BIG MART SALES PRICE PREDICTION

PROJECT 1

By:
Priyanka
Chandramohan

1. Business Problem We are trying to Solve

- The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store.

- Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales.

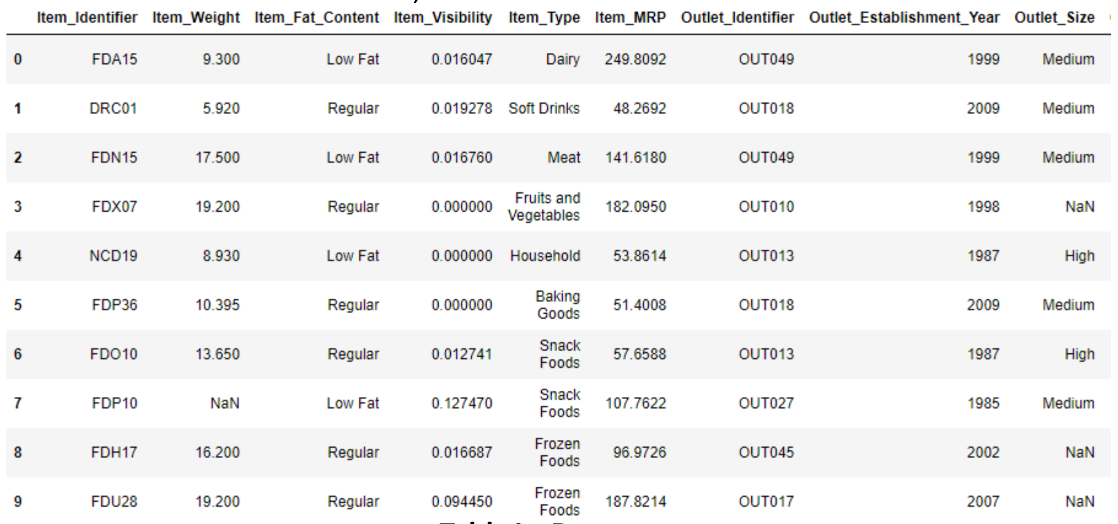
Main Goal:

- Create an analytical framework to understand Key factors impacting Sales.
- Develop a modeling framework
- To estimate the stores which play a key role in increasing sales.

Variable Description

- ProductID : unique product ID
- Weight : weight of products
- FatContent : specifies whether the product is low on fat or not
- Visibility : percentage of total display area of all products in a store allocated to the particular product
- ProductType : the category to which the product belongs
- MRP : Maximum Retail Price (listed price) of the products
- OutletID : unique store ID
- EstablishmentYear : year of establishment of the outlets
- OutletSize : the size of the store in terms of ground area covered
- LocationType : the type of city in which the store is located
- OutletType : specifies whether the outlet is just a grocery store or some sort of supermarket
- OutletSales : (target variable) sales of the product in the particular store

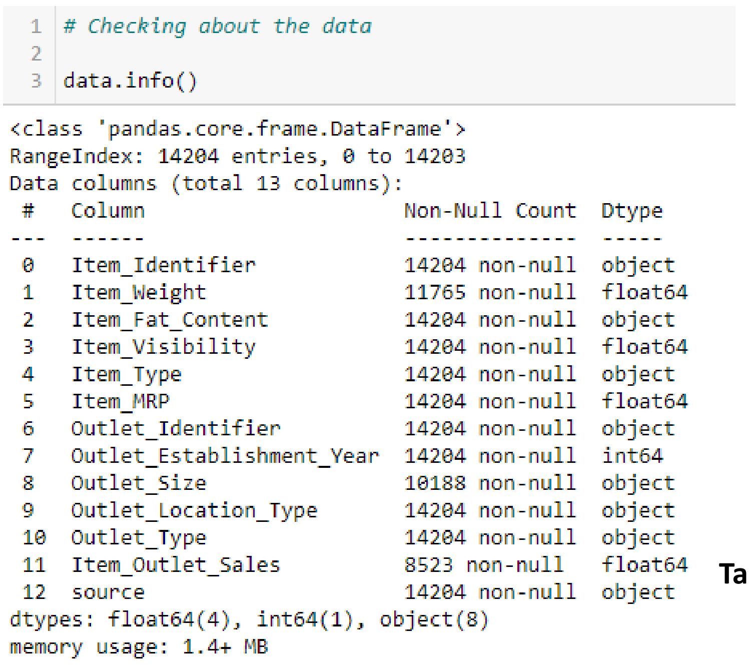
This is what the dataset looks like,



	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High
5	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium
6	FDO10	13.650	Regular	0.012741	Snack Foods	57.6588	OUT013	1987	High
7	FDP10	NaN	Low Fat	0.127470	Snack Foods	107.7622	OUT027	1985	Medium
8	FDH17	16.200	Regular	0.016687	Frozen Foods	96.9726	OUT045	2002	NaN
9	FDU28	19.200	Regular	0.094450	Frozen Foods	187.8214	OUT017	2007	NaN

Table1 - Dataset

We have 14204 rows and 13 columns in the dataset.(1 added to identifying train and test data, originally 12 columns)



```
# Checking about the data

data.info()
```

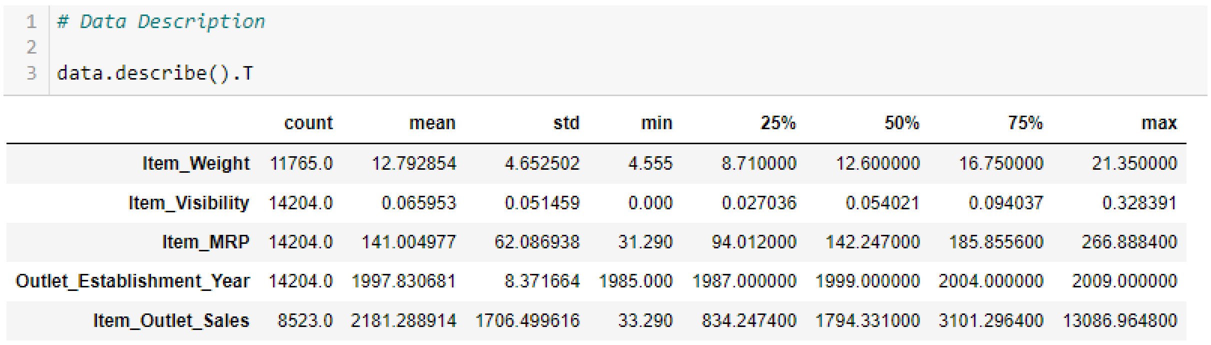
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            14204 non-null  object 
 1   Item_Weight                11765 non-null  float64
 2   Item_Fat_Content           14204 non-null  object 
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object 
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object 
 7   Outlet_Establishment_Year  14204 non-null  int64  
 8   Outlet_Size                10188 non-null  object 
 9   Outlet_Location_Type       14204 non-null  object 
 10  Outlet_Type                14204 non-null  object 
 11  Item_Outlet_Sales          8523 non-null   float64
 12  source                     14204 non-null  object 
dtypes: float64(4), int64(1), object(8)
memory usage: 1.4+ MB
```

Table2 - 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



```
# Data Description

data.describe().T
```

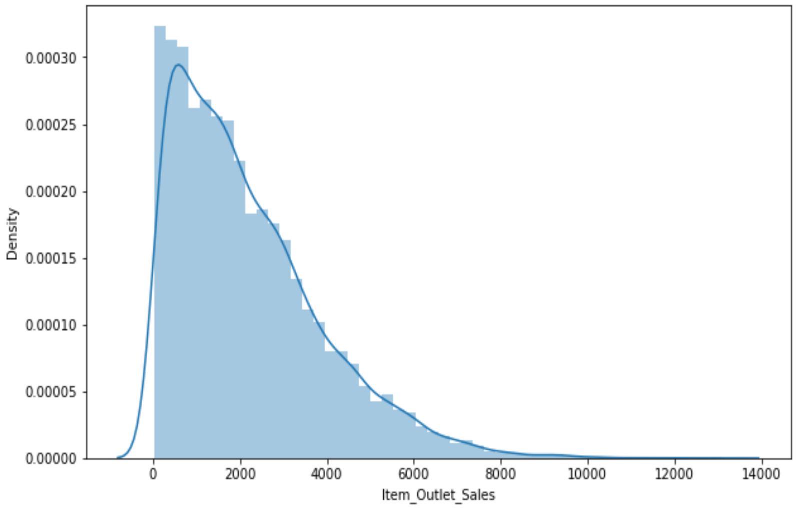
	count	mean	std	min	25%	50%	75%	max
Item_Weight	11765.0	12.792854	4.652502	4.555	8.710000	12.600000	16.750000	21.350000
Item_Visibility	14204.0	0.065953	0.051459	0.000	0.027036	0.054021	0.094037	0.328391
Item_MRP	14204.0	141.004977	62.086938	31.290	94.012000	142.247000	185.855600	266.888400
Outlet_Establishment_Year	14204.0	1997.830681	8.371664	1985.000	1987.000000	1999.000000	2004.000000	2009.000000
Item_Outlet_Sales	8523.0	2181.288914	1706.499616	33.290	834.247400	1794.331000	3101.296400	13086.964800

Table3 – Data Description

Observation:

- By looking at the 75% and max value we can say that there are outliers present in the many columns of dataset where there is large gap between these two values.
- Item_Visibility has a min value of zero. This makes no practical sense because when a product is being sold in a store, the visibility cannot be 0.
- Outlet_Establishment_Years vary from 1985 to 2009. The values might not be apt in this form. Rather, if we can convert them to how old the particular store is, it should have a better impact on sales.

Analysis of Price (Target Variable)

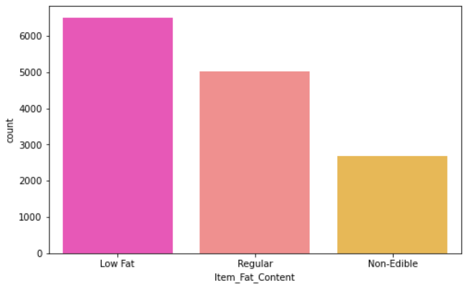


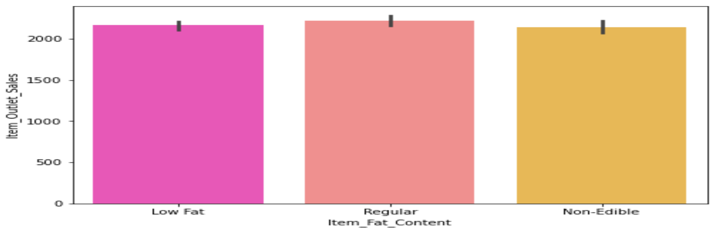
Plot1 – Analysis on Price

Most of the Stores have Sales less than 6000.

The above graph shows that Item Outlet Sales has right skewness. And we know that the assumption of linear regression tells us that the distribution of dependent variable has to be normal, so we should perform some operation to make it normal.

Item Fat Content





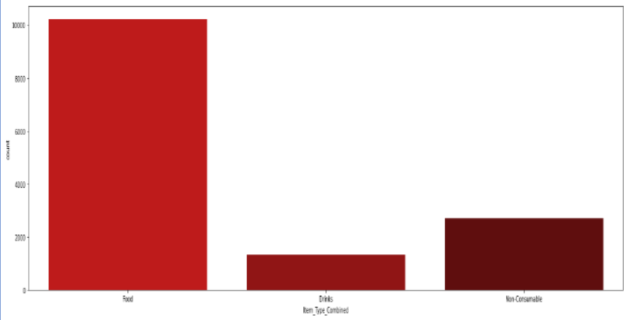
Plot2 – Analysis on Item Fat Content

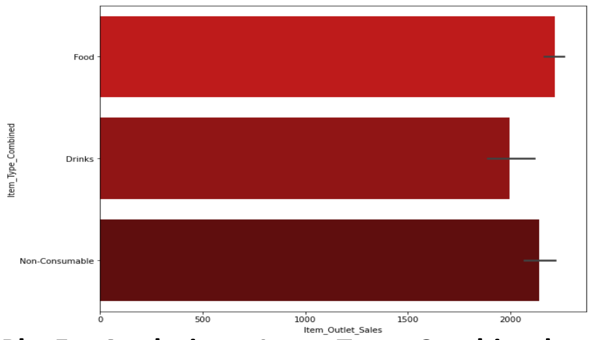
Plot3 – Analysis on Item Fat Content and Sales

Low FAT products are bought more followed by Regular and Non Edible.

But Item Outles sales are almost same for all Low Fat,Regular Item Content and Non-Edible.

Item_Type_Combined



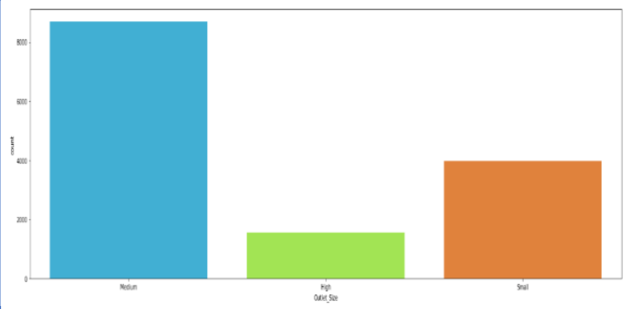


Plot4 – Analysis on Item_Type_Combined

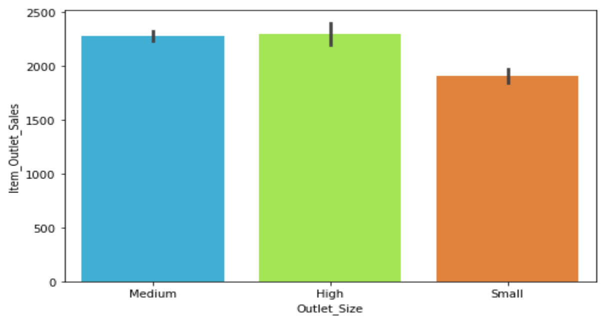
Plot5 – Analysis on Item_Type_Combined and Sales

The product category of food is mostly available but the sales of Food ,Non consumable also along with Drinks seems high hence the sales can be improved with having stock of products that are most bought by customers.

Outlet Size



Plot6 – Analysis on Outlet Size

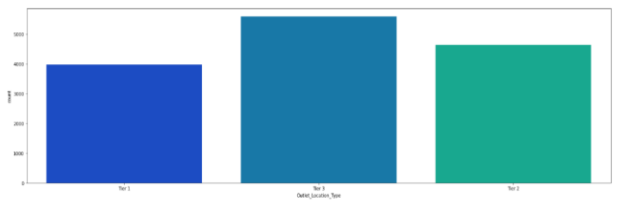


Plot7 – Analysis on Outlet Size and Sales

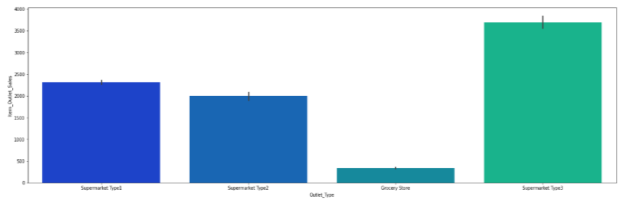
The Outlets are more of Medium Size

But Outlet Sales is maximum for Medium and High sized Outlets so may be with High size Outlets can improve the Outlet Sales

Outlet Location Type



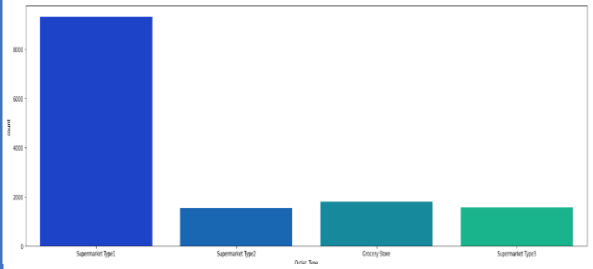
Plot8 – Analysis on Outlet Location Type

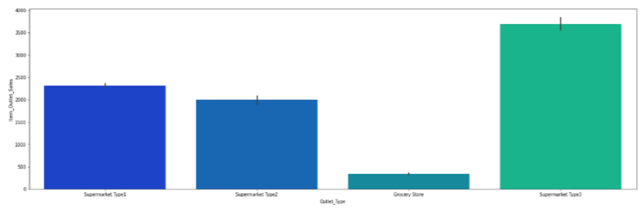


Plot9 – Analysis on Outlet Location Type and Sales

The Outlets are more of Supermarket Type1. But sales are more on Type 3

Outlet Type

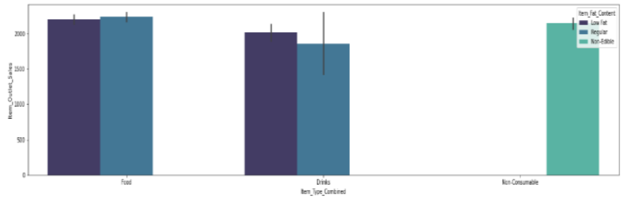




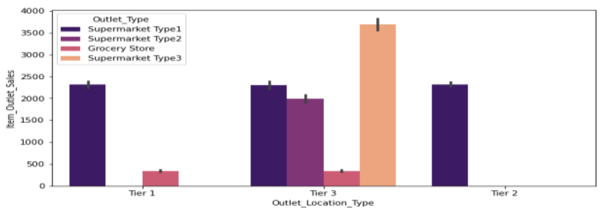
Plot10 – Analysis on Outlet Type **Plot11 – Analysis on Outlet Type and Sales**

The Outlets are more of Supermarket Type1. But sales are more on Type 3

Item_Type_Combined



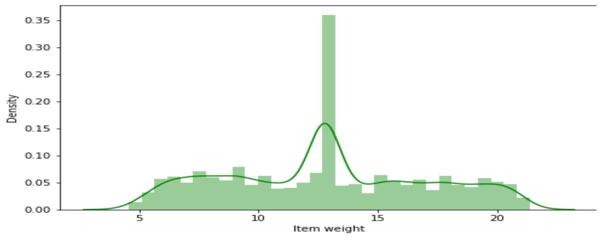
Plot12 – Analysis on Item_Type_Combined



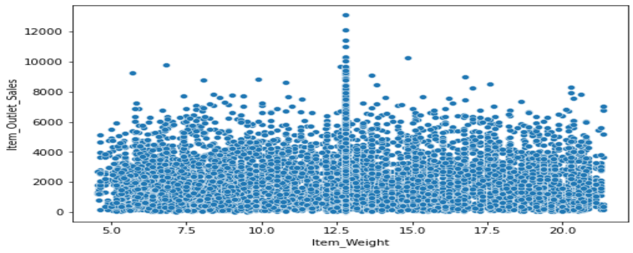
Plot13 – Analysis on Item_Type_Combined and Sales

The Tier-3 location type has all types of Outlet type and has high sales margin.

Item weight



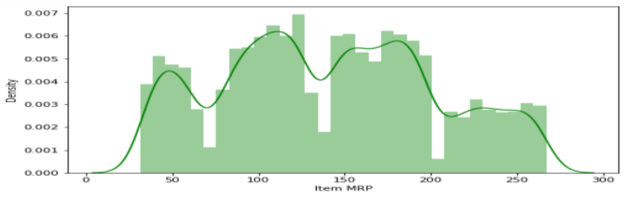
Plot14 – Analysis on Item weight



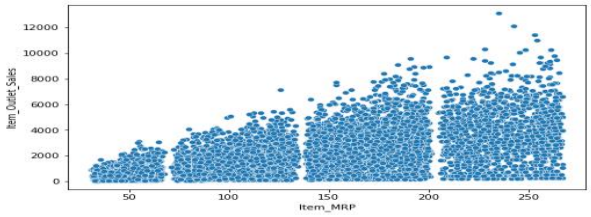
Plot15 – Analysis on Item weight and Sales

We have more products of weight around 12.5, and sell is maximum for that weight

Item MRP



Plot16 – Analysis on Item MRP

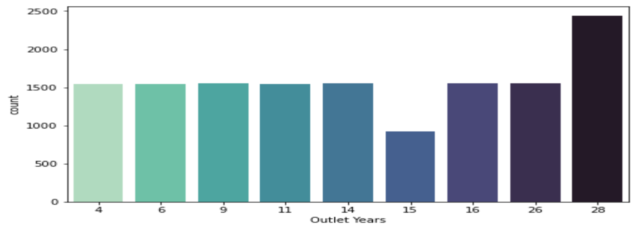


Plot17 – Analysis on Item MRP and Sales

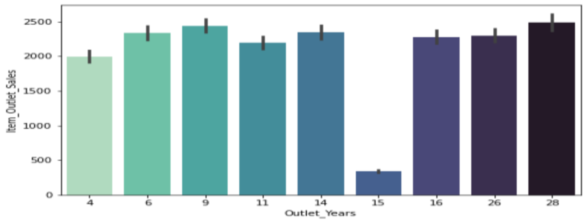
We have good amount of products for 50 MRP, 100 MRP ,180 MRP

But MRP ranging from 200-250 dollars is having high Sales.

Outlet Year



Plot18 – Analysis on Outlet Year

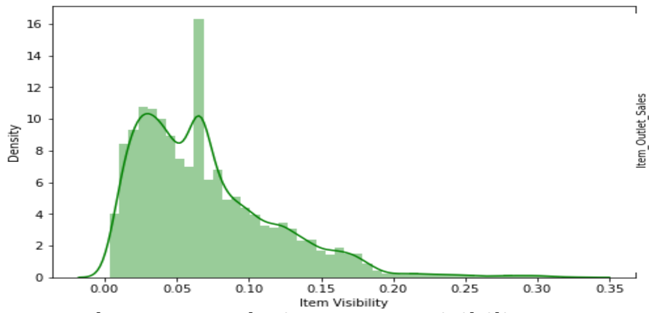


Plot19 – Analysis on Outlet Year and Sales

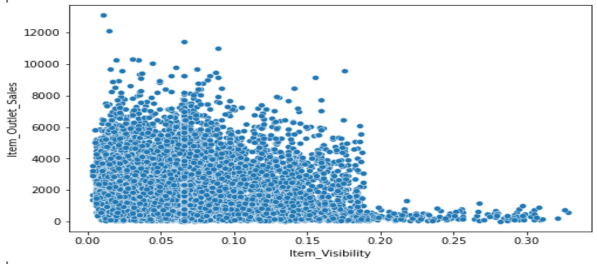
It is quiet evident that Outlets established 28 years before is having good Sales margin.

We also have a outlet which was established before 15 years has the lowest sales margin, so established years wouldn't improve the Sales unless the products are sold according to customer's interest.

Item Visibility



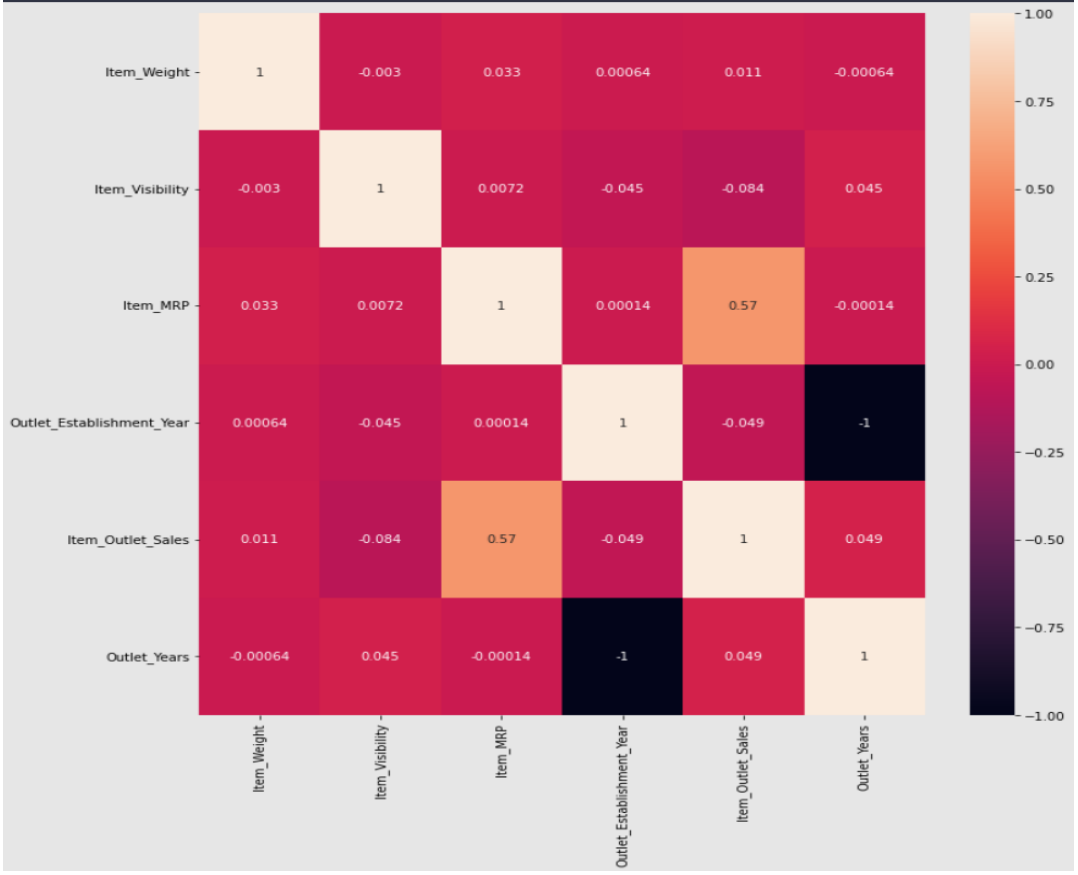
Plot20 – Analysis on Item Visibility



Plot21 – Analysis on Item Visibility and Sales

We have Items having Visibility 0 to 0.2 is more. And Items having Visibility around 0.05 is maximum. Sales is more for Items having Visibility 0 to 0.2

Heatmap



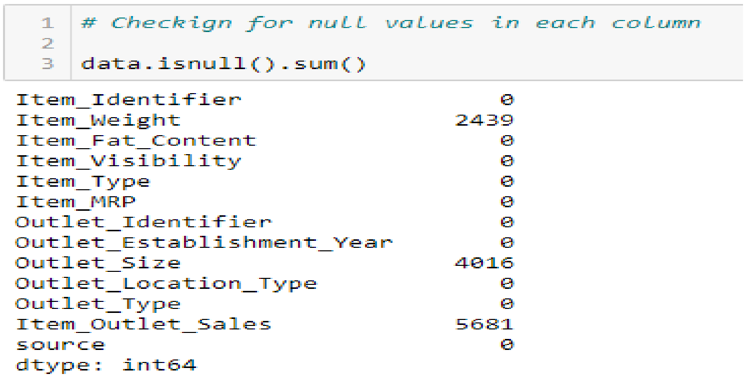
Plot22 – Heatmap

We can see Item_Outlet_Sales is highly correlated with Item_MRP, i.e. if Item_MRP increases, Item_Outlet_Sales increases. Item Visibility column has slightly negative correlation with the Item Outlet Sales means Sales decreses with increase in visibility Outlet Years and Item weight have no correlation with the Item Outlet Sales.

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. Un-cleaned 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:-

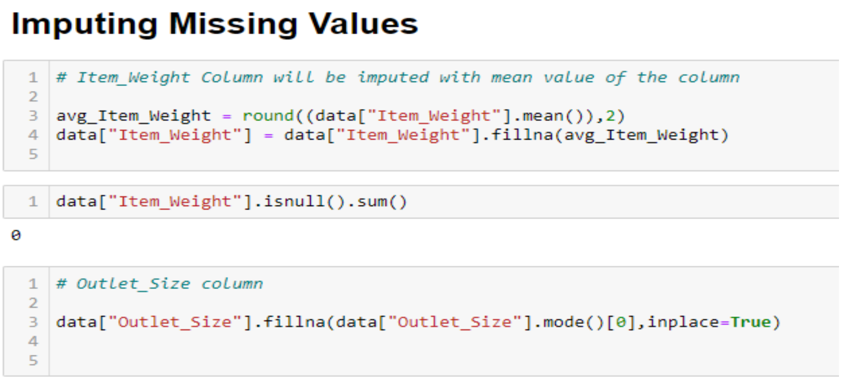


```
# Checkign for null values in each column

data.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                  2439
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  4016
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales            5681
source                          0
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 body and acidity 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().



Imputing Missing Values

```
# Item_Weight Column will be imputed with mean value of the column

avg_Item_Weight = round((data["Item_Weight"].mean()),2)
data["Item_Weight"] = data["Item_Weight"].fillna(avg_Item_Weight)

```

```
data["Item_Weight"].isnull().sum()
```

```
0
```

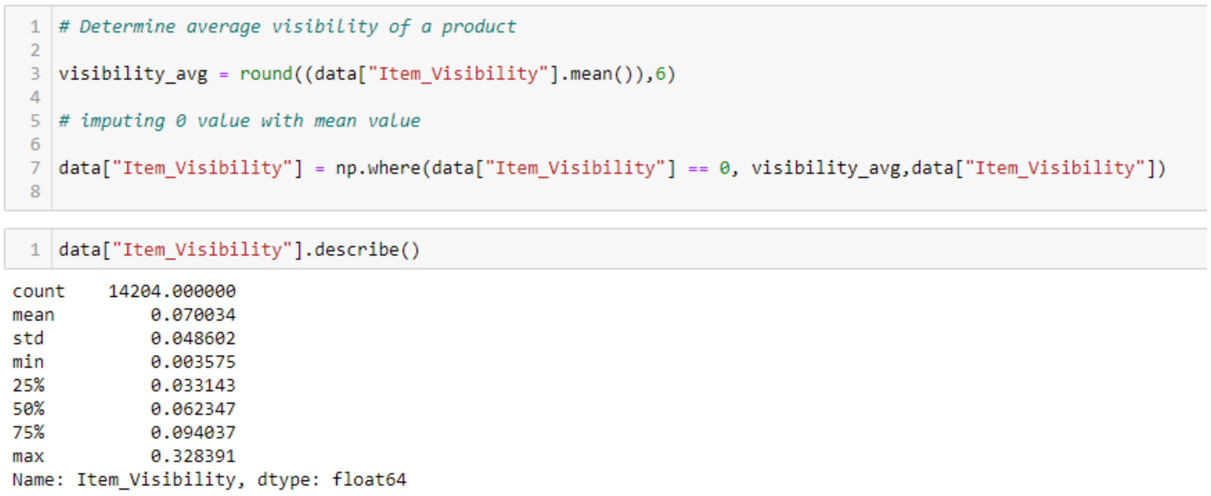
```
# Outlet_Size column

data["Outlet_Size"].fillna(data["Outlet_Size"].mode()[0],inplace=True)


```

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.

<u>Need for Variable Transformation</u>:-

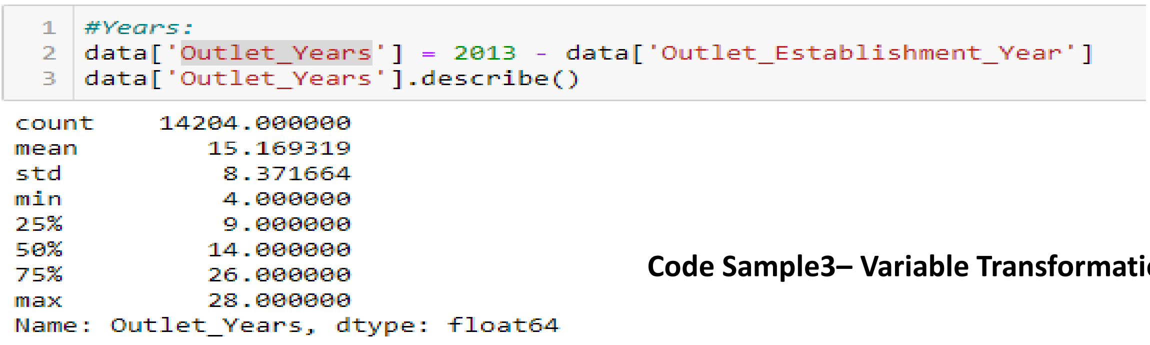
- Variable transformation is a way to make the data work better in your model. Here specifically data type of year column is object , we need to convert it in numerical type. The need for this is because we need our model to have a good score and accuracy which will make good predictions. So to feed the data to our model we must ensure to take these steps and make our data insightful.



```
# Determine average visibility of a product

visibility_avg = round((data["Item_Visibility"].mean()),6)

# imputing 0 value with mean value

data["Item_Visibility"] = np.where(data["Item_Visibility"] == 0, visibility_avg,data["Item_Visibility"])

```

```
data["Item_Visibility"].describe()
```

```
count    14204.000000
mean         0.070034
std          0.048602
min          0.003575
25%          0.033143
50%          0.062347
75%          0.094037
max          0.328391
Name: Item_Visibility, dtype: float64
```



```
#Years:
data['Outlet_Years'] = 2013 - data['Outlet_Establishment_Year']
data['Outlet_Years'].describe()
```

```
count    14204.000000
mean        15.169319
std          8.371664
min          4.000000
25%          9.000000
50%         14.000000
75%         26.000000
max         28.000000
Name: Outlet_Years, dtype: float64
```

Code Sample3– Variable Transformation

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.

- 'Item_Identifier','Outlet_Identifier' are just id , dont have any relation with the sales
- 'Item_Type','source' are also irrelevant column with the context

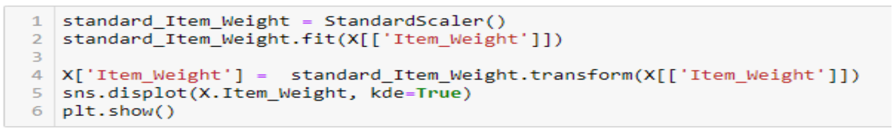
Feature Scaling

- Feature scaling is important for every algorithm where distance matter. Two famous techniques for Feature Scaling are:

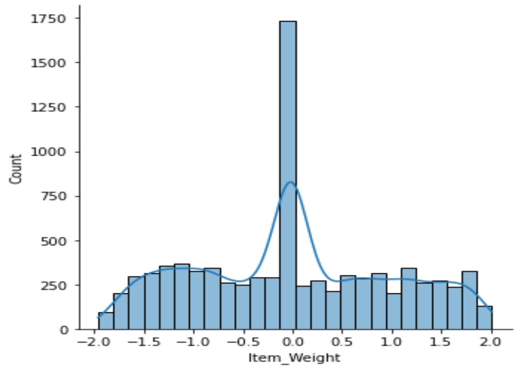
1. Normalization
2. Standardization

Standardization is useful when the feature distribution is Normal or Gaussian, otherwise we do Normalization.

1. Item Weight

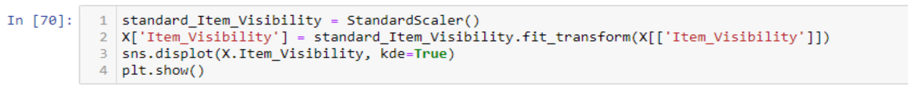


```
standard_Item_Weight = StandardScaler()
standard_Item_Weight.fit(X[['Item_Weight']])

X['Item_Weight'] =  standard_Item_Weight.transform(X[['Item_Weight']])
sns.displot(X.Item_Weight, kde=True)
plt.show()
```

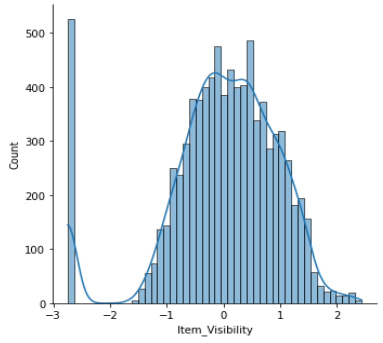


Code Sample4 – Standardization on Item Weight

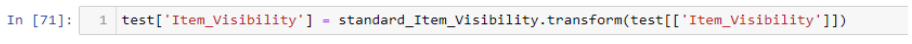
Item Visibility



```
In [70]: standard_Item_Visibility = StandardScaler()
         X['Item_Visibility'] = standard_Item_Visibility.fit_transform(X[['Item_Visibility']])
         sns.displot(X.Item_Visibility, kde=True)
         plt.show()
```

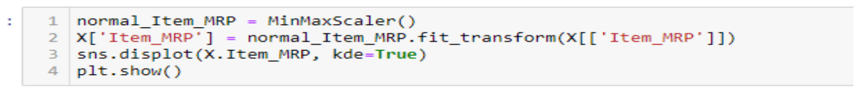


Code Sample5 – Standardization on Item Visibility

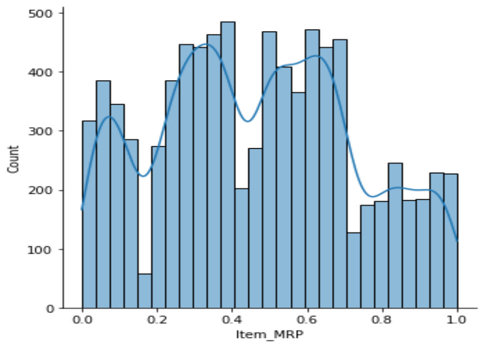


```
In [71]: test['Item_Visibility'] = standard_Item_Visibility.transform(test[['Item_Visibility']])
```

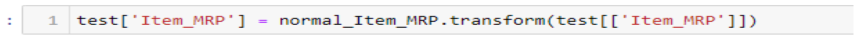
Item MRP



```
normal_Item_MRP = MinMaxScaler()
X['Item_MRP'] = normal_Item_MRP.fit_transform(X[['Item_MRP']])
sns.displot(X.Item_MRP, kde=True)
plt.show()
```



Code Sample6 – Normalization on Item MRP



```
test['Item_MRP'] = normal_Item_MRP.transform(test[['Item_MRP']])
```

4. Modeling Building

Model Selection and Why?

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

Machine Learning is a technique that analyses 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.

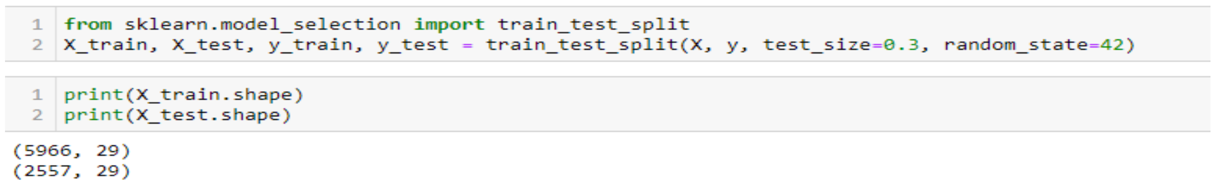
First, we need to separate the dataset into two parts: features (property attributes) and labels (prices) which is the required format for any model to be trained on.

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.



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

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

```
(5966, 29)
(2557, 29)
```

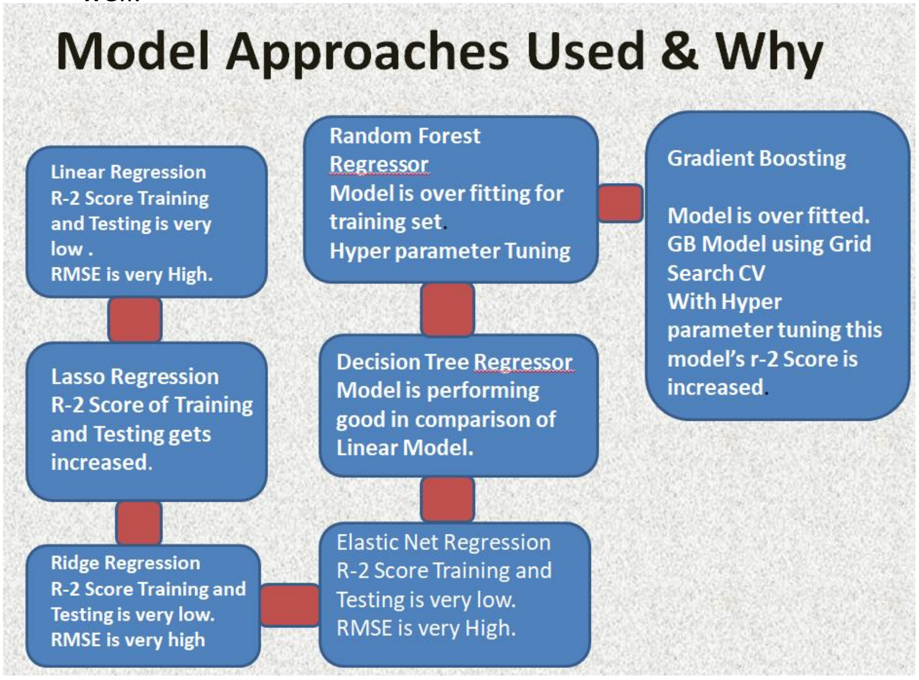
Code Sample 7: Splitting Data

We need to build different regression algorithms and using the testing 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:

Following Models are used for predicting Sales of Big Mart Store wise:

- **Linear Regression**
- **Lasso Regression**
- **Ridge Regression**
- **Elastic net Regression**
- **Decision Tree**
- **Random Forest**
- **Gradient Boosting**

- Initially, we've tried Linear Regression and its variants Lasso (l1 norm) , Ridge (l2 norm) Regression and Elastic net Regression , but they are performing quite similarly on the 2 sets (train & test).
- They are not giving good result on train, neither test set , with lasso giving low r-2 score for both train test set which is not good.
- Let's look at some other algorithms, and how they are performing as compared to linear regression
- So, **Decision Tree Regression** is giving score of 0.59 on the train set and similarly on the valid and test set.
- If we look at the Random Forest Regressor is giving quite good score of 0.94 on the train set, but not that good on the valid and test set, looks like it is overfitting on the training data.
- 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.
- Same Case with the Gradient boosting regressor , Over fitting is seen in both of these as well.

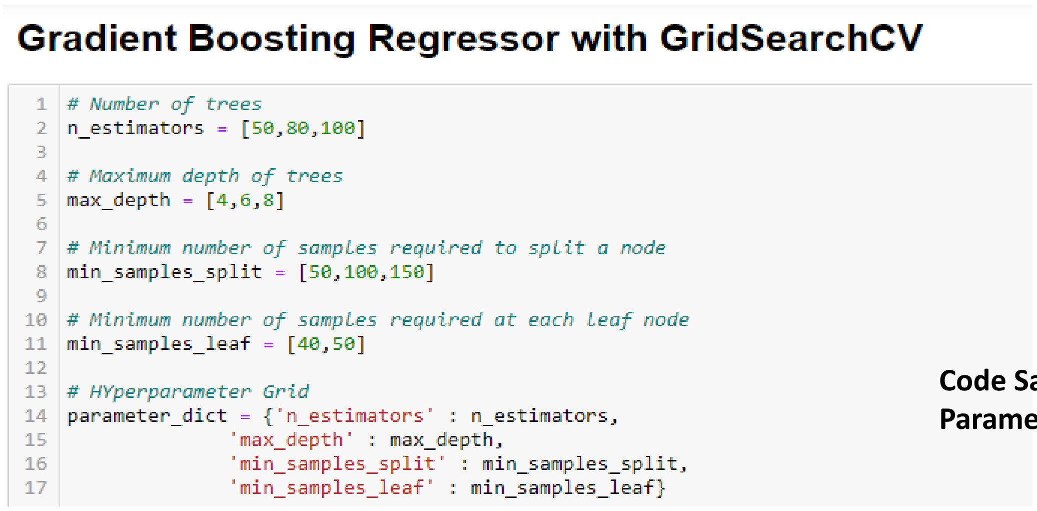

Model Approaches Used & Why
Linear Regression R-2 Score Training and Testing is very low . RMSE is very High.
Lasso Regression R-2 Score of Training and Testing gets increased.
Ridge Regression R-2 Score Training and Testing is very low. RMSE is very high
Elastic Net Regression R-2 Score Training and Testing is very low. RMSE is very High.
Decision Tree Regressor Model is performing good in comparison of Linear Model.
Random Forest Regressor Model is over fitting for training set. Hyper parameter Tuning
Gradient Boosting Model is over fitted. GB Model using Grid Search CV With Hyper parameter tuning this model's r-2 Score is increased.

Flow Diagram 1- Flow of the Model Used and Why

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 hyper parameters 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.

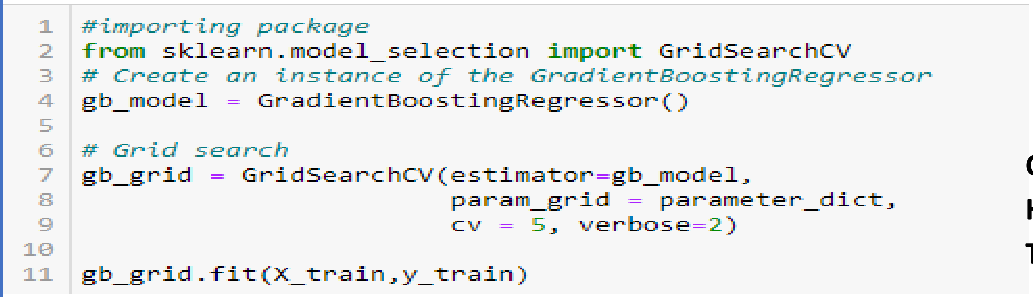


Gradient Boosting Regressor with GridSearchCV

```python
# Number of trees
n_estimators = [50,80,100]

# Maximum depth of trees
max_depth = [4,6,8]

# Minimum number of samples required to split a node
min_samples_split = [50,100,150]

# Minimum number of samples required at each leaf node
min_samples_leaf = [40,50]

# HYperparameter Grid
parameter_dict = {'n_estimators' : n_estimators,
              'max_depth' : max_depth,
              'min_samples_split' : min_samples_split,
              'min_samples_leaf' : min_samples_leaf}
```

Code Sample8 – Best Parameter Grid

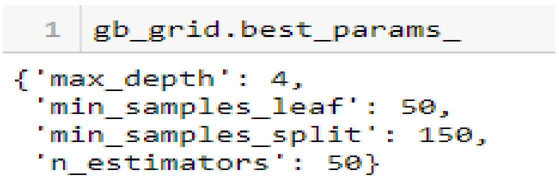
Then we need to provide it with the estimator and specify other things such as how many cross-validations sets to evaluate upon.



```python
#importing package
from sklearn.model_selection import GridSearchCV
# Create an instance of the GradientBoostingRegressor
gb_model = GradientBoostingRegressor()

# Grid search
gb_grid = GridSearchCV(estimator=gb_model,
                       param_grid = parameter_dict,
                       cv = 5, verbose=2)

gb_grid.fit(X_train,y_train)
```

Code Sample9 – Hyper parameter Tuning

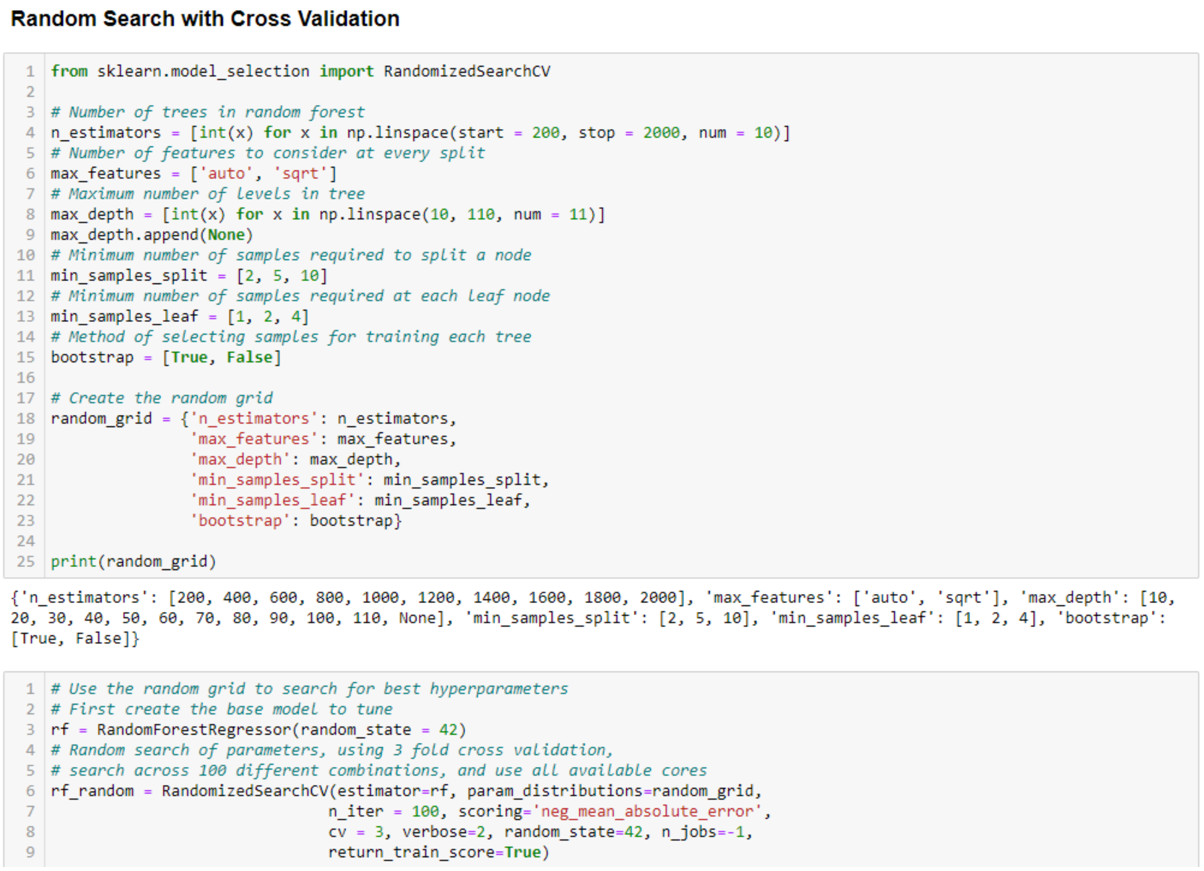
After fitting GridSearchCV it returns those params with which it got the best score.



```python
gb_grid.best_params_
```

```
{'max_depth': 4,
 'min_samples_leaf': 50,
 'min_samples_split': 150,
 'n_estimators': 50}
```

Similarly, we've done it using RandomSearchCV with Random Forest Regression , 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 Random Forest Regression Model can be used to make the final predictions.


Random Search with Cross Validation

```python
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [10,
20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap':
[True, False]}
```

```python
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor(random_state = 42)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                              n_iter = 100, scoring='neg_mean_absolute_error',
                              cv = 3, verbose=2, random_state=42, n_jobs=-1,
                              return_train_score=True)
```

Code Sample10 – Hyper parameter Tuning

Performance Metrics

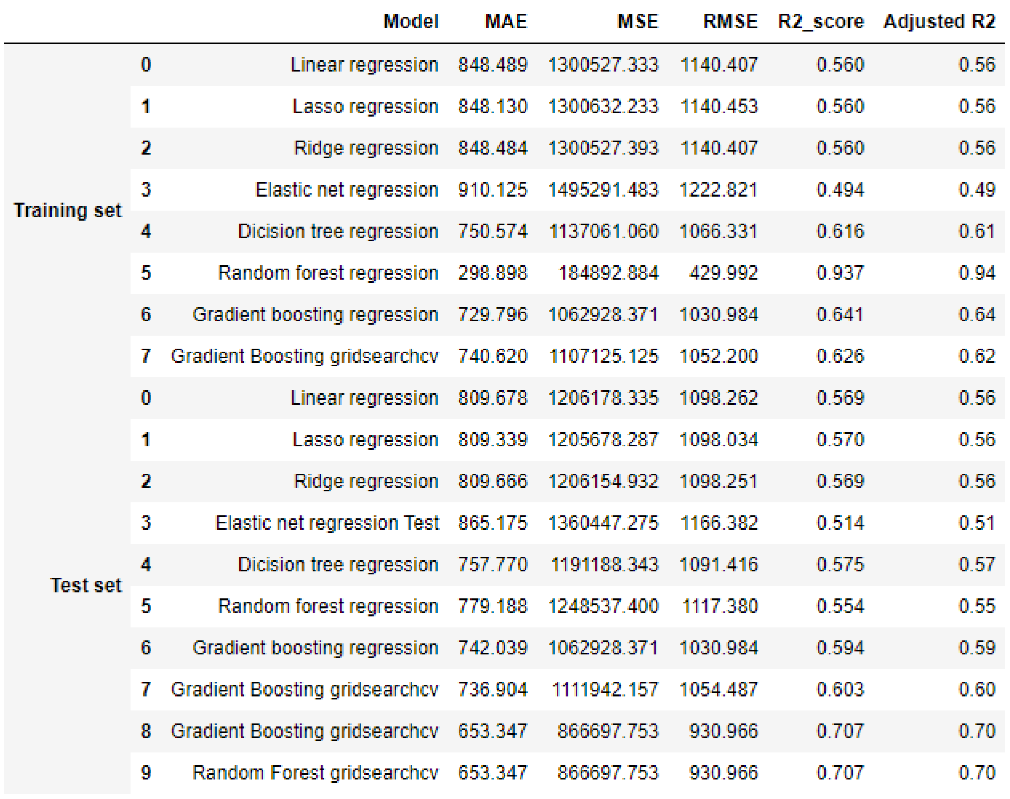
Just building is not enough, as we need to evaluate it using different metrics based on the problem we're solving. Model Validation helps us to understand how well the model is generalizing on the real-world data, the data which it has not seen during the training phase.

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 R2

The R2 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



		Model	MAE	MSE	RMSE	R2_score	Adjusted R2
Training set	0	Linear regression	848.489	1300527.333	1140.407	0.560	0.56
	1	Lasso regression	848.130	1300632.233	1140.453	0.560	0.56
	2	Ridge regression	848.484	1300527.393	1140.407	0.560	0.56
	3	Elastic net regression	910.125	1495291.483	1222.821	0.494	0.49
	4	Dicision tree regression	750.574	1137061.060	1066.331	0.616	0.61
	5	Random forest regression	298.898	184892.884	429.992	0.937	0.94
	6	Gradient boosting regression	729.796	1062928.371	1030.984	0.641	0.64
	7	Gradient Boosting gridsearchcv	740.620	1107125.125	1052.200	0.626	0.62
Test set	0	Linear regression	809.678	1206178.335	1098.262	0.569	0.56
	1	Lasso regression	809.339	1205678.287	1098.034	0.570	0.56
	2	Ridge regression	809.666	1206154.932	1098.251	0.569	0.56
	3	Elastic net regression Test	865.175	1360447.275	1166.382	0.514	0.51
	4	Dicision tree regression	757.770	1191188.343	1091.416	0.575	0.57
	5	Random forest regression	779.188	1248537.400	1117.380	0.554	0.55
	6	Gradient boosting regression	742.039	1062928.371	1030.984	0.594	0.59
	7	Gradient Boosting gridsearchcv	736.904	1111942.157	1054.487	0.603	0.60
	8	Gradient Boosting gridsearchcv	653.347	866697.753	930.966	0.707	0.70
	9	Random Forest gridsearchcv	653.347	866697.753	930.966	0.707	0.70

Table4 - Final Algorithm Performance Table

5. Final Interpretation/Recommendation

In our analysis, we initially did EDA on all the features of our datset. We first analysed our dependent variable i.e, 'Item_Outlet_Sales'and also transformed it. Next we analysed categorical variable and performed feature engineering on some column dropped off the irrelevant ones. we also analysed numerical variable, check out the correlation, distribution and their relationship with the dependent variable. We also removed some numerical features and hot encoded the categorical variables.

overfitting is seen for random forest regression.

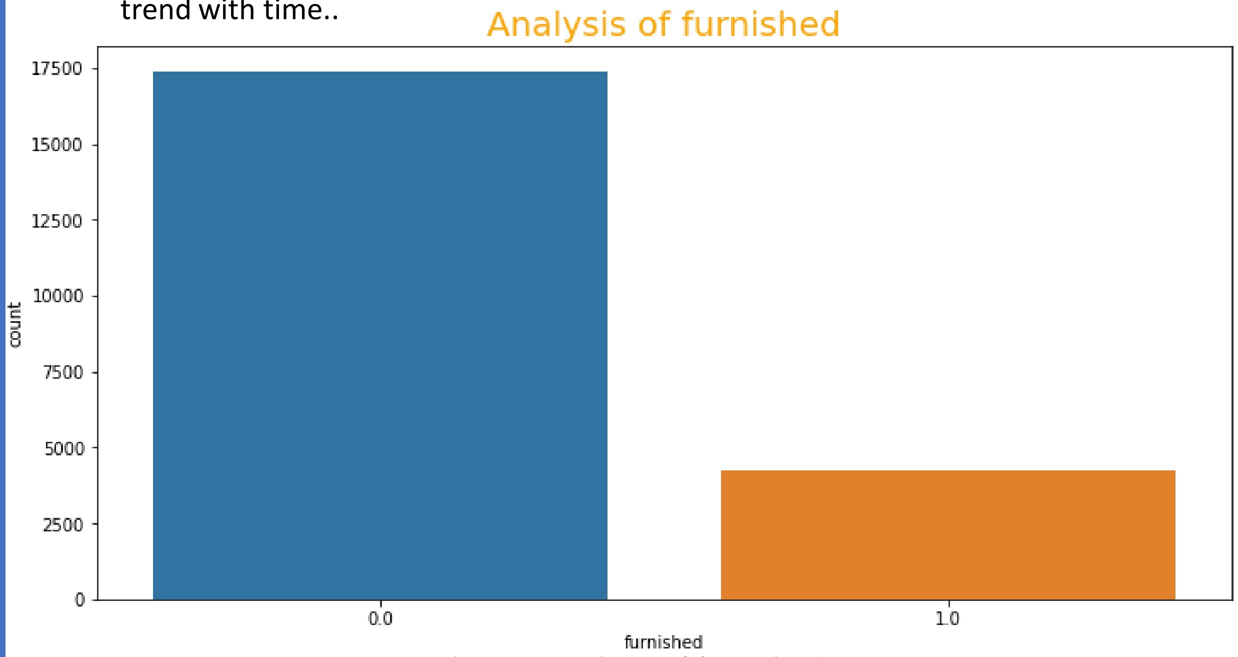
Random forest Regressor gives the highest R2 score of 94% for Train Set and Gradient Boosting gridsearchcv gives the highest R2 score of 70% for Test set,which is decent.

We can deploy this model.

Analysis of furnished:

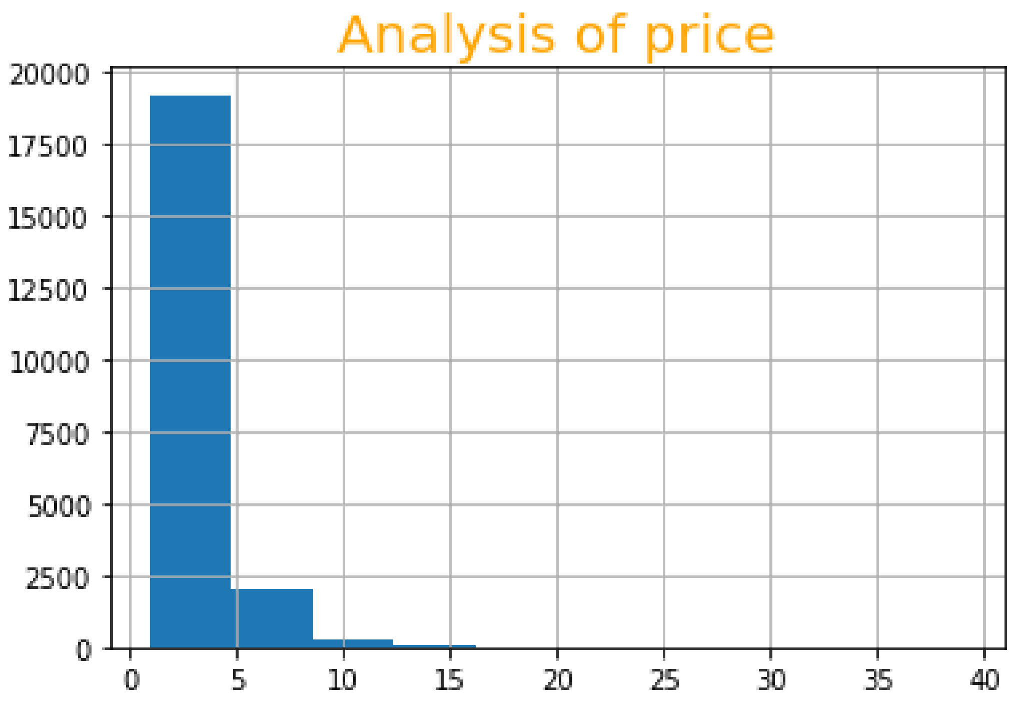
Observation:

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



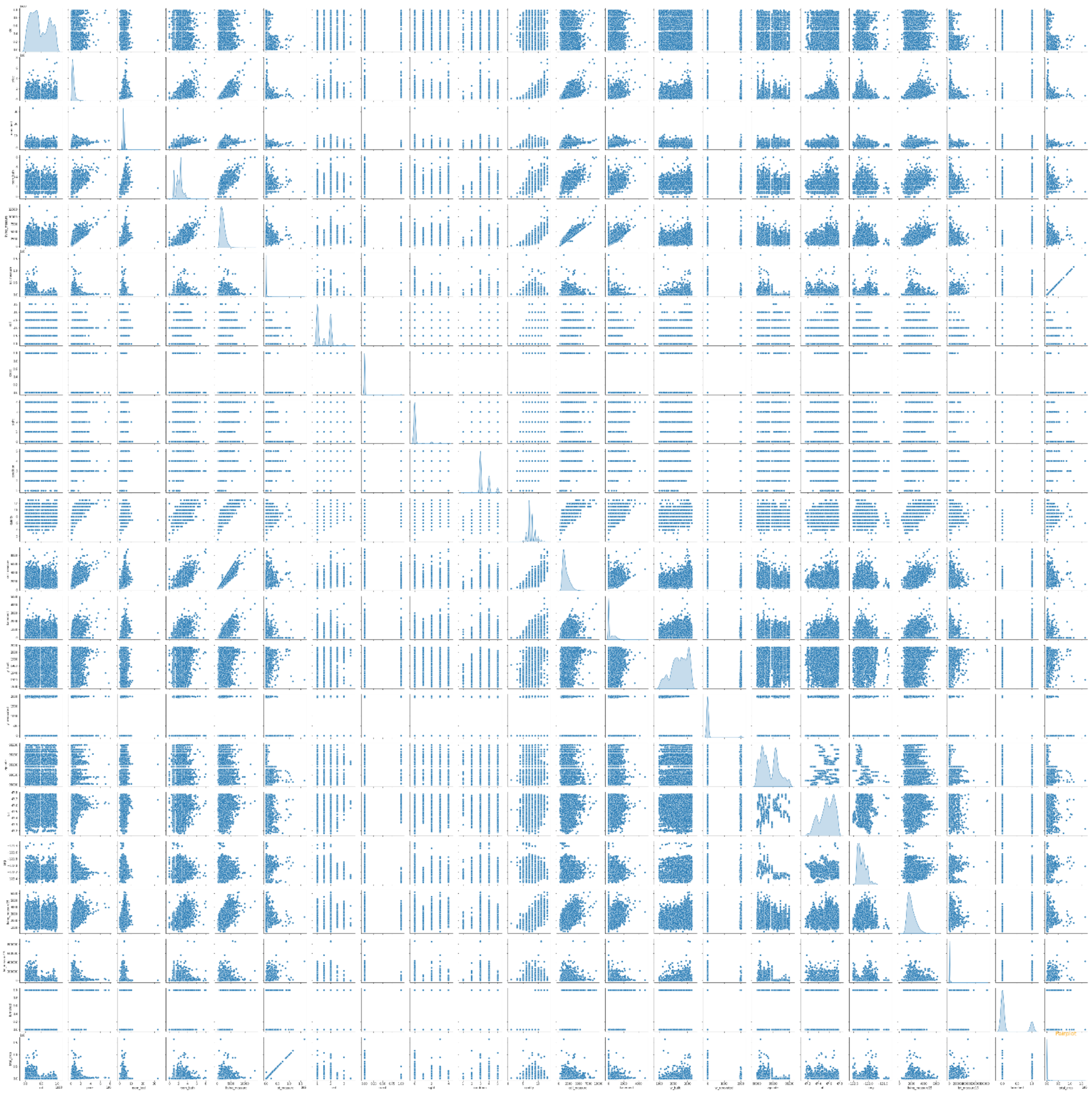
Plot 15- Analysis of furnished

Analysis of price:



Plot 16- Analysis of price

Bivariant Analysis- PairPlot



Plot 17- PairPlot

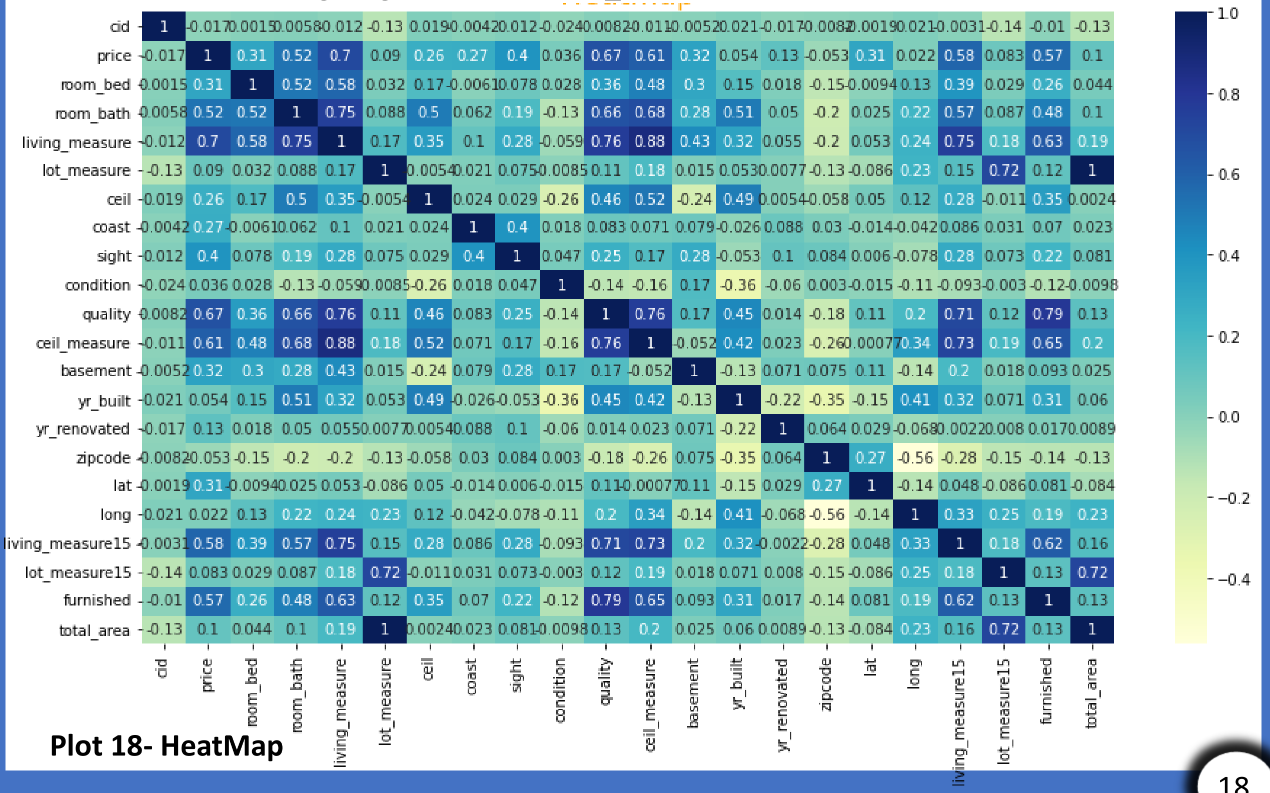
Observation:

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
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.



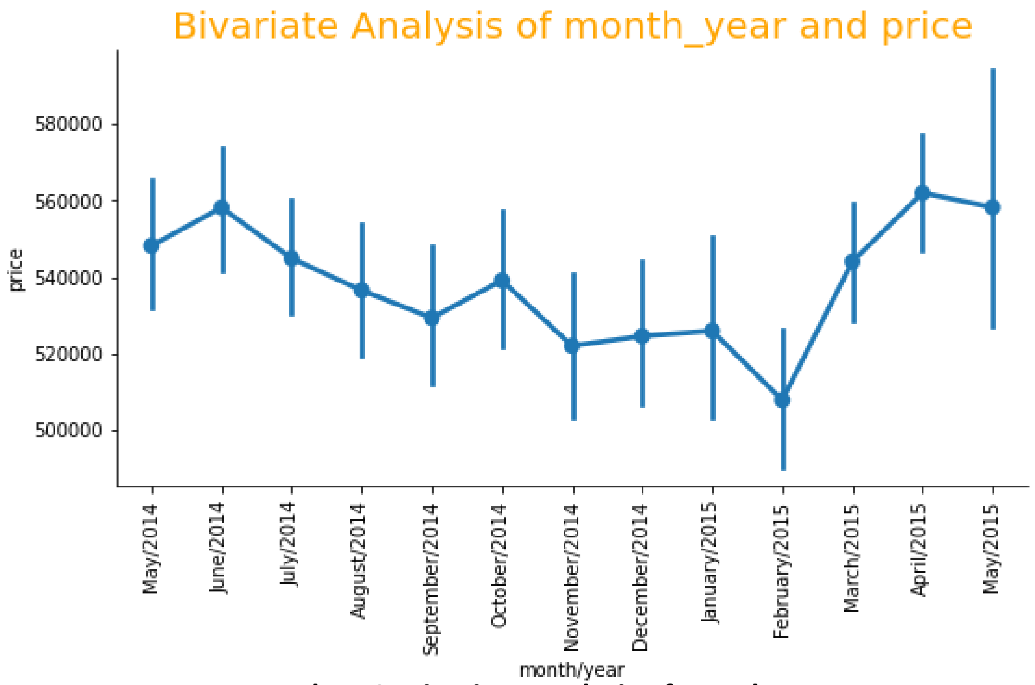
Plot 18- HeatMap

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.

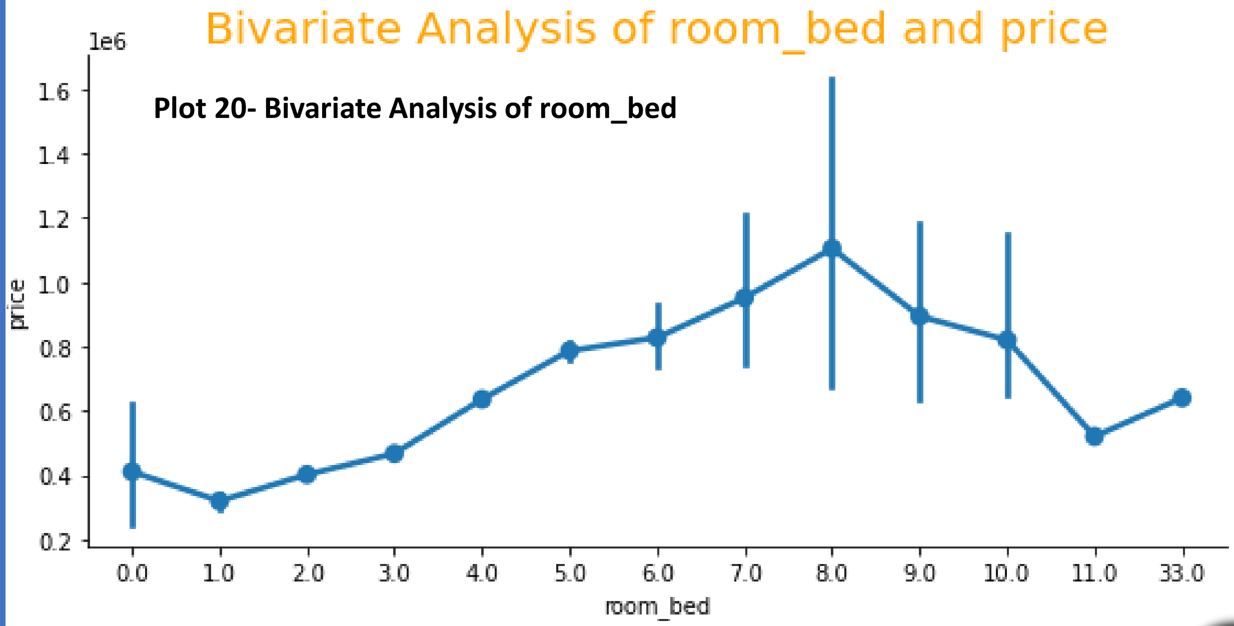


Plot 19- Bivariate Analysis of month year

Bivariate Analysis of room_bed

Observation:

- There is clear increasing trend in price with room_bed

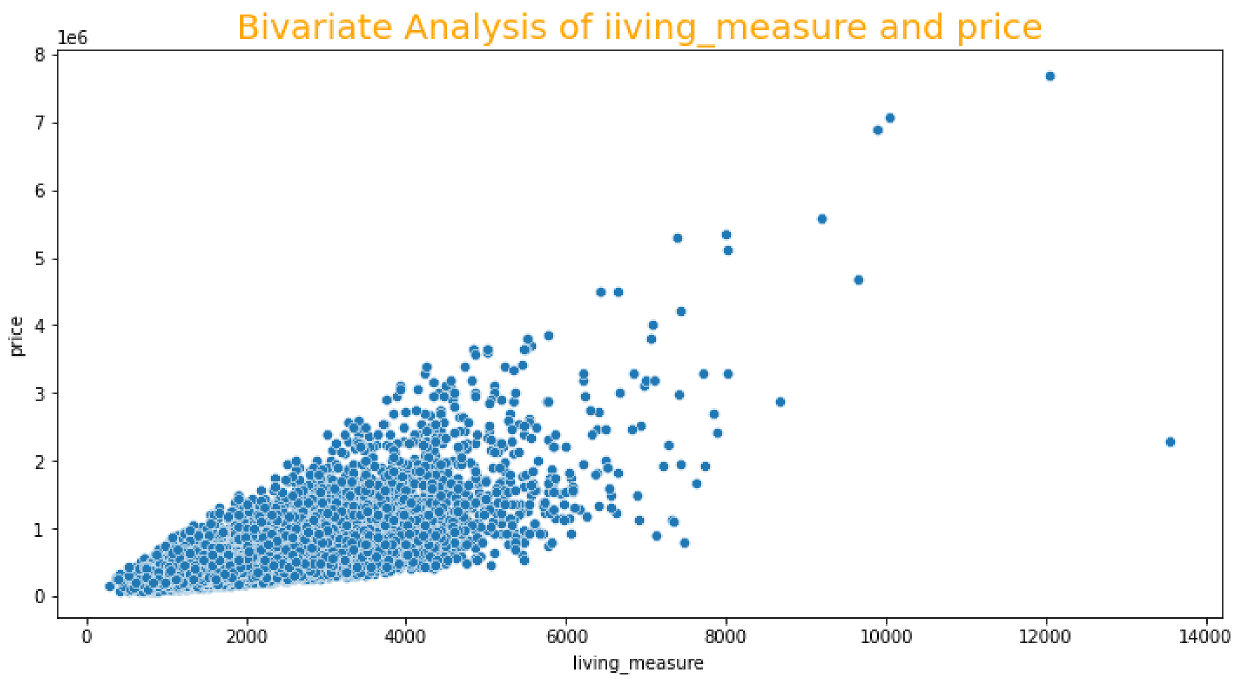


Plot 20- Bivariate Analysis of room_bed

Bivariate Analysis of living _measure

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



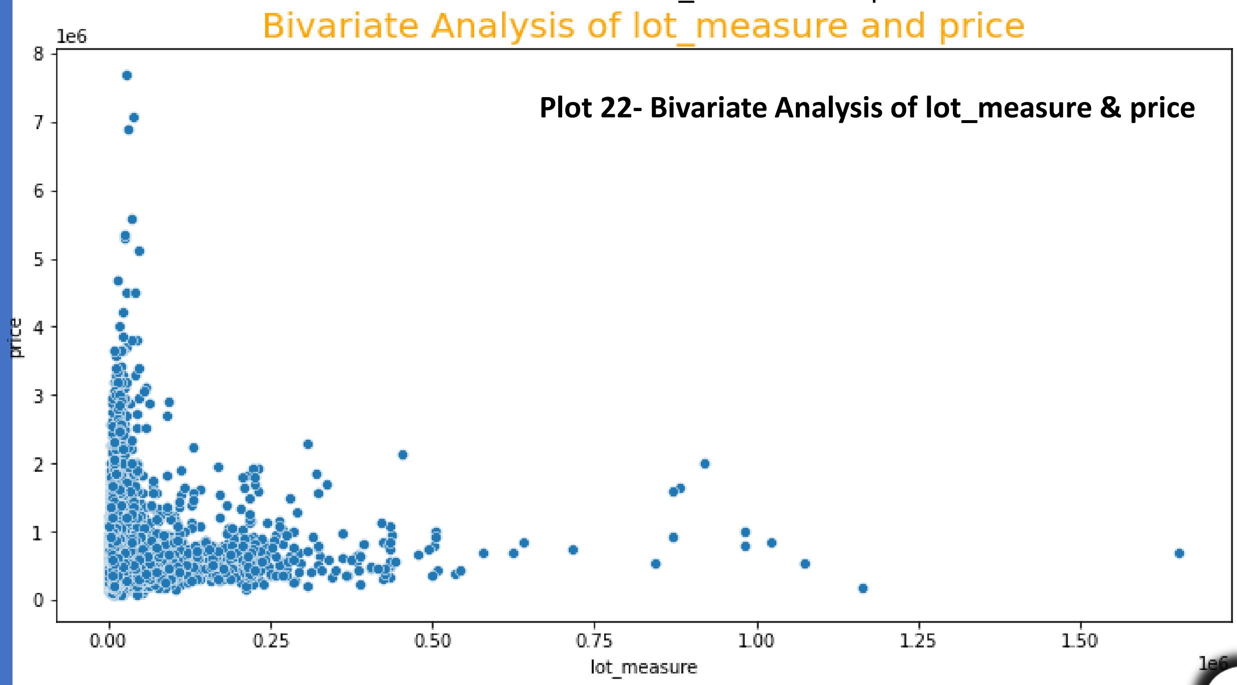
Plot 21- Bivariate Analysis of living_measure & price

Bivariate Analysis of lot_measure

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

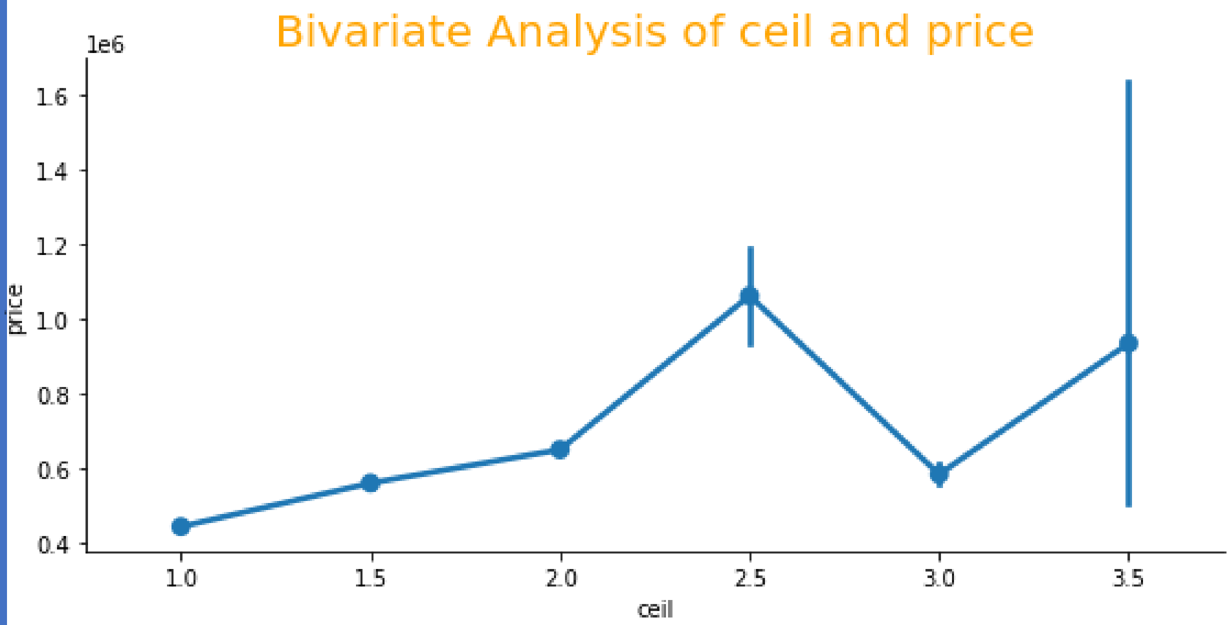


Plot 22- Bivariate Analysis of lot_measure & price

Bivariate Analysis of ceil

Observation:

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

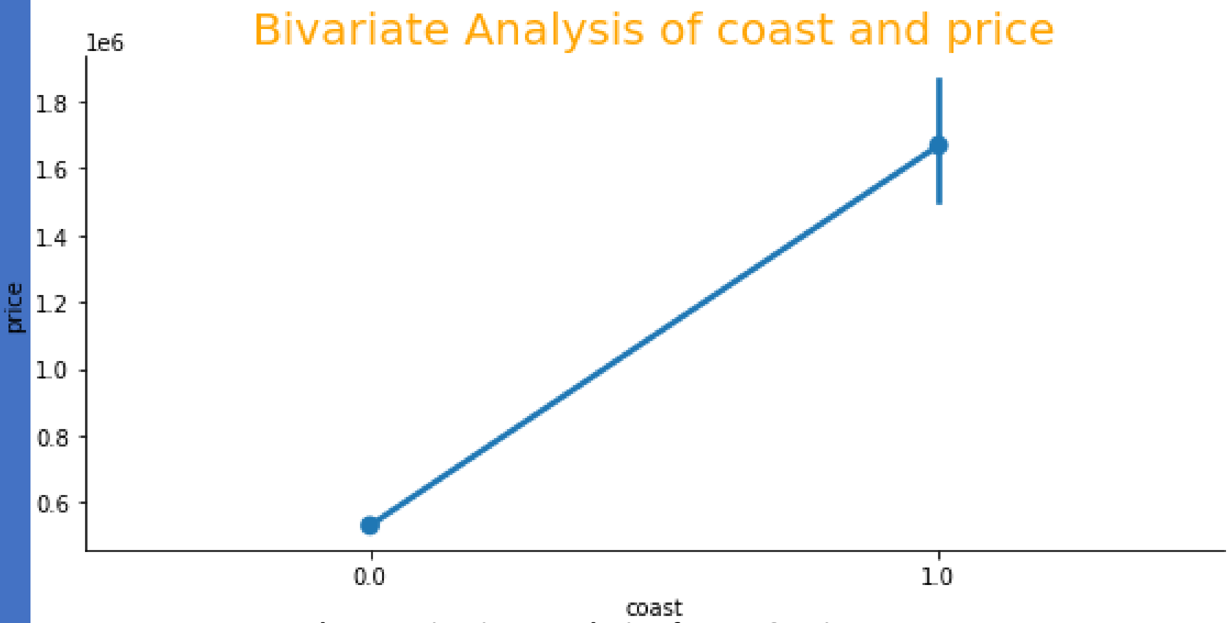


Plot 23- Bivariate Analysis of ceil & price

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

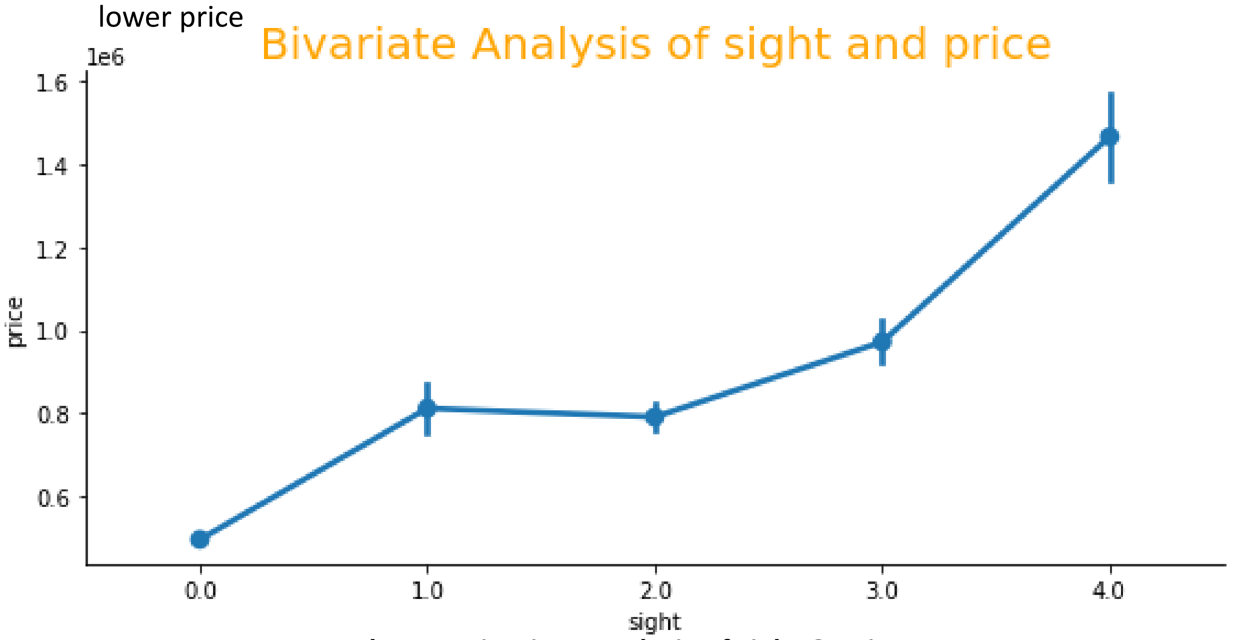


Plot 24- Bivariate Analysis of coast & price

Bivariate Analysis of sight and price

Observation:

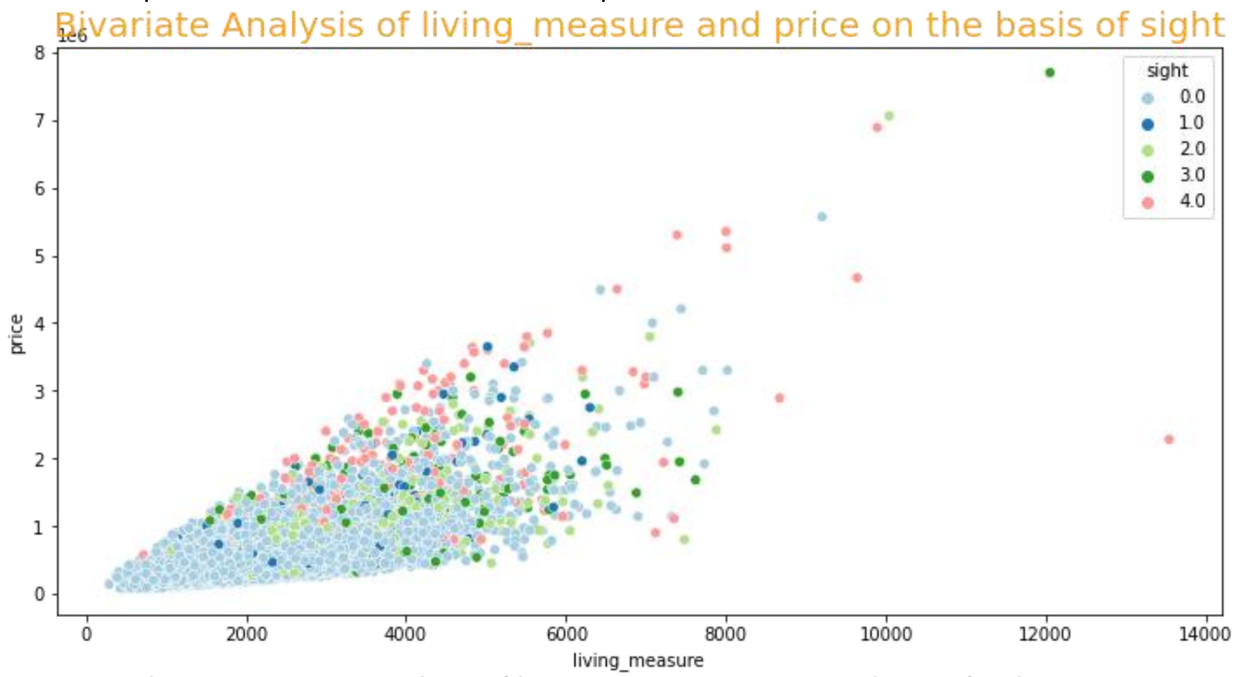
- It contains outliers
- The house sighted more have high price (mean and median) and have large living area as well.
- Properties with higher price have more no.of sights compared to that of houses with lower price



