HOUSE PRICE PREDICTION

CAPSTONE PROJECT

Aaditya Desai 9/12/21 Data Science and Business Analytics

INDEX

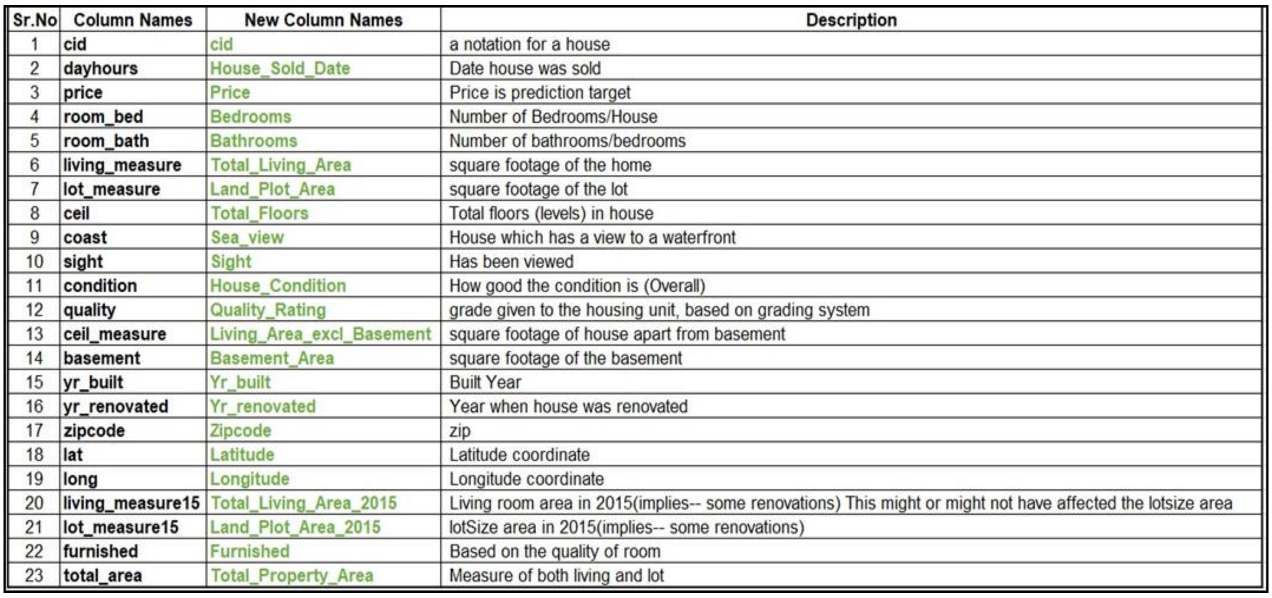
INTRODUCTION OF THE BUSINESS PROBLEM

• Defining 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 may expect — 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.

We have around 21600 rows and 23 columns of data which we need to process to get the useful information and make insights from it.



Sr.No	Column Names	New Column Names	Description
1	cid	cid	a notation for a house
2	dayhours	House_Sold_Date	Date house was sold
3	price	Price	Price is prediction target
4	room_bed	Bedrooms	Number of Bedrooms/House
5	room_bath	Bathrooms	Number of bathrooms/bedrooms
6	living_measure	Total_Living_Area	square footage of the home
7	lot_measure	Land_Plot_Area	square footage of the lot
8	ceil	Total_Floors	Total floors (levels) in house
9	coast	Sea_view	House which has a view to a waterfront
10	sight	Sight	Has been viewed
11	condition	House_Condition	How good the condition is (Overall)
12	quality	Quality_Rating	grade given to the housing unit, based on grading system
13	ceil_measure	Living_Area_excl_Basement	square footage of house apart from basement
14	basement	Basement_Area	square footage of the basement
15	yr_built	Yr_built	Built Year
16	yr_renovated	Yr_renovated	Year when house was renovated
17	zipcode	Zipcode	zip
18	lat	Latitude	Latitude coordinate
19	long	Longitude	Longitude coordinate
20	living_measure15	Total_Living_Area_2015	Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area
21	lot_measure15	Land_Plot_Area_2015	lotSize area in 2015(implies-- some renovations)
22	furnished	Furnished	Based on the quality of room
23	total_area	Total_Property_Area	Measure of both living and lot

• Need of the Study/Project

Buying a house is generally a dream for many. Considering all the parameters that contribute to the variation in price of buying a home, we have gathered data for 21.5 thousand homes that have been sold in 2014 and 2015 in various cities (24) in King County, Washington.

Internal factors and quantitative ones like number of bedrooms, number of bathrooms, furniture in the house, basement, size of living area, lawns/plot area, etc. play an important part in contribution to the price of the house.

External factors like sea view, location of the house i.e. latitude and longitude, zipcode, sight play a handy role in classifying the area and determining the price.

Qualitative factors like condition of the house, Quality rating, Year in which house was built, renovated and sold play along as factors setting the price of the house, as in the house that was renovated later will be a little costly compared to the one that was not renovated.

So, we use these factors, analyze their combination and try to fit them in a model to predict the houses based on the newer inputs and the combinations of these parameters. This will automate and stabilize the rates based on what actually the cost should be.

• Understanding Business/Social Opportunity

The case has various business and social aspects. The model that we build to predict the house price based on the features can be useful for the buyers as well as the sellers.

Supposedly, if you are shifting to a newer locality and you wish to purchase a home, it may be difficult to know the right amount to spend on the property in that location, so you may be either need to find a real estate consultant who would help you find houses in your budget and give them their commission for the work or use this model to find the perfect fit in your budget.

If you’re a seller and want to know the right price of your house which will help you sell the house quickly, again, you can refer to this model. The model uses the data of houses sold in your neighborhood and based on the features in your house, suggest you the best price.

This model sound realistic with respect to its application point of view. Now coming to its business applications, we can make a web application/phone application and use some cloud storage for data and make an interactive website which works on this model and then present it/sell it to different real estate companies to store data, predict price and widen its cliental base- both on buyer and seller side. We can also sell the model to the state (government) and then it may make it public for public welfare and social use.

We can launch an online website/portal where sellers and buyers can upload their house details and all the values for these parameters and hence, this can act as the mediator- portals that will eliminate the brokers and the customers may hence save their money on brokerages.

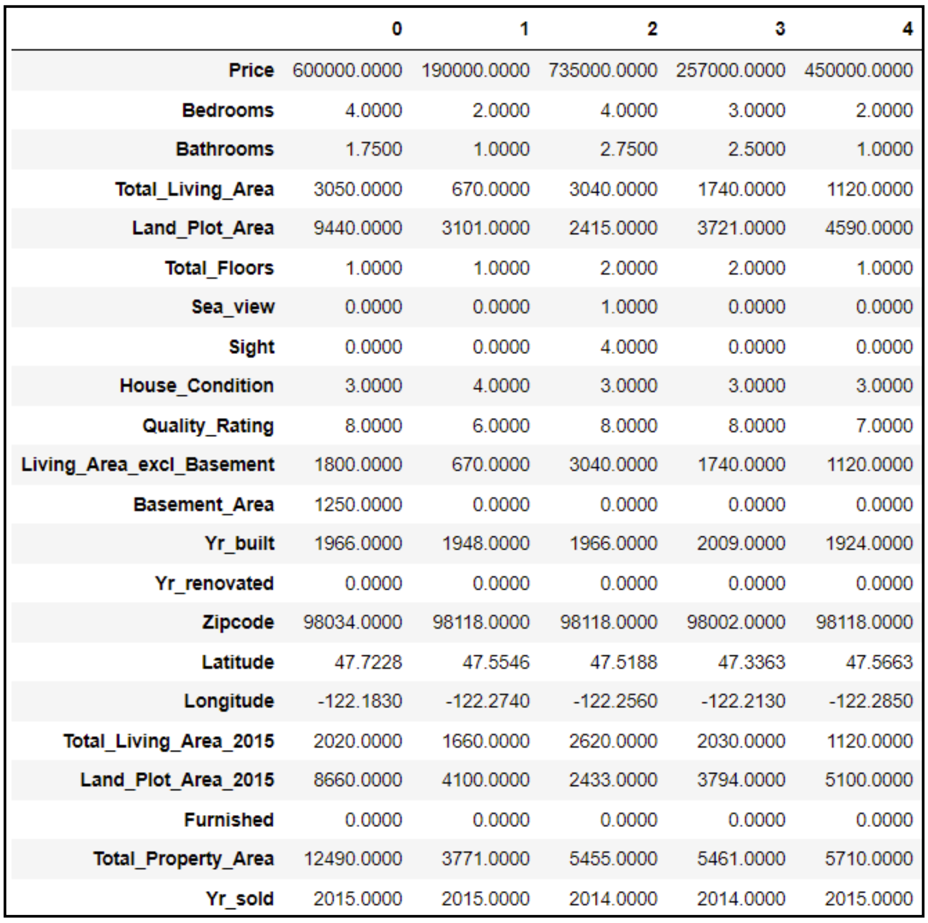
DATA REPORT

• Understanding the data collection

The data consists of 21613 observations (houses) across 24 cities in King County (Washington) that were sold in 2014 and 2015. We identified the location using the latitude, longitude and zipcode provided in the data. The data consists of various parameters like number of bedrooms, bathrooms, floors, furnished, Sea view, basement and living area, plot area, etc.

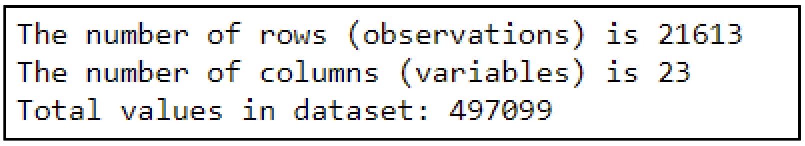
• Visual inspection of data

We can see the top 5 rows using the head () function.



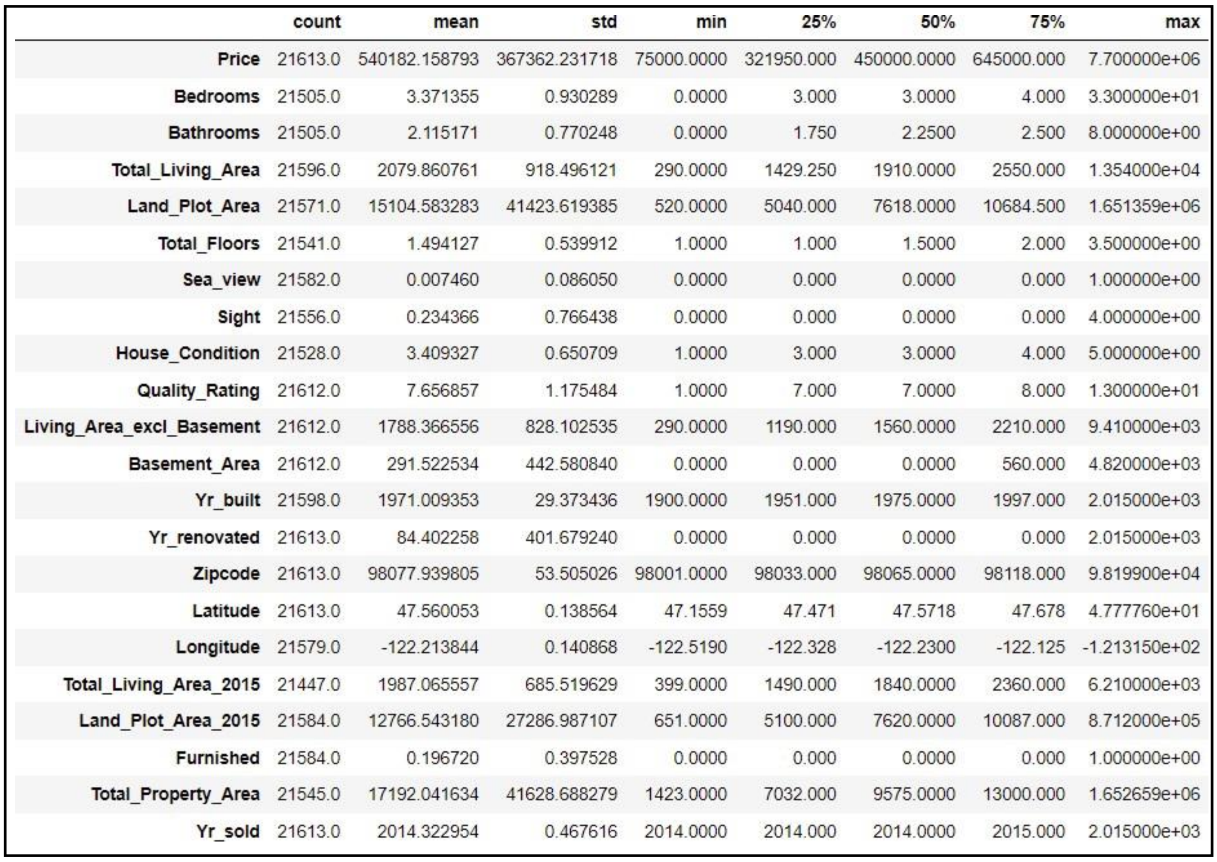
	0	1	2	3	4
Price	600000.0000	190000.0000	735000.0000	257000.0000	450000.0000
Bedrooms	4.0000	2.0000	4.0000	3.0000	2.0000
Bathrooms	1.7500	1.0000	2.7500	2.5000	1.0000
Total_Living_Area	3050.0000	670.0000	3040.0000	1740.0000	1120.0000
Land_Plot_Area	9440.0000	3101.0000	2415.0000	3721.0000	4590.0000
Total_Floors	1.0000	1.0000	2.0000	2.0000	1.0000
Sea_view	0.0000	0.0000	1.0000	0.0000	0.0000
Sight	0.0000	0.0000	4.0000	0.0000	0.0000
House_Condition	3.0000	4.0000	3.0000	3.0000	3.0000
Quality_Rating	8.0000	6.0000	8.0000	8.0000	7.0000
Living_Area_excl_Basement	1800.0000	670.0000	3040.0000	1740.0000	1120.0000
Basement_Area	1250.0000	0.0000	0.0000	0.0000	0.0000
Yr_built	1966.0000	1948.0000	1966.0000	2009.0000	1924.0000
Yr_renovated	0.0000	0.0000	0.0000	0.0000	0.0000
Zipcode	98034.0000	98118.0000	98118.0000	98002.0000	98118.0000
Latitude	47.7228	47.5546	47.5188	47.3363	47.5663
Longitude	-122.1830	-122.2740	-122.2560	-122.2130	-122.2850
Total_Living_Area_2015	2020.0000	1660.0000	2620.0000	2030.0000	1120.0000
Land_Plot_Area_2015	8660.0000	4100.0000	2433.0000	3794.0000	5100.0000
Furnished	0.0000	0.0000	0.0000	0.0000	0.0000
Total_Property_Area	12490.0000	3771.0000	5455.0000	5461.0000	5710.0000
Yr_sold	2015.0000	2015.0000	2014.0000	2014.0000	2015.0000

We can also check the size of dataset (rows and columns) using shape. We see that there are 21613 rows and 23 columns.



```
The number of rows (observations) is 21613
The number of columns (variables) is 23
Total values in dataset: 497099
```

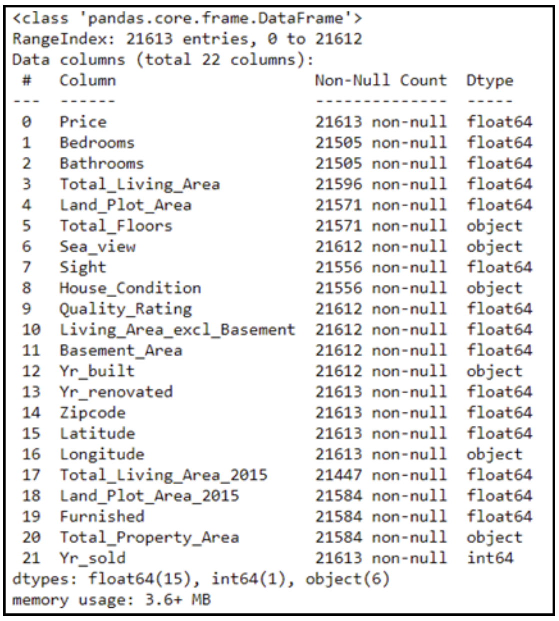
We check the 5 point summary for variables using describe ()



	count	mean	std	min	25%	50%	75%	max
Price	21613.0	540182.158793	367362.231718	75000.0000	321950.000	450000.0000	645000.000	7.700000e+06
Bedrooms	21505.0	3.371355	0.930289	0.0000	3.000	3.0000	4.000	3.300000e+01
Bathrooms	21505.0	2.115171	0.770248	0.0000	1.750	2.2500	2.500	8.000000e+00
Total_Living_Area	21596.0	2079.860761	918.496121	290.0000	1429.250	1910.0000	2550.000	1.354000e+04
Land_Plot_Area	21571.0	15104.583283	41423.619385	520.0000	5040.000	7618.0000	10684.500	1.651359e+06
Total_Floors	21541.0	1.494127	0.539912	1.0000	1.000	1.5000	2.000	3.500000e+00
Sea_view	21582.0	0.007460	0.086050	0.0000	0.000	0.0000	0.000	1.000000e+00
Sight	21556.0	0.234366	0.766438	0.0000	0.000	0.0000	0.000	4.000000e+00
House_Condition	21528.0	3.409327	0.650709	1.0000	3.000	3.0000	4.000	5.000000e+00
Quality_Rating	21612.0	7.656857	1.175484	1.0000	7.000	7.0000	8.000	1.300000e+01
Living_Area_excl_Basement	21612.0	1788.366556	828.102535	290.0000	1190.000	1560.0000	2210.000	9.410000e+03
Basement_Area	21612.0	291.522534	442.580840	0.0000	0.000	0.0000	560.000	4.820000e+03
Yr_built	21598.0	1971.009353	29.373436	1900.0000	1951.000	1975.0000	1997.000	2.015000e+03
Yr_renovated	21613.0	84.402258	401.679240	0.0000	0.000	0.0000	0.000	2.015000e+03
Zipcode	21613.0	98077.939805	53.505026	98001.0000	98033.000	98065.0000	98118.000	9.819900e+04
Latitude	21613.0	47.560053	0.138564	47.1559	47.471	47.5718	47.678	4.777760e+01
Longitude	21579.0	-122.213844	0.140868	-122.5190	-122.328	-122.2300	-122.125	-1.213150e+02
Total_Living_Area_2015	21447.0	1987.065557	685.519629	399.0000	1490.000	1840.0000	2360.000	6.210000e+03
Land_Plot_Area_2015	21584.0	12766.543180	27286.987107	651.0000	5100.000	7620.0000	10087.000	8.712000e+05
Furnished	21584.0	0.196720	0.397528	0.0000	0.000	0.0000	0.000	1.000000e+00
Total_Property_Area	21545.0	17192.041634	41628.688279	1423.0000	7032.000	9575.0000	13000.000	1.652659e+06
Yr_sold	21613.0	2014.322954	0.467616	2014.0000	2014.000	2014.0000	2015.000	2.015000e+03

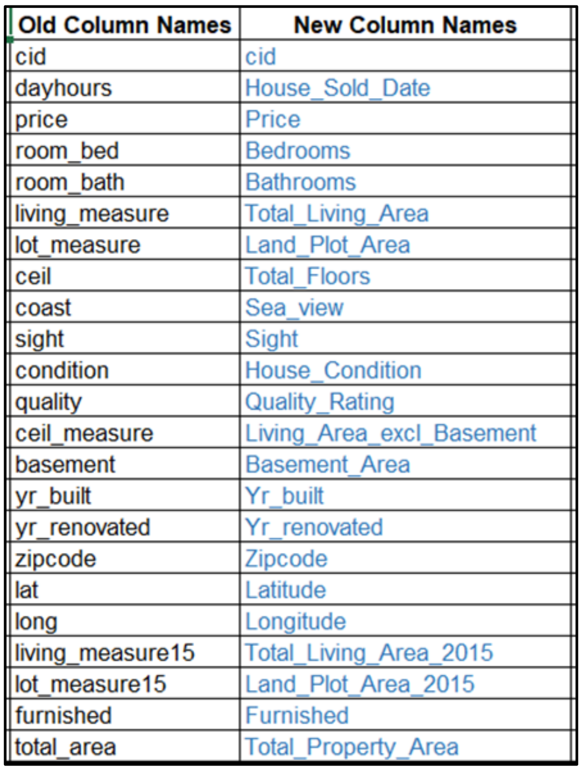
- ## Understanding of attributes

We check the datatype, number of non-null values wrt columns using info () function



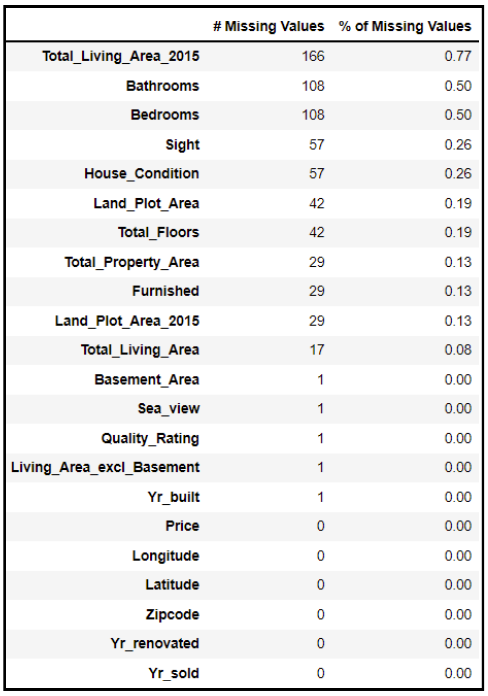
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 22 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Price                      21613 non-null  float64
 1   Bedrooms                   21505 non-null  float64
 2   Bathrooms                  21505 non-null  float64
 3   Total_Living_Area          21596 non-null  float64
 4   Land_Plot_Area             21571 non-null  float64
 5   Total_Floors               21571 non-null  object
 6   Sea_view                   21612 non-null  object
 7   Sight                      21556 non-null  float64
 8   House_Condition            21556 non-null  object
 9   Quality_Rating             21612 non-null  float64
 10  Living_Area_excl_Basement  21612 non-null  float64
 11  Basement_Area              21612 non-null  float64
 12  Yr_built                   21612 non-null  object
 13  Yr_renovated               21613 non-null  float64
 14  Zipcode                    21613 non-null  float64
 15  Latitude                   21613 non-null  float64
 16  Longitude                  21613 non-null  object
 17  Total_Living_Area_2015     21447 non-null  float64
 18  Land_Plot_Area_2015        21584 non-null  float64
 19  Furnished                  21584 non-null  float64
 20  Total_Property_Area        21584 non-null  object
 21  Yr_sold                    21613 non-null  int64
dtypes: float64(15), int64(1), object(6)
memory usage: 3.6+ MB
```

We have renamed the columns to make them more interpretable using the rename (columns='',inplace=True) function. Inplace=True will make the change permanent.



Old Column Names	New Column Names
cid	cid
dayhours	House_Sold_Date
price	Price
room_bed	Bedrooms
room_bath	Bathrooms
living_measure	Total_Living_Area
lot_measure	Land_Plot_Area
ceil	Total_Floors
coast	Sea_view
sight	Sight
condition	House_Condition
quality	Quality_Rating
ceil_measure	Living_Area_excl_Basement
basement	Basement_Area
yr_built	Yr_built
yr_renovated	Yr_renovated
zipcode	Zipcode
lat	Latitude
long	Longitude
living_measure15	Total_Living_Area_2015
lot_measure15	Land_Plot_Area_2015
furnished	Furnished
total_area	Total_Property_Area

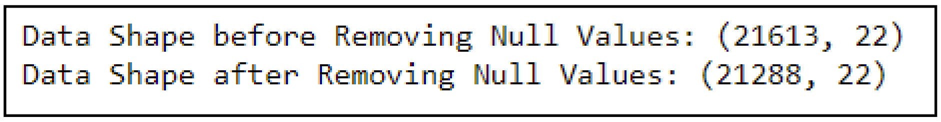
Finally we check the number of missing values and proportion of missing values out of total entries using isna().sum() and df.isnull().sum()/df.isnull().count() respectively. We see that variables have missing values but less than 1% ,so we can impute them later and not drop them.



	# Missing Values	% of Missing Values
Total_Living_Area_2015	166	0.77
Bathrooms	108	0.50
Bedrooms	108	0.50
Sight	57	0.26
House_Condition	57	0.26
Land_Plot_Area	42	0.19
Total_Floors	42	0.19
Total_Property_Area	29	0.13
Furnished	29	0.13
Land_Plot_Area_2015	29	0.13
Total_Living_Area	17	0.08
Basement_Area	1	0.00
Sea_view	1	0.00
Quality_Rating	1	0.00
Living_Area_excl_Basement	1	0.00
Yr_built	1	0.00
Price	0	0.00
Longitude	0	0.00
Latitude	0	0.00
Zipcode	0	0.00
Yr_renovated	0	0.00
Yr_sold	0	0.00

EXPLORATORY DATA ANALYSIS

For EDA, we create a copy of the dataset and remove the null values and do the visualizations on it so that the visualizations aren't bias.



