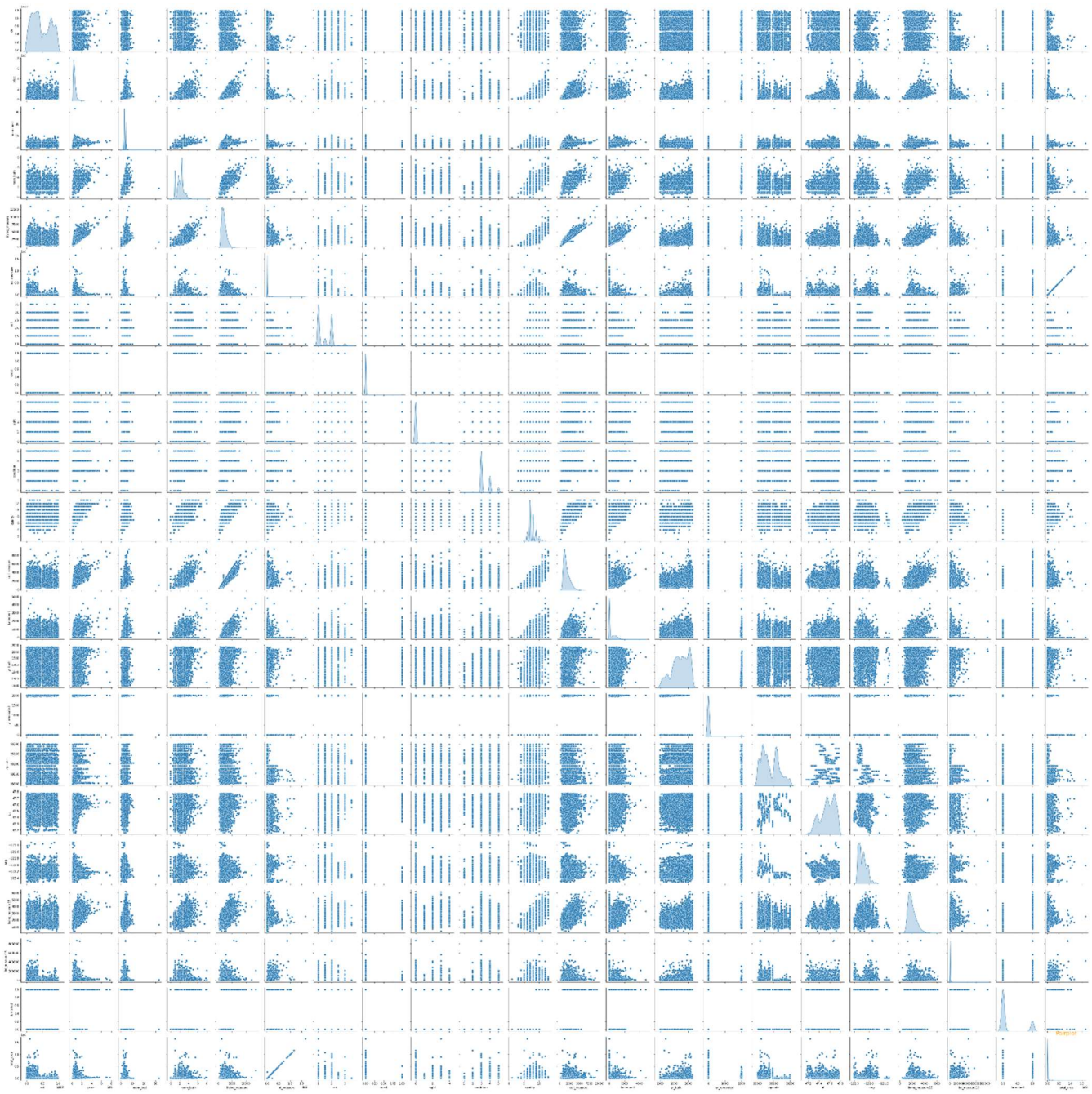
Plot 15 - Analysis of furnished

Analysis of Price



Plot 16 - Analysis of Price

Bivariant Analysis- PairPlot



Plot 17 - PairPlot

Observation:

price: price distribution is Right-Skewed as we deduced earlier from our 5-factor analysis
room_bed: our target variable (price) and room_bed plot is not linear. It's distribution have lot of gaussians
room_bath: It's plot with price has somewhat linear relationship. Distribution has number of gaussians.
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.
lot_measure: No clear relationship with price.
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.
coast: No clear relationship with price. Clearly it's categorical variable with 2 unique values.
sight: No clear relationship with price. This has 5 unique values. Can be converted to Categorical variable.
condition: No clear relationship with price. This has 5 unique values. Can be converted to Categorical variable.
quality: Somewhat linear relationship with price. Has discrete values from 1 - 13. Can be converted to Categorical variable.
ceil_measure: Strong linear relationship with price. Also with room_bath and living_measure features. Distribution is Right-Skewed.
basement: No clear relationship with price.
living_measure15: Somewhat linear relationship with target feature. It's same as living_measure. Therefore we can drop this variable.
lot_measure15: No clear relationship with price or any other feature.

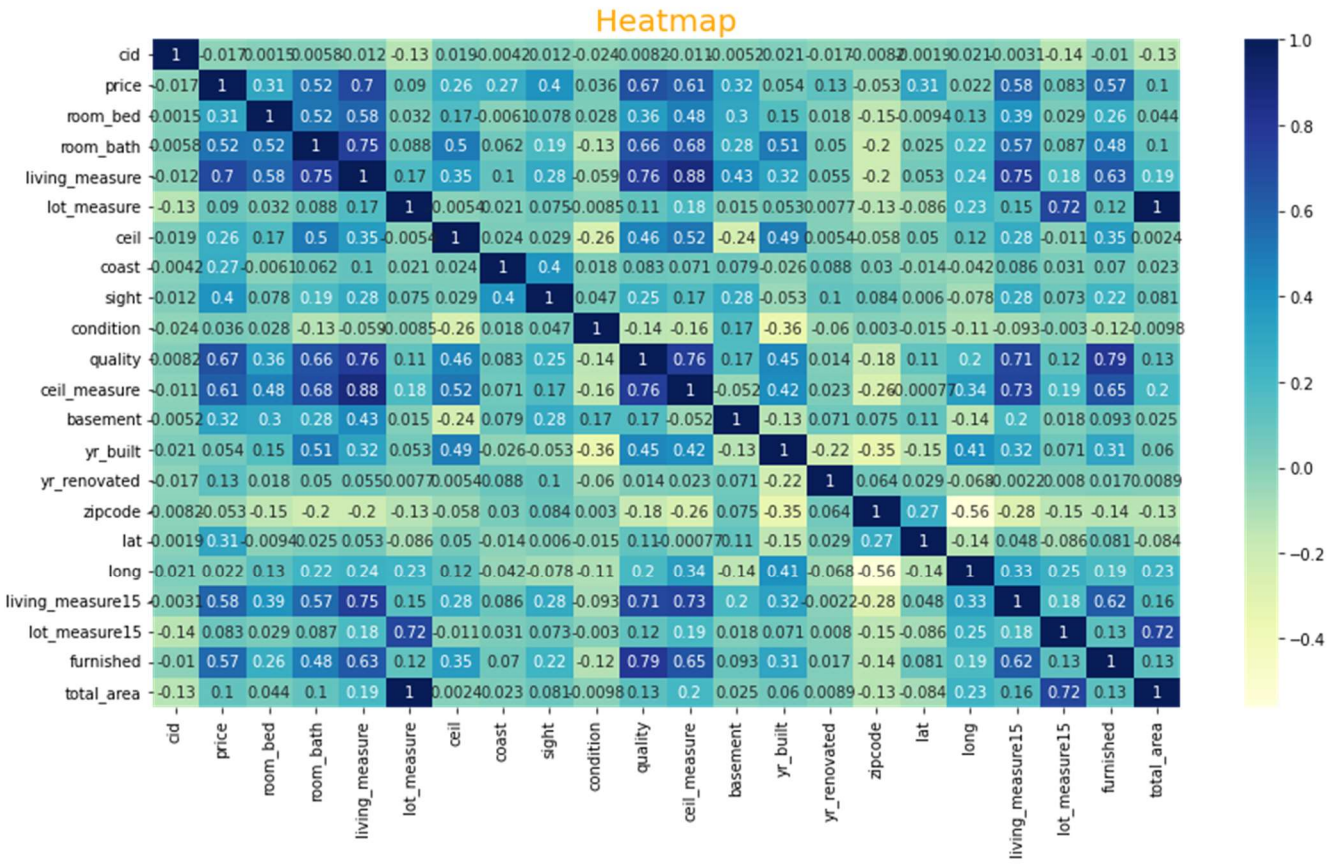
Bivariant Analysis- Heatmap

Here we are looking which feature can be dropped according to correlation between them.