Plot 25- Bivariate Analysis of sight & 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

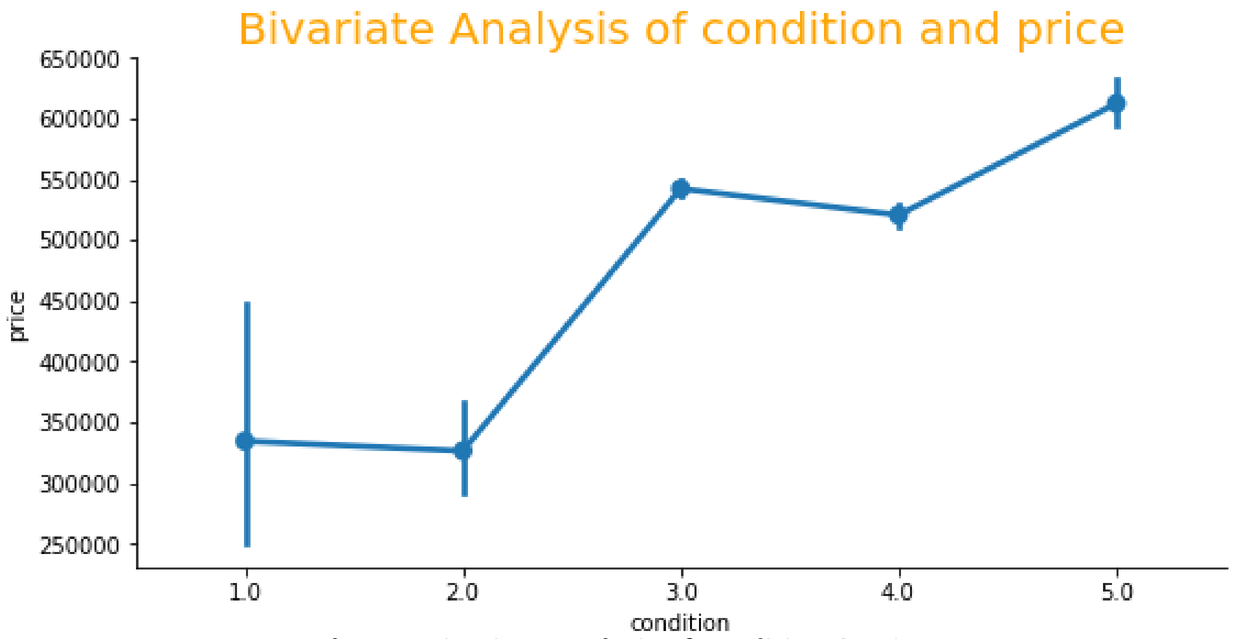


Plot 26- Bivariate Analysis of living_measure & price on basis of sight

Bivariate Analysis of condition

Observation:

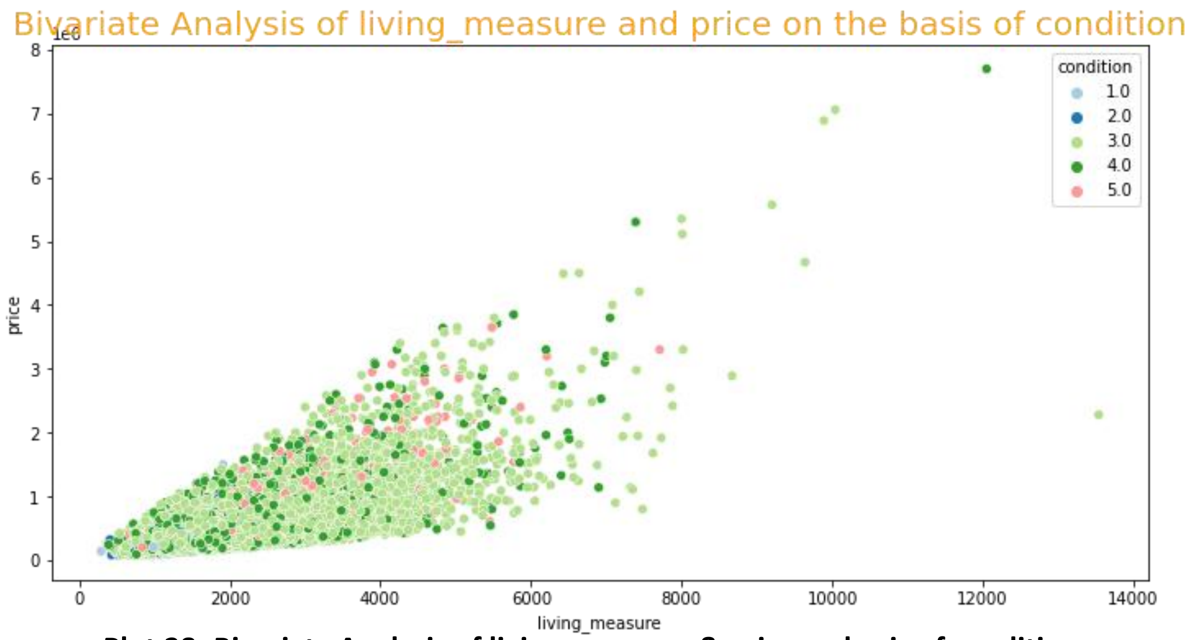
- As the condition rating increases its price and living measure mean and median also increases.
- The price of the house increases with condition rating of the house



Plot 27- Bivariate Analysis of condition & price

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

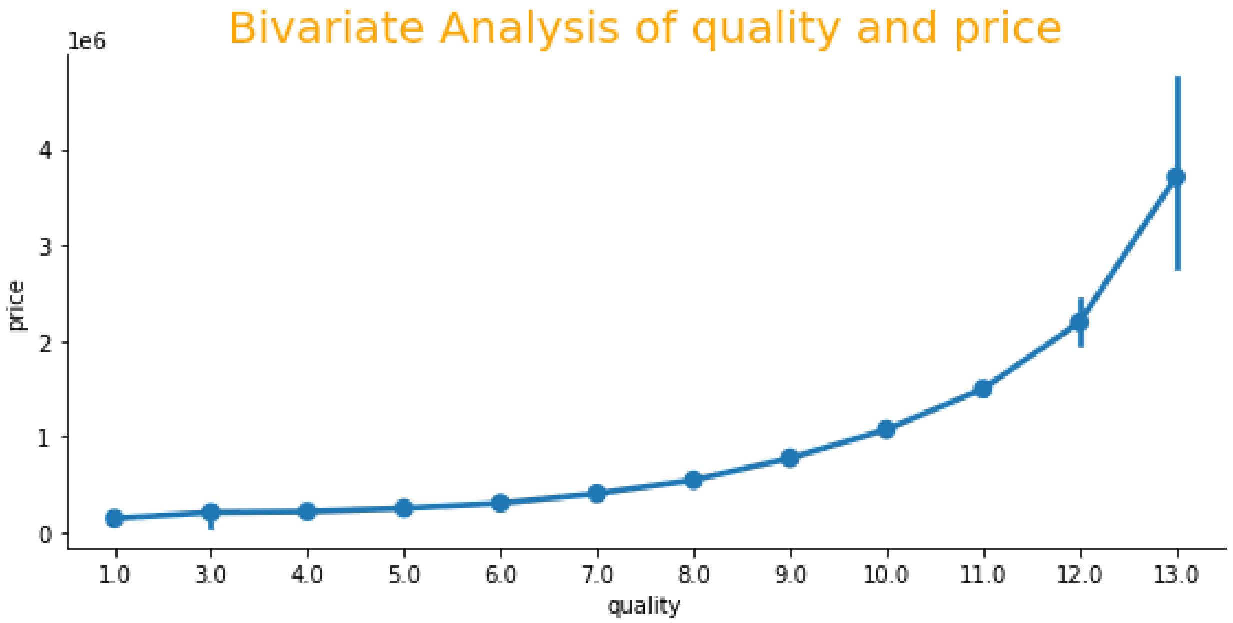


Plot 28- Bivariate Analysis of living_measure & price on basis of condition

Bivariate Analysis of quality

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

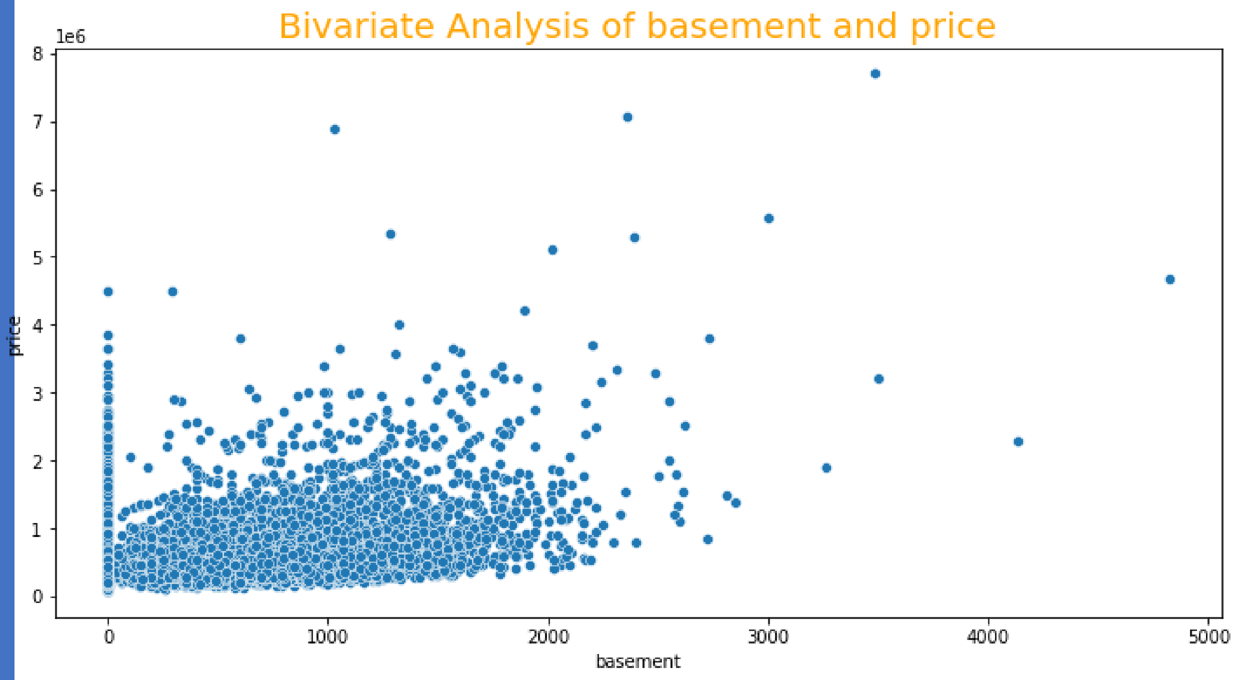


Plot 29- Bivariate Analysis of quality & price

Bivariate Analysis of basement

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

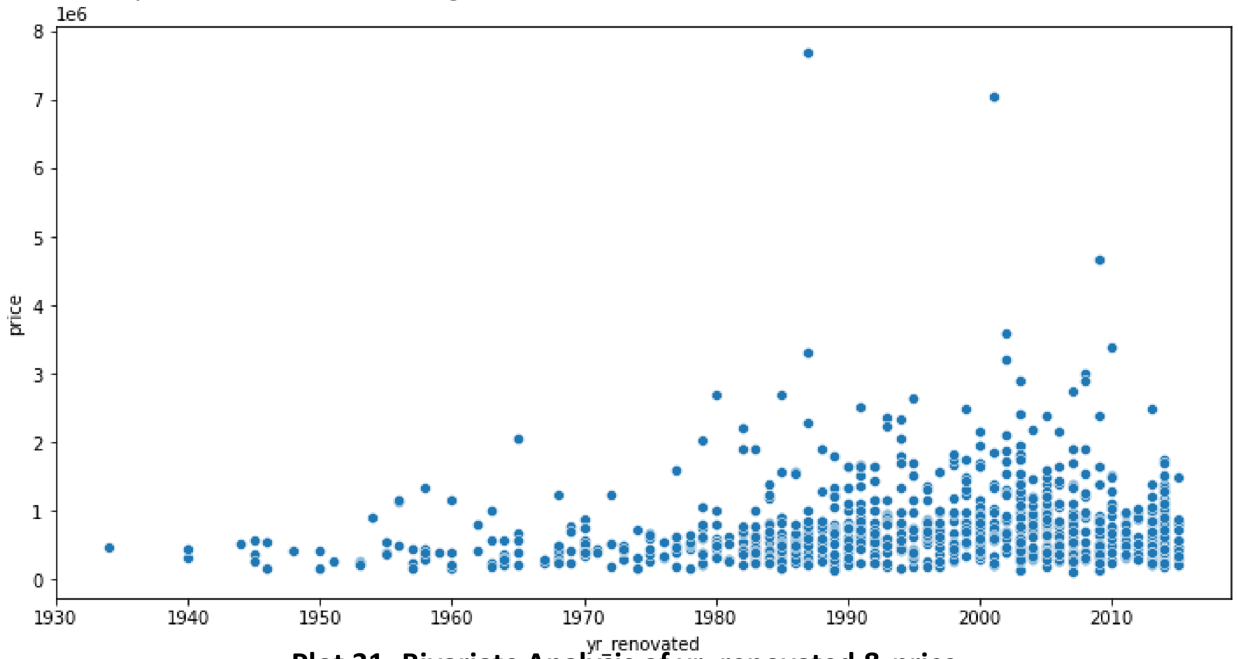


Plot 30- Bivariate Analysis of basement & price

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.

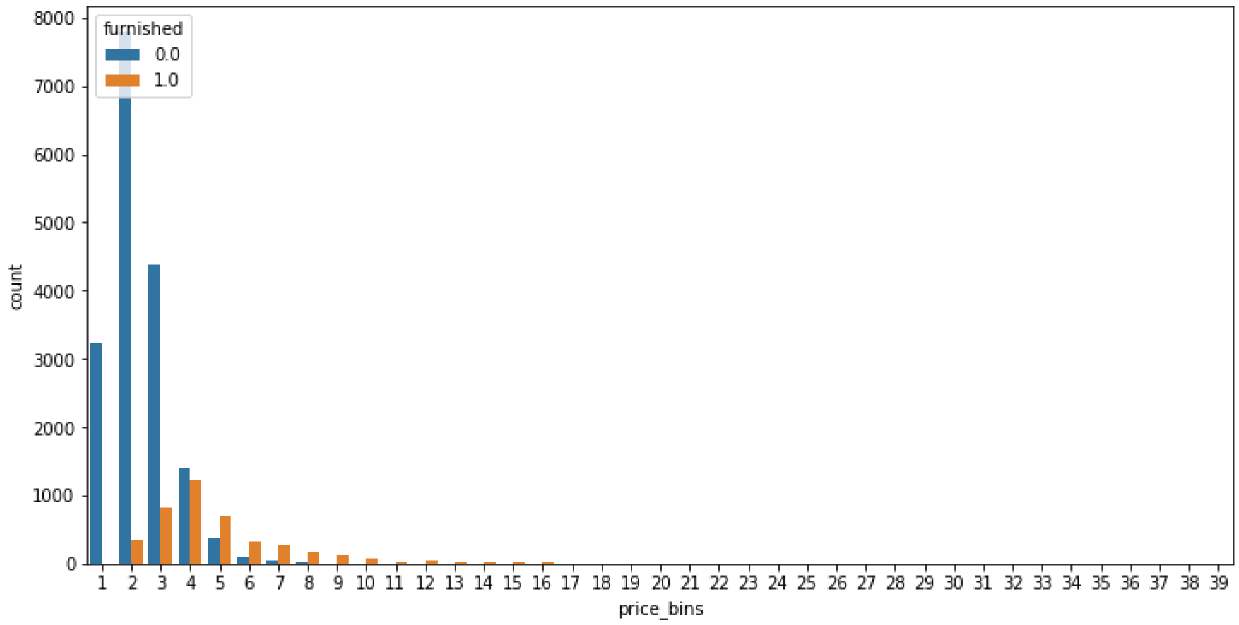


Plot 31- Bivariate Analysis of yr_renovated & price

Bivariate Analysis of furnished

Observation:

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

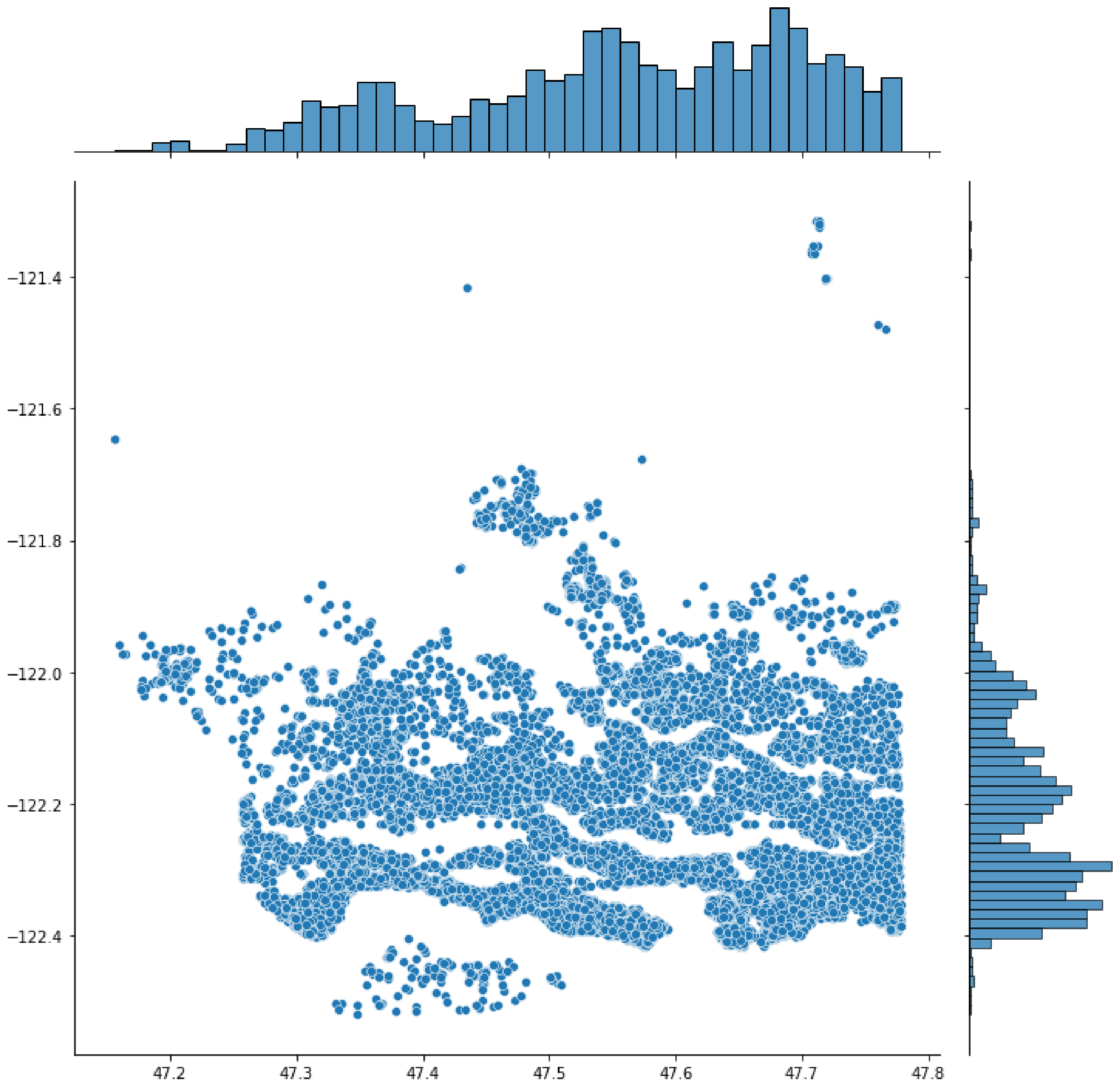


Plot 32- Bivariate Analysis of furnished & price

Visualising the location of the houses based on latitude and longitude

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

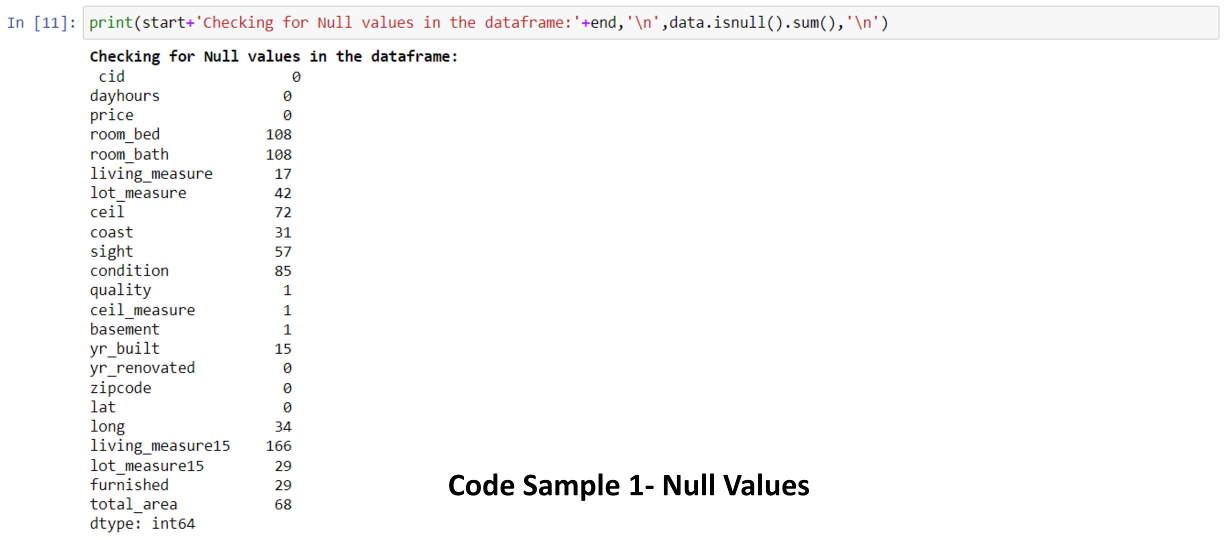


Plot 33- Visualizing the location of houses

3. Data Cleaning and 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.

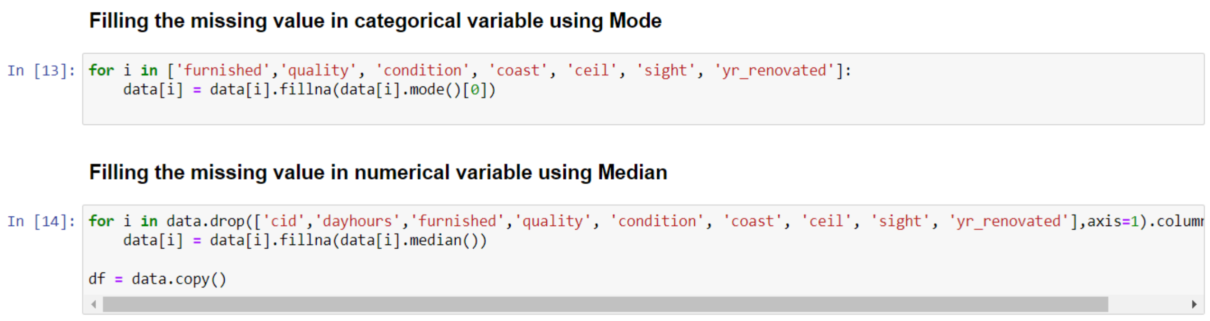
The approach used for identifying and treating missing values and outlier treatment:-



```
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])
```

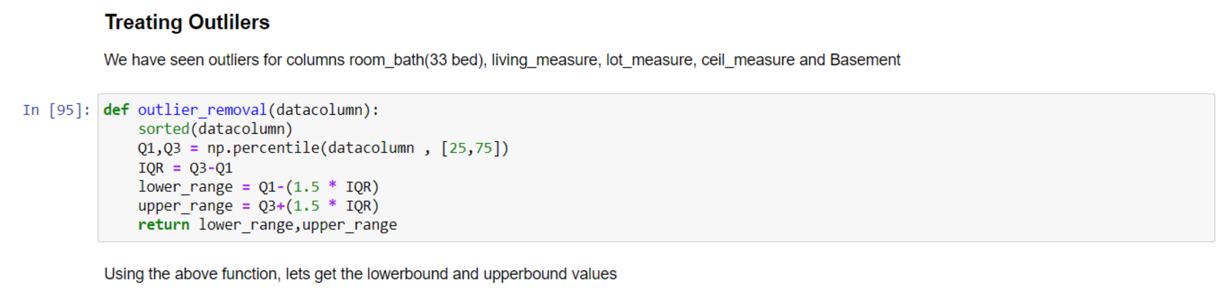
Filling the missing value in numerical variable using Median

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

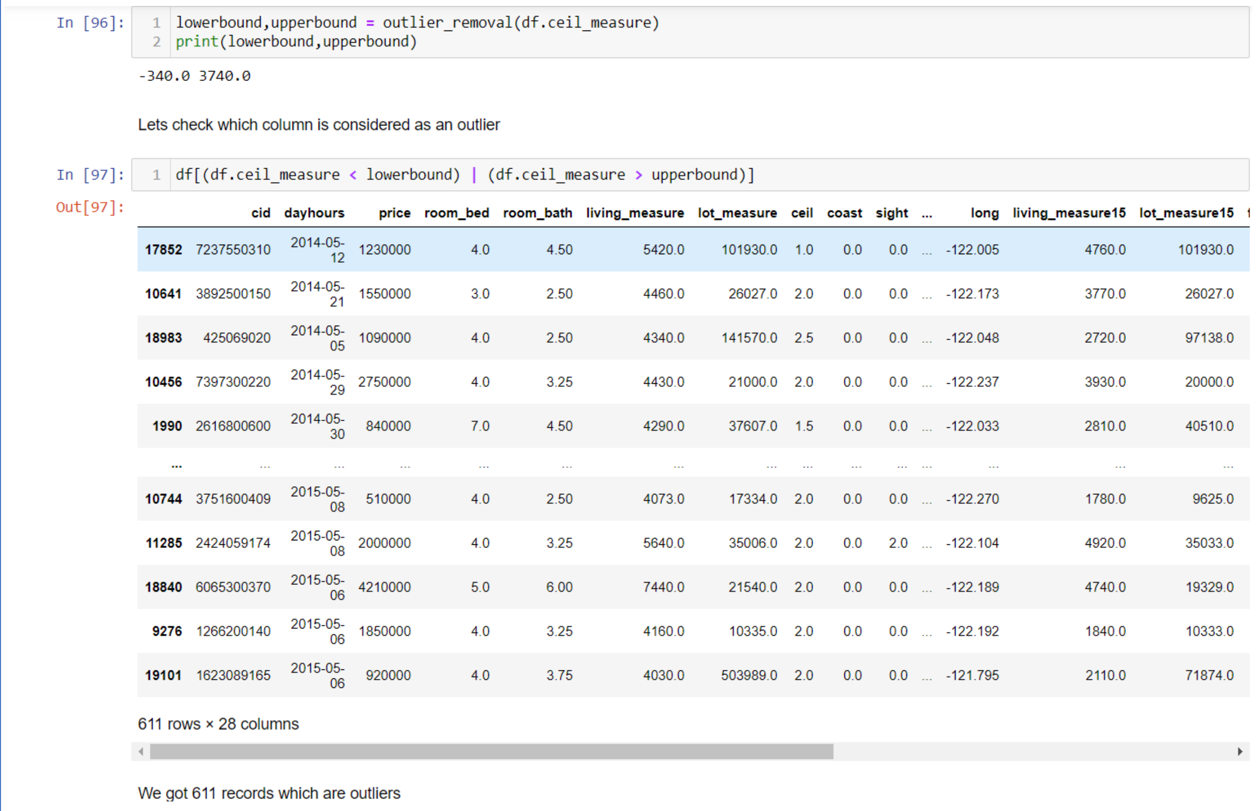
We have seen outliers for columns room_bath(33 bed), living_measure, lot_measure, ceil_measure and Basement

```
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
```

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

Code Sample 3- Outlier Detection

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]:
```

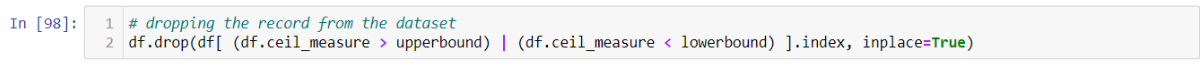
	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	long	living_measure15	lot_measure15
17852	7237550310	2014-05-12	1230000	4.0	4.50	5420.0	101930.0	1.0	0.0	0.0	...	-122.005	4760.0	101930.0
10641	3892500150	2014-05-21	1550000	3.0	2.50	4460.0	26027.0	2.0	0.0	0.0	...	-122.173	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

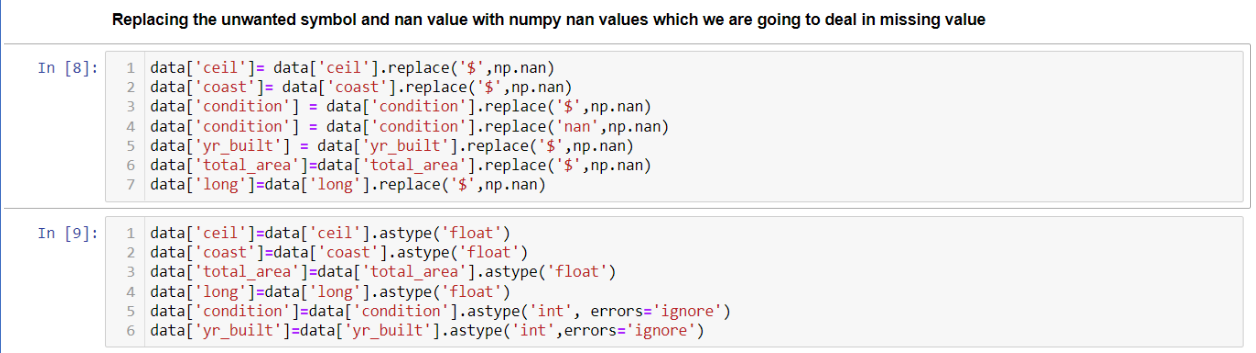
Here is a code snippet of a column named ceil_measure, were using our function outlier_removal we have got 611 records that are outliers for that particular column.



```
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 TransformationC

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.
We have added month/year for analyzing the dayhours columns for getting to know in which months sales of houses are more and another column that we have added is has_basement which for analyzing how basement are affecting the prices of the houses and we have removed certain columns from it using p-value by checking the significance of each variable. So, the removed feature are -
basement,ceil_2.0,furnished_1.0,room_bath_5.75,room_bed_10.0, room_bed_11.0,room_bed_4.0,room_bed_5.0,room_bed_6.0,room_bed_7.0,room_bed_8.0 ,room_bed_9.0 and the test we are performing the one hot encoding to the variables.

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.

```python
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)
```

```python
print(X_train.shape, y_train.shape)
print(X_test.shape,y_test.shape)
print(X_val.shape, y_val.shape)
```

Code Sample 5- Splitting Data

```
(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.

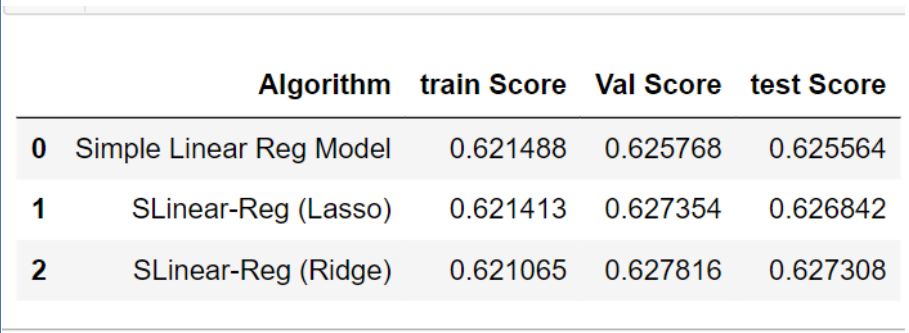
Here is the list of all the algorithms we've to build and evaluated

```python
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']
```

Code Sample 6- List of Algo used

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).



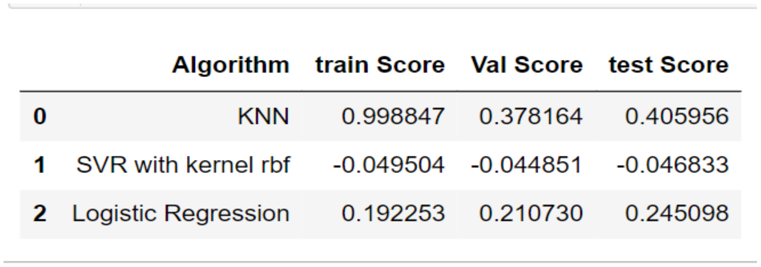
	Algorithm	train Score	Val Score	test Score
0	Simple Linear Reg Model	0.621488	0.625768	0.625564
1	SLinear-Reg (Lasso)	0.621413	0.627354	0.626842
2	SLinear-Reg (Ridge)	0.621065	0.627816	0.627308

Table 5 i- Algorithm score

From the score column, we can see that the train, valid 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.

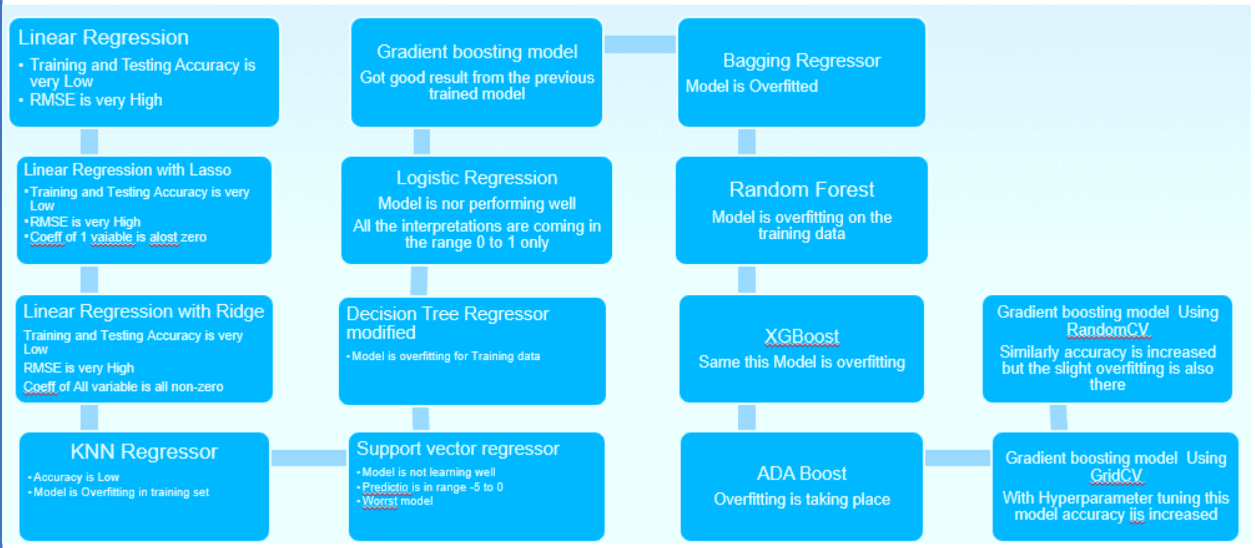
So **KNN** is giving a very good score of 0.99 on the train set, but not that good on the valid and test set, looks like it is overfitting on the training data. If we look at the **Support Vector Regressor** the performance is worse on the train set as well as the test set, same is the case for **Logistic Regression**.



	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

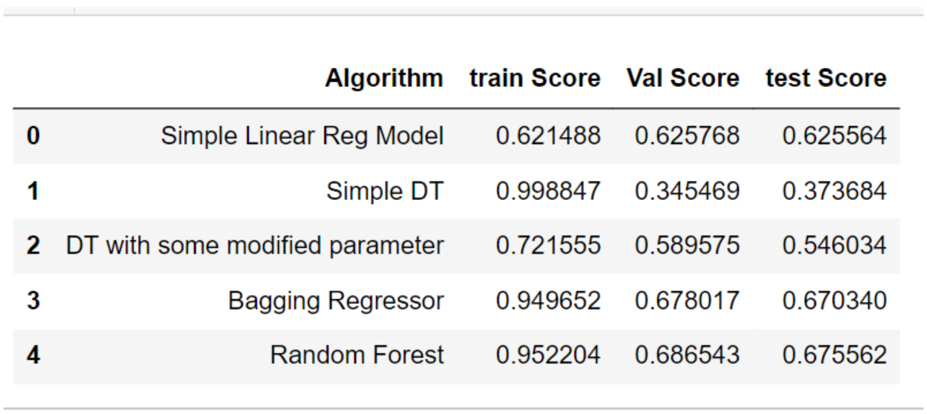
Table 5 ii- Algorithm score

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



Linear Regression
• Training and Testing Accuracy is very Low
• RMSE is very High
Linear Regression with Lasso
•Training and Testing Accuracy is very Low
•RMSE is very High
•Coeff of 1 vaiable is alost zero
Linear Regression with Ridge
Training and Testing Accuracy is very Low
RMSE is very High
Coeff of All variable is all non-zero
KNN Regressor
•Accuracy is Low
•Model is Overfitting in training set
Support vector regressor
•Model is not learning well
•Predictio is in range -5 to 0
•Worrst model
Decision Tree Regressor modified
•Model is overfitting for Training data
Logistic Regression
Model is nor performing well
All the interpretations are coming in the range 0 to 1 only
Gradient boosting model
Got good result from the previous trained model
Bagging Regressor
Model is Overfitted
Random Forest
Model is overfitting on the training data
XGBoost
Same this Model is overfitting
ADA Boost
Overfitting is taking place
Gradient boosting model Using GridCV
With Hyperparameter tuning this model accuracy iis increased
Gradient boosting model Using RandomCV
Similarly accuracy is increased but the slight overfitting is also there

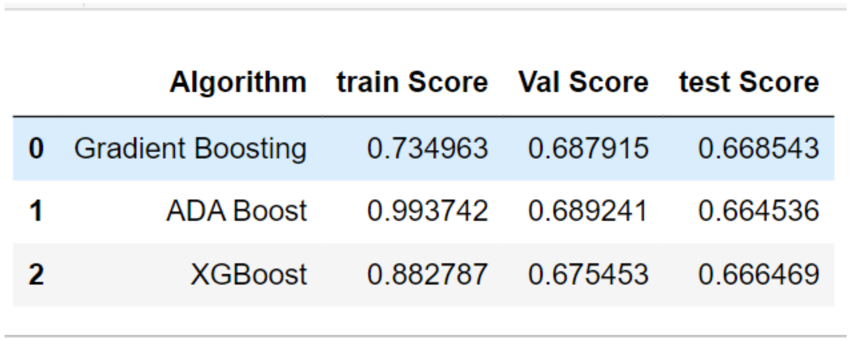
Flow Diagram 1- Flow of the model used and why



	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

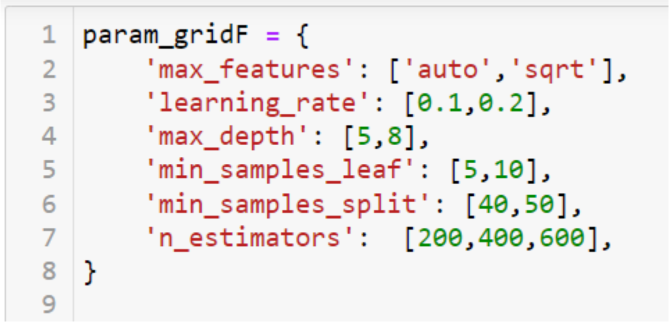
Efforts to improve model performance

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



```
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],
}
```

Code sample 7- Best parameter grid

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

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

Code sample 8- Hyperparameter Tuning

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.

	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

Table 5v-Algorithm score

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:

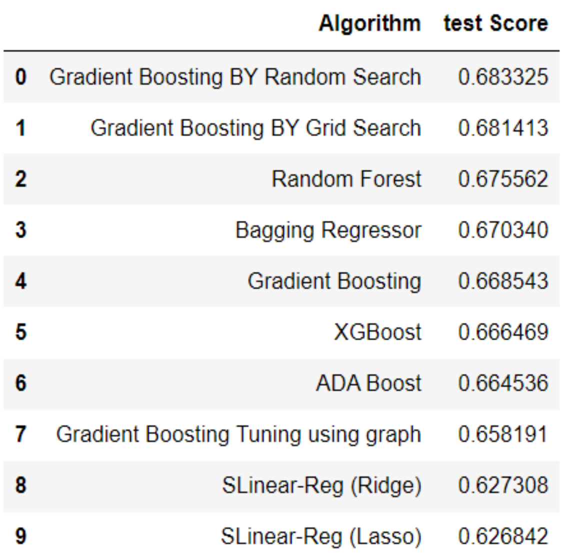
- **RMSE** (Root Mean Squared Error)
- **MSE** (Mean Squared Error)
- **MAE** (Mean Absolute Error)
- **MAPE** (Mean Absolute Percentage Error)
- **Adjusted R^2**

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.

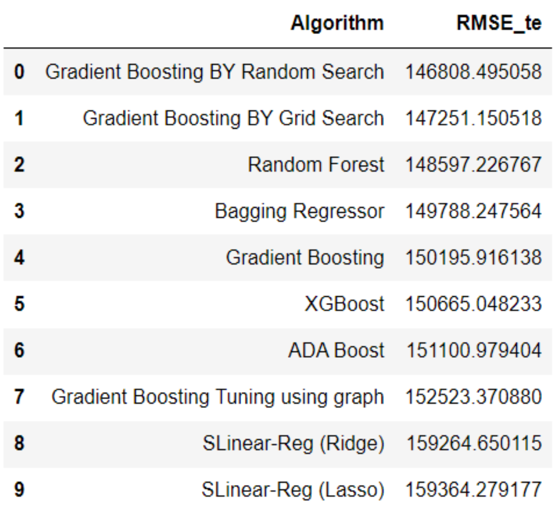
MAE is on average how far those predicted values are from actual values, whereas MSE is the average of squared differences between actual and predicted values.

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

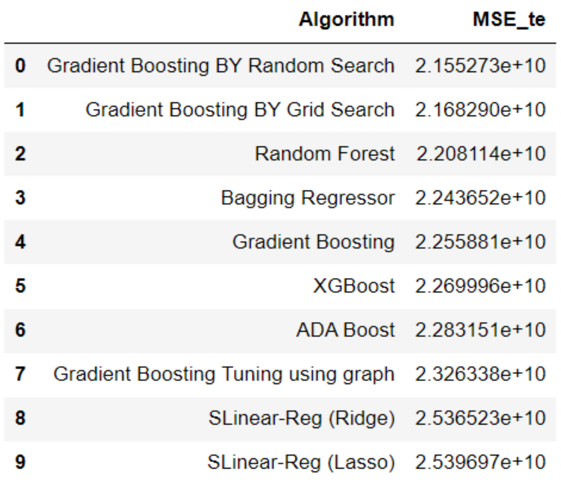
Table 5 vi- Algorithm Performance



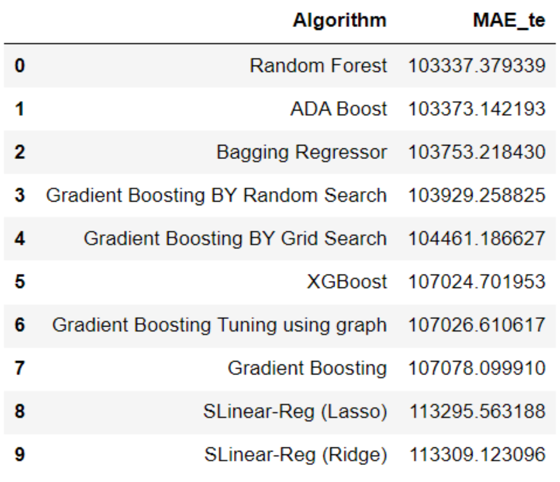
	Algorithm	test Score
0	Gradient Boosting BY Random Search	0.683325
1	Gradient Boosting BY Grid Search	0.681413
2	Random Forest	0.675562
3	Bagging Regressor	0.670340
4	Gradient Boosting	0.668543
5	XGBoost	0.666469
6	ADA Boost	0.664536
7	Gradient Boosting Tuning using graph	0.658191
8	SLinear-Reg (Ridge)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



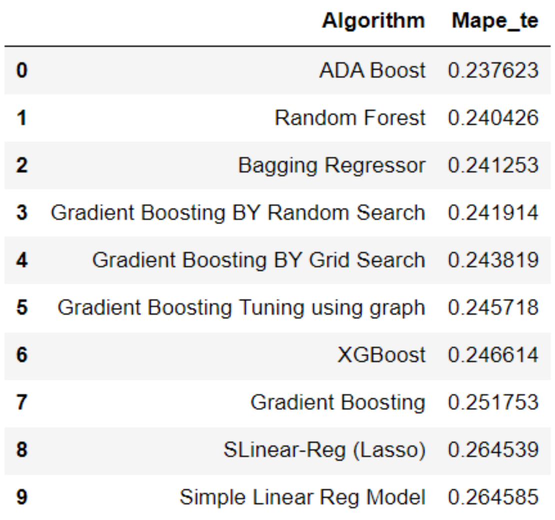
	Algorithm	RMSE_te
0	Gradient Boosting BY Random Search	146808.495058
1	Gradient Boosting BY Grid Search	147251.150518
2	Random Forest	148597.226767
3	Bagging Regressor	149788.247564
4	Gradient Boosting	150195.916138
5	XGBoost	150665.048233
6	ADA Boost	151100.979404
7	Gradient Boosting Tuning using graph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



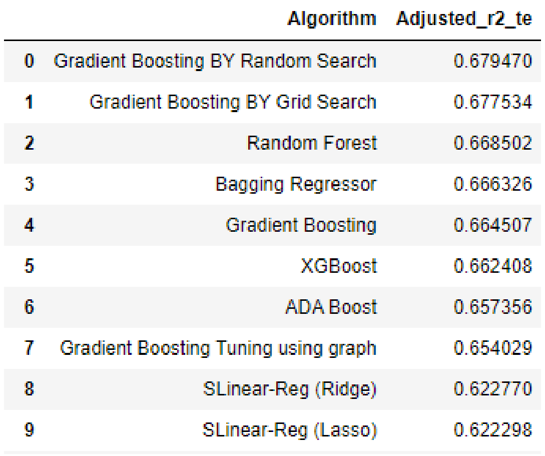
	Algorithm	MSE_te
0	Gradient Boosting BY Random Search	2.155273e+10
1	Gradient Boosting BY Grid Search	2.168290e+10
2	Random Forest	2.208114e+10
3	Bagging Regressor	2.243652e+10
4	Gradient Boosting	2.255881e+10
5	XGBoost	2.269996e+10
6	ADA Boost	2.283151e+10
7	Gradient Boosting Tuning using graph	2.326338e+10
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



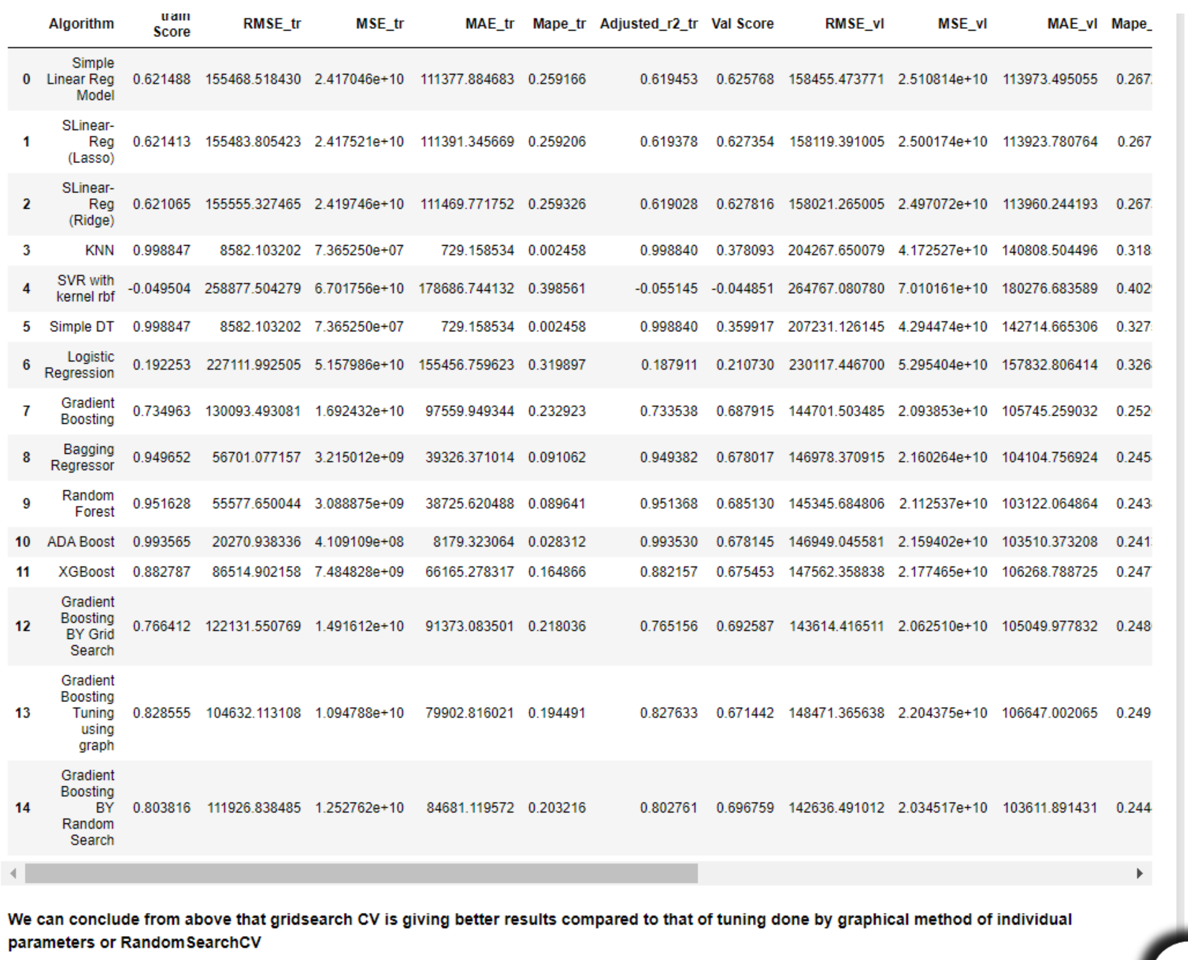
	Algorithm	Adjusted_r2_te
0	Gradient Boosting BY Random Search	0.679470
1	Gradient Boosting BY Grid Search	0.677534
2	Random Forest	0.668502
3	Bagging Regressor	0.666326
4	Gradient Boosting	0.664507
5	XGBoost	0.662408
6	ADA Boost	0.657356
7	Gradient Boosting Tuning using graph	0.654029
8	SLinear-Reg (Ridge)	0.622770
9	SLinear-Reg (Lasso)	0.622298

6. Final Interpretation/ Recommendation

- 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.

Table 6- Final Algorithm Performance Table



	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	0.619453	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	0.619378	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	0.619028	0.627816	158021.265005	2.497072e+10	113960.244193	0.267
3	KNN	0.998847	8582.103202	7.365250e+07	729.158534	0.002458	0.998840	0.378093	204267.650079	4.172527e+10	140808.504496	0.318
4	SVR with kernel rbf	-0.049504	258877.504279	6.701756e+10	178686.744132	0.398561	-0.055145	-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	0.998840	0.359917	207231.126145	4.294474e+10	142714.665306	0.327
6	Logistic Regression	0.192253	227111.992505	5.157986e+10	155456.759623	0.319897	0.187911	0.210730	230117.446700	5.295404e+10	157832.806414	0.326
7	Gradient Boosting	0.734963	130093.493081	1.692432e+10	97559.949344	0.232923	0.733538	0.687915	144701.503485	2.093853e+10	105745.259032	0.252
8	Bagging Regressor	0.949652	56701.077157	3.215012e+09	39326.371014	0.091062	0.949382	0.678017	146978.370915	2.160264e+10	104104.756924	0.245
9	Random Forest	0.951628	55577.650044	3.088875e+09	38725.620488	0.089641	0.951368	0.685130	145345.684806	2.112537e+10	103122.064864	0.243
10	ADA Boost	0.993565	20270.938336	4.109109e+08	8179.323064	0.028312	0.993530	0.678145	146949.045581	2.159402e+10	103510.373208	0.241
11	XGBoost	0.882787	86514.902158	7.484828e+09	66165.278317	0.164866	0.882157	0.675453	147562.358838	2.177465e+10	106268.788725	0.247
12	Gradient Boosting BY Grid Search	0.766412	122131.550769	1.491612e+10	91373.083501	0.218036	0.765156	0.692587	143614.416511	2.062510e+10	105049.977832	0.248
13	Gradient Boosting Tuning using graph	0.828555	104632.113108	1.094788e+10	79902.816021	0.194491	0.827633	0.671442	148471.365638	2.204375e+10	106647.002065	0.249
14	Gradient Boosting BY Random Search	0.803816	111926.838485	1.252762e+10	84681.119572	0.203216	0.802761	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 parameters or RandomSearchCV

- 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.