```
Data Shape before Removing Null Values: (21613, 22)
Data Shape after Removing Null Values: (21288, 22)
```

Now, we check the distribution of variables. Hence, we see the Skewness and Kurtosis.

Kurtosis: -

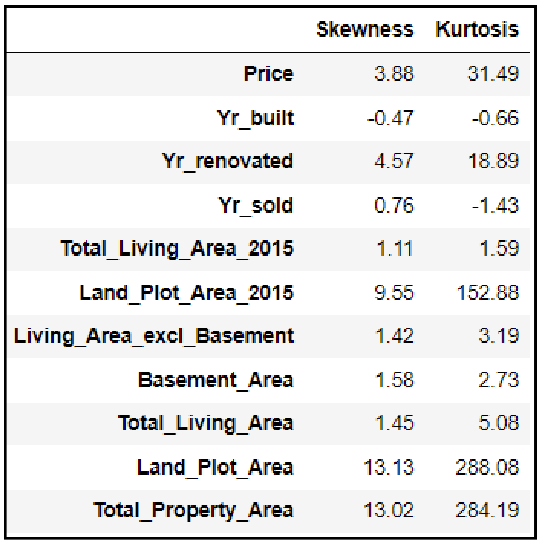
Kurtosis is a statistical measure used to describe the degree to which scores cluster in the tails or the peak of a frequency distribution. The peak is the tallest part of the distribution, and the tails are the ends of the distribution. There are three types of kurtosis:

- mesokurtic (0): similar to normal distribution.
- leptokurtic(+ve): has a higher peak and taller tails than a normal distribution.
- platykurtic(-ve): is flatter when compared with the normal distribution, with fewer values in its shorter tails.

Skewness: -

Skewness is the measure of how much the probability distribution of a random variable deviates from the normal distribution.

- symmetric (0): similar to normal distribution i.e. mean=median=mode.
- Positively skewed (+ve): Here, mode < median < mean.
- Negatively skewed (-ve): Here, mean < median < mode.

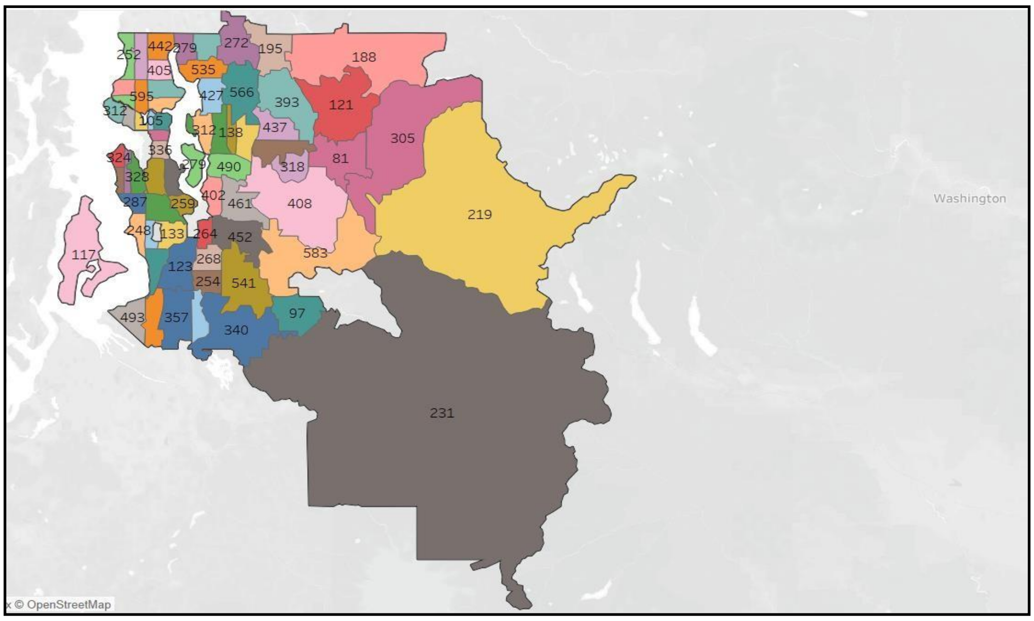


	Skewness	Kurtosis
Price	3.88	31.49
Yr_built	-0.47	-0.66
Yr_renovated	4.57	18.89
Yr_sold	0.76	-1.43
Total_Living_Area_2015	1.11	1.59
Land_Plot_Area_2015	9.55	152.88
Living_Area_excl_Basement	1.42	3.19
Basement_Area	1.58	2.73
Total_Living_Area	1.45	5.08
Land_Plot_Area	13.13	288.08
Total_Property_Area	13.02	284.19

We can see that Yr_built and Yr_sold are platykurtic and negatively skewed. All the other variables are leptokurtic and positively skewed. For continuous variables, we do boxplots and distribution plots and for categorical variables, we can do barplots, countplots, etc.

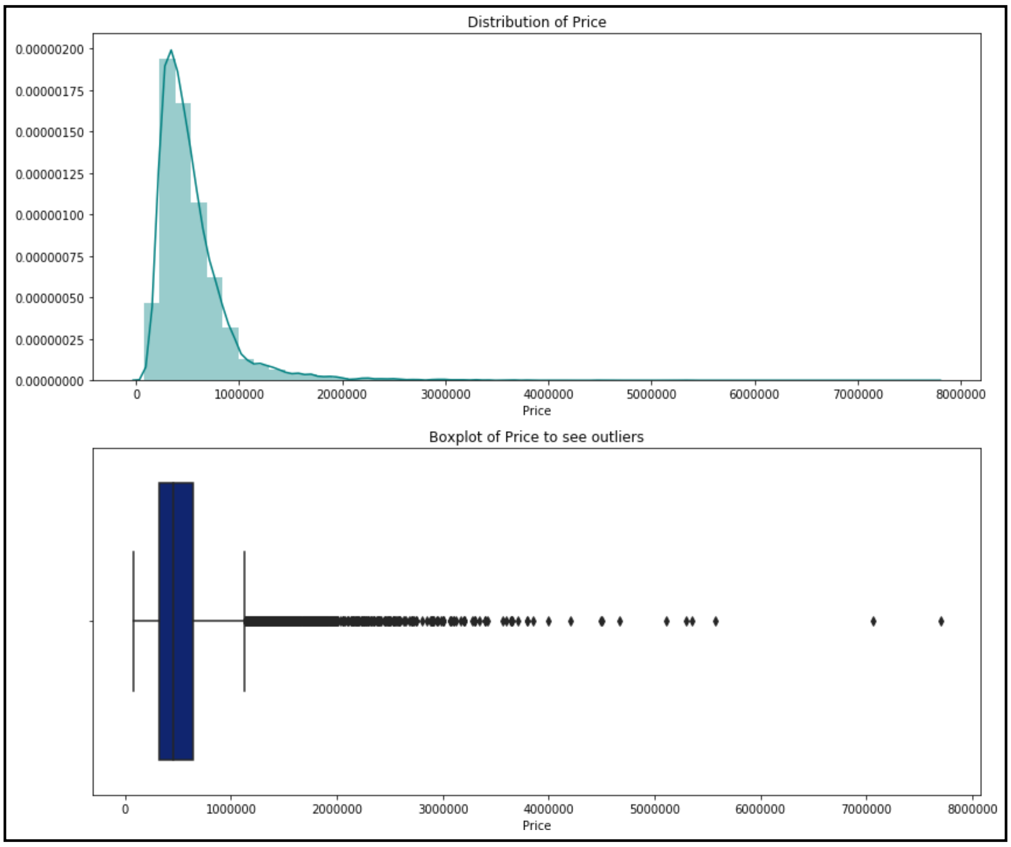
• Univariate analysis(Continuous Variables)

The map shows the distribution of houses by pincode in King County, Washington.



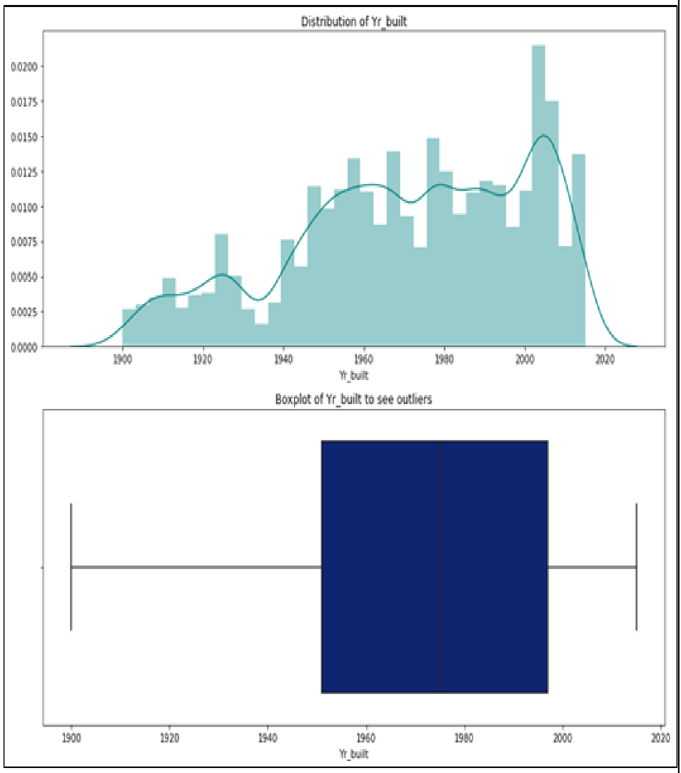
Now we do the Univariate analysis for the distribution of Continuous variables for which we have defined a function that plots the distribution plot and box plot for the variables that we give as an input.

Price (Target variable) -



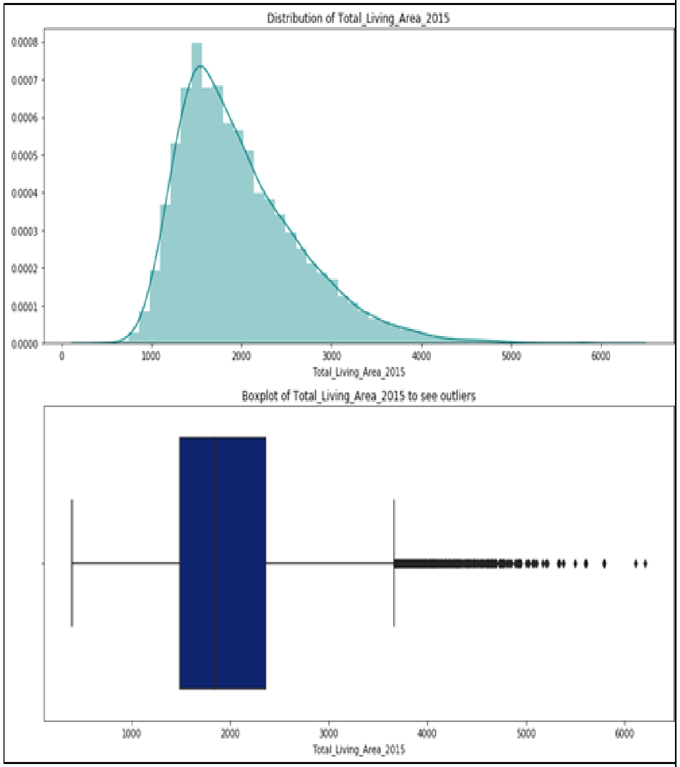
We can see that the data has outliers to the right of the box and is skewed positively.

Yr_built-



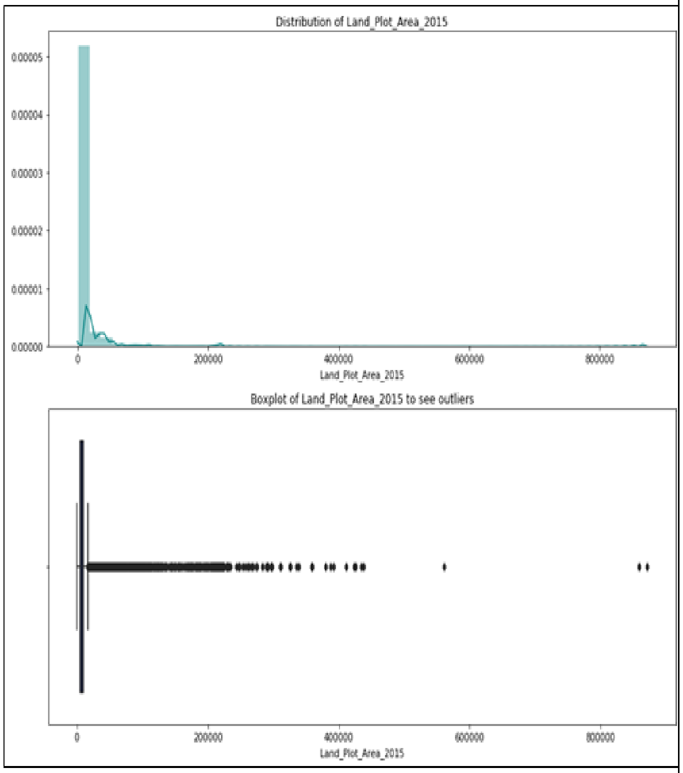
We can see that the data has no outliers and is slightly left skewed.

Total Living Area 2015-



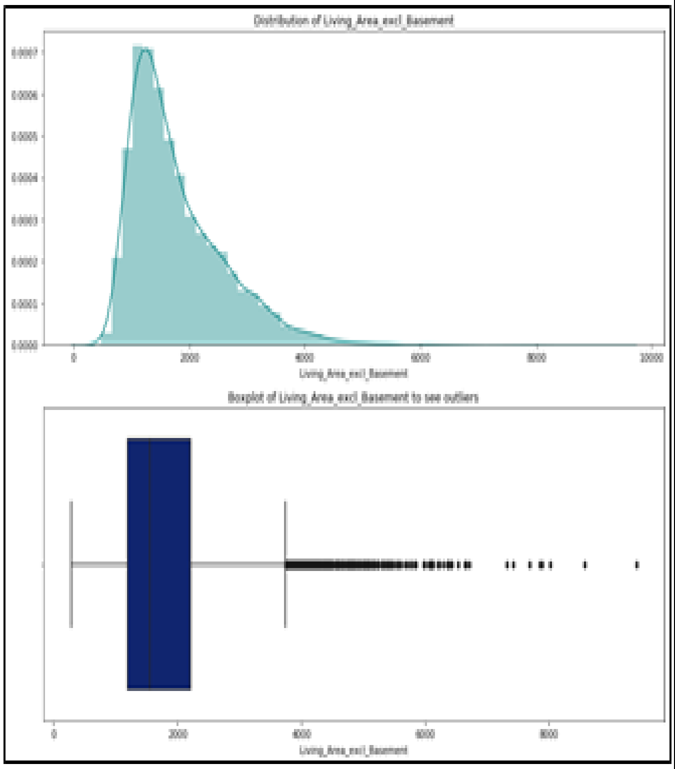
There are outliers to the right of the box and the data is slightly skewed positively.

Land Plot Area 2015-



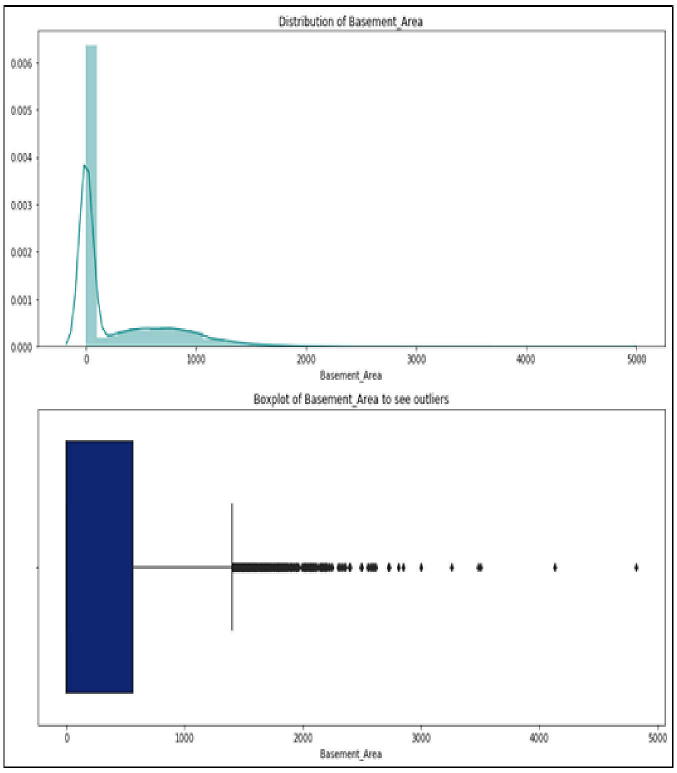
The data is highly positively skewed and hence we can see the outliers to the right of the box.

Living Area excl Basement-



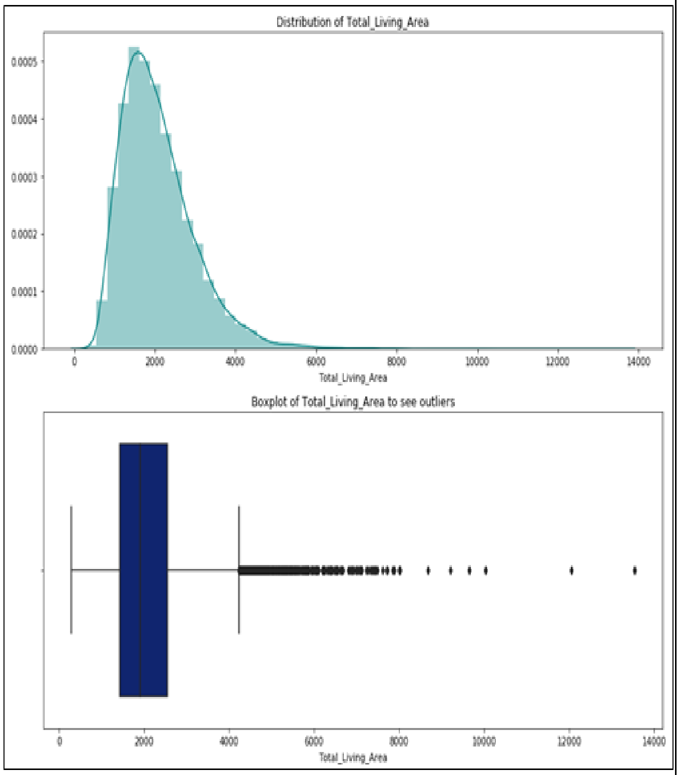
The data is positively skewed and has outliers to the tight of the box.

Basement Area-



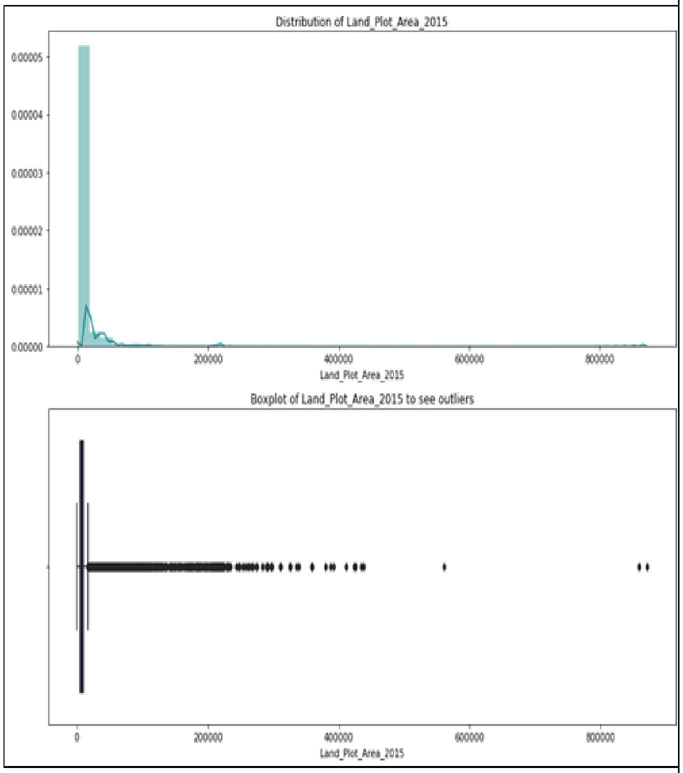
The data is positively skewed and there are outliers to the right of the box.

Total Living Area-



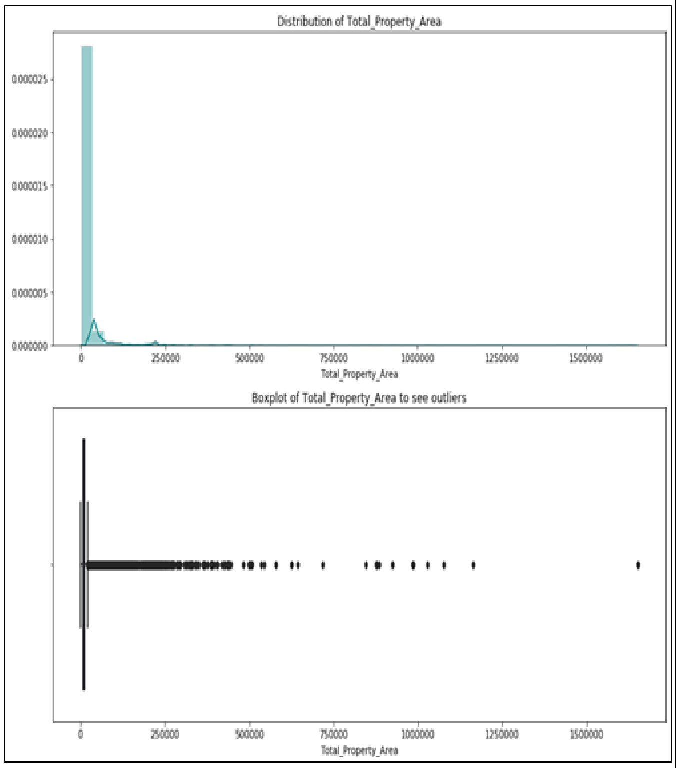
There are outliers to the right of the box and the data is slightly skewed positively.

Land Plot Area-



The data is highly positively skewed and hence we can see the outliers to the right of the box.

Total Property Area-

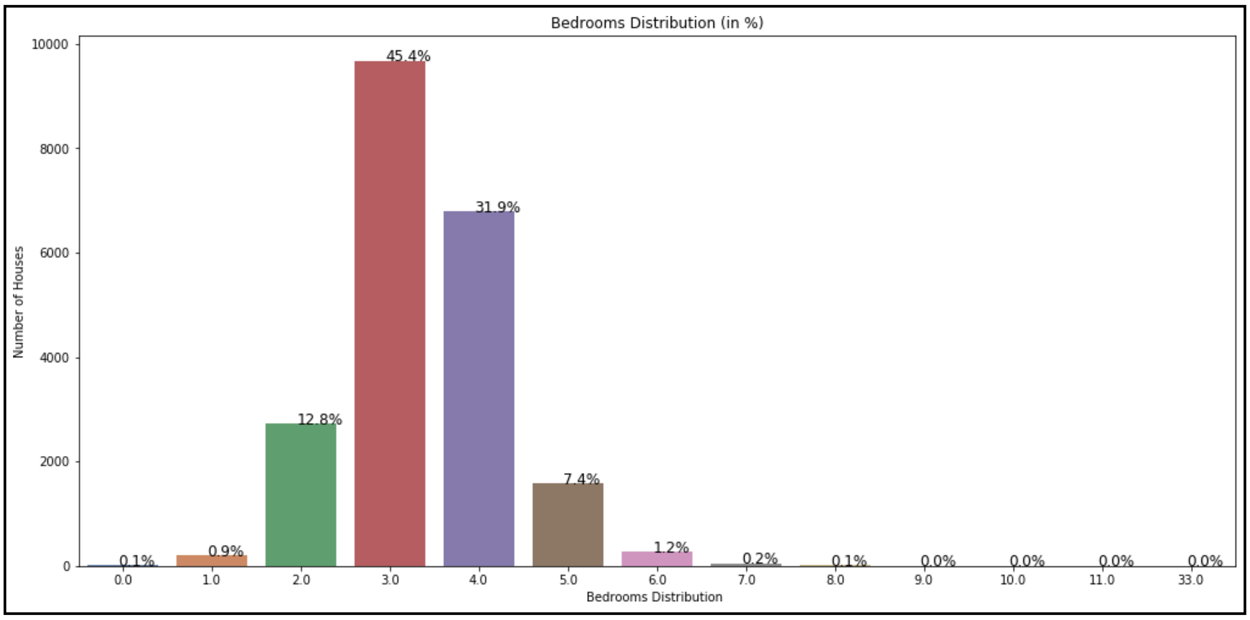


The data is highly skewed and we can see outliers to the right end of the box.

• Univariate analysis(Categorical Variables)

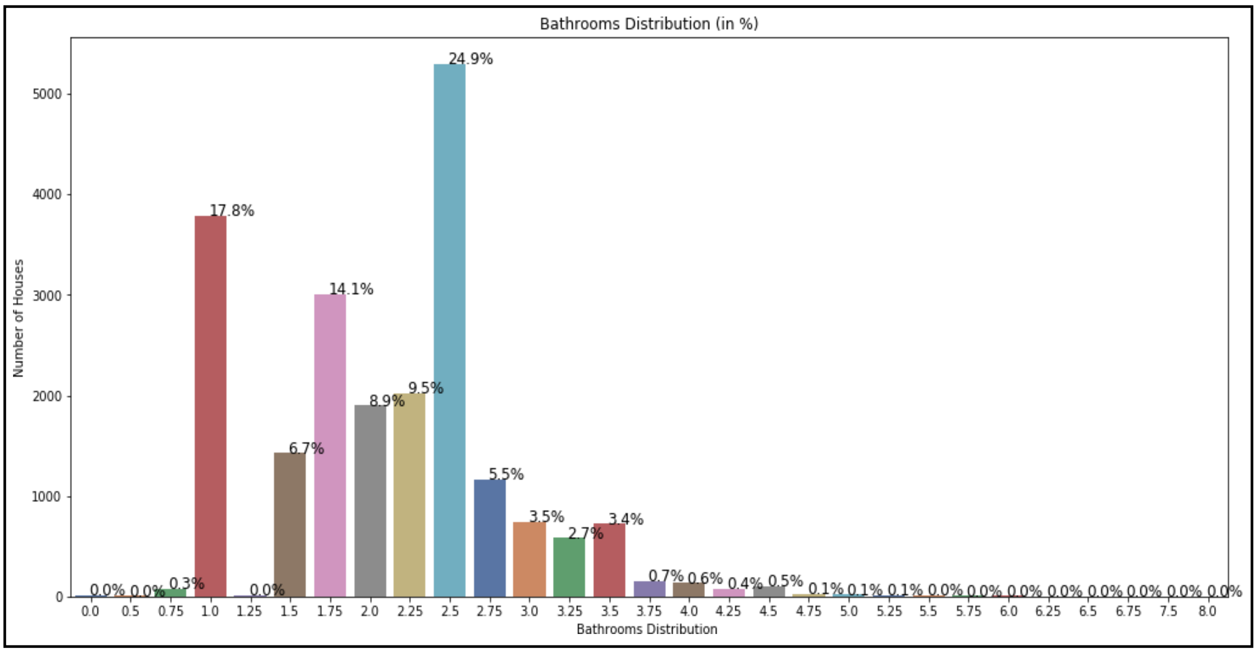
We have made a custom function for generating countplots for the categorical variables which will plot the graph and show us the proportion or distribution of data percentage wise for every category in that variable.

Bedrooms-



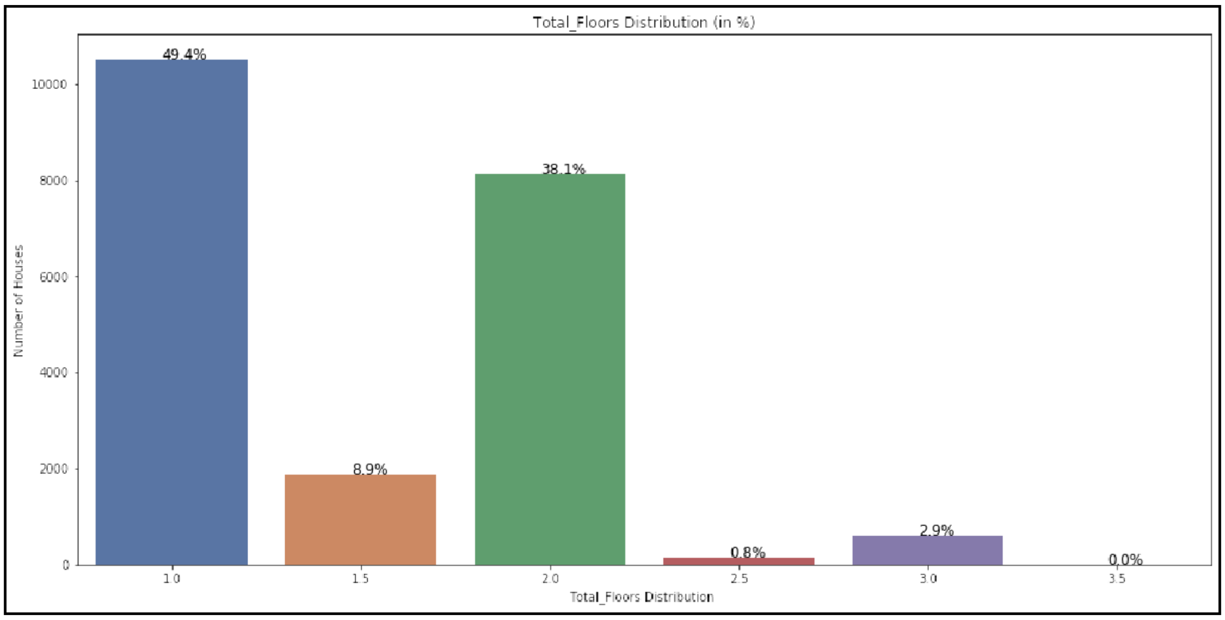
We can see that most of the houses i.e. ~77% have 3 or 4 Bedrooms and there are barely any houses with 7+ bedrooms or <1 bedrooms.

Bathrooms-



Majority of houses i.e. ~57% have either 1 or 1.75 or 2.5 Bathrooms, followed by 2.0 ,2.25 and 1.5 bathrooms. So, it seems that the people in Kings County either have different schedules.

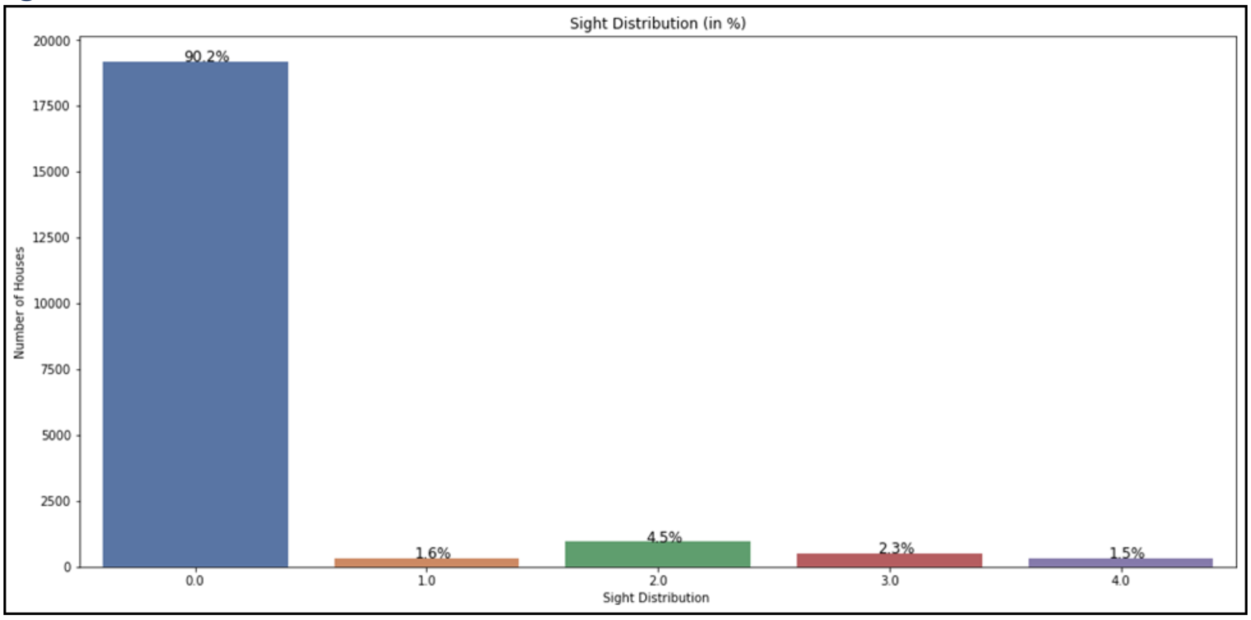
Total_Floors-



Total Floors	1	1.5	2	2.5	3	3.5
Number of Houses	10718	1905	8210	161	610	8

Most of the houses are either 2 floors or less than that. It is possible that there might be more elevation cost in construction or people don’t prefer multi-storey houses in the place which is clearly visible i.e. ~97% homes are 2 or <2 floors.

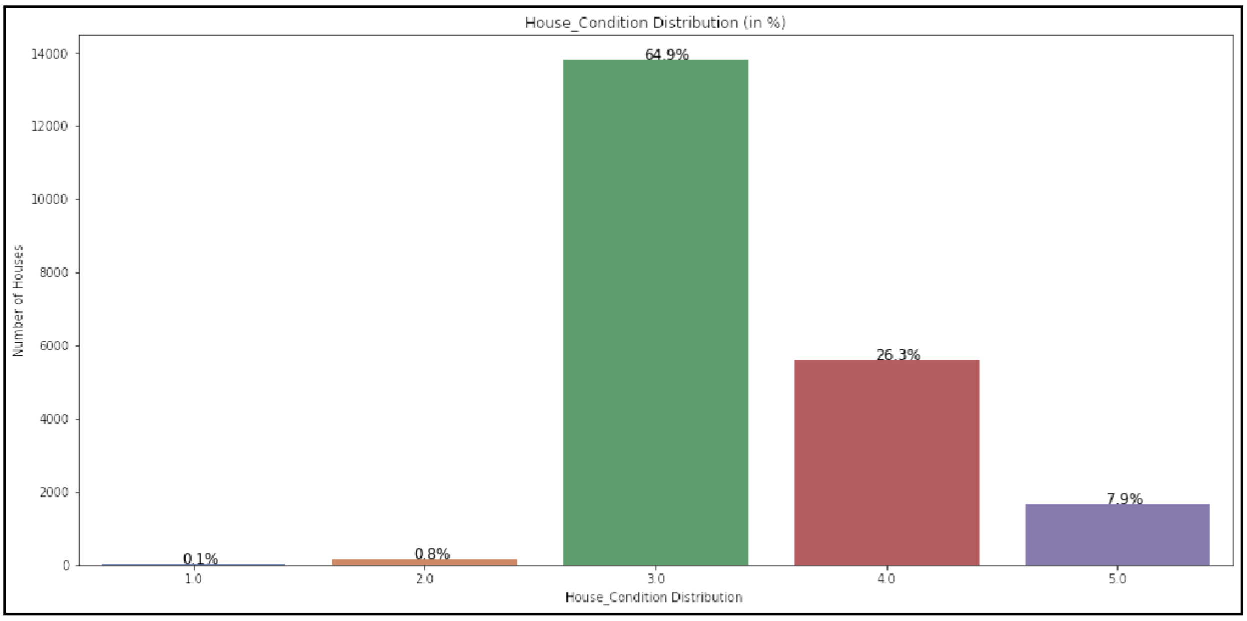
Sight-



Sight	Number of Houses
0	19489
1	335
2	960
3	510
4	318

Most of the houses i.e. ~90% haven’t been viewed. It seems that people have not been visiting the property but viewing them online or through pictures that might have been uploaded on the website.

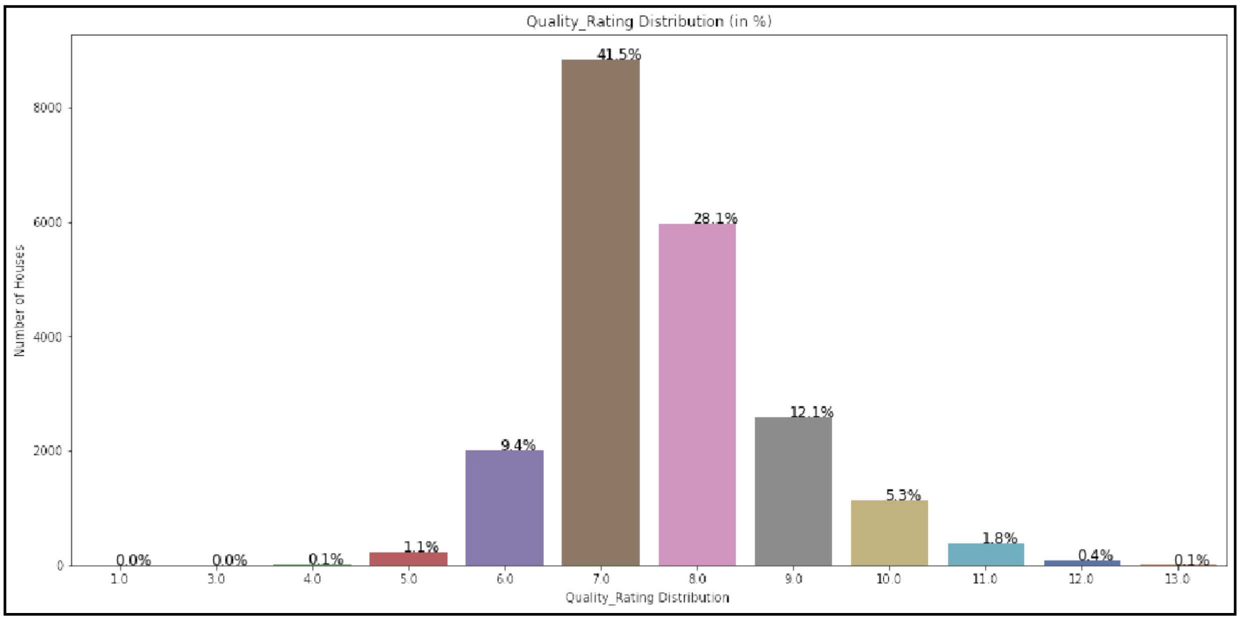
House_Condition-



HouseCondition	Number of Houses
1	30
2	171
3	14043
4	5674
5	1694

The house condition of majority of houses are better than average. Almost 65% houses are 3/5 and the remaining ones are even better i.e. either 4/5 or 5/5.

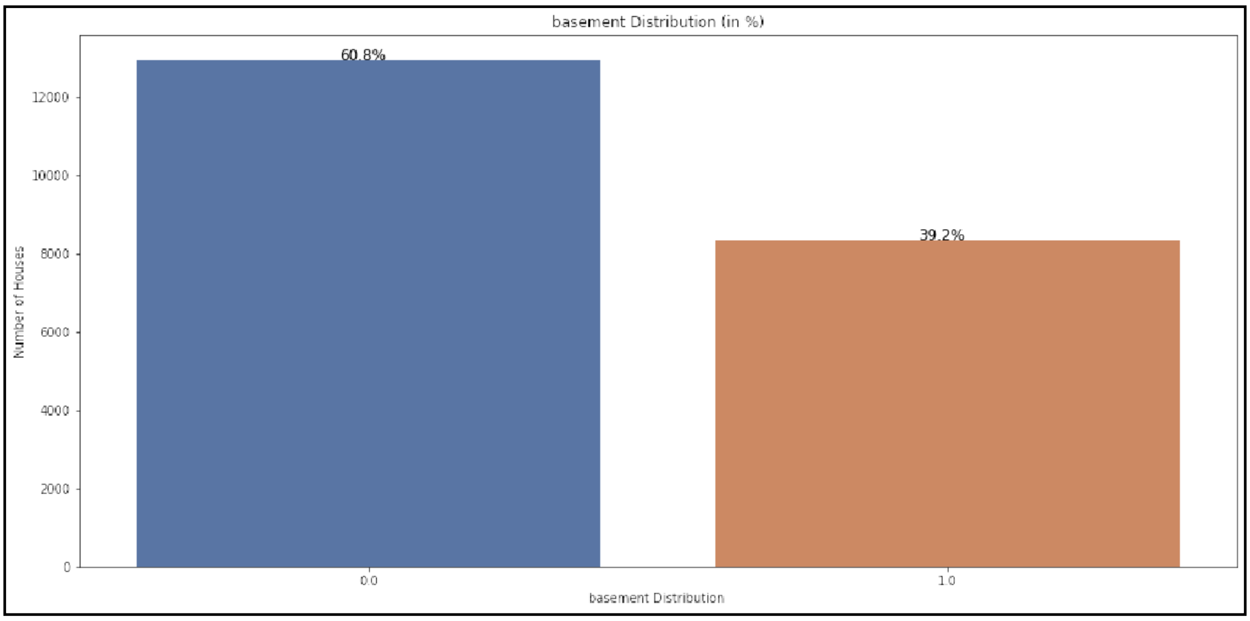
Quality_Rating-



QualityRating	1	3	4	5	6	7	8	9	10	11	12	13
Number of Houses	1	3	29	242	2038	8981	6067	2615	1134	399	90	13

Most of the houses have a more than average quality rating , as we can see that ~80% houses have rating between 6 and 8. Some houses have an even better quality rating, so it is possible that those houses might have extra features and are costly.

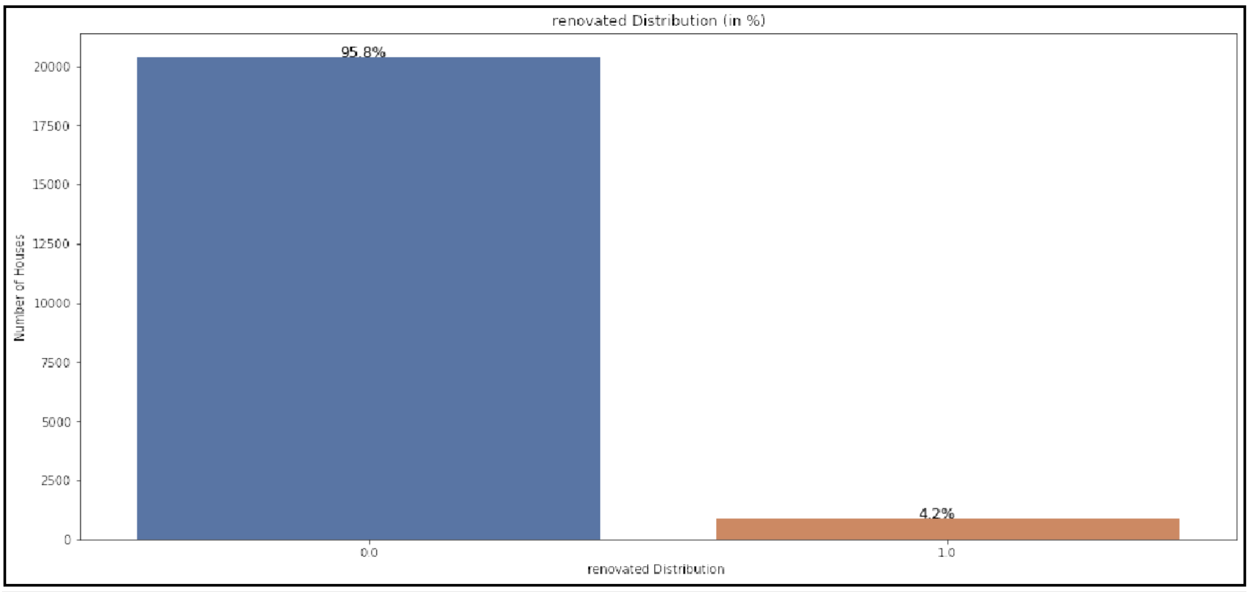
Basement-



Has_Basement	Number of Houses
No	13125
Yes	8487

Around 60% of the houses don't have a basement, whilst 40% houses have a basement. It is possible that the houses that have a basement are distant from the downtown or markets and people store their goods or non-daily products in the basements.

Renovation-



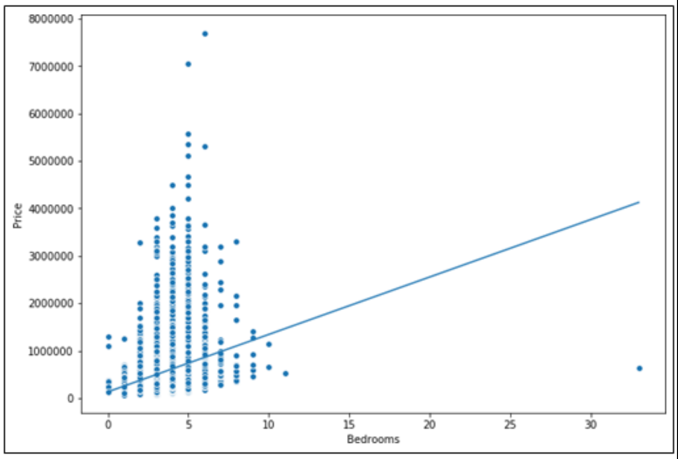
Is Renovated	Number of Houses
No	20698
Yes	914

Almost 96% of the houses aren't renovated, so it is possible that the 4% that were renovated were too old or had to be sold very quickly and hence were renovated.

• Bivariate and Multivariate Analysis

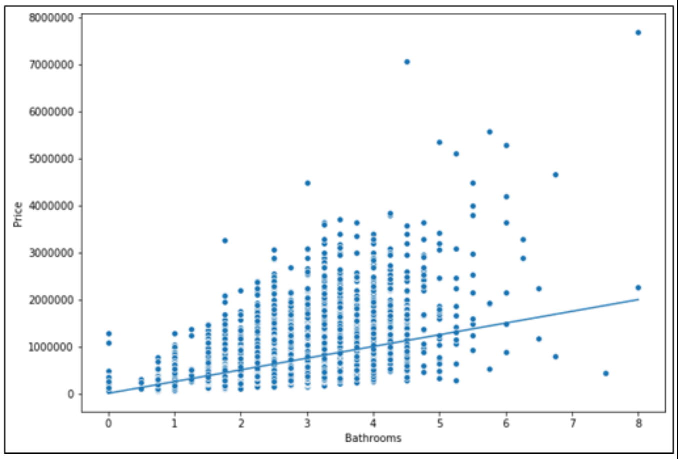
We have defined a function that plots the scatterplot for the variables that we give as an input against our Target variable ‘Price’. We have also used polyfit() to plot the trend using the coefficients of the variables and shows the direction or the trend.

Bedrooms-



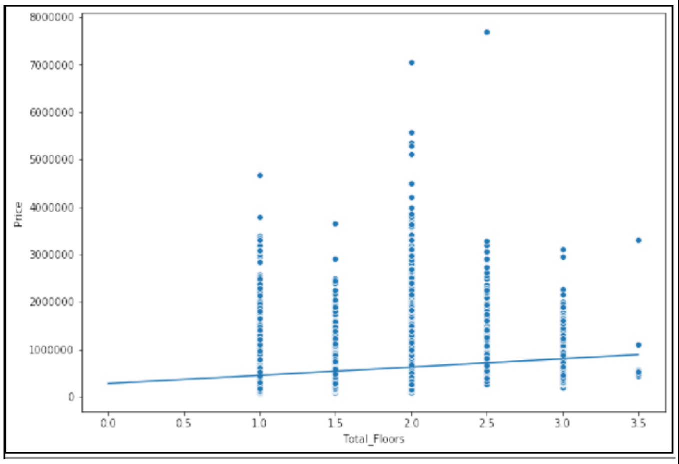
We can see a linear increase in Price with increase in number of bedrooms. The only one outlier i.e. 33 rooms.

Bathrooms-



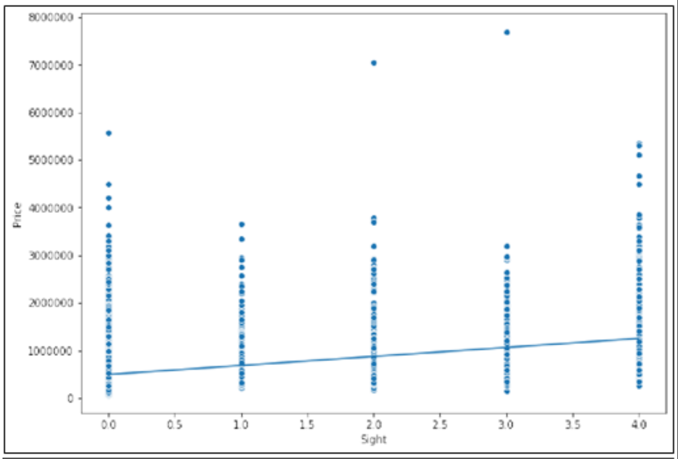
Even in bathrooms, there is a linear upward trend in Price.

Total Floors-



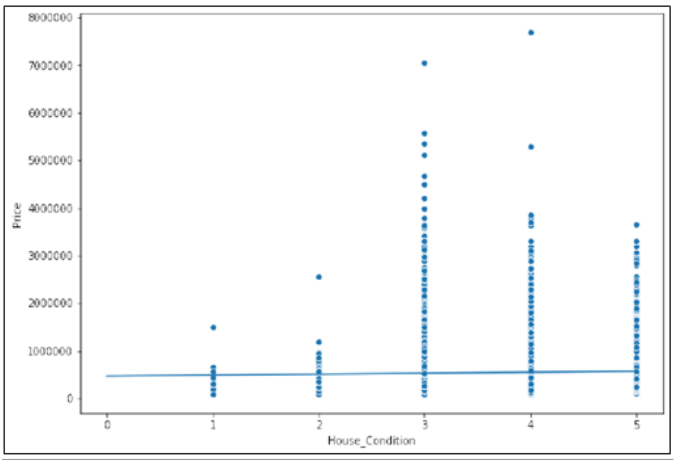
The number of floors has the slightest of impact on the Price directly as we can see that trend is similar of all the categories.

Sight-



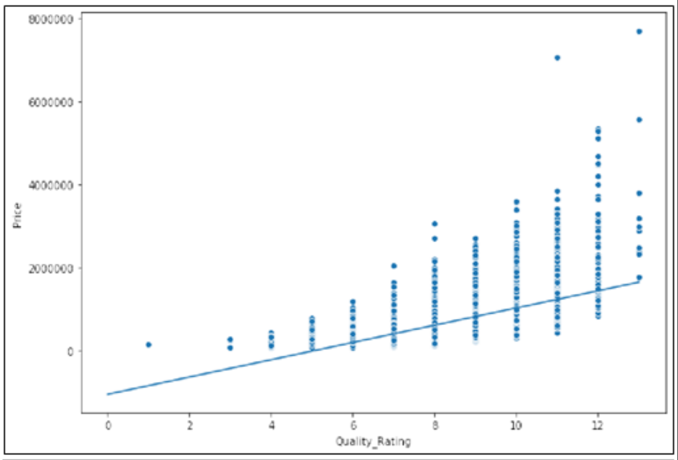
The sight has the some impact on the Price directly as we can see that trend is slightly upward across all the categories.

House Condition-



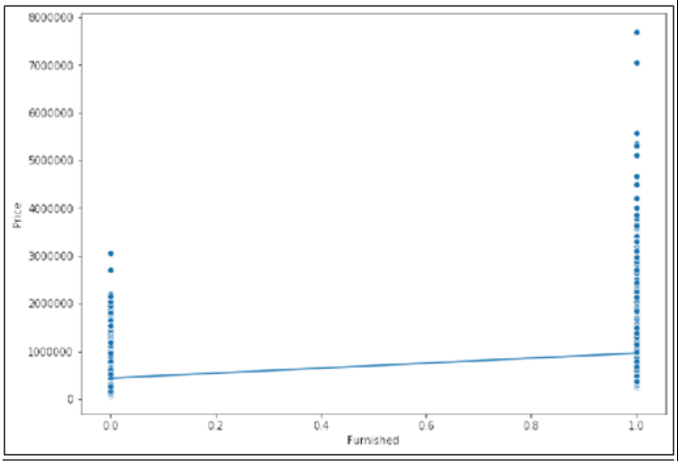
We cannot see direct relation or impact of house condition on Price.

Quality Rating-



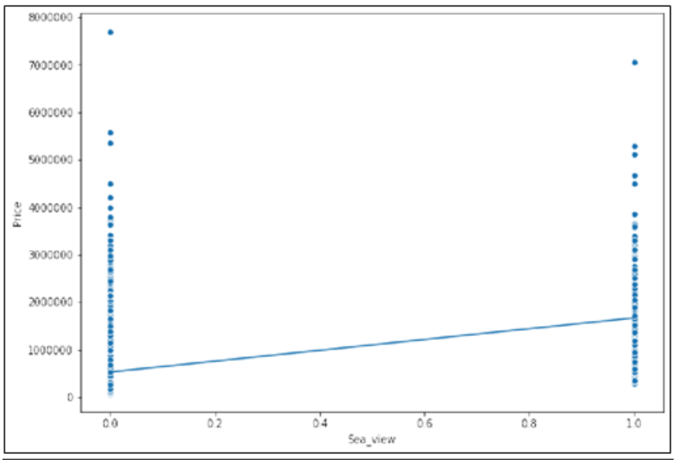
We can notice that Rating has direct and good impact on Price as with increase in rating, the price moment is high and linearly proportional.

Furnished-



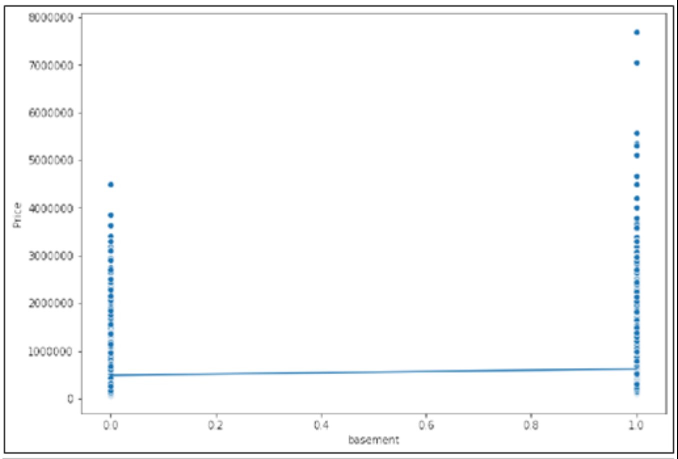
Furnished houses have higher prices.

Sea view-



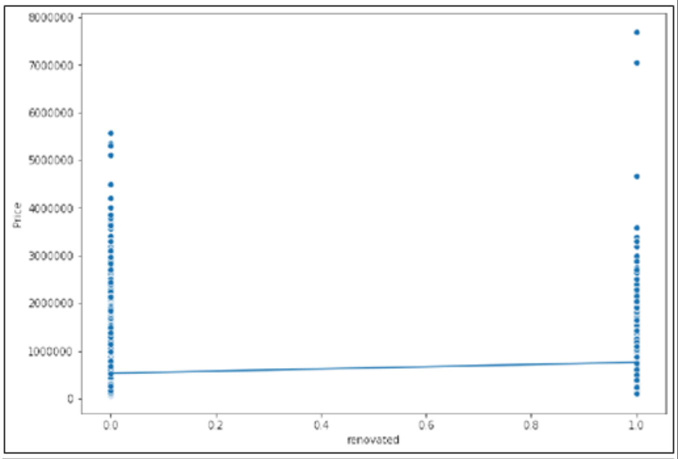
The houses having sea view are costlier than the ones with no sea view.

Basement-



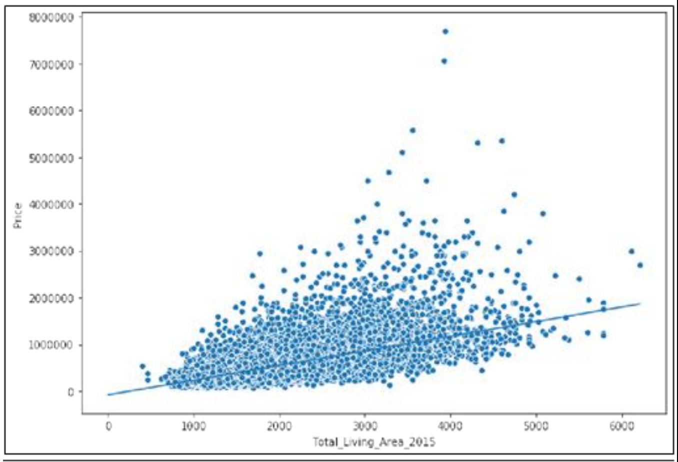
Basement has some slight influence on House Pricing.

Renovated-



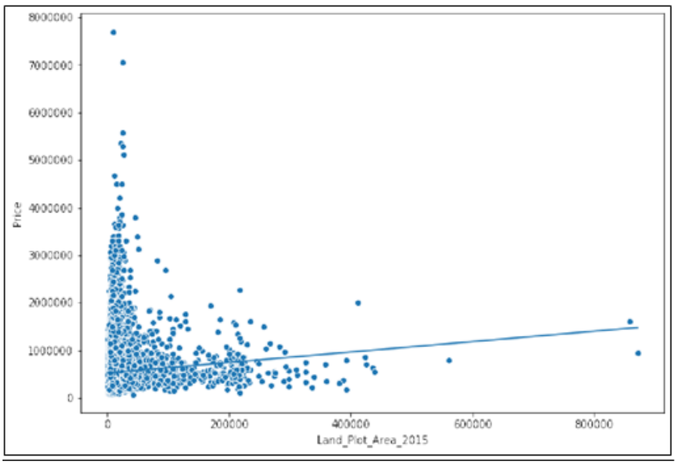
When the houses are renovated, the price increases slightly.

Total Living Area 2015-



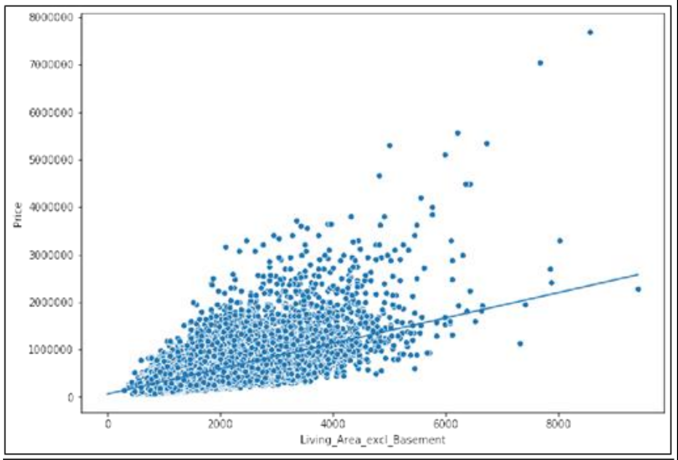
Total Living Area is linearly proportional to Price, as the area increases, the price also increases.

Land Plot Area 2015-



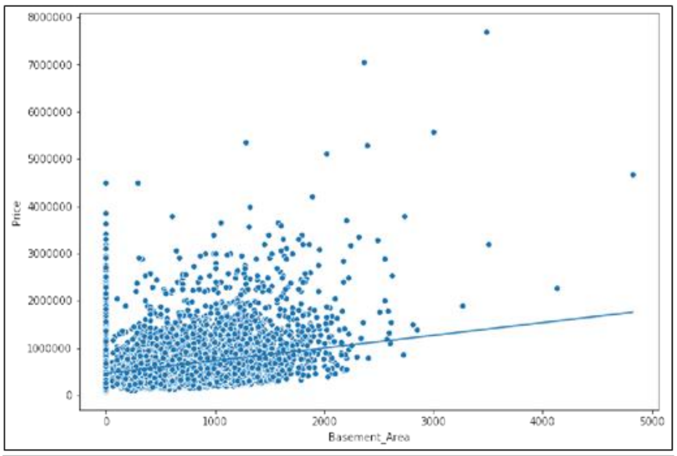
Total LandPlot Area is linearly proportional to Price, as the area increases, the price also increases.

Living Area excl Basement-



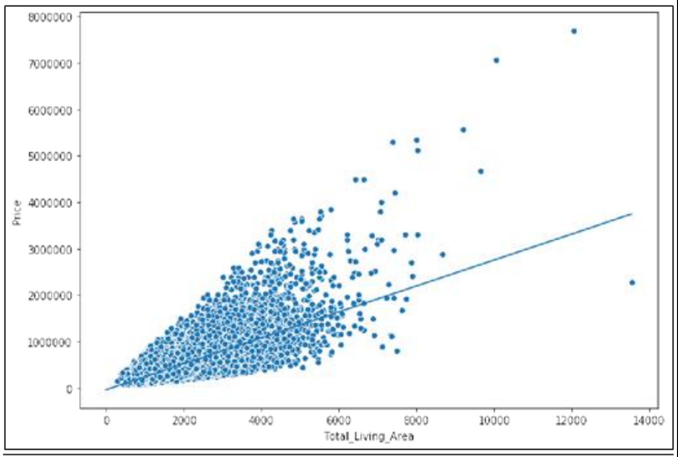
Living Area excl Basement is linearly proportional to Price, as the area increases, the price also increases.

Basement Area-



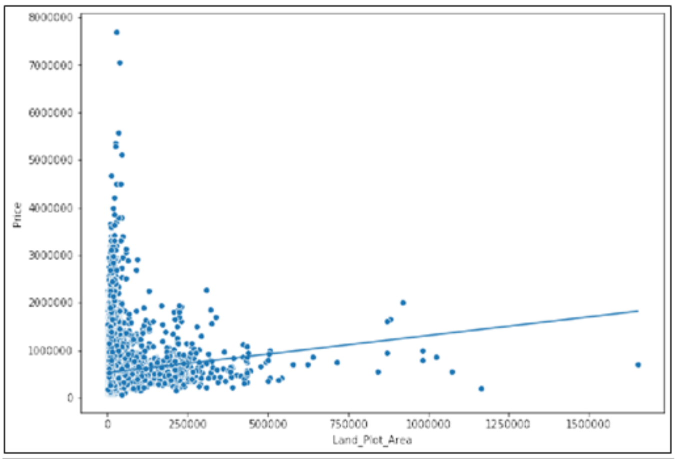
Basement Area is linearly proportional to Price, as the area increases, the price also increases.

Total Living Area-



Total Living Area is linearly proportional to Price, as the area increases, the price also increases.

Land Plot Area-



Total Land Plot Area is linearly proportional to Price, as the area increases, the price also increases.

• Multivariate Analysis:

<u>Correlation Plot-</u>

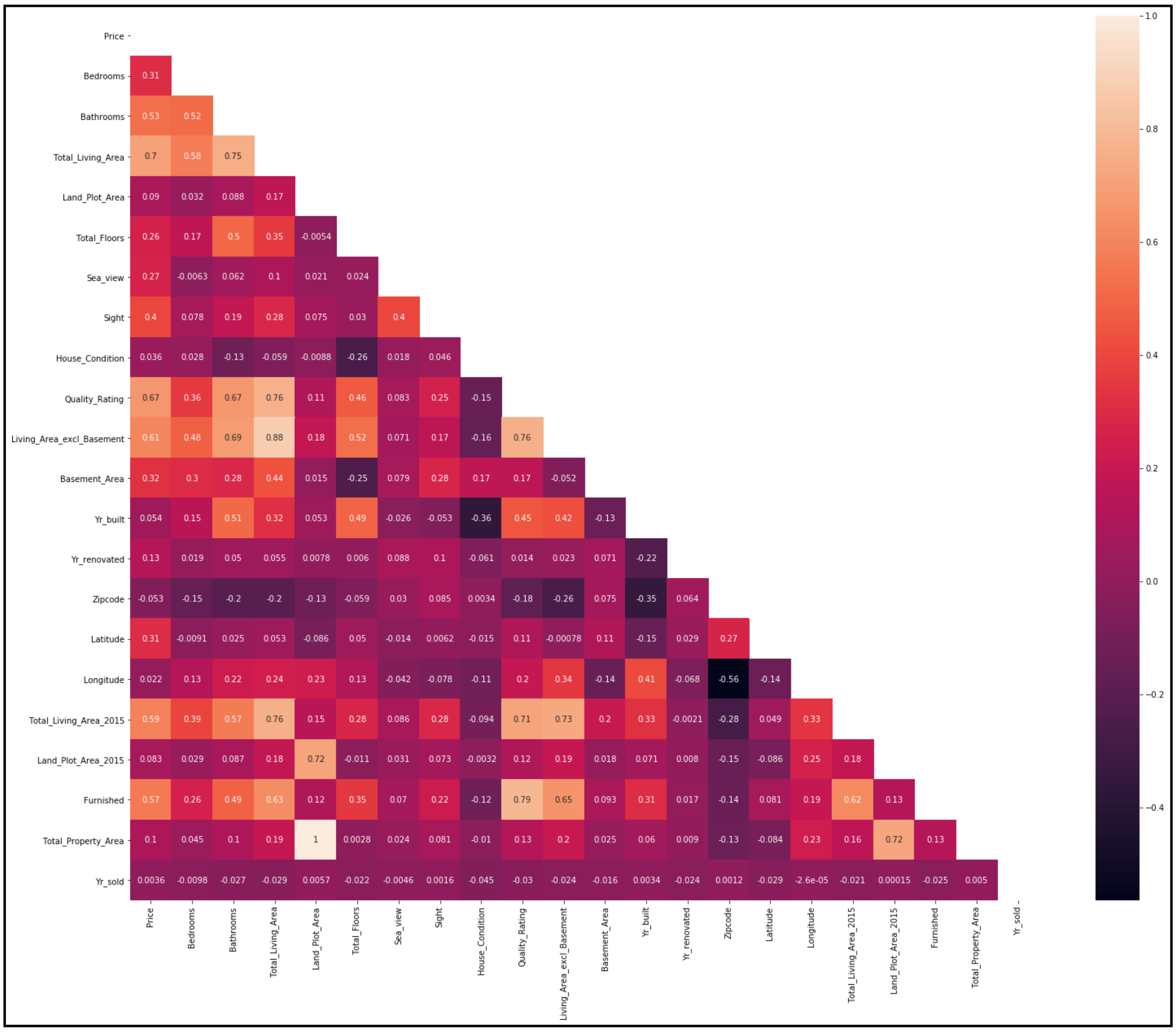
We calculate the linear relationship i.e. correlation among the variables using the corr()

and then plot it using a heatmap.

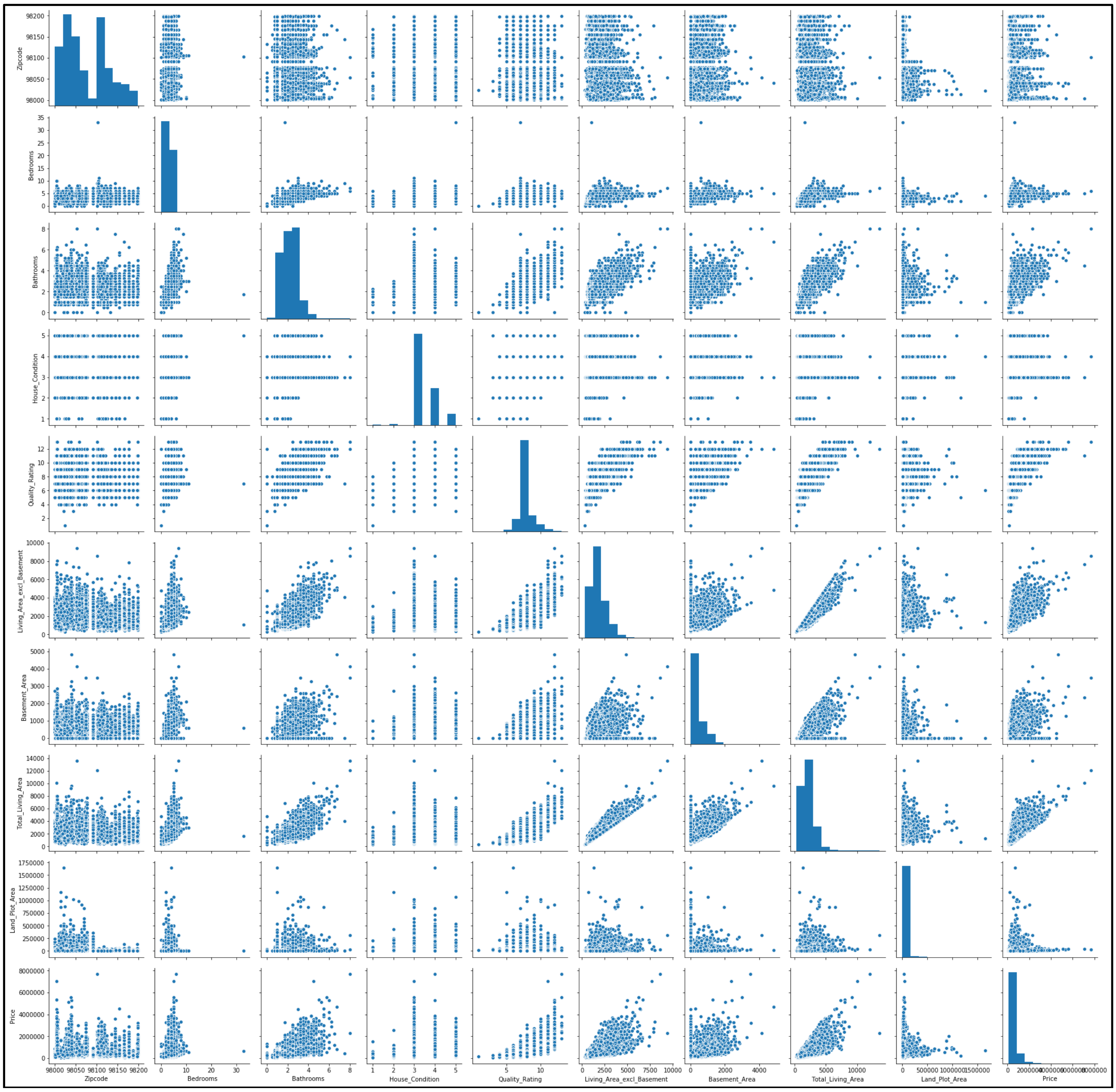
We can see that the variables are not very strongly correlated except for Total_Propert_Area and Land_Plot_Area, that means that either of the variables are strongly dependant on the other, the change in one will definitely influence the proportional change in the other.

Some correlations like Living_Area_excl_Basement and Total_Living_Area have correlation that are reasonably strong to influence the other. This is because Total Living Area= Living_Area_excl_Basement + Basement_Area, and Total_Property_Area = Total_Living_Area

+ Land_Plot_Area. There are some variables which have somewhat negative correlations like Zipcode and Yr_built and Yr_built and House_Condition, etc.

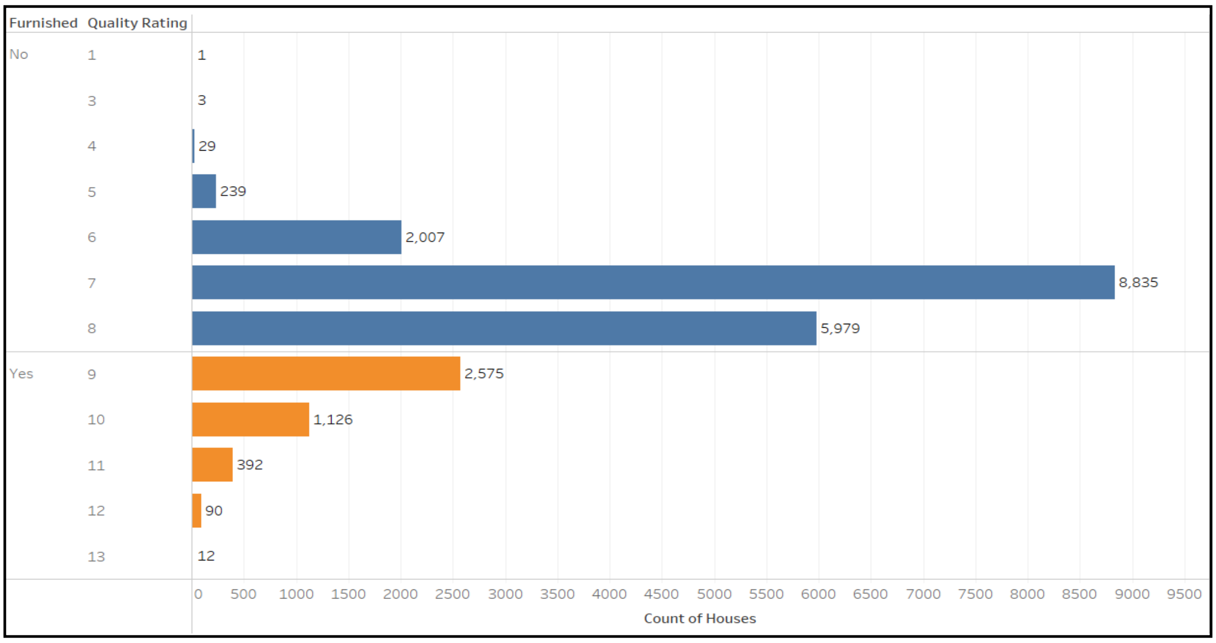


Pairplot to show relation and distribution among variables-



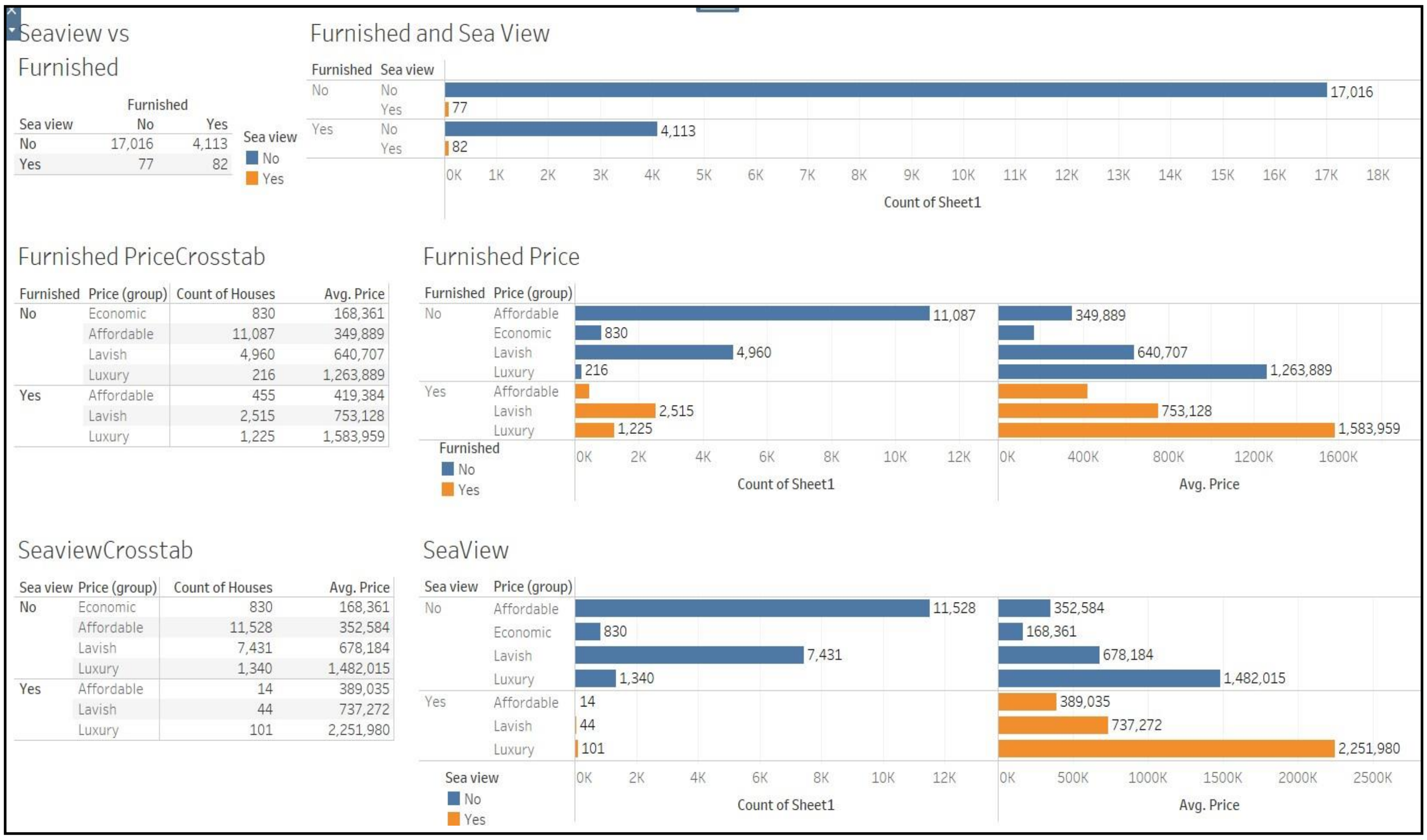
The pairplot shows the distribution of variables with respect to another variables in the dataset. It is a concatenation of scatterplots of different variables and how they are distributed.

Influence of Furniture on Quality Rating-



We can see that furniture has direct influence on Price and Quality Ratings, if the houses are furnished, they have higher Quality Ratings.

Impact of SeaView and Furniture on House Price-


Seaview vs Furnished

Sea view	Furnished No	Furnished Yes
No	17,016	4,113
Yes	77	82

Furnished PriceCrosstab

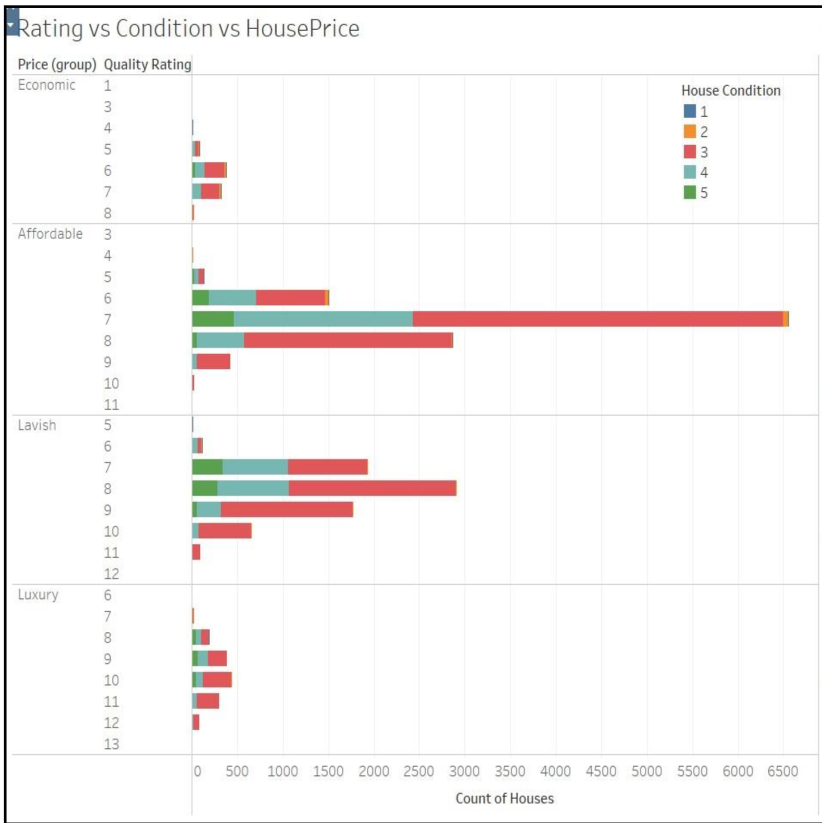
Furnished	Price (group)	Count of Houses	Avg. Price
No	Economic	830	168,361
	Affordable	11,087	349,889
	Lavish	4,960	640,707
	Luxury	216	1,263,889
Yes	Affordable	455	419,384
	Lavish	2,515	753,128
	Luxury	1,225	1,583,959

SeaviewCrosstab

Sea view	Price (group)	Count of Houses	Avg. Price
No	Economic	830	168,361
	Affordable	11,528	352,584
	Lavish	7,431	678,184
	Luxury	1,340	1,482,015
Yes	Affordable	14	389,035
	Lavish	44	737,272
	Luxury	101	2,251,980

We can notice that the houses with sea view and furniture are costlier, and their categories of price range vary accordingly. Majority of houses don’t have a sea view and the houses with sea view have higher average prices, similarly, the houses with furniture have higher average prices. Houses without furniture are affordable.

Influence of HouseCondition and QualityRAting on House Price-



Condition

House Condition	Count of Houses	Avg. Price
1	29	326,998
2	171	326,423
3	13,820	542,168
4	5,595	520,585
5	1,673	612,324

Rating

Quality Rating	Count of Houses	Avg. Price
1	1	142,000
3	3	205,667
4	29	214,381
5	239	247,693
6	2,007	302,066
7	8,835	402,403
8	5,979	542,266
9	2,575	773,602
10	1,126	1,073,031
11	392	1,495,801
12	90	2,192,500
13	12	3,445,833

Rating vs Condition

	House Condition				
Quality Rating	1	2	3	4	5
1	1				
3		1	1		1
4	1	5	13	10	
5	8	15	98	84	34
6	11	58	1,013	680	245
7	6	75	5,148	2,786	820
8	2	13	4,207	1,374	383
9		2	2,013	439	121
10		2	916	153	55
11			327	54	11
12			74	13	3
13			10	2	

The houses with higher quality rating have higher prices, conditions do impact the price but we can see that majority of houses have a moderate house condition. We can see the statistics in the above image.

• Removing of Unwanted variables

We remove cid, House_Sold_Date and Total_Property_Area from the dataset because :

- cid is just like a primary key or unique identifier and does not influence the model anyways.
- we have already created Yr_sold from House_Sold_Date taking only the year part, hence this variable is of no use anymore and
- Total_Property_Area is summation of Total_Living_Area and Land_Plot_Area and has a perfect correlation with Land_Plot_Area which may later be a problem when we prepare the prediction model.

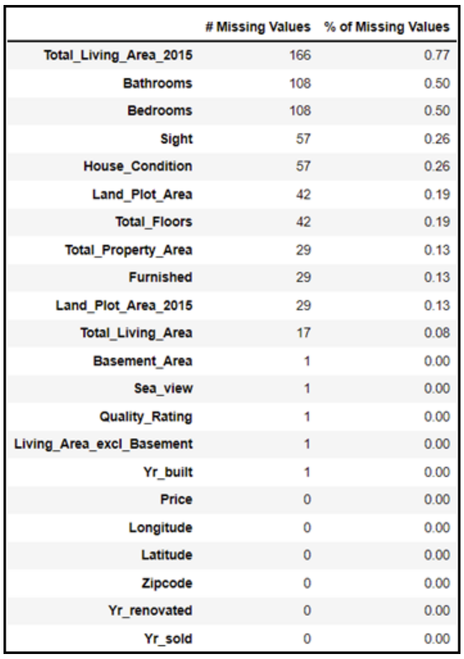
So we use the drop(axis=1,inplace=True) function to remove these columns as axis=1 means columns and inplace=True will make the deletion persistent.

• Outlier Treatment

When we looked at the boxplots, we could see the outliers, but since most of those values seemed legitimate, we can ask the client if those values are true. Considering that these values despite being outliers are true, we choose not to go with the outlier treatment.

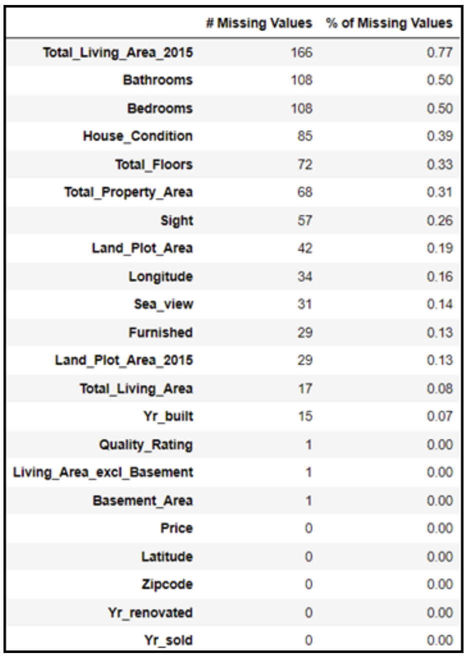
• Missing Value Treatment

There were many missing values along with some incorrect values like ‘$’ in the data : 12 variables have missing values.



	# Missing Values	% of Missing Values
Total_Living_Area_2015	166	0.77
Bathrooms	108	0.50
Bedrooms	108	0.50
Sight	57	0.26
House_Condition	57	0.26
Land_Plot_Area	42	0.19
Total_Floors	42	0.19
Total_Property_Area	29	0.13
Furnished	29	0.13
Land_Plot_Area_2015	29	0.13
Total_Living_Area	17	0.08
Basement_Area	1	0.00
Sea_view	1	0.00
Quality_Rating	1	0.00
Living_Area_excl_Basement	1	0.00
Yr_built	1	0.00
Price	0	0.00
Longitude	0	0.00
Latitude	0	0.00
Zipcode	0	0.00
Yr_renovated	0	0.00
Yr_sold	0	0.00

Now we converted ‘$’ as NaN or null values using replace() function and checked the missing values again. 13 variables have missing values now and we can see increase in missing values in some variables because of $ being converted to nulls.

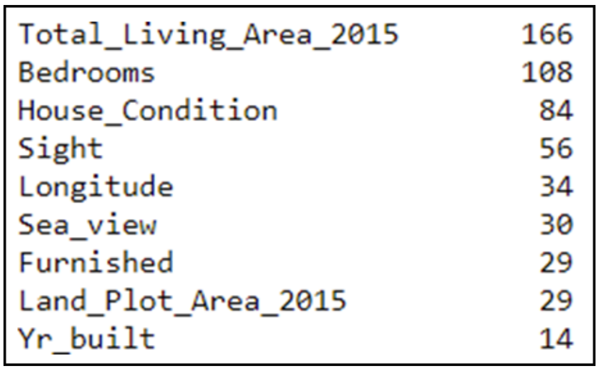


	# Missing Values	% of Missing Values
Total_Living_Area_2015	166	0.77
Bathrooms	108	0.50
Bedrooms	108	0.50
House_Condition	85	0.39
Total_Floors	72	0.33
Total_Property_Area	68	0.31
Sight	57	0.26
Land_Plot_Area	42	0.19
Longitude	34	0.16
Sea_view	31	0.14
Furnished	29	0.13
Land_Plot_Area_2015	29	0.13
Total_Living_Area	17	0.08
Yr_built	15	0.07
Quality_Rating	1	0.00
Living_Area_excl_Basement	1	0.00
Basement_Area	1	0.00
Price	0	0.00
Latitude	0	0.00
Zipcode	0	0.00
Yr_renovated	0	0.00
Yr_sold	0	0.00

Looking at the data we identified that Total Property Area = Land Plot Area + Total Living Area, also Total Living Area = Living Area excl Basement + Basement Area. So we impute those missing values using the fillna() using the same math calculation.

Since there is only one missing value for Quality_Rating, Living Area excl Basement and Basement Area, we saw that all of them were in the same row i.e. index=21226, so we dropped the row.

Now for the categorical column that had decima values i.e. Bathrooms and Total Floors, we have imputed the missing values using mode of column as the missing values.



```
Total_Living_Area_2015       166
Bedrooms                     108
House_Condition               84
Sight                         56
Longitude                     34
Sea_view                      30
Furnished                     29
Land_Plot_Area_2015           29
Yr_built                      14
```

Now we have only these as missing values, so we use the KNN Imputer from sklearn.impute library and do the KNN imputation for these missing values and then round them off. KNN Imputation is a technique that imputes the value based on k-nearest neighbors clustering, so we gave the input as 200 neighbors to impute the missing data.

After imputing the data, we use the round function, since these values aren't float but integer values, hence we round them off. Now we have imputed all the missing values.

• Addition of new variables

We have just added Basement and Renovated variables for doing the EDA but won't include them while preparing the model. All the houses with basement area > 0 has basements and all the houses with renovation year != 0 have been renovated. Also we have created a new variable Yr_sold from House_year_sold using the datepart() function and using the year from it.

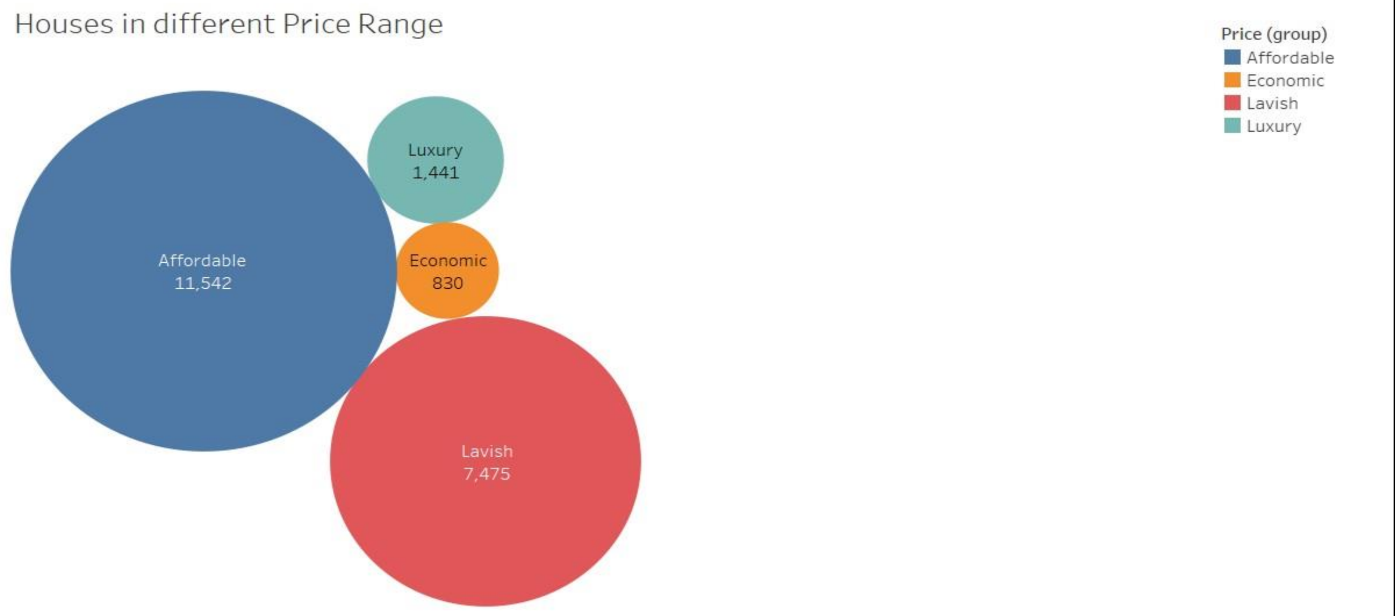
BUSINESS INSIGHTS FROM EDA

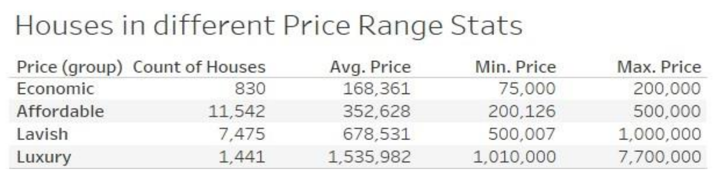
• Is the data unbalanced? If so, what can be done? Please explain in the context of the business

In our case, since we need to do regression and not classification, the data balance is out of context. Hence, we do not need to check the balance of the data.

• Any business insights using clustering

Price (Target Variable)-





Houses in different Price Range Stats

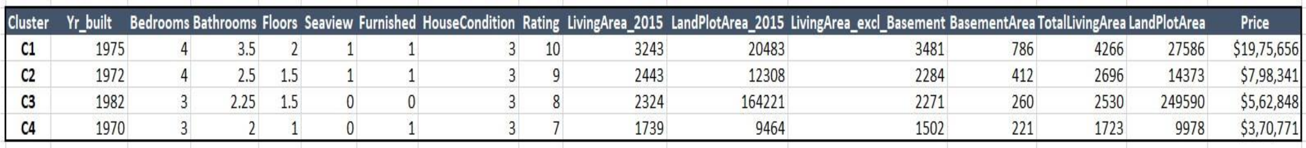
Price (group)	Count of Houses	Avg. Price	Min. Price	Max. Price
Economic	830	168,361	75,000	200,000
Affordable	11,542	352,628	200,126	500,000
Lavish	7,475	678,531	500,007	1,000,000
Luxury	1,441	1,535,982	1,010,000	7,700,000

The target variable is Price, it is a continuous variable and we have seen its distribution in a distribution plot and a box plot above. But now, we have made 4 classes in the variable where –

- Price less than equal to $ 200,000 are classified as Economic houses,
- Price between $200,001 and $500,000 are classified as Affordable houses
- Prices between $500,001 and $1,000,000 are classified as Lavish houses
- Price greater than equal to $ 1,000,001 are classified as Luxury houses.

So, we observe that majority of the records we have for the houses are either Affordable ones or Lavish houses. So, it is possible that the standard of living of people is really good.

We have also done clustering (K-Means) using KNIME – 4 clusters and based on that we get the following data per feature (average and rounding-off) :



Cluster	Yr_built	Bedrooms	Bathrooms	Floors	Seaview	Furnished	HouseCondition	Rating	LivingArea_2015	LandPlotArea_2015	LivingArea_excl_Basement	BasementArea	TotalLivingArea	LandPlotArea	Price
C1	1975	4	3.5	2	1	1	3	10	3243	20483	3481	786	4266	27586	$19,75,656
C2	1972	4	2.5	1.5	1	1	3	9	2443	12308	2284	412	2696	14373	$7,98,341
C3	1982	3	2.25	1.5	0	0	3	8	2324	164221	2271	260	2530	249590	$5,62,848
C4	1970	3	2	1	0	1	3	7	1739	9464	1502	221	1723	9978	$3,70,771

We can see that the cluster distribution is similar to the categorization of Price Category we did earlier in the visualization. We can see that the as the Price goes higher, the number of Bedrooms, Bathrooms increase, the quality ratings and conditions get better, also the Lavish and Luxury house offers seaView for some houses compared to affordable or economic. The living area and basement areas are more in the costlier houses. An odd observation that we can see is that there is more LandPlotArea in the C3 i.e. Affordable cluster, so it is possible that those houses might be located a little away from the cities.

• Any other business insights

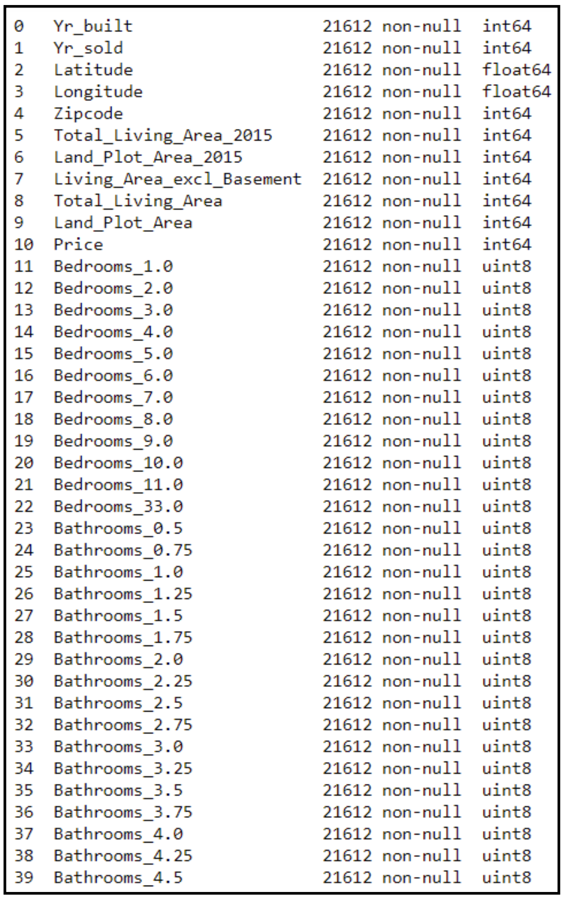
So if we want the price of house to be more, it has to be furnished and have a sea view or water front. Also the number of bathrooms should be more. The areas definitely impact the pricing but the quality rating impacts the housing the most. Also, majority of houses don’t have a sea view and have a moderate house condition.

From a buyers point of view, the houses that don’t have a basement area are also good, since the living area is sufficient and the floors don’t matter much when it comes to buying a house, so the houses that are far from the city are bigger in size but affordable compared to the ones in the prime locations.

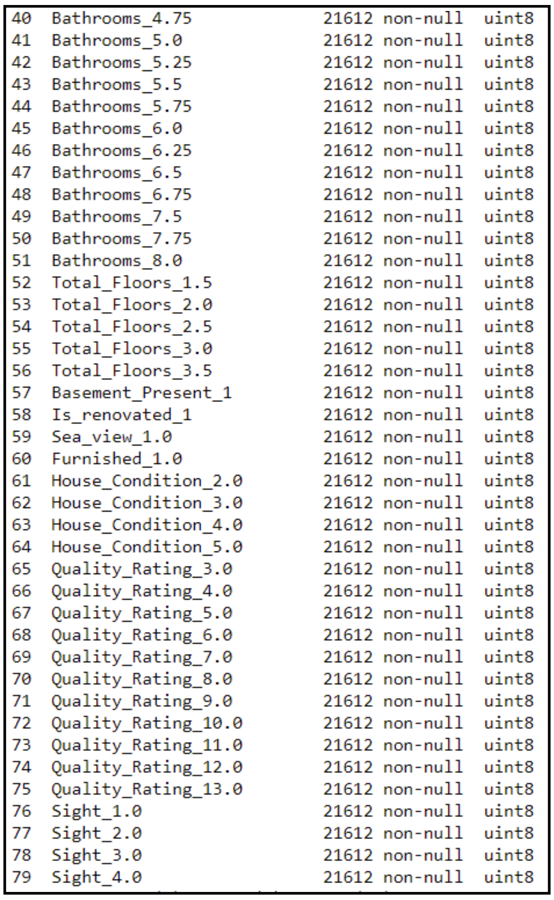
From sellers point of view, the houses should be renovated once and have high Quality Rating, furnished houses fetches some extra dollars and ratings, since they go hand in hand.

MODEL BUILDING AND INTERPRETATION

- In this project of predicting house prices in King County (Washington), the dataset we had contained 23 variables out of which some variables like cid didn't add any value to the model.
- Other variables like House_Sold_Date were less significant , so we extracted the Year part
- from it and added a column "Year sold".
- Similarly, Year_renovated didn't add any value to with respect to building a model, so we formulated a new column "Is renovated" which contains the value 0 for Not Renovated and 1 for Renovated.
- We did the same with Basement_Area to "Basement Present" where 1 meant that the
- house has Basement and 0 stands for Basement not present. We have also dropped "Total Property Area" as it had a perfect correlation with "Land Plot Area", otherwise we would get biased or incorrect results due to heavy collinearity among the variables.
- Now, for the categorical variables i.e. "Bedrooms", "Bathrooms", "Total Floors",
- "Basement Present", "Is renovated", "Sea view", "Furnished", "House Condition", "Quality Ratings" and "Sight", we do encoding using pd.get_dummies() to convert all the categories as unique variables. Since the machine learning algorithms cannot process categories as categories, we do encoding to use these categories as numeric representations of these categories and feed them as input to the models. We also drop the first category from all variables so that the it reduces the correlation among the created dummy variables. These are the columns on which we will build our models: -



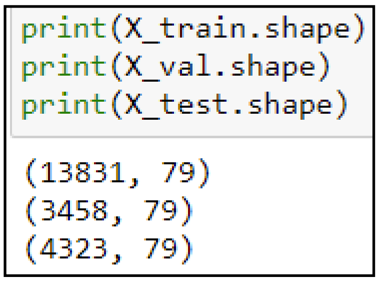
```
0   Yr_built                  21612 non-null  int64
1   Yr_sold                   21612 non-null  int64
2   Latitude                  21612 non-null  float64
3   Longitude                 21612 non-null  float64
4   Zipcode                   21612 non-null  int64
5   Total_Living_Area_2015    21612 non-null  int64
6   Land_Plot_Area_2015       21612 non-null  int64
7   Living_Area_excl_Basement 21612 non-null  int64
8   Total_Living_Area         21612 non-null  int64
9   Land_Plot_Area            21612 non-null  int64
10  Price                     21612 non-null  int64
11  Bedrooms_1.0              21612 non-null  uint8
12  Bedrooms_2.0              21612 non-null  uint8
13  Bedrooms_3.0              21612 non-null  uint8
14  Bedrooms_4.0              21612 non-null  uint8
15  Bedrooms_5.0              21612 non-null  uint8
16  Bedrooms_6.0              21612 non-null  uint8
17  Bedrooms_7.0              21612 non-null  uint8
18  Bedrooms_8.0              21612 non-null  uint8
19  Bedrooms_9.0              21612 non-null  uint8
20  Bedrooms_10.0             21612 non-null  uint8
21  Bedrooms_11.0             21612 non-null  uint8
22  Bedrooms_33.0             21612 non-null  uint8
23  Bathrooms_0.5             21612 non-null  uint8
24  Bathrooms_0.75            21612 non-null  uint8
25  Bathrooms_1.0             21612 non-null  uint8
26  Bathrooms_1.25            21612 non-null  uint8
27  Bathrooms_1.5             21612 non-null  uint8
28  Bathrooms_1.75            21612 non-null  uint8
29  Bathrooms_2.0             21612 non-null  uint8
30  Bathrooms_2.25            21612 non-null  uint8
31  Bathrooms_2.5             21612 non-null  uint8
32  Bathrooms_2.75            21612 non-null  uint8
33  Bathrooms_3.0             21612 non-null  uint8
34  Bathrooms_3.25            21612 non-null  uint8
35  Bathrooms_3.5             21612 non-null  uint8
36  Bathrooms_3.75            21612 non-null  uint8
37  Bathrooms_4.0             21612 non-null  uint8
38  Bathrooms_4.25            21612 non-null  uint8
39  Bathrooms_4.5             21612 non-null  uint8
```



```
40  Bathrooms_4.75            21612 non-null  uint8
41  Bathrooms_5.0             21612 non-null  uint8
42  Bathrooms_5.25            21612 non-null  uint8
43  Bathrooms_5.5             21612 non-null  uint8
44  Bathrooms_5.75            21612 non-null  uint8
45  Bathrooms_6.0             21612 non-null  uint8
46  Bathrooms_6.25            21612 non-null  uint8
47  Bathrooms_6.5             21612 non-null  uint8
48  Bathrooms_6.75            21612 non-null  uint8
49  Bathrooms_7.5             21612 non-null  uint8
50  Bathrooms_7.75            21612 non-null  uint8
51  Bathrooms_8.0             21612 non-null  uint8
52  Total_Floors_1.5          21612 non-null  uint8
53  Total_Floors_2.0          21612 non-null  uint8
54  Total_Floors_2.5          21612 non-null  uint8
55  Total_Floors_3.0          21612 non-null  uint8
56  Total_Floors_3.5          21612 non-null  uint8
57  Basement_Present_1        21612 non-null  uint8
58  Is_renovated_1            21612 non-null  uint8
59  Sea_view_1.0              21612 non-null  uint8
60  Furnished_1.0             21612 non-null  uint8
61  House_Condition_2.0       21612 non-null  uint8
62  House_Condition_3.0       21612 non-null  uint8
63  House_Condition_4.0       21612 non-null  uint8
64  House_Condition_5.0       21612 non-null  uint8
65  Quality_Rating_3.0        21612 non-null  uint8
66  Quality_Rating_4.0        21612 non-null  uint8
67  Quality_Rating_5.0        21612 non-null  uint8
68  Quality_Rating_6.0        21612 non-null  uint8
69  Quality_Rating_7.0        21612 non-null  uint8
70  Quality_Rating_8.0        21612 non-null  uint8
71  Quality_Rating_9.0        21612 non-null  uint8
72  Quality_Rating_10.0       21612 non-null  uint8
73  Quality_Rating_11.0       21612 non-null  uint8
74  Quality_Rating_12.0       21612 non-null  uint8
75  Quality_Rating_13.0       21612 non-null  uint8
76  Sight_1.0                 21612 non-null  uint8
77  Sight_2.0                 21612 non-null  uint8
78  Sight_3.0                 21612 non-null  uint8
79  Sight_4.0                 21612 non-null  uint8
```

• Build various models.

Now we use the train_test_split from sklearn.model_selection to split our train and test sets. The model learns on the train set and validates its results on the test set. So, in our case, we split 80% as train set data and 20% as test set data. We further split the train set into train and validation set in proportion of 80:20 as train set : validation set (similar to test set), but once we get the optimum model, we use train set to build the model and validate it on test set, till then we use the validation set as test set.



```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(13831, 79)
(3458, 79)
(4323, 79)
```

Our problem is a Regression problem i.e. Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).

Regression takes a group of random variables, thought to be predicting Y, and tries to find a mathematical relationship between them. This relationship is typically in the form of a straight line (linear regression) that best approximates all the individual data points. In multiple regression, the separate variables are differentiated by using subscripts.

Metrics for RegressionModels-

- Mean Squared Error (MSE) = 1 / N * sum for i to N (y_i – yhat_i)^2
- Root Mean Squared Error (RMSE) = sqrt(1 / N * sum for i to N (y_i – yhat_i)^2)
- Mean Absolute Error (MAE) = 1 / N * sum for i to N abs(y_i – yhat_i)
- r2_score

Now, we start building models for predicting the house prices and the measure that we consider for selecting the model are r2 score and RMSE values. We will use the following models: -

- Linear Regression
- KNN Regressor
- Decision Tree Regressor
- Ensemble techniques –
    - ➢ Random Forest Regressor
    - ➢ Bagging Regressor
    - ➢ Boosting Regressor (AdaBoost and GradientBoost).

LINEAR REGRESSION and LASSO and RIDGE: -

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable. This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. As the name suggests, linear regression follows the linear mathematical model for determining the value of one dependent variable from value of one given independent variable. Remember the linear equation from school?

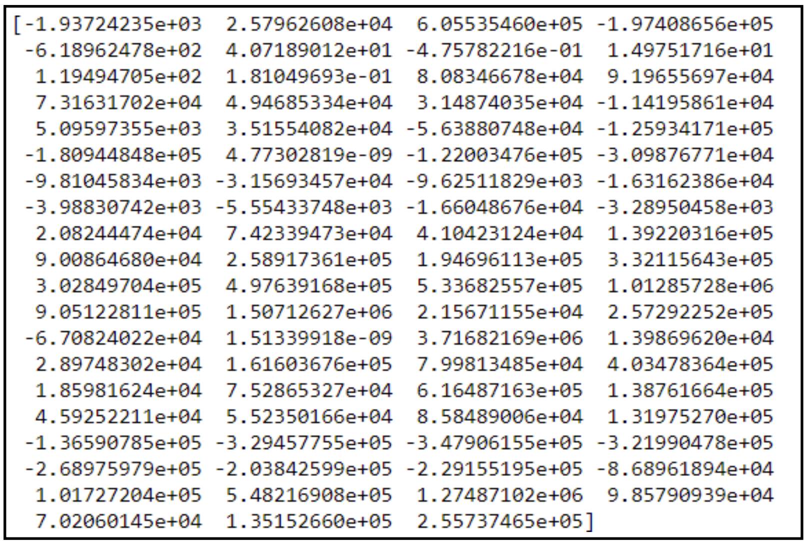
$y=mx+c$

where y is the dependent variable, m is slope, x is the independent variable and c is the intercept for a given line.

Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a "least squares" method to discover the best-fit line for a set of paired data. You then estimate the value of X (dependent variable) from Y (independent variable).

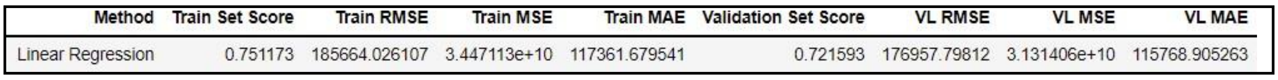
So, we import the LinearRegression from sklearn. linear_model and fit a model on train set and predict values for train and validation set i.e. test set and compare our predictions. Foloowing is the output of our model: -

Following are the coefficients of all the variables in our data: -



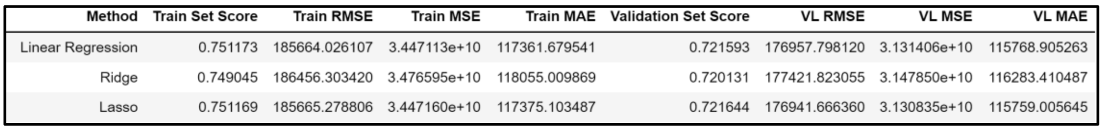
```
[-1.93724235e+03  2.57962608e+04  6.05535460e+05 -1.97408656e+05
 -6.18962478e+02  4.07189012e+01 -4.75782216e-01  1.49751716e+01
  1.19494705e+02  1.81049693e-01  8.08346678e+04  9.19655697e+04
  7.31631702e+04  4.94685334e+04  3.14874035e+04 -1.14195861e+04
  5.09597355e+03  3.51554082e+04 -5.63880748e+04 -1.25934171e+05
 -1.80944848e+05  4.77302819e-09 -1.22003476e+05 -3.09876771e+04
 -9.81045834e+03 -3.15693457e+04 -9.62511829e+03 -1.63162386e+04
 -3.98830742e+03 -5.55433748e+03 -1.66048676e+04 -3.28950458e+03
  2.08244474e+04  7.42339473e+04  4.10423124e+04  1.39220316e+05
  9.00864680e+04  2.58917361e+05  1.94696113e+05  3.32115643e+05
  3.02849704e+05  4.97639168e+05  5.33682557e+05  1.01285728e+06
  9.05122811e+05  1.50712627e+06  2.15671155e+04  2.57292252e+05
 -6.70824022e+04  1.51339918e-09  3.71682169e+06  1.39869620e+04
  2.89748302e+04  1.61603676e+05  7.99813485e+04  4.03478364e+05
  1.85981624e+04  7.52865327e+04  6.16487163e+05  1.38761664e+05
  4.59252211e+04  5.52350166e+04  8.58489006e+04  1.31975270e+05
 -1.36590785e+05 -3.29457755e+05 -3.47906155e+05 -3.21990478e+05
 -2.68975979e+05 -2.03842599e+05 -2.29155195e+05 -8.68961894e+04
  1.01727204e+05  5.48216908e+05  1.27487102e+06  9.85790939e+04
  7.02060145e+04  1.35152660e+05  2.55737465e+05]
```

Following is the metric output: -



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Linear Regression	0.751173	185664.026107	3.447113e+10	117361.679541	0.721593	176957.79812	3.131406e+10	115768.905263

We also tried Lasso and Ridge regression as found the output as –



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Linear Regression	0.751173	185664.026107	3.447113e+10	117361.679541	0.721593	176957.798120	3.131406e+10	115768.905263
Ridge	0.749045	186456.303420	3.476595e+10	118055.009869	0.720131	177421.823055	3.147850e+10	116283.410487
Lasso	0.751169	185665.278806	3.447160e+10	117375.103487	0.721644	176941.666360	3.130835e+10	115759.005645

KNN REGRESSOR (K-Nearest Neighbors) –

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood. The size of the neighbourhood needs to be set by the analyst or can be chosen using cross-validation (we will see this later) to select the size that minimises the mean-squared error.

While the method is quite appealing, it quickly becomes impractical when the dimension increases, i.e., when there are many independent variables.

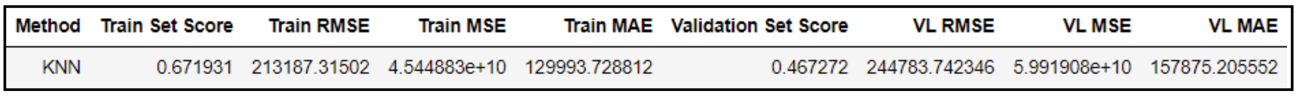
The KNN algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

Below is a stepwise explanation of the algorithm:

- First, the distance between the new point and each training point is calculated.
- The closest k data points are selected (based on the distance)
- The average of these data points is the final prediction for the new point.

So, we import the KNeighborsRegressor from sklearn. neighbors and fit a model on train set and predict values for train and validation set i.e. test set and compare our predictions.

Foloowing is the output of our model: -



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
KNN	0.671931	213187.31502	4.544883e+10	129993.728812	0.467272	244783.742346	5.991908e+10	157875.205552

DECISION TREE REGRESSOR –

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

A decision tree is arriving at an estimate by asking a series of questions to the data, each question narrowing our possible values until the model get confident enough to make a single prediction. The order of the question as well as their content are being determined by the model. In addition, the questions asked are all in a True/False form. For each True and False answer there are separate branches. No matter the answers to the questions, we eventually reach a prediction (leaf node). Start at the root node at the top and progress through the tree answering the questions along the way. So, given any pair of X1, X2. One

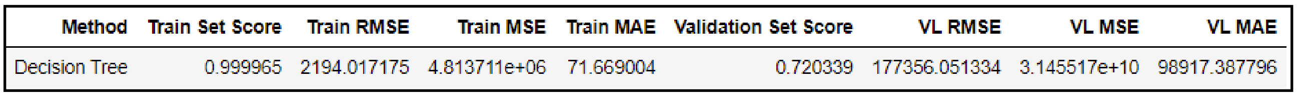
aspect of the decision tree I should mention is how it actually learns (how the 'questions' are formed and how the thresholds are set). As a supervised machine learning model, a decision tree learns to map data to outputs in what is called the training phase of model building.

During training, the model is fitted with any historical data that is relevant to the problem domain and the true value we want the model to learn to predict. The model learns any relationships between the data and the target variable. After the training phase, the decision tree produces a tree similar to the one shown above, calculating the best questions as well as their order to ask in order to make the most accurate estimates possible. When we want to make a prediction the same data format should be provided to the model in order to make a prediction. The prediction will be an estimate based on the train data that it has been trained on.

How split is done –

- A single number that represents how good a split is which is the weighted average of the mean squared errors of the two groups that create.
- A way to find the best split which is to try every variable and to try every possible value of that variable and see which variable and which value gives us a split with the best score.

So, we import the DecisionTreeRegressor from sklearn.tree and fit a model on train set and predict values for train and validation set i.e. test set and compare our predictions. Foloowing is the output of our model: -



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Decision Tree	0.999965	2194.017175	4.813711e+06	71.669004	0.720339	177356.051334	3.145517e+10	98917.387796

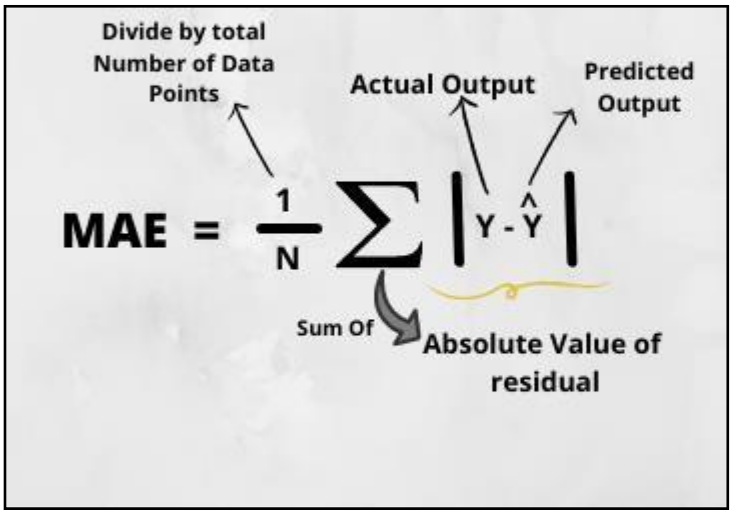
- Test your predictive model against the test set using various appropriate performance metrics.

Evaluation Metrics-

1) Mean Absolute Error (MAE)-

MAE is a very simple metric which calculates the absolute difference between actual and predicted values.

The MAE of your model which is basically a mistake made by the model known as an error. Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset. So, sum all the errors and divide them by a total number of observations And this is MAE. And we aim to get a minimum MAE because this is a loss.


Divide by total Number of Data Points
Actual Output
Predicted Output
MAE = 1/N ∑ |Y - Ŷ|
Sum Of
Absolute Value of residual

Advantages of MAE-

- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers.

Disadvantages of MAE-

- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

2) Mean Squared Error (MSE)-

MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value. So, above we are finding the absolute difference and here we are finding the squared difference.

What actually the MSE represents? It represents the squared distance between actual and predicted values. we perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

Advantages of MSE-

- The graph of MSE is differentiable, so you can easily use it as a loss function.

Disadvantages of MSE-

- The value you get after calculating MSE is a squared unit of output. for example, the output variable is in meter(m) then after calculating MSE the output we get is in meter squared.
- If you have outliers in the dataset then it penalizes the outliers most and the calculated MSE is bigger. So, in short, It is not Robust to outliers which were an advantage in MAE.

$$\textbf{R2 Squared} = 1 - \frac{\textbf{SSr}}{\textbf{SSm}}$$

SSr = Squared sum error of regression line

SSm = Squared sum error of mean line

3) Root Mean Squared Error(RMSE)

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

$$\textbf{RMSE} = \sqrt{\textbf{MSE}}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)}$$

Advantages of RMSE-

- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

Disadvantages of RMSE-

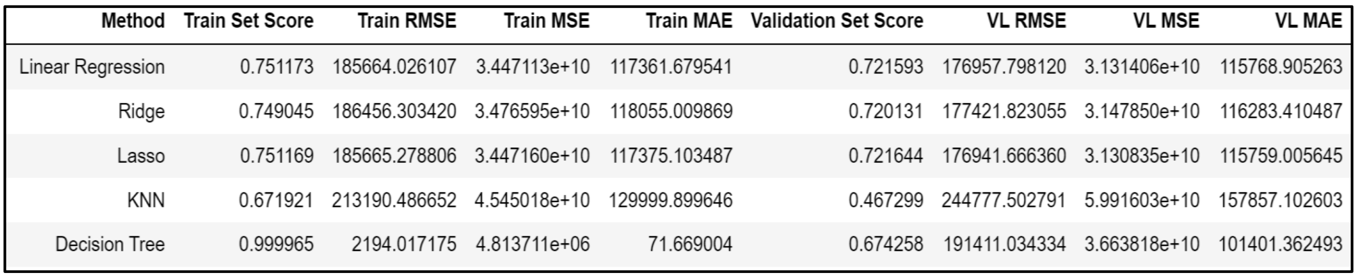
- It is not that robust to outliers as compared to MAE.

4) R Squared (R2)-

R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how well did your model perform. In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context. So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides. The same we have in classification problems which we call a threshold which is fixed at 0.5. So, basically R2 squared calculates how must regression line is better than a mean line. Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit. If the R2 score is 0 then the above regression line by mean line is equal means 1 so 1-1 is zero. So, in this case, both lines are overlapping means model performance is worst, It is not capable to take advantage of the output column. Now the second case is when the R2 score is 1, it means when the division term is zero and it will happen when the regression line does not make any mistake, it is perfect. In the real world, it is not possible. So, we can conclude that as our regression line moves towards perfection, R2 score move towards one. And the model performance improves.

• Interpretation of the models.

We have 5 models i.e. Linear Regression including Ridge and Lasso methods, K-Nearest Neighbors and Decision Tree without tuning any parameters for the initial part of the model building. As we can see that the model scores are not upto the mark and the RMSE values are also very high. We have used r2 score as score, RMSE, MSE and MAE as measures in these models.



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Linear Regression	0.751173	185664.026107	3.447113e+10	117361.679541	0.721593	176957.798120	3.131406e+10	115768.905263
Ridge	0.749045	186456.303420	3.476595e+10	118055.009869	0.720131	177421.823055	3.147850e+10	116283.410487
Lasso	0.751169	185665.278806	3.447160e+10	117375.103487	0.721644	176941.666360	3.130835e+10	115759.005645
KNN	0.671921	213190.486652	4.545018e+10	129999.899646	0.467299	244777.502791	5.991603e+10	157857.102603
Decision Tree	0.999965	2194.017175	4.813711e+06	71.669004	0.674258	191411.034334	3.663818e+10	101401.362493

But we can see that the performances in these models isn’t good enough, in Linear Regression model, the train set score and Test set score are decent but a score of ~ 75 is not really great while predicting. The RMSE and MSE scores are better than KNN and Decision Tree.

The Decision Tree model is an overfit model and we can see it looking at the difference in scores and RMSE values, so to improve the model, we will have to tune the model parameters.

The KNN model has a poor performance as the score on test set is not even 0.5, and the RMSE scores are poor too.

Hence, we will have to tune the parameters of these models and also try different ensemble methods to optimize the model performance.

MODEL TUNING AND BUSINESS IMPLICATION

• Ensemble modelling

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would. This has been the case in a number of machine learning competitions, where the winning solutions used ensemble methods.

- Bagging involves fitting many decision trees on different samples of the same dataset and averaging the predictions.
- Boosting involves adding ensemble members sequentially that correct the predictions made by prior models and outputs a weighted average of the predictions.
- Random Forest Models. Random Forest Models can be thought of as BAGGing, with a slight tweak. When deciding where to split and how to make decisions, BAGGed Decision Trees have the full disposal of features to choose from.

RANDOM FOREST REGRESSOR

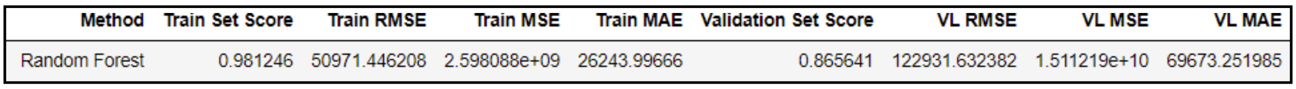
A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. What is bagging you may ask? Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Predictions are made by averaging the predictions of each decision tree. Or, to extend the analogy—much like a forest is a collection of trees, the random forest model is also a collection of decision tree models. This makes random forests a strong modeling technique

that's much more powerful than a single decision tree.

Each tree in a random forest is trained on the subset of data provided. The subset is obtained both with respect to rows and columns. This means each random forest tree is trained on a random data point sample, while at each decision node, a random set of features is considered for splitting.

So, we import the DecisionTreeRegressor from sklearn.tree and fit a model on train set and predict values for train and validation set i.e. test set and compare our predictions. Foloowing is the output of our model: -



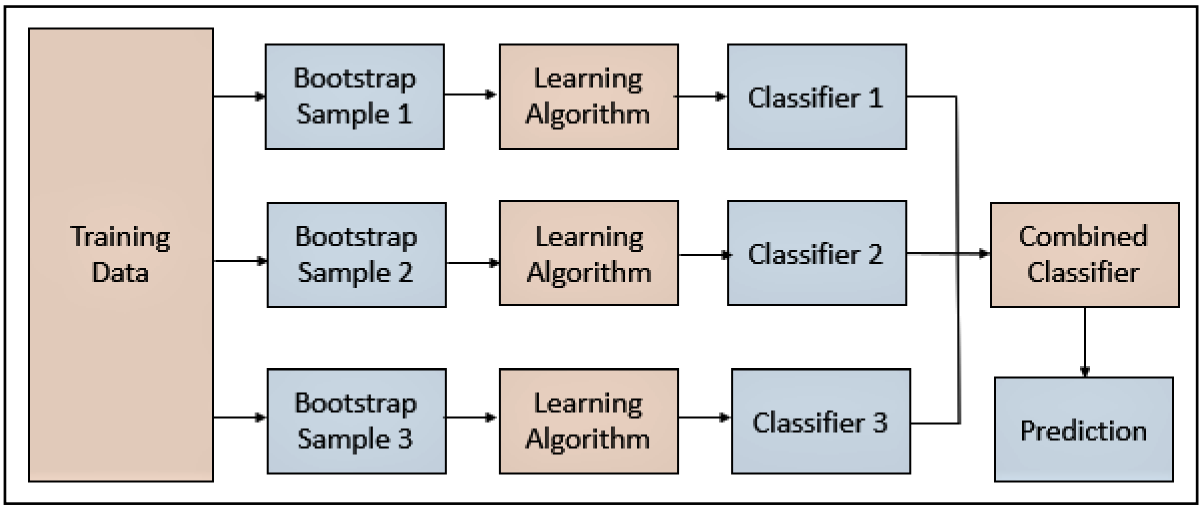
Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Random Forest	0.981246	50971.446208	2.598088e+09	26243.99666	0.865641	122931.632382	1.511219e+10	69673.251985

BAGGING REGRESSOR

A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

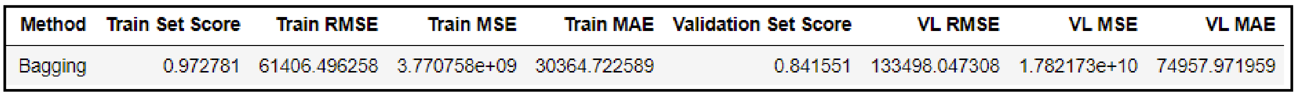
This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting. If samples are drawn with replacement, then the method is known as Bagging. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches.

Bagging is used with decision trees, where it significantly raises the stability of models in improving accuracy and reducing variance, which eliminates the challenge of overfitting.



Bagging in ensemble machine learning takes several weak models, aggregating the predictions to select the best prediction. The weak models specialize in distinct sections of the feature space, which enables bagging leverage predictions to come from every model to reach the utmost purpose.

So, we import the BaggingRegressor from sklearn.ensemble and fit a model on train set and predict values for train and validation set i.e. test set and compare our predictions. Foloowing is the output of our model: -



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Bagging	0.972781	61406.496258	3.770758e+09	30364.722589	0.841551	133498.047308	1.782173e+10	74957.971959

BOOSTING

Boosting is an ensemble learning method that combines a set of weak learners into a strong learner to minimize training errors. In boosting, a random sample of data is selected, fitted with a model and then trained sequentially—that is, each model tries to compensate for the weaknesses of its predecessor. With each iteration, the weak rules from each individual classifier are combined to form one, strong prediction rule.

- The main difference between Bagging and Boosting is the way in which they are trained. In bagging, weak learners are trained in parallel, but in boosting, they learn sequentially. This means that a series of models are constructed and with each new model iteration, the weights of the misclassified data in the previous model are increased. This redistribution of weights helps the algorithm identify the parameters that it needs to focus on to improve its performance. AdaBoost, which stands for "adaptative boosting algorithm," is one of the most popular boosting algorithms as it was one of the first of its kind. Other types of boosting algorithms include XGBoost, GradientBoost, and BrownBoost.
- Another difference between bagging and boosting is in how they are used. For example, bagging methods are typically used on weak learners that exhibit high variance and low bias, whereas boosting methods are leveraged when low variance and high bias is observed. While bagging can be used to avoid overfitting, boosting methods can be more prone to this

Advantages-

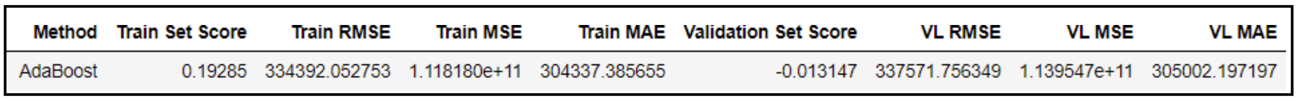
- Ease of Implementation (hypertuning is easy)
- Reduction of bias (sequential approach)
- Efficiency (reduce dimension and increase prediction accuracy)

Disadvantages-

- Overfitting
- Intense computation

ADABOOSTING REGRESSOR

This method operates iteratively, identifying misclassified data points and adjusting their weights to minimize the training error. The model continues optimize in a sequential fashion until it yields the strongest predictor.



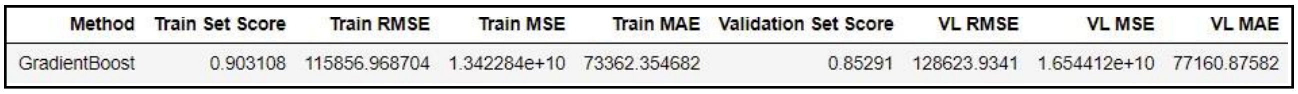
Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
AdaBoost	0.19285	334392.052753	1.118180e+11	304337.385655	-0.013147	337571.756349	1.139547e+11	305002.197197

GRADIENT BOSSTING REGRESSOR

This method works by sequentially adding predictors to an ensemble with each one correcting for the errors of its predecessor. However, instead of changing weights of data points like AdaBoost, the gradient boosting trains on the residual errors of the previous predictor. The name, gradient boosting, is used since it combines the gradient descent algorithm and boosting method.

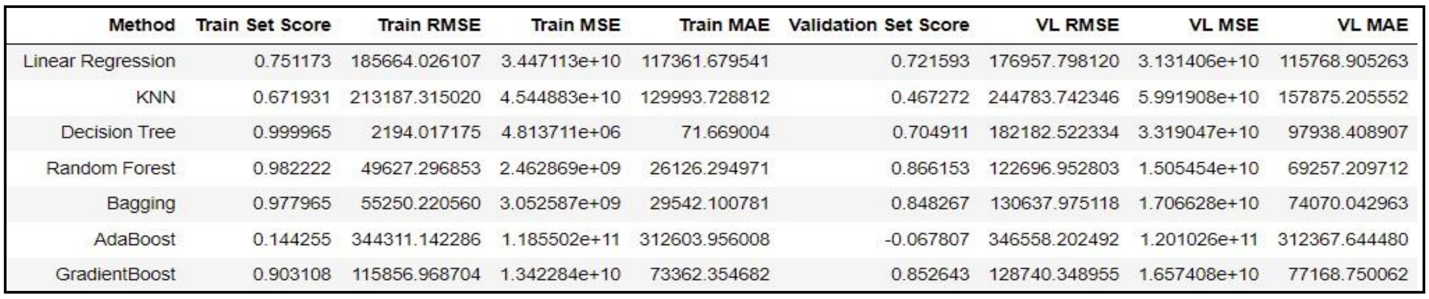
Gradient boosting differs from AdaBoost in the manner that decision stumps (one node & two leaves) are used in AdaBoost whereas decision trees of fixed size are used in Gradient Boosting.

The process of fitting the model starts with the constant such as mean value of the target values. In subsequent stages, the decision trees or the estimators are fitted to predict the negative gradients of the samples. The gradients are updated in the each iterator (for every subsequent estimators). A learning rate is used to shrink the outcome or the contribution from each subsequent trees or estimators.



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
GradientBoost	0.903108	115856.968704	1.342284e+10	73362.354682	0.85291	128623.9341	1.654412e+10	77160.87582

Now, we can see all the different models that we have built along with the output metrics: -



Method	Train Set Score	Train RMSE	Train MSE	Train MAE	Validation Set Score	VL RMSE	VL MSE	VL MAE
Linear Regression	0.751173	185664.026107	3.447113e+10	117361.679541	0.721593	176957.798120	3.131406e+10	115768.905263
KNN	0.671931	213187.315020	4.544883e+10	129993.728812	0.467272	244783.742346	5.991908e+10	157875.205552
Decision Tree	0.999965	2194.017175	4.813711e+06	71.669004	0.704911	182182.522334	3.319047e+10	97938.408907
Random Forest	0.982222	49627.296853	2.462869e+09	26126.294971	0.866153	122696.952803	1.505454e+10	69257.209712
Bagging	0.977965	55250.220560	3.052587e+09	29542.100781	0.848267	130637.975118	1.706628e+10	74070.042963
AdaBoost	0.144255	344311.142286	1.185502e+11	312603.956008	-0.067807	346558.202492	1.201026e+11	312367.644480
GradientBoost	0.903108	115856.968704	1.342284e+10	73362.354682	0.852643	128740.348955	1.657408e+10	77168.750062

Looking at the above image, we can see that the Ensemble models have been doing better than the Linear models, KNN and Decision Tree, Random Forest and Bagging are doing good but they seem to be overfit, so we can tune the parameters and optimize their performances, AdaBoost is performing poorly and GradientBoost is doing really well looking at the score and RMSE values on both- Train set and Validation Set.

So, we will select KNN, Decision Tree and other ensemble models and tune the parameters and check for their performances (even though we can go for GradientBoost model directly as it seems to perform better than all the models in terms of performance and fitting).

• Any other model tuning measures

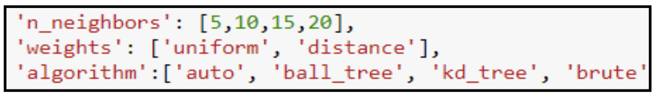
Tuning is usually a trial-and-error process by which you change some hyperparameters (for example, the number of trees in a tree-based algorithm or the value of alpha in a linear algorithm), run the algorithm on the data again, then compare its performance on your validation set in order to determine which set of hyperparameters results in the most accurate model. All machine learning algorithms have a “default” set of hyperparameters that are like a configuration that is external to the model and whose value cannot be estimated from data. Different algorithms consist of different hyperparameters. For example, regularized regression models have coefficients penalties, decision trees have a set number of branches, and neural networks have a set number of layers. When building models, analysts and data scientists choose the default configuration of these hyperparameters after running the model on several datasets.

We first import GridSearchCV from sklearn.model_selection for model tuning, then give input as a set of different parameters which the models compute by fitting different combinations and give the output as the best combination of those features, we then use those features and then fit a model on them. We need to tweak and test a lot of different combinations to build an optimal model.

KNN-

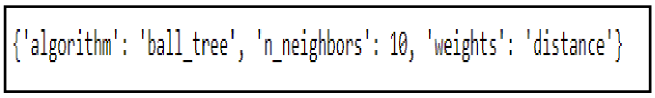
We give the parameter gird and get the following as best parameters for tuning model: -

Input-



```
'n_neighbors': [5,10,15,20],
'weights': ['uniform', 'distance'],
'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'
```

Output-

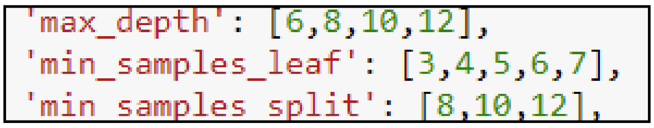


```
{'algorithm': 'ball_tree', 'n_neighbors': 10, 'weights': 'distance'}
```

Decision Tree -

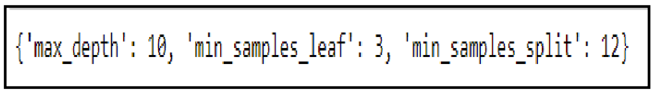
We give the parameter gird and get the following as best parameters for tuning model: -

Input-



```
'max_depth': [6,8,10,12],
'min_samples_leaf': [3,4,5,6,7],
'min_samples_split': [8,10,12],
```

Output-

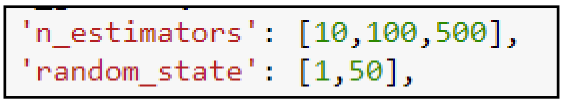


```
{'max_depth': 10, 'min_samples_leaf': 3, 'min_samples_split': 12}
```

Bagging-

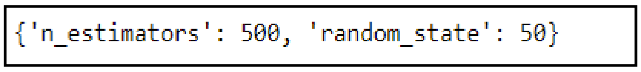
We give the parameter gird and get the following as best parameters for tuning model: -

Input-



```
'n_estimators': [10,100,500],
'random_state': [1,50],
```

Output-

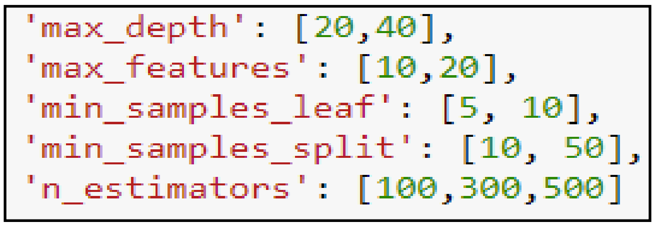


```
{'n_estimators': 500, 'random_state': 50}
```

Random Forest -

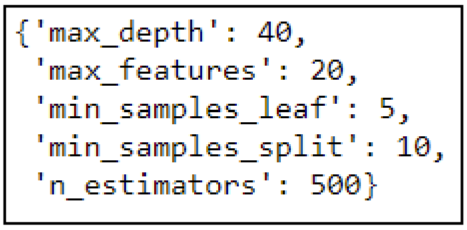
We give the parameter gird and get the following as best parameters for tuning model: -

Input-



```
'max_depth': [20,40],
'max_features': [10,20],
'min_samples_leaf': [5, 10],
'min_samples_split': [10, 50],
'n_estimators': [100,300,500]
```

Output-

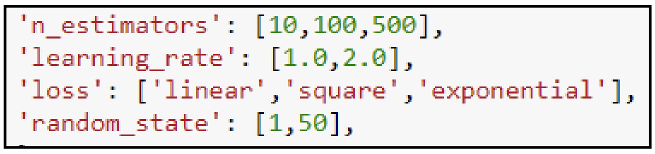


```
{'max_depth': 40,
 'max_features': 20,
 'min_samples_leaf': 5,
 'min_samples_split': 10,
 'n_estimators': 500}
```

AdaBoost-

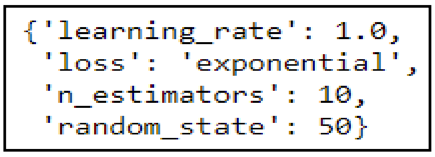
We give the parameter gird and get the following as best parameters for tuning model: -

Input-



```
'n_estimators': [10,100,500],
'learning_rate': [1.0,2.0],
'loss': ['linear','square','exponential'],
'random_state': [1,50],
```

Output-

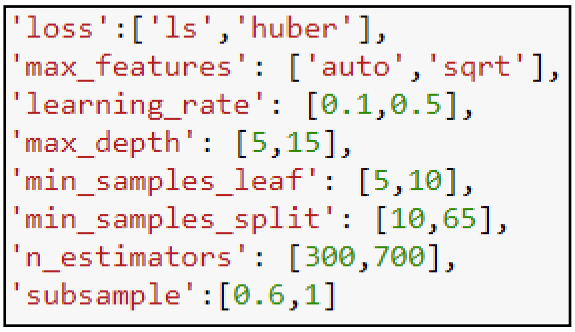


```
{'learning_rate': 1.0,
 'loss': 'exponential',
 'n_estimators': 10,
 'random_state': 50}
```

GradientBoost-

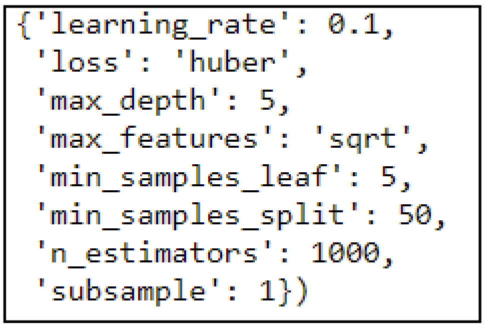
We give the parameter gird and get the following as best parameters for tuning model: -

Input-



```
'loss':['ls','huber'],
'max_features': ['auto','sqrt'],
'learning_rate': [0.1,0.5],
'max_depth': [5,15],
'min_samples_leaf': [5,10],
'min_samples_split': [10,65],
'n_estimators': [300,700],
'subsample':[0.6,1]
```

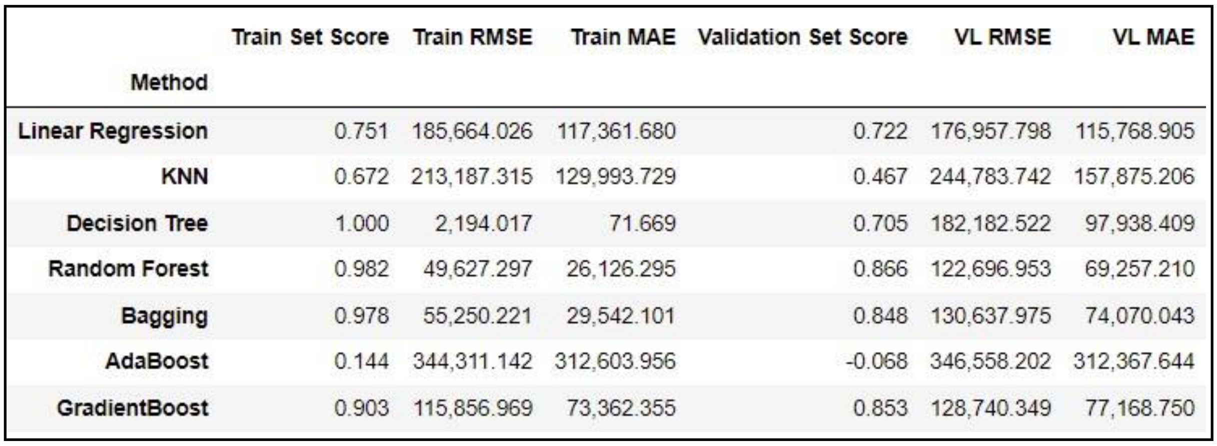
Output-



```
{'learning_rate': 0.1,
 'loss': 'huber',
 'max_depth': 5,
 'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 50,
 'n_estimators': 1000,
 'subsample': 1})
```

• Interpretation of the most optimum model and its implication on the business

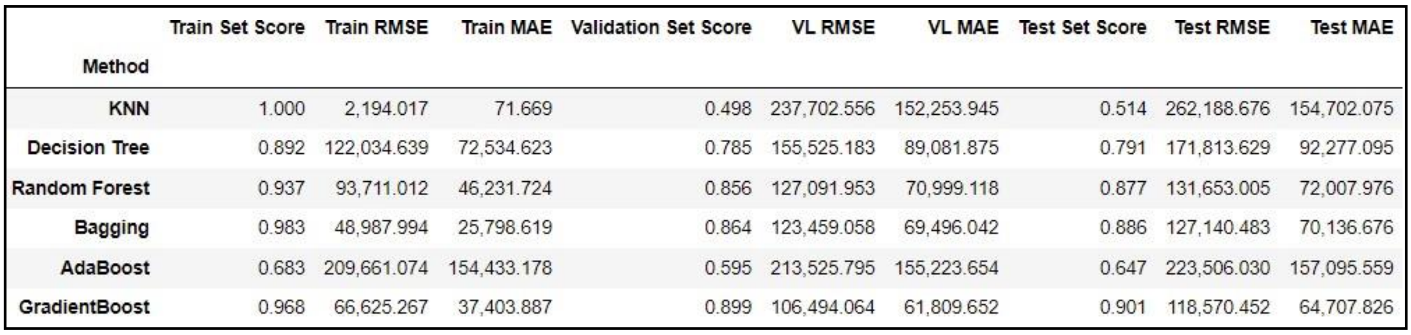
Models before Tuning-



Method	Train Set Score	Train RMSE	Train MAE	Validation Set Score	VL RMSE	VL MAE
Linear Regression	0.751	185,664.026	117,361.680	0.722	176,957.798	115,768.905
KNN	0.672	213,187.315	129,993.729	0.467	244,783.742	157,875.206
Decision Tree	1.000	2,194.017	71.669	0.705	182,182.522	97,938.409
Random Forest	0.982	49,627.297	26,126.295	0.866	122,696.953	69,257.210
Bagging	0.978	55,250.221	29,542.101	0.848	130,637.975	74,070.043
AdaBoost	0.144	344,311.142	312,603.956	-0.068	346,558.202	312,367.644
GradientBoost	0.903	115,856.969	73,362.355	0.853	128,740.349	77,168.750

Vs

Models after Tuning-



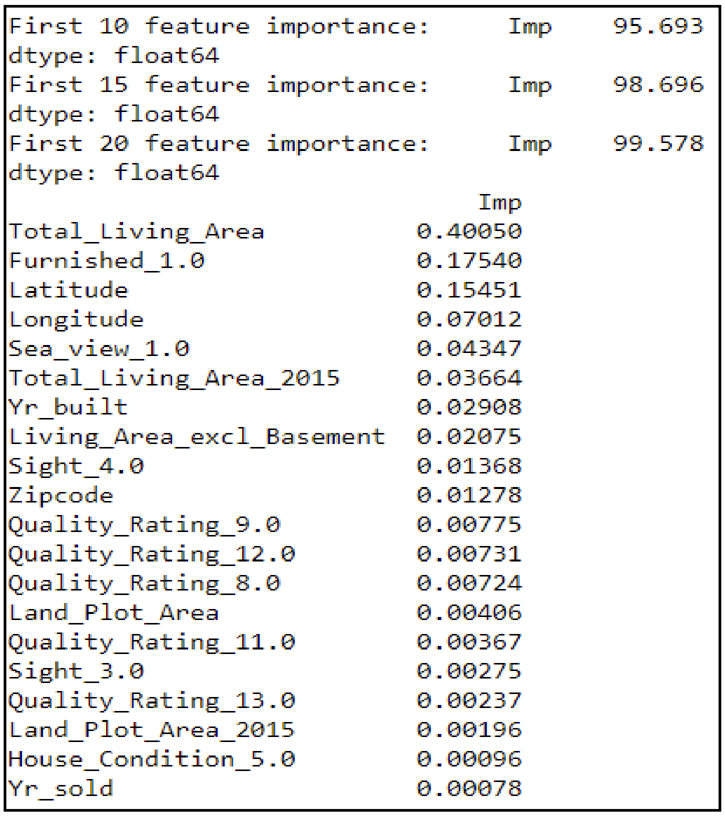
Method	Train Set Score	Train RMSE	Train MAE	Validation Set Score	VL RMSE	VL MAE	Test Set Score	Test RMSE	Test MAE
KNN	1.000	2,194.017	71.669	0.498	237,702.556	152,253.945	0.514	262,188.676	154,702.075
Decision Tree	0.892	122,034.639	72,534.623	0.785	155,525.183	89,081.875	0.791	171,813.629	92,277.095
Random Forest	0.937	93,711.012	46,231.724	0.856	127,091.953	70,999.118	0.877	131,653.005	72,007.976
Bagging	0.983	48,987.994	25,798.619	0.864	123,459.058	69,496.042	0.886	127,140.483	70,136.676
AdaBoost	0.683	209,661.074	154,433.178	0.595	213,525.795	155,223.654	0.647	223,506.030	157,095.559
GradientBoost	0.968	66,625.267	37,403.887	0.899	106,494.064	61,809.652	0.901	118,570.452	64,707.826

We can clearly see improvements in models after tuning, the R2 scores have increased and the RMSE and MAE values have decreased. We can see that the Ensemble models have done the best, KNN is now overfitting, Decision Tree, Random Forest, Bagging and Gradient Boosting have done well after Hypertuning the parameters and these models don't have an effect of outliers. GradientBoost does the best followed by Bagging, Random Forest and Decision Tree in that order. Since GradientBoost follows a sequential approach and has better efficiency and is effective with tuning parameters, it seems that this is the optimum model. The R2 score i.e. goodness of the fit of the prediction to the actual values is ~97% on train set and ~90% on Test set.

The RMSE shows the magnitude of the error and the model with least RMSE is the best model, so according to these numbers GradientBoost has the least RMSE in both Train and Test sets.

THE MAE is the magnitude of the average residual of the errors, lesser the MAE, better the model. Similarly, Gradient Boosting has the least MAE for the Test set.

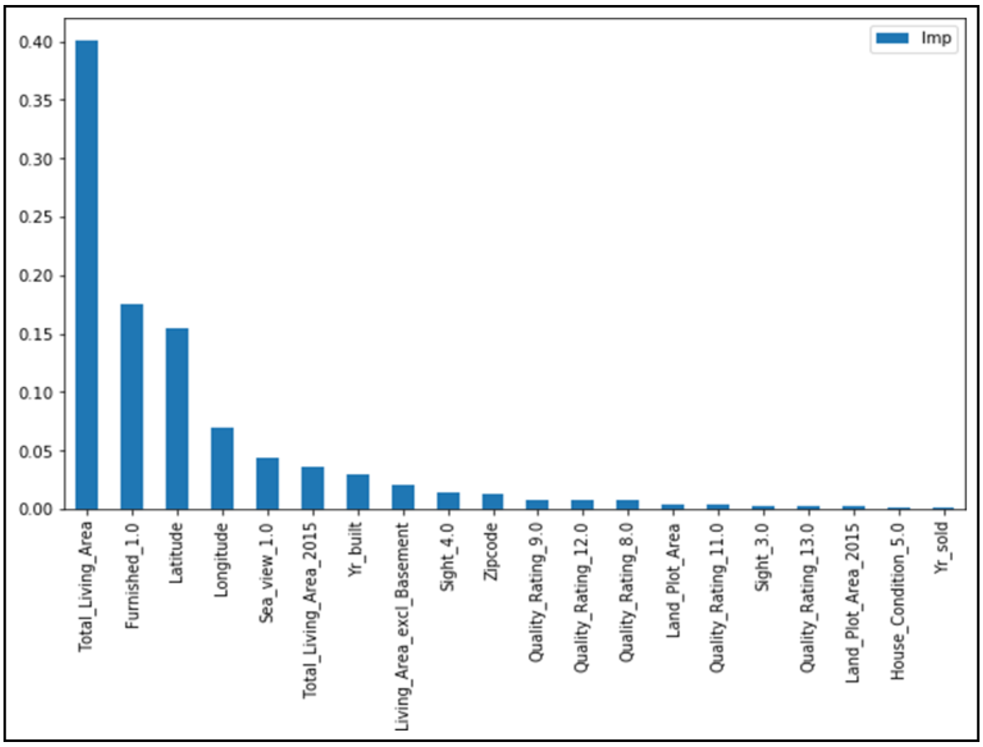
We can also see the best features that contribute to the model using feature_importance



```
First 10 feature importance:      Imp    95.693
dtype: float64
First 15 feature importance:      Imp    98.696
dtype: float64
First 20 feature importance:      Imp    99.578
dtype: float64
                                Imp
Total_Living_Area           0.40050
Furnished_1.0               0.17540
Latitude                    0.15451
Longitude                   0.07012
Sea_view_1.0                0.04347
Total_Living_Area_2015      0.03664
Yr_built                    0.02908
Living_Area_excl_Basement   0.02075
Sight_4.0                   0.01368
Zipcode                     0.01278
Quality_Rating_9.0          0.00775
Quality_Rating_12.0         0.00731
Quality_Rating_8.0          0.00724
Land_Plot_Area              0.00406
Quality_Rating_11.0         0.00367
Sight_3.0                   0.00275
Quality_Rating_13.0         0.00237
Land_Plot_Area_2015         0.00196
House_Condition_5.0         0.00096
Yr_sold                     0.00078
```

The top 10 features itself contribute to ~96%, 15 features contribute to ~99% and 20 features itself form ~100% of the model.

We can also show them in visualization:



Business Applications- Recommendations and Scope

- It will help the buyer to know what are the different prices in different areas depending upon all the features that are provided or the facilities that are available in the house. So, any new buyer can use the system to keep in check if they aren't paying extra for something that they can avail in lesser prices.
- With the sellers' perspectives, it will help the seller to know the right price of the property in which the house will easily get sold. The seller may just have to feed in all the input parameters such as house ratings, sea view, furnished, area sizes, etc. and the model will compute the right price of the property in accordance to other properties registered on the system.
- We can recommend the renovations so that the house condition score increases and so does the quality rating, also adding some basic furniture adds the quality rating to above 9 which will increase the house prices significantly. This will be a win-win for sellers.
- We can build an interactive website for the buyers and the sellers and charge a nominal fee as the registration fee which will enable them to view different properties and calculate the right price for selling and buying the properties, this will reduce or eliminate the need of estate agents that charge their brokerage from both- the buyer and the seller. This website will be a live website and will add a lot of value to the application of this model. We can tie up with different organization or governments to enable them provide this as a public feature.
- This application of predicting prices will be beneficial in many uncounted ways like – limiting the ballooning up of property prices and thereby keeping in check the prices, all the houses in the system will be verified by the agents of the system and hence the properties can be marked safe and the chances of being duped will drop down drastically.
- Further, we can also develop a mobile application for the same and increase the scope further (in terms of geographic locations) i.e. from county to state and then to country levels.
- Also, we can then encourage customers to rent their apartments to tourists or students by leasing them for a time period and we can thereby increase the scope of application which will then calculate the rent prices. We can also integrate our system with banks or payment services which will remind the customers for payments and act as a wallet for the owners, thereby getting timely rents.