On left the name of the column is written and on right the columns which are highly correlated with it is written.

price: room_bath, living_measure, quality, living_measure15, furnished

Plot 18 - Heatmap



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

quality: price, room_bath, living_measure

ceil_measure: price, room_bath, living_measure, quality

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

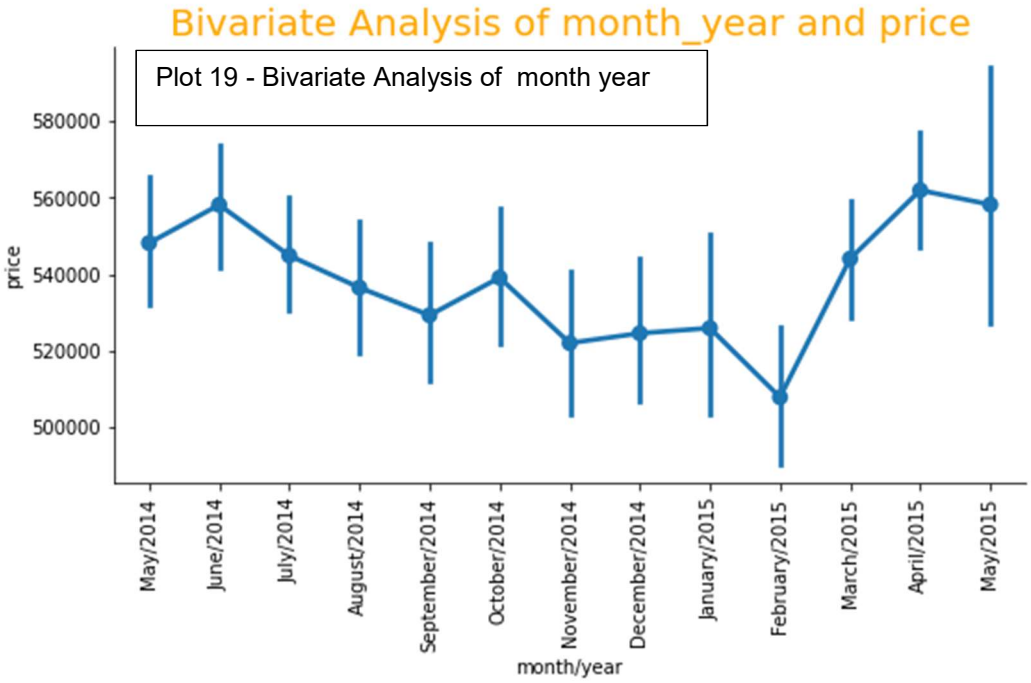
lot_measure15: lot_measure. Therefore, we can consider dropping lot_measure15, as it's giving same info.

furnished: quality

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.

Bivariant Analysis of Each Column with Price

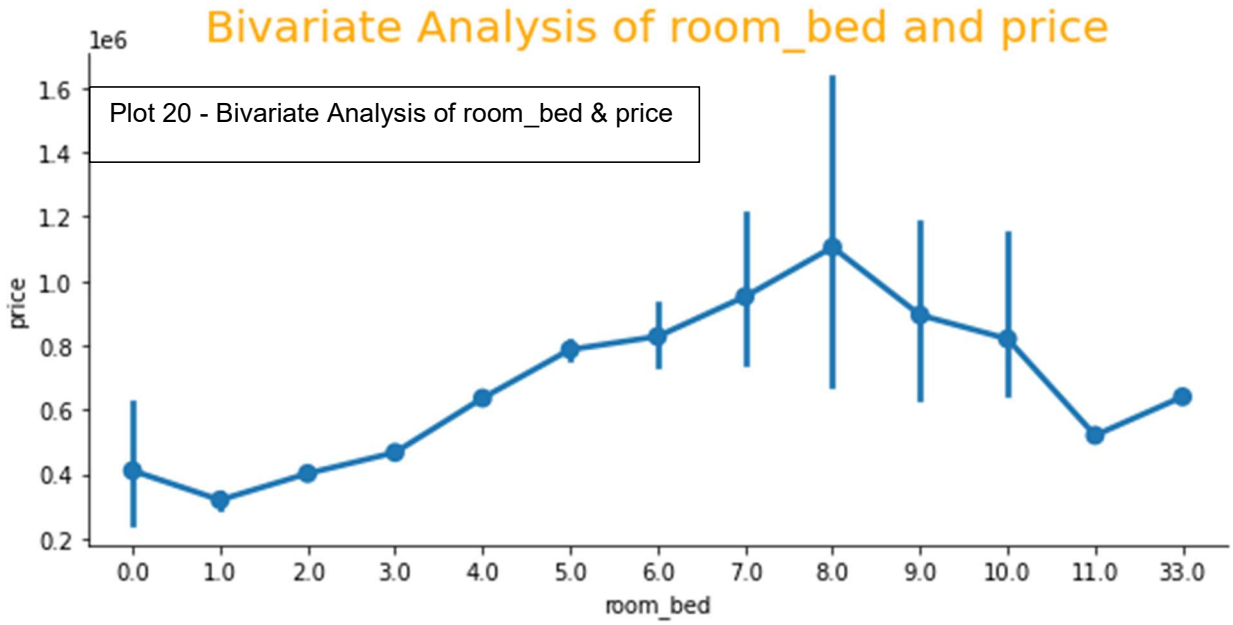
Bivariate Analysis of month year



Observation:

The mean price of the houses tend to be high during March,April, May as compared to that of September, October, November,December period.

Bivariate Analysis of room_bed

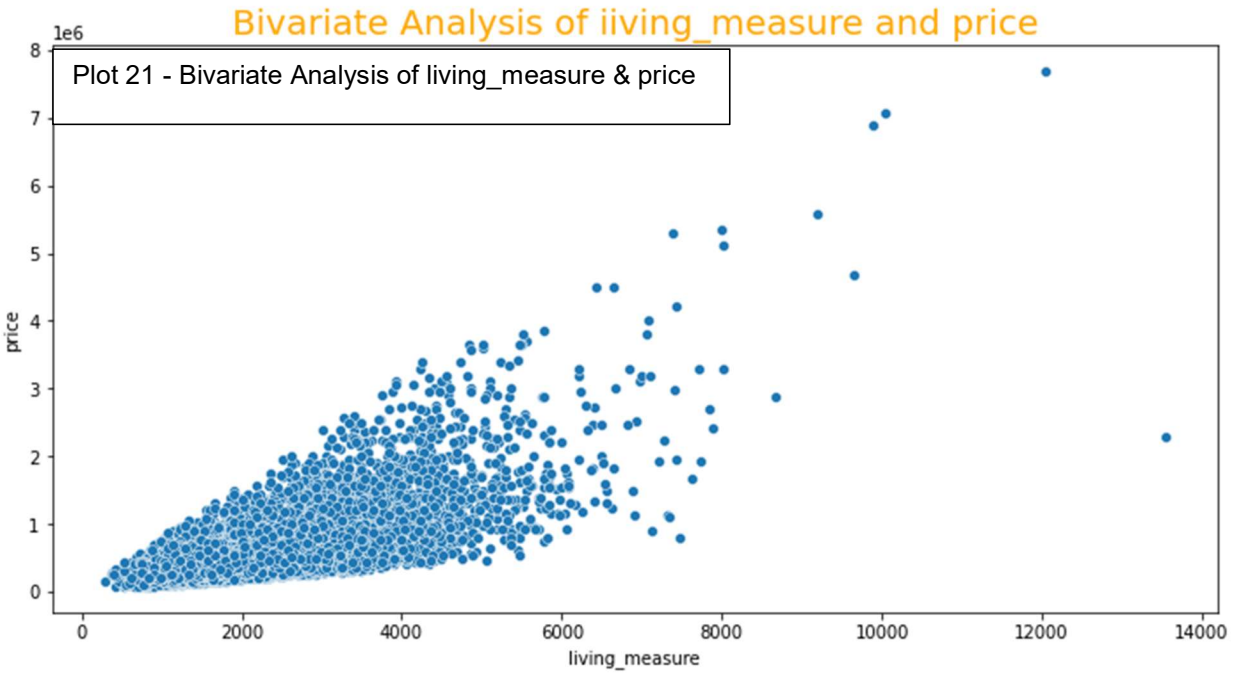


Plot 20 - Bivariate Analysis of room_bed & price

Observation:

- There is clear increasing trend in price with room_bed

Bivariate Analysis of living_measure

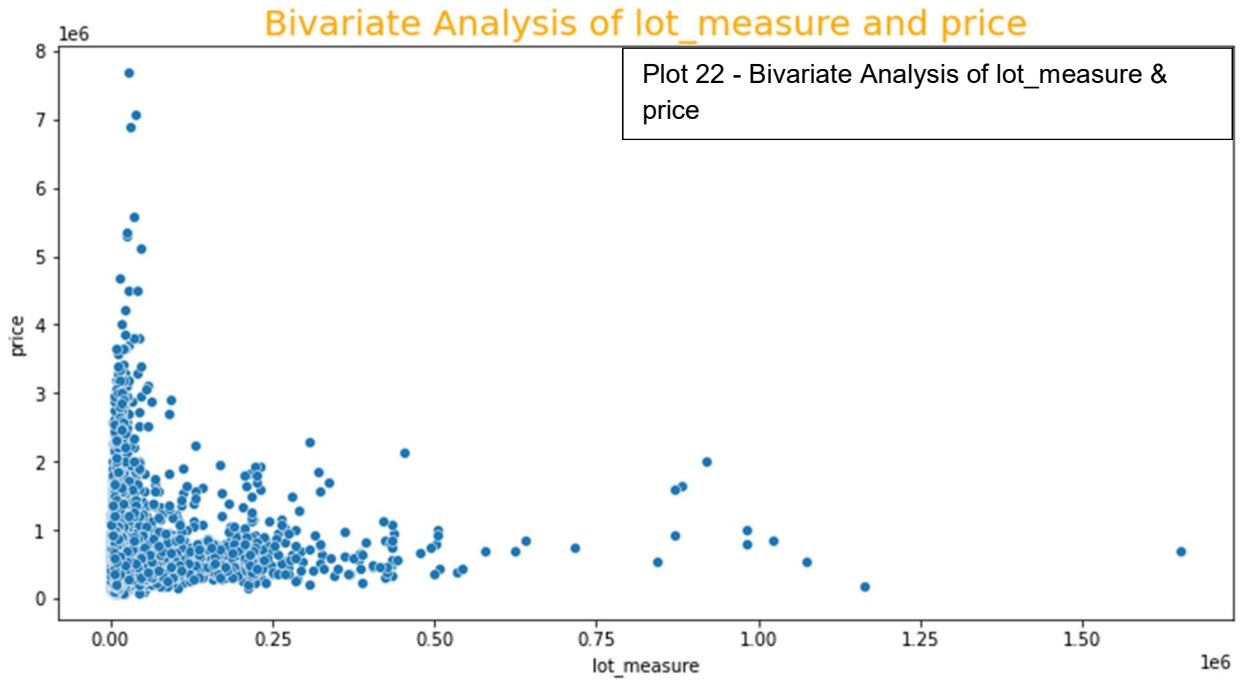


Plot 21 - Bivariate Analysis of living_measure & price

Observation:

- There is clear increment in price of the property with increment in the living measure But there seems to be one outlier to this trend. Need to evaluate the same

Bivariate Analysis of lot_measure

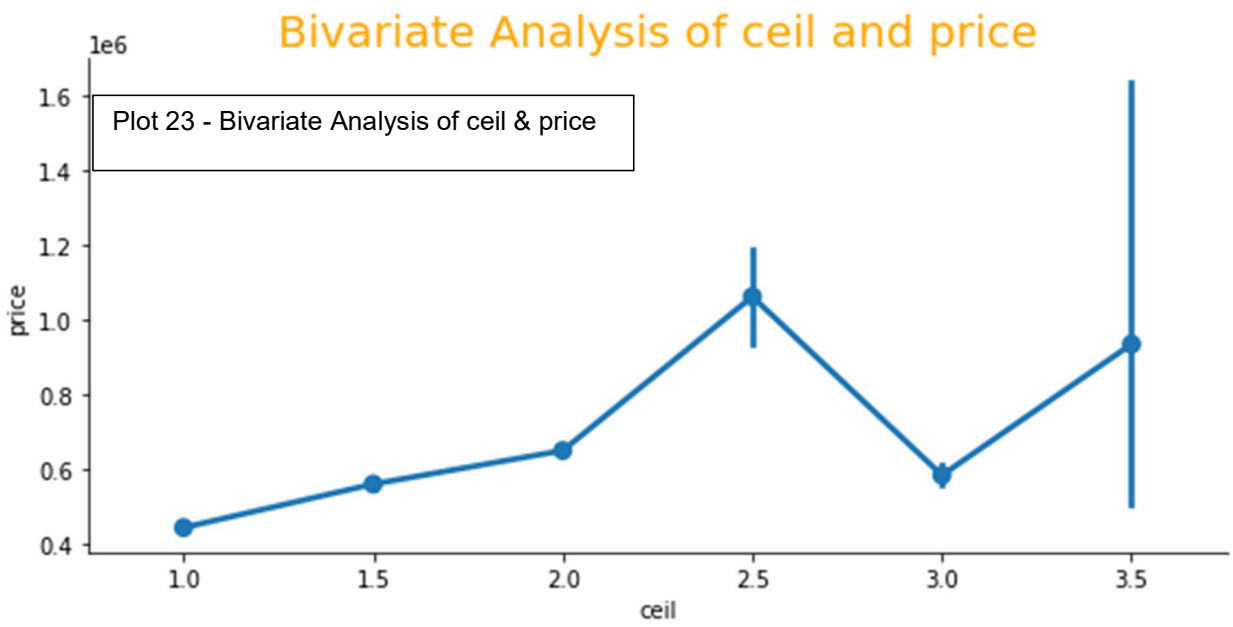


Plot 22 - Bivariate Analysis of lot_measure & price

Observation:

- There seems to be no relation between lot_measure and price.
- Data value range is very large so breaking it get better view.
- There doesnt seem to be no relation between lot_measure and price trend

Bivariate Analysis of ceil



Plot 23 - Bivariate Analysis of ceil & price

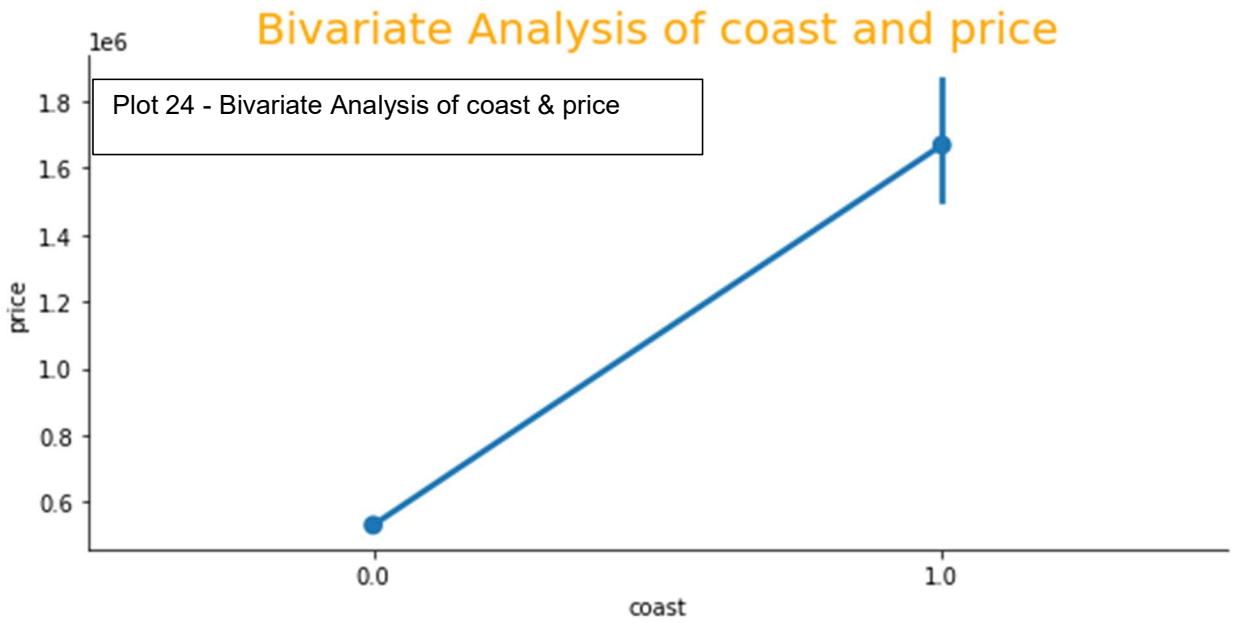
Observation:

- Median price increases initially and then fall
- There is some slight upward trend in price with the ceil

Bivariate Analysis of coast

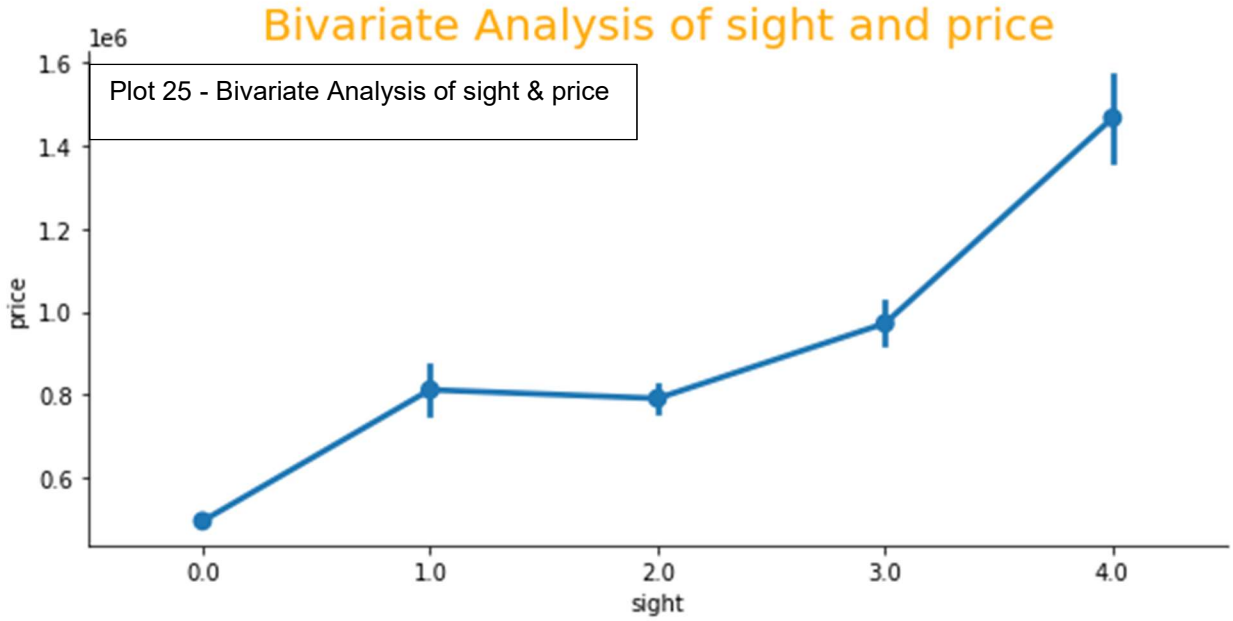
Observation:

- Mean and median of waterfront view is high however such houses are very small in compare to non-waterfront
- 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



Plot 24 - Bivariate Analysis of coast & price

Bivariate Analysis of sight and price

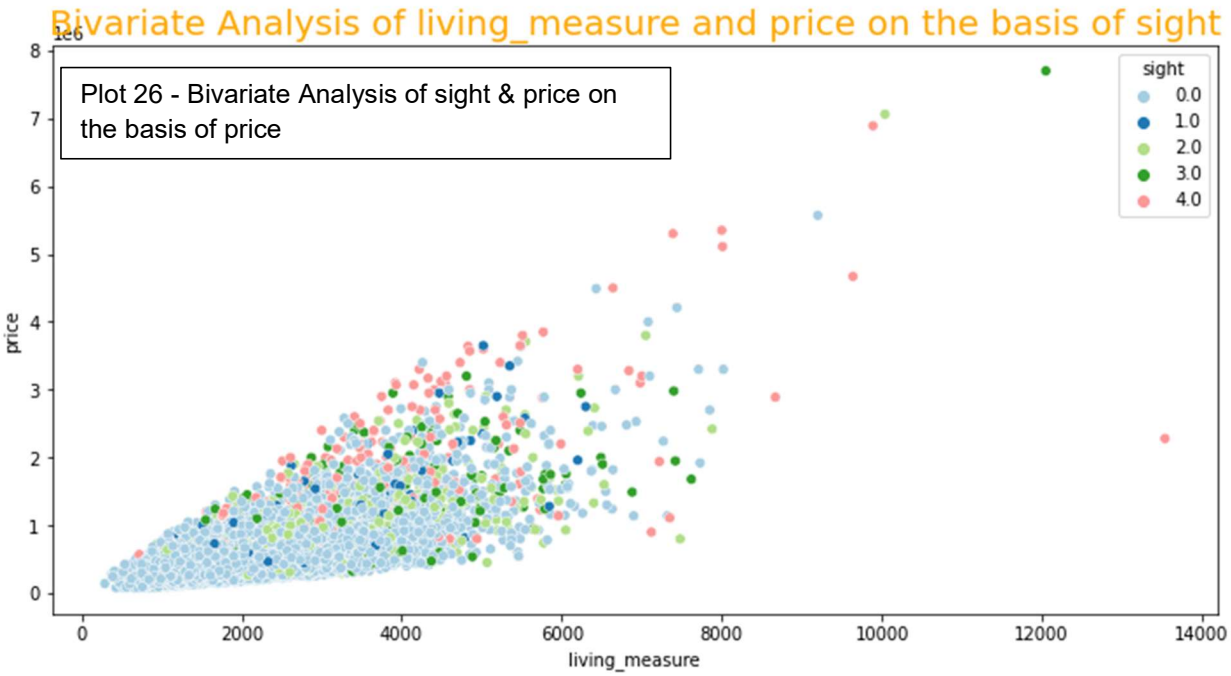


Plot 25 - Bivariate Analysis of sight & price

Observation:

- It contains 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

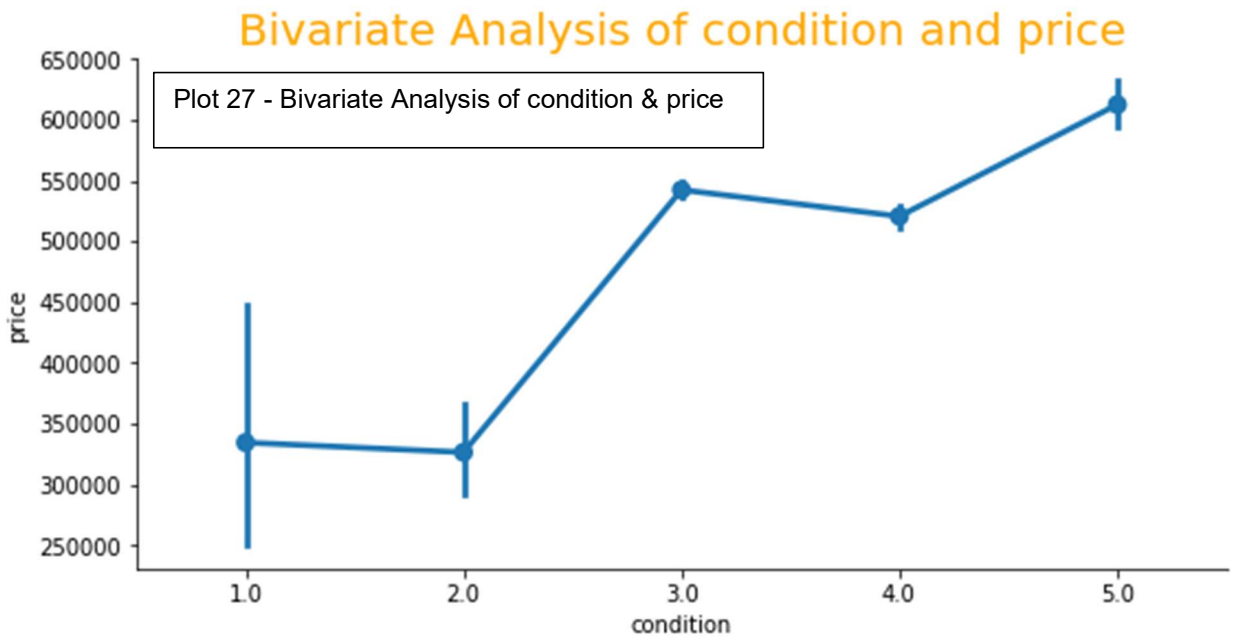


Plot 26 - Bivariate Analysis of sight & price on the basis of price

Observation:

- Viewed in relation with price and living_measure
- Costlier houses with large living area are sighted more.
- The above graph also justify that: Properties with higher price have more no.of sights compared to that of houses with lower price

Bivariate Analysis of condition

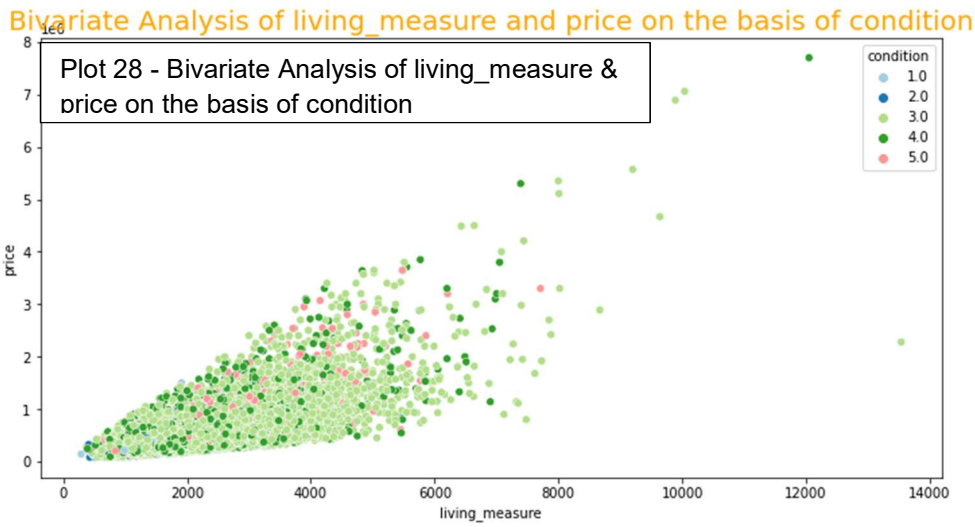


Plot 27 - Bivariate Analysis of condition & price

Observation:

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



Plot 28 - Bivariate Analysis of living_measure & price on the basis of condition

Observation:

- Viewed in relation with price and living_measure. Most houses are rated as 3 or more.
- We can see some outliers as well
- So we found out that smaller houses are in better condition and better condition houses are having higher prices

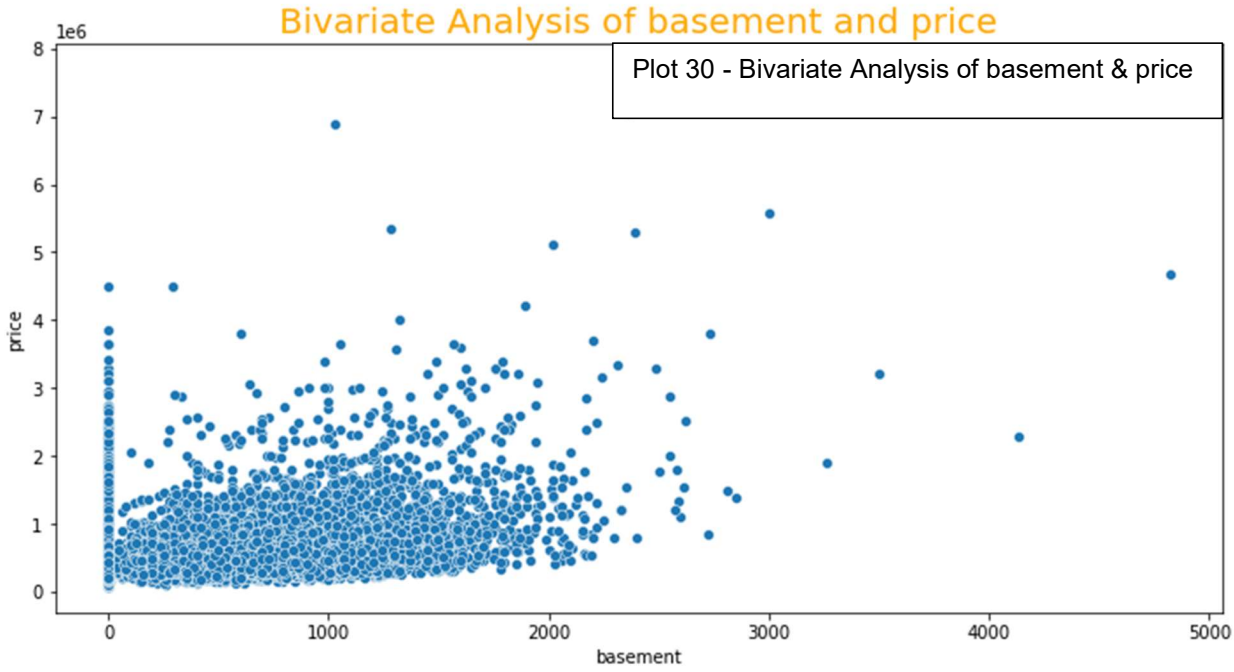
Bivariate Analysis of quality

Plot 29 - Bivariate Analysis of quality & price

Observation:

- 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

Bivariate Analysis of basement



Plot 30 - Bivariate Analysis of basement & price

Observation:

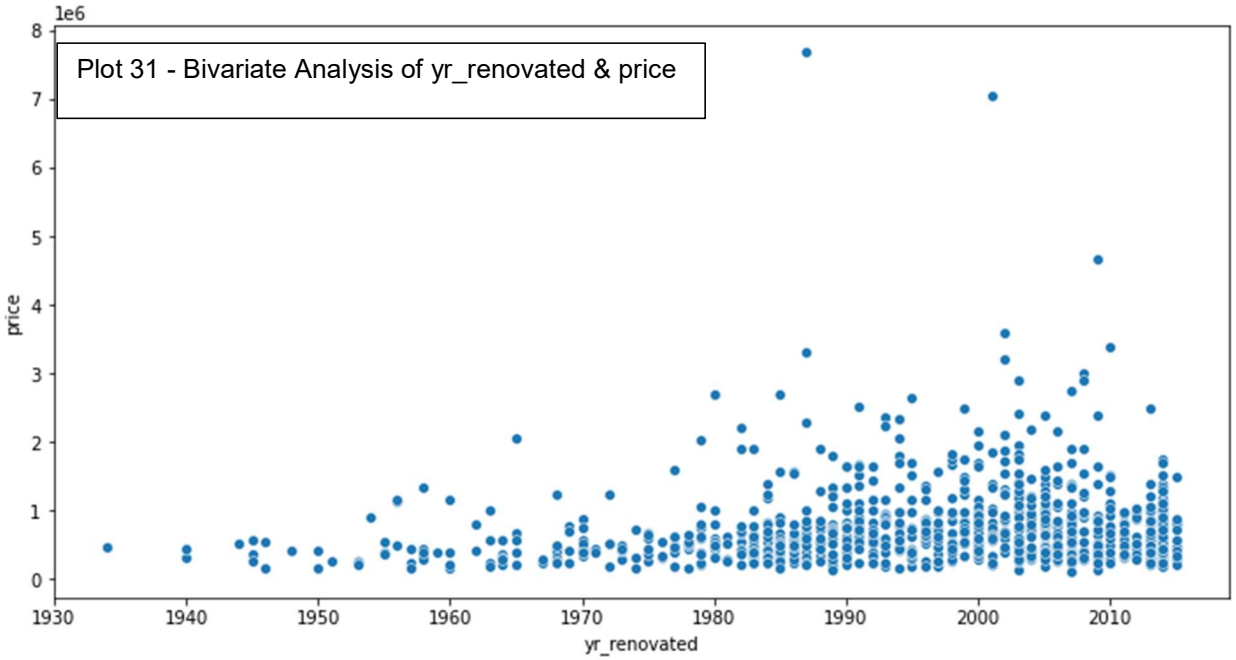
- We will create the categorical variable for basement 'has_basement' for houses with basement and no basement.This categorical variable will be used for further analysis.
- Price increases with increase in ceil measure

Bivariate Analysis of yr_renovated

Observation:

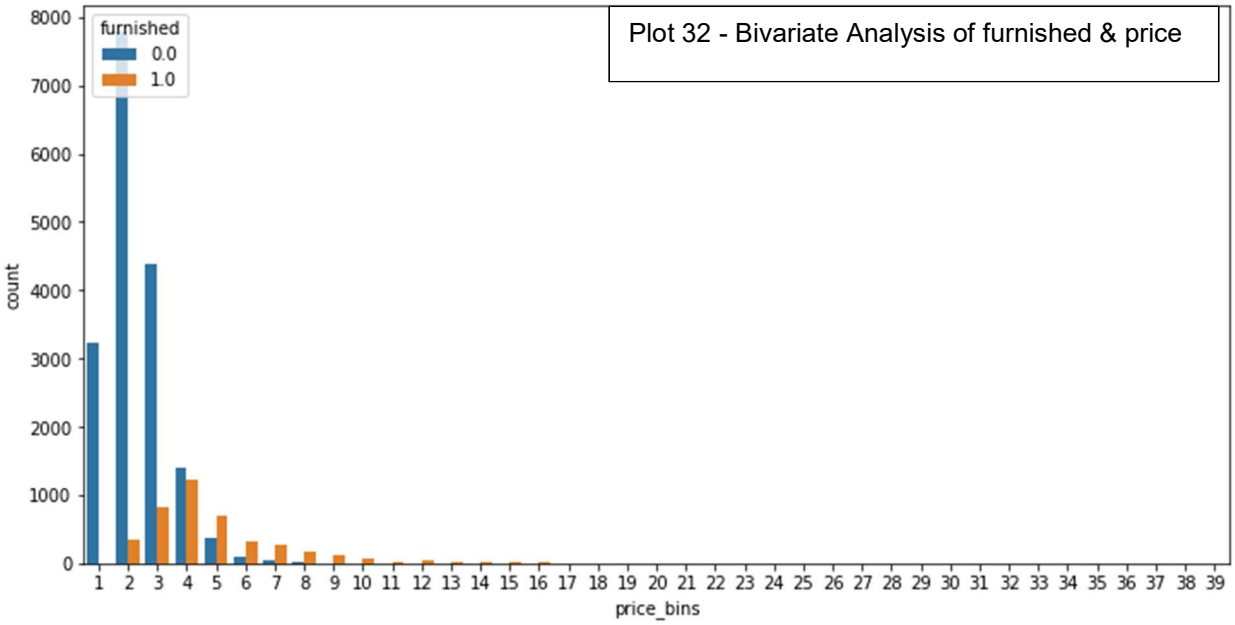
- So most houses are renovated after 1980's. We will create new categorical variable 'has_renovated' to categorize the property as renovated and non-

renovated. For further ananlysis we will use this categorical variable.



Plot 31 - Bivariate Analysis of yr_renovated & price

Bivariate Analysis of furnished



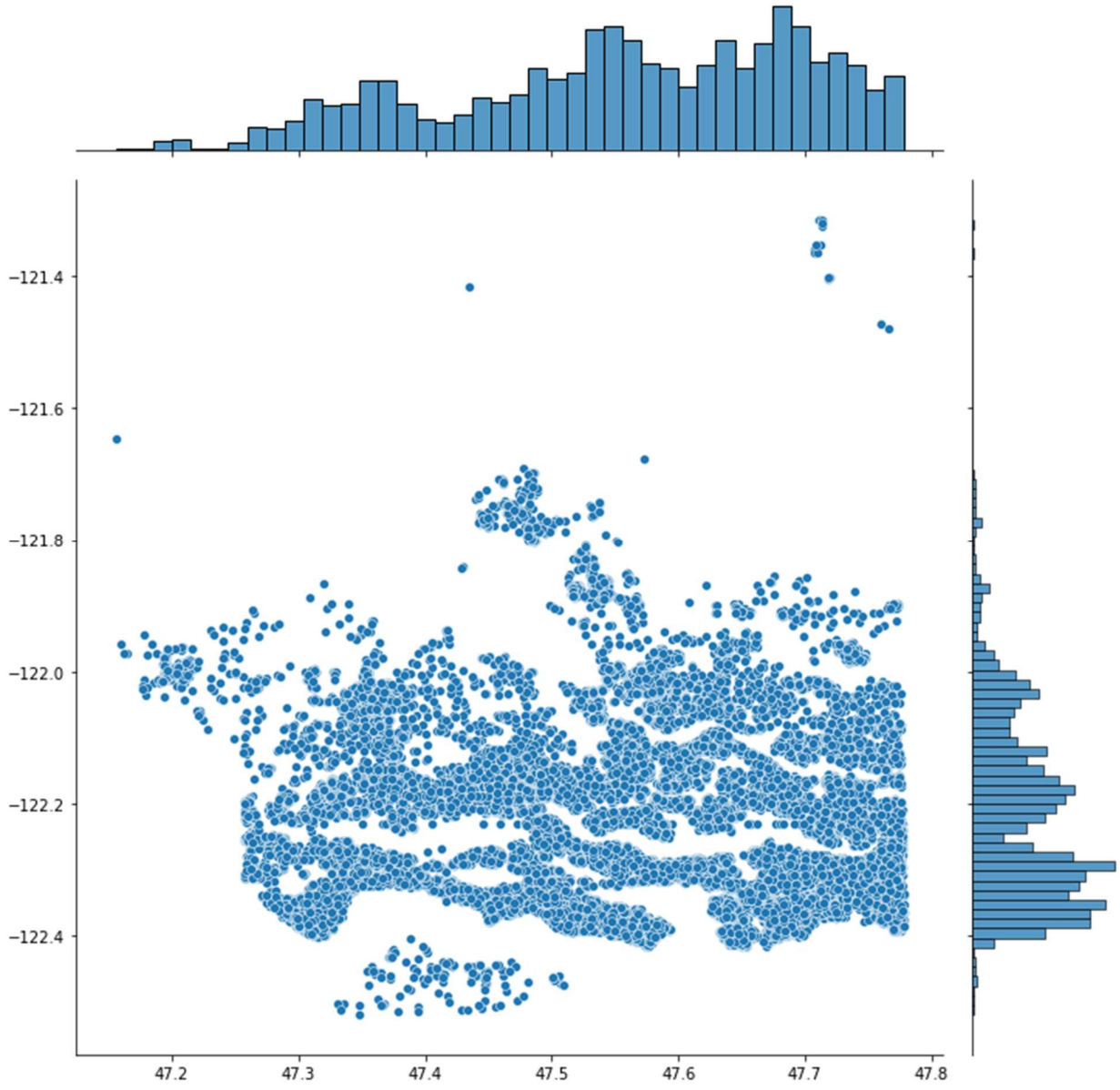
Plot 32 - Bivariate Analysis of furnished & price

Observation:

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

Visualizing the location of the houses based on latitude and longitude

Plot 33 – Visualizing the location of the houses



Observation:

We can see that for latitude between -47.3 and 47.8 and for longitude between -122.0 to -122.4 there are many houses

3. Data Cleaning & Pre-processing

Data Cleaning is an important phase in any data science project, if our data is clean then only we can provide it to our machine learning model. Uncleaned Data can further lead our model with low accuracy. And, If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset.

The approach used for identifying and treating missing values and outlier treatment:-

```
In [11]: print(start+'Checking for Null values in the dataframe:'+end,'\n',data.isnull().sum(),'\n')

Checking for Null values in the dataframe:
 cid                0
dayhours            0
price               0
room_bed          108
room_bath         108
living_measure     17
lot_measure        42
ceil               72
coast              31
sight              57
condition          85
quality             1
ceil_measure        1
basement            1
yr_built           15
yr_renovated        0
zipcode             0
lat                 0
long               34
living_measure15  166
lot_measure15      29
furnished          29
total_area         68
dtype: int64
```

Code Sample 1- Null Values

To identify any missing values in our data set we have used **Pandas** pre built function **isnull()** to detect any missing values in our datasets. As we can see that column living_measure15 has a high number of missing values. So our next step is how to handle a large number of missing values. One approach is, that we will delete the column if we don't need that column for further analysis. And, what if we need that column for further analysis then we have use an approach will is a predefined function in **Pandas** called fillna().

Filling the missing value in categorical variable using Mode

```
In [13]: for i in ['furnished','quality', 'condition', 'coast', 'ceil', 'sight', 'yr_renovated']:
             data[i] = data[i].fillna(data[i].mode()[0])
```

Filling the missing value in numerical variable using Median

```
In [14]: for i in data.drop(['cid','dayhours','furnished','quality', 'condition', 'coast', 'ceil', 'sight', 'yr_renovated'],axis=1).columr
             data[i] = data[i].fillna(data[i].median())

         df = data.copy()
```

Code Sample 2- Missing Values

As you can have a look, How we have filled the missing values in a **categorical** variable using **mode**. And, How we have filled the missing values in a **numerical** variable using **Median**. This is how we have an approach for identifying and handling missing values.

While performing Preprocessing and Data cleaning we have to also deal with outliers. Dealing with outliers is also a necessary step to be taken for further analysis and model building. Outliers are data points in a data set that is distant from all other observations. A data point that lies outside the overall distribution of the dataset

Treating Outlilers

We have seen outliers for columns room_bath(33 bed), living_measure, lot_measure, ceil_measure and Basement

```
In [95]: def outlier_removal(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
```

Code Sample 2- Outlier Detection

Using the above function, lets get the lowerbound and upperbound values

After performing Uni-variate, Bi-variate, and Multi-variate analysis we can get an idea about outliers present in our dataset. We have seen outliers for columns room_bath(33

bed), living_measure, lot_measure, ceil_measure, and Basement. We have created a function named outlier_removal which we return as lower bound and upper bound.

In [96]:
```
lowerbound,upperbound = outlier_removal(df.ceil_measure)
print(lowerbound,upperbound)
```

```
-340.0 3740.0
```

Lets check which column is considered as an outlier

In [97]:
```
df[(df.ceil_measure < lowerbound) | (df.ceil_measure > upperbound)]
```

Out[97]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	long	living_measure15	lot_measure15	f
17852	7237550310	2014-05-12	1230000	4.0	4.50	5420.0	101930.0	1.0	0.0	0.0	...	-122.005	4760.0	101930.0	
10641	3892500150	2014-05-21	1550000	3.0	2.50	4460.0	26027.0	2.0	0.0	0.0	...	-122.173	3770.0	26027.0	
18983	425069020	2014-05-05	1090000	4.0	2.50	4340.0	141570.0	2.5	0.0	0.0	...	-122.048	2720.0	97138.0	
10456	7397300220	2014-05-29	2750000	4.0	3.25	4430.0	21000.0	2.0	0.0	0.0	...	-122.237	3930.0	20000.0	
1990	2616800600	2014-05-30	840000	7.0	4.50	4290.0	37607.0	1.5	0.0	0.0	...	-122.033	2810.0	40510.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10744	3751600409	2015-05-08	510000	4.0	2.50	4073.0	17334.0	2.0	0.0	0.0	...	-122.270	1780.0	9625.0	
11285	2424059174	2015-05-08	2000000	4.0	3.25	5640.0	35006.0	2.0	0.0	2.0	...	-122.104	4920.0	35033.0	
18840	6065300370	2015-05-06	4210000	5.0	6.00	7440.0	21540.0	2.0	0.0	0.0	...	-122.189	4740.0	19329.0	
9276	1266200140	2015-05-06	1850000	4.0	3.25	4160.0	10335.0	2.0	0.0	0.0	...	-122.192	1840.0	10333.0	
19101	1623089165	2015-05-06	920000	4.0	3.75	4030.0	503989.0	2.0	0.0	0.0	...	-121.795	2110.0	71874.0	

611 rows × 28 columns

We got 611 records which are outliers

Table 3- Outlier Detection

Here is a code snippet of a column named ceil_measure, were using our function outlier_removal we have got 611 records that are outliers for that particular column.

In [98]:
```
# dropping the record from the dataset
df.drop(df[ (df.ceil_measure > upperbound) | (df.ceil_measure < lowerbound) ].index, inplace=True)
```

Likewise, we have dropped the outliers. So, we have followed the same procedure for the columns having outliers.

Need For Variable Transformation:-

Replacing the unwanted symbol and nan value with numpy nan values which we are going to deal in missing value

```
In [8]:
data['ceil']= data['ceil'].replace('$',np.nan)
data['coast']= data['coast'].replace('$',np.nan)
data['condition'] = data['condition'].replace('$',np.nan)
data['condition'] = data['condition'].replace('nan',np.nan)
data['yr_built'] = data['yr_built'].replace('$',np.nan)
data['total_area']=data['total_area'].replace('$',np.nan)
data['long']=data['long'].replace('$',np.nan)
```

```
In [9]:
data['ceil']=data['ceil'].astype('float')
data['coast']=data['coast'].astype('float')
data['total_area']=data['total_area'].astype('float')
data['long']=data['long'].astype('float')
data['condition']=data['condition'].astype('int', errors='ignore')
data['yr_built']=data['yr_built'].astype('int',errors='ignore')
```

Code Sample 4 – Variable Transformation

Variable transformation is a way to make the data work better in your model. Here specifically we have Replaced the unwanted symbol and nan value with NumPy nan values i.e np. nan which we are going to deal in missing value. And we have also converted the data types of few columns which will help build our model. The need for this is because we need our model to have a good score and accuracy which will make good predictions. So to feed the data to our model we must ensure to take these steps and make our data insightful.

Variables removed or added and why?

Variables are removed in such a scenario where we have a large number of null values in any columns particularly or else in our dataset, to make our data clean and ready for modeling. Whereas, variables are added in such a scenario where Consider an example where we have a column as start-date and another column as end-date so we can create a column named ‘difference’ where we subtract the start-date and end-date and have a number of days between them in a column named ‘difference’ and later drop start-date and end-date as if they are of no use.

4. Modeling Building

Model Selection and Why?

After cleaning and processing the data then comes the modeling part which includes building Machine Learning models, let's first understand in brief what Machine Learning is?

Machine Learning is a technique that analyzes past data and tries to extract meaningful insights and patterns from them which can be further used to perform predictions in future. For example, classifying whether a tumor is benign or malignant, predicting stock prices, etc. One such application which we're using right here is predicting house prices. Before making predictions first we need to build a model and train it using past data.

First, we need to separate the dataset into two parts: features (property attributes) and labels (prices) which is the required format for any model to be trained on.

Then the data needs to be split into 3 sets

1. Training set - This will be the part of the dataset which the model will be using to train itself, the size should be at least 60-70% of the total data we've.
2. Validation set - This set is used for validating our model's performance for a different set of hyperparameters. After taking out the train set, the remaining set can be split into validation and test set.

3. Testing set - To evaluate how the model is performing on the unseen data on which the model will be doing future predictions on, test set is used. It helps to understand how much error is there between actual and predicted values.

Code Sample 5- Splitting Data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=10)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=10)
```

```
3 print(X_train.shape, y_train.shape)
4 print(X_test.shape,y_test.shape)
5 print(X_val.shape, y_val.shape)
```

```
(10288, 55) (10288,)
(4573, 55) (4573,)
(3430, 55) (3430,)
```

We need to build different regression algorithms and using the validation set we can determine which model to keep for making final predictions.

Here is the list of all the algorithms we've to build and evaluated:

Code Sample 6- List of Algo used

```
1 list(model_comp['Algorithm'].values)
```

```
['Simple Linear Reg Model',
 'SLinear-Reg (Lasso)',
 'SLinear-Reg (Ridge)',
 'KNN',
 'SVR with kernel rbf',
 'Simple DT',
 'DT with some modified parameter',
 'Logistic Regression',
 'Gradient Boosting',
 'Bagging Regressor',
 'Random Forest',
 'ADA Boost',
 'XGBoost',
 'Gradient Boosting BY Grid Search',
 'Gradient Boosting Tuning using graph',
 'Gradient Boosting BY Random Search']
```

Initially, we've tried **Linear Regression** and its variants **Lasso** (l1 norm) and **Ridge** (l2 norm) **Regression**, but they are performing quite similarly on the 3 sets (train, valid & test).

Table 5 i- Algorithm Score

	Algorithm	train Score	Val Score	test Score
0	Simple Linear Reg Model	0.621488	0.625768	0.625564
1	SLinear-Reg (Lasso)	0.621413	0.627354	0.626842
2	SLinear-Reg (Ridge)	0.621065	0.627816	0.627308

From the score column, we can see that the train, vali,d and test scores are 0.621, 0.625, and 0.625 respectively.

Let's look at some other algorithms, and how they are performing as compared to linear regression.

So **KNN** is giving a very good score of 0.99 on the train set, but not that good on the valid and test set, looks like it is overfitting on the training data. If we look at the **Support Vector**

Regressor the performance is worse on the train set as well as the test set, same is the case for **Logistic Regression**.

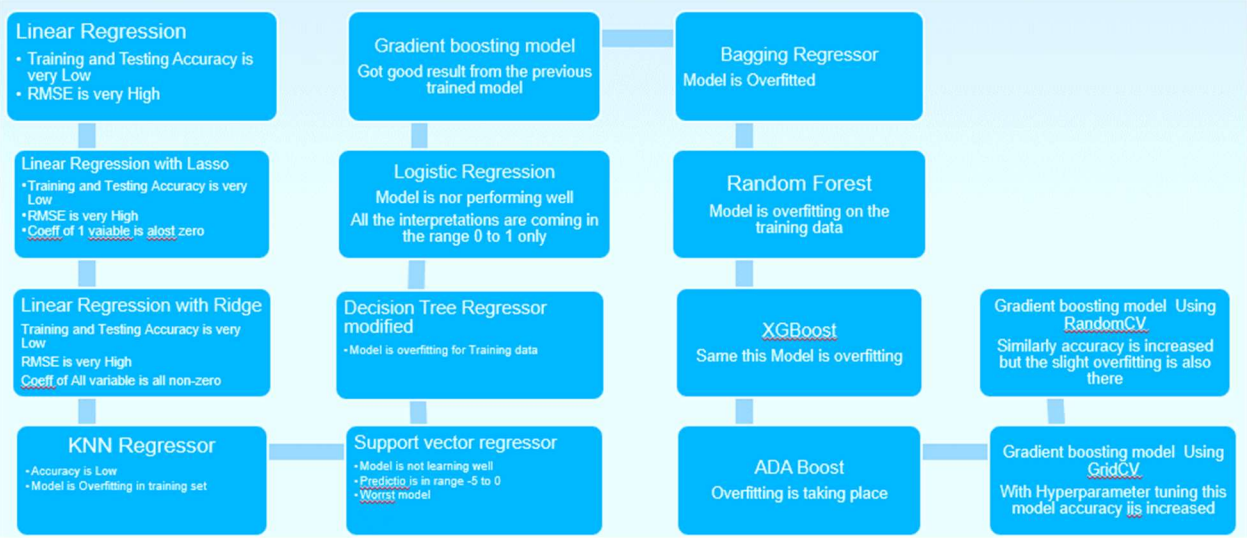
Table 5 ii- Algorithm Score

	Algorithm	train Score	Val Score	test Score
0	KNN	0.998847	0.378164	0.405956
1	SVR with kernel rbf	-0.049504	-0.044851	-0.046833
2	Logistic Regression	0.192253	0.210730	0.245098

We aren't getting that good results with these algorithms, what about Decision Tree and the more powerful tree algorithms based on decision trees such as Random Forest, and Bagging Regressor? If we look at their performance, the **Decision Tree** with default

Flow Diagram 1- Flow of the Model Used and Why

parameters is kinda overfitting, but **Bagging Regressor** and **Random Forest** have little improvement over the **Linear Regressor**.



	Algorithm	train Score	Val Score	test Score
0	Simple Linear Reg Model	0.621488	0.625768	0.625564
1	Simple DT	0.998847	0.345469	0.373684
2	DT with some modified parameter	0.721555	0.589575	0.546034
3	Bagging Regressor	0.949652	0.678017	0.670340
4	Random Forest	0.952204	0.686543	0.675562

Table 5 iii- Algorithm Score

Finally, now we can try those boosting algorithms and see where they are getting us? Generally boosting algorithms give a very good performance, and they are giving on the training set but there isn't any significant improvement on the test set as compared to Tree-based models.

	Algorithm	train Score	Val Score	test Score
0	Gradient Boosting	0.734963	0.687915	0.668543
1	ADA Boost	0.993742	0.689241	0.664536
2	XGBoost	0.882787	0.675453	0.666469

Table 5 iv- Algorithm Score

Efforts to improve model performance

Hyperparameter Tuning

Hyperparameter tuning is the process of trying out a different set of model parameters, actually, they are algorithm's parameter for example **theta1** and **theta2** in the hypothesis function for Linear Regression is model parameter, but **lambda** which is a factor that decides the amount of regularization is algorithm's parameter called as a hyperparameter. Tuning hyperparameters helps us to find out the optimal parameter values for which the model is giving less error and a better overall score. We've performed tuning for Gradient Boosting Regressor with both the methods **RandomizedSearchCV** as well as **GridSearchCV.** What random search does is from the given set of parameter values, it tries out **n** number of combinations whereas grid search builds a model with all the possible combinations from the given set of parameters and gives us the optimal values.

Grid Search CV

These are the parameters that we've given to Grid Search to find out the best one.

Code Sample 7- Best parameter grid

```
param_gridF = {
    'max_features': ['auto','sqrt'],
    'learning_rate': [0.1,0.2],
    'max_depth': [5,8],
    'min_samples_leaf': [5,10],
    'min_samples_split': [40,50],
    'n_estimators':  [200,400,600],
}

```

Then we need to provide it with the estimator and specify other things such as how many cross-validations sets to evaluate upon.

Code Sample 8- Hyparparamter tuning

```
grid_searchF = GridSearchCV(estimator = GBR_test, param_grid = param_gridF,
                            cv = 2, n_jobs = -1, verbose = 1)
grid_searchF.fit(X_train,y_train)
```

After fitting GridSearchCV it returns those params with which it got the best score.

```
grid_searchF.best_score_,grid_searchF.best_params_
```

```
Fitting 2 folds for each of 96 candidates, totalling 192 fits

(0.6603042018549291,
 {'learning_rate': 0.1,
  'max_depth': 5,
  'max_features': 'sqrt',
  'min_samples_leaf': 10,
  'min_samples_split': 40,
  'n_estimators': 200})
```

Similarly, we've done it using RandomSearchCV, even if they have improved the final test score by a little difference, they are performing the best when compared with all the other algorithms. So this tuned Gradient Boosted Model can be used to make the final predictions.

Table 5 v- Algorithm Score

	Algorithm	train Score	Val Score	test Score
0	Gradient Boosting BY Grid Search	0.766412	0.692587	0.681413
1	Gradient Boosting BY Random Search	0.803816	0.696759	0.683325

5. Model Validation

Checking significance of variable using p value

- Firstly we are performing the one hot encoding on the scaled data then we are checking the significance of each column by p-value.
- *These are the selected feature using p-value which we are going to use for analysis-*

living_measure, lot_measure, ceil_measure, yr_built,living_measure15, lot_measure15, total_area, house_land_ratio,room_bed_1.0, room_bed_2.0, room_bed_3.0, room_bath_0.5,room_bath_0.75, room_bath_1.0, room_bath_1.25, room_bath_1.5,room_bath_1.75, room_bath_2.0, room_bath_2.25, room_bath_2.5,room_bath_2.75, room_bath_3.0, room_bath_3.25, room_bath_3.5,room_bath_3.75, room_bath_4.0, room_bath_4.25, room_bath_4.5,room_bath_4.75, room_bath_5.0, room_bath_5.25, ceil_1.5,ceil_2.5, ceil_3.0, ceil_3.5, coast_1.0, sight_1.0,sight_2.0, sight_3.0, sight_4.0, condition_2.0, condition_3.0,condition_4.0, condition_5.0, quality_4.0, quality_5.0,quality_6.0, quality_7.0, quality_8.0, quality_9.0,quality_10.0, quality_11.0, quality_12.0, has_basement_Yes,has_renovated_Yes

Performance Metrices

Just building is not enough, as we need to evaluate it using different metrics based on the problem we're solving. Model Validation helps us to understand how well the model is generalizing on the real-world data, the data which it has not seen during the training phase.

For regression problems the evaluation metrics we've used are:

- **RMSE** (Root Mean Squared Error)
- **MSE** (Mean Squared Error)
- **MAE** (Mean Absolute Error)
- **MAPE** (Mean Absolute Percentage Error)
- **Adjusted R^2**

The R^2 score is between 0 and 1 (it can also get -ve when the model is performing worse). The closer the value to 1 the better the performance of the model will be and a model which always predicts constant value irrespective of the input values gets an R2 score of 0.

MAE is on average how far those predicted values are from actual values, whereas MSE is the average of squared differences between actual and predicted values.

Let's look at these metrics for the best-performing algorithms on the test set,
Here are the top 10 algorithms based on the test score, MAE, MSE, MAPE, and Adjusted R2. Depending on increasing or decreasing which metric helps solve the business problem, we can pick the appropriate model for final deployment.
Note: Other than Test scores all scores are sorted in ascending order

Table 5 vi- Algorithm Performance

	Algorithm	test Score
0	Gradient Boosting BY Random Search	0.683325
1	Gradient Boosting BY Grid Search	0.681413
2	Random Forest	0.675562
3	Bagging Regressor	0.670340
4	Gradient Boosting	0.668543
5	XGBoost	0.666469
6	ADA Boost	0.664536
7	Gradient Boosting Tuning using graph	0.658191
8	SLinear-Reg (Ridge)	0.627308
9	SLinear-Reg (Lasso)	0.626842

	Algorithm	RMSE_te
0	Gradient Boosting BY Random Search	146808.495058
1	Gradient Boosting BY Grid Search	147251.150518
2	Random Forest	148597.226767
3	Bagging Regressor	149788.247564
4	Gradient Boosting	150195.916138
5	XGBoost	150665.048233
6	ADA Boost	151100.979404
7	Gradient Boosting Tuning using graph	152523.370880
8	SLinear-Reg (Ridge)	159264.650115
9	SLinear-Reg (Lasso)	159364.279177

	Algorithm	MSE_te
0	Gradient Boosting BY Random Search	2.155273e+10
1	Gradient Boosting BY Grid Search	2.168290e+10
2	Random Forest	2.208114e+10
3	Bagging Regressor	2.243652e+10
4	Gradient Boosting	2.255881e+10
5	XGBoost	2.269996e+10
6	ADA Boost	2.283151e+10
7	Gradient Boosting Tuning using graph	2.326338e+10
8	SLinear-Reg (Ridge)	2.536523e+10
9	SLinear-Reg (Lasso)	2.539697e+10

	Algorithm	MAE_te
0	Random Forest	103337.379339
1	ADA Boost	103373.142193
2	Bagging Regressor	103753.218430
3	Gradient Boosting BY Random Search	103929.258825
4	Gradient Boosting BY Grid Search	104461.186627
5	XGBoost	107024.701953
6	Gradient Boosting Tuning using graph	107026.610617
7	Gradient Boosting	107078.099910
8	SLinear-Reg (Lasso)	113295.563188
9	SLinear-Reg (Ridge)	113309.123096

	Algorithm	Mape_te
0	ADA Boost	0.237623
1	Random Forest	0.240426
2	Bagging Regressor	0.241253
3	Gradient Boosting BY Random Search	0.241914
4	Gradient Boosting BY Grid Search	0.243819
5	Gradient Boosting Tuning using graph	0.245718
6	XGBoost	0.246614
7	Gradient Boosting	0.251753
8	SLinear-Reg (Lasso)	0.264539
9	Simple Linear Reg Model	0.264585

	Algorithm	Adjusted_r2_te
0	Gradient Boosting BY Random Search	1.061958
1	Gradient Boosting BY Grid Search	1.062332
2	Random Forest	1.063477
3	Bagging Regressor	1.064499
4	Gradient Boosting	1.064850
5	XGBoost	1.065256
6	ADA Boost	1.065634
7	Gradient Boosting Tuning using graph	1.066876
8	SLinear-Reg (Ridge)	1.072918
9	SLinear-Reg (Lasso)	1.073009

6. Final Interpretation/Recommendation

Table 6 - Final Algorithm Permomance Table

Out[59]:

	Algorithm	train Score	RMSE_tr	MSE_tr	MAE_tr	Mape_tr	Adjusted_r2_tr	Val Score	RMSE_vl	MSE_vl	MAE_vl	Mape_
0	Simple Linear Reg Model	0.621488	155468.518430	2.417046e+10	111377.884683	0.259166	1.074057	0.625768	158455.473771	2.510814e+10	113973.495055	0.267
1	SLinear-Reg (Lasso)	0.621413	155483.805423	2.417521e+10	111391.345669	0.259206	1.074071	0.627354	158119.391005	2.500174e+10	113923.780764	0.267
2	SLinear-Reg (Ridge)	0.621065	155555.327465	2.419746e+10	111469.771752	0.259326	1.074140	0.627816	158021.265005	2.497072e+10	113960.244193	0.267
3	KNN	0.998847	8582.103202	7.365250e+07	729.158534	0.002458	1.000226	0.378164	204255.885469	4.172047e+10	140777.742441	0.318
4	SVR with kernel rbf	-0.049504	258877.504279	6.701756e+10	178686.744132	0.398561	1.205338	-0.044851	264767.080780	7.010161e+10	180276.683589	0.402
5	Simple DT	0.998847	8582.103202	7.365250e+07	729.158534	0.002458	1.000226	0.372515	205181.617732	4.209950e+10	141627.837318	0.325
6	DT with some modified parameter	0.721555	133343.513951	1.778049e+10	97392.963931	NaN	NaN	0.589047	166047.753895	2.757186e+10	118299.186379	N
7	Logistic Regression	0.192253	227111.992505	5.157986e+10	155456.759623	0.319897	1.158037	0.210730	230117.446700	5.295404e+10	157832.806414	0.326
8	Gradient Boosting	0.734963	130093.493081	1.692432e+10	97559.949344	0.232923	1.051855	0.687915	144701.503485	2.093853e+10	105745.259032	0.252
9	Bagging Regressor	0.949652	56701.077157	3.215012e+09	39326.371014	0.091062	1.009851	0.678017	146978.370915	2.160264e+10	104104.756924	0.245
10	Random Forest	0.951876	55435.122905	3.073053e+09	38776.966675	0.090071	1.009416	0.685103	145352.001749	2.112720e+10	103022.832113	0.243
11	ADA Boost	0.993702	20053.605927	4.021471e+08	7954.145087	0.027689	1.001232	0.682158	146030.011107	2.132476e+10	103023.322351	0.241
12	XGBoost	0.882787	86514.902158	7.484828e+09	66165.278317	0.164866	1.022933	0.675453	147562.358838	2.177465e+10	106268.788725	0.247
13	Gradient Boosting BY Random Search	0.803816	111926.838485	1.252762e+10	84681.119572	0.203216	1.038384	0.696759	142636.491012	2.034517e+10	103611.891431	0.244

We can conclude from above that gridsearch CV is giving better results compared to that of tuning done by graphical method of individual

- The ensemble models have performed well compared to that of linear, KNN, SVR models, and logistic.
- The best performance is given by the Gradient boosting model
- The top key features that drive the price of the property are: 'furnished_1', 'yr_built', 'living_measure','quality_8', 'HouseLandRatio', 'lot_measure15', 'quality_9', 'ceil_measure', 'total_area'.
- The above data is also reinforced by the analysis done during bivariate analysis.
- The linear regression model performed with scores 0.62, 0.61 & 0.62 in training data set, validation data and test data set respectively
- The lasso linear regression model performed with scores 0.62, 0.61 & 0.62 in training data set, validation data and test data set respectively. The coefficients of 1 variable in lasso model is almost '0', signifying that the variable with '0' coefficient can be dropped.
- The Ridge linear regression model performed with scores 0.62, 0.61 & 0.62 in training data set, validation data and test data set respectively. The coefficients of variables in ridge model are all non-zero, indicating that non of the variables can be dropped.
- Linear models have performed almost with similar results in both regularized model and non-regularized models From the graph we can also see that the over model is in ideal state as regularized model and non-regularized models
- Though KNN regressor performed well in training set, the performance score in validation set is very less. This shows that the model is overfitted in training set.
- Negative scores in SVR model is due to non-learning of the model in the training set which results in non-performance in validation set.
- Initial Decision tree model shows overfit in training set with 0.99 score and low performance in validation and test set. It is not the ideal solution.
- Decision tree model with modified parameter has better performed on the training set and validation set compared to initial decision tree model. But overall decision tree has not performed well than linear regression models.
- The Logistic regression model with modified parameters has not performed well with just ~0.22 in training tesing and validation data sets.

- Gradient boosting model has provided good scores in both training and validation sets
- Bagging model also performed well in training and validation sets. There seems to be overfitting in training set. We need to analyze further by hyper tuning
- Random forest model has performed fine not that much good. There is scope of further analysis on this model. But the overfitting is taken place which is not good.
- Ensemble models: in summary ensemble models have performed well on training and validation sets. These models will be selected for further analysis with hyper tuning and feature selection
- Ada Boost model has performed fine not that much good. There is scope of further analysis on this model. But the overfitting is taken place which is not good.
- XGBoost has performed fine not that much good. There is scope of further analysis on this model. But the overfitting is taken place which is not good.
- Ensemble methods are performing better than linear models.
- Of all the ensemble models, Gradient boosting regressor is giving better R2 score.
- We identified top 30 features that are explaining the 95% variation in model(Random Forest). Will further hyper tune the model to improve the model performance. Will further explore and evaluate the features while hyper turning the ensemble models

Recommendations:

- Classification algorithms can be used to predict if a house is near a coastal area.
- Clustering algorithms can be used to identify similar neighbors
- Machine learning algorithms can predict the value of a building or asset over the next decade, based on data from the last few years. You can see exactly how much profit you will get in the next 10 years. !!

- Using machine learning algorithms in recommender systems helps buyers find property details and unique insights of it.

Final Words:

- We can conclude from above that GridSearchCV is giving better results compared to that of tuning done by graphical method of individual parameters or RandomSearchCV
- The ensemble models have performed well compared to that of linear,KNN,SVR models, logistic
- The best performance is given by Gradient boosting model
- The top key features that drive the price of the property are: 'furnished_1', 'yr_built', 'living_measure','quality_8', 'HouseLandRatio', 'lot_measure15', 'quality_9', 'ceil_measure', 'total_area'
- The above data is also reinforced by the analysis done during bivariate analysis.