GREAT LEARNING BANGALORE

HOUSE VALUE PREDICTOR

Part of Machine Learning Capstone Project

Presented By:

Nishita Baskaran

Sanjay Kumar N

Kritika Hiremath

Sumendra Mohanty

Mentored By:

Sanjoy Ghosh

TABLE OF CONTENTS:

1. ABSTRACT:

Machine learning participates in a significant role in every single area of technology as per today's scenario. It has been employed for many sectors since past decades like image processing, pattern recognition, medical diagnosis, and predictive analysis, product recommendation. House prices change every year, so it is mandatory for a structure to foresee house prices in the future. House price prediction can help in fixing and thereby predicting house prices and customers can evaluate it. Our intention is to predict house prices using several machine learning techniques. House price of location does depend on various factors like lot size, bedrooms, bathrooms, location, drawing room, material used in house, interiors, parking area and mainly on square feet per area. Currently to find house price, a concerned person must manually try to find similar properties in the neighborhood, and according to that estimate and house price which requires a lot of time. And again, in the same neighborhood some other person must sell or buy a house, again he must manually do the whole steps. That person also needs to find the appreciation/depreciation of property rates in that neighborhood, in addition he/she also needs to consider inflation rate, real- estate agent charges & associated closing costs from tax & bank. We employ different machine learning techniques for predicting the price based on past data. While testing the data, we used different algorithms like Multiple linear, Decision tree, Random forest, KNN and gradient boosting regression techniques, out of these Random Forest and gradient boosting regression techniques are performing better.

2. 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 do not know the price which you can take—it cannot be too low or too high. To find house prices you usually try to find similar properties in your neighborhood and based on gathered data you will try to assess your house price.

2.1 AIM:

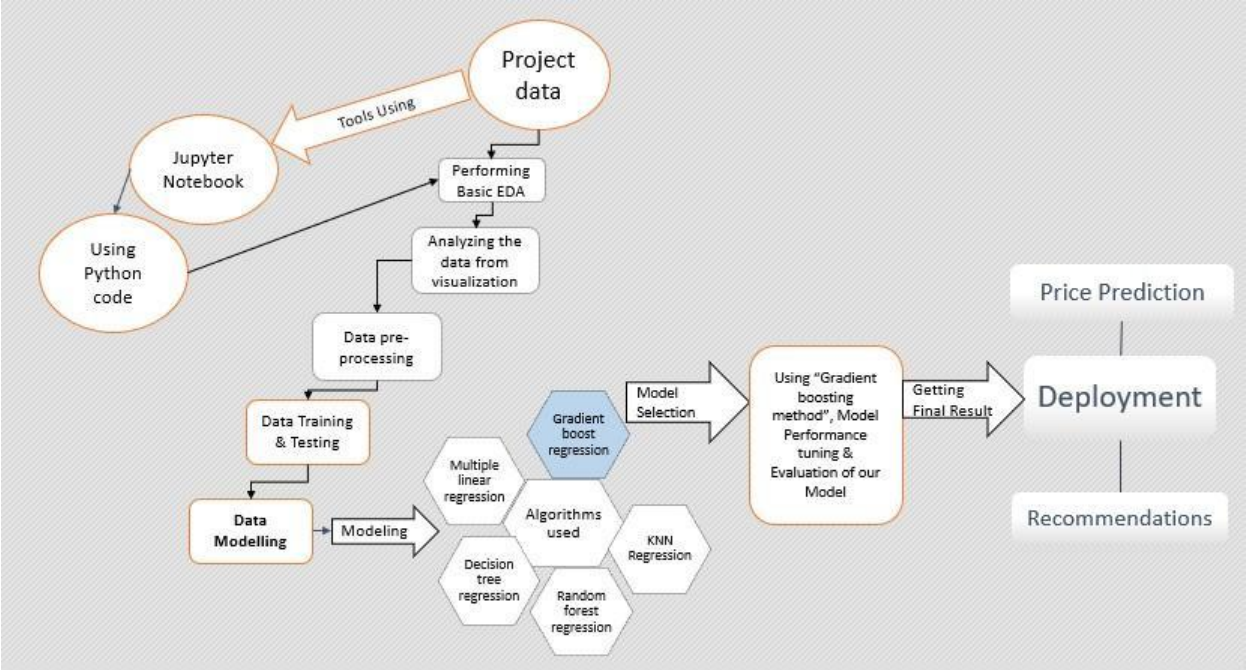
To build a machine learning model to predict the house of price based on the past data, without going through different applications. Below are the parameters on which we will evaluate our model.

- Create an effective price prediction model.
- Validate the model's prediction accuracy.
- Identify the important home price attributes which feed the model's predictive power.

3. PROCESS FLOW OF THE PROJECT:

Process flows define which phases are implemented during a machine learning project. The typical phases include data collection, data pre-processing, building datasets, model training and refinement, evaluation, and deployment to production.

Below is process flow chart of the project:



4. DATA DESCRIPTION:

The raw data used for training the machine learning models comprises several attributes. House value prediction data set has 21613 rows and 23 columns which has following details:

1. **cid**: Identification number for a house.
2. **Day hours**: Date house was sold.
3. **price**: Price of the house.
4. **room_bed**: Number of Bedrooms in the property.
5. **room_bath**: Number of bathrooms in the property.
6. **living_measure**: Square footage of the carpet area.
7. **lot_measure**: Square footage of the super built-up.
8. **ceil**: Total floors (levels) in house.
9. **coast**: House which has a view to a waterfront.
10. **sight**: Has been viewed by how many times.
11. **condition**: Condition of the property.
12. **quality**: Grade given to the housing unit, based on grading system of frames.
13. **ceil_measure**: Square footage of house apart from basement.
14. **basement_measure**: Square footage of the basement.
15. **yr_built**: Built Year of the property.
16. **yr_renovated**: Year when house was renovated.
17. **zip code**: Locality of the property by zip code.
18. **lat**: Latitude coordinate (The angular distance of a house north or south of the earth's equator).
19. **long**: Longitude coordinate (The angular distance of a house east or west of the Greenwich meridian, or west of the standard meridian of a celestial object, usually expressed in degrees)
20. **living_measure15**: Square footage of the carpet area in 2015 (after renovations) This might or might not have affected the lot area.
21. **lot_measure15**: Square footage of the super built-up in 2015 (after renovations).
22. **furnished**: Based on the furnishings of the house, like tiles, marbles, and plain surfaced wall with colored.
23. **total_area**: Total Measure of the property including living and lot area.

5. EDA AND PREPROCESSING:

Before proceeding with the training of machine learning models, we are interested in getting some insights about the data at hand. It would be interesting to know how different variables affect the price of the house, to understand their potential quality as predictors, we do some data analysis using different methods.

5.1 DETAILED DESCRIPTION OF DATA:

- In the dataset provided, we have a different feature which helps us in calculating the difference like latitude and longitude which helps to derive location of the house in turn helps us to estimate the house value.
- From the Year_built in the dataset we can predict how the old house is in turn helps to roughly calculate the current value of the house.
- Value of the house can be adjusted by taking the following things into consideration: number of bedrooms & bathrooms, carpet area, renovation & basement area.
- Before starting to work with the data, we have proceeded to clean some of the data. From the data information output, it can be observed that there are no Nan values. All the variables are of either int dtype or float dtype. The day hours are the only object datatype.

```
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   cid               21613 non-null  int64
 1   dayhours          21613 non-null  object
 2   price             21613 non-null  int64
 3   room_bed          21613 non-null  int64
 4   room_bath         21613 non-null  float64
 5   living_measure    21613 non-null  int64
 6   lot_measure       21613 non-null  int64
 7   ceil              21613 non-null  float64
 8   coast             21613 non-null  int64
 9   sight             21613 non-null  int64
 10  condition         21613 non-null  int64
 11  quality           21613 non-null  int64
 12  ceil_measure      21613 non-null  int64
 13  basement          21613 non-null  int64
 14  yr_built          21613 non-null  int64
 15  yr_renovated      21613 non-null  int64
 16  zipcode           21613 non-null  int64
 17  lat               21613 non-null  float64
 18  long              21613 non-null  float64
 19  living_measure15  21613 non-null  int64
 20  lot_measure15     21613 non-null  int64
 21  furnished         21613 non-null  int64
 22  total_area        21613 non-null  int64
dtypes: float64(4), int64(18), object(1)
memory usage: 3.8+ MB
```

Fig. 5.1.1

Inference from 5.1.1: All attributes except Day hours are int or float.

5.2 STATISTICAL ANALYSIS OF THE DATA :

Below image shows the statistical analysis of the data.

	cid	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition	...	21
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	...	
mean	4.580302e+09	5.401822e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	...	
std	2.876566e+09	3.673622e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	...	
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	...	
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	...	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	...	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	...	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	...	4

8 rows × 22 columns

Fig. 5.2.1

Inference from 5.2.1:

- Bedrooms: min is 0, max is 33.
- Bathrooms: min is 0, max is 8.
- Living area: min is 290, max is 13540.
- Floors: min is 1, max is 3.5.
- Coast: min is 0, max is 1.
- Sight: min is 0, max is 4.
- Condition: min is 1, max is 5.

5.3 IDENTIFYING THE MISSING VALUES:

Below image shows missing values present in the data.

```
cid                0
dayhours           0
price              0
room_bed           0
room_bath          0
living_measure     0
lot_measure        0
ceil               0
coast              0
sight              0
condition          0
quality            0
ceil_measure       0
basement           0
yr_built           0
yr_renovated       0
zipcode            0
lat                0
long               0
living_measure15   0
lot_measure15      0
furnished          0
total_area         0
dtype: int64
```

Fig. 5.3.1

Inference from 5.3.1: There are no missing values in the data.

5.4 DATA VISUALIZATION:

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. Data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions. We have used bivariate analysis to do the data visualization, with the target attribute price being the constant one and comparing with the other attributes of dependent variables.

Below is the image showing the distribution of quality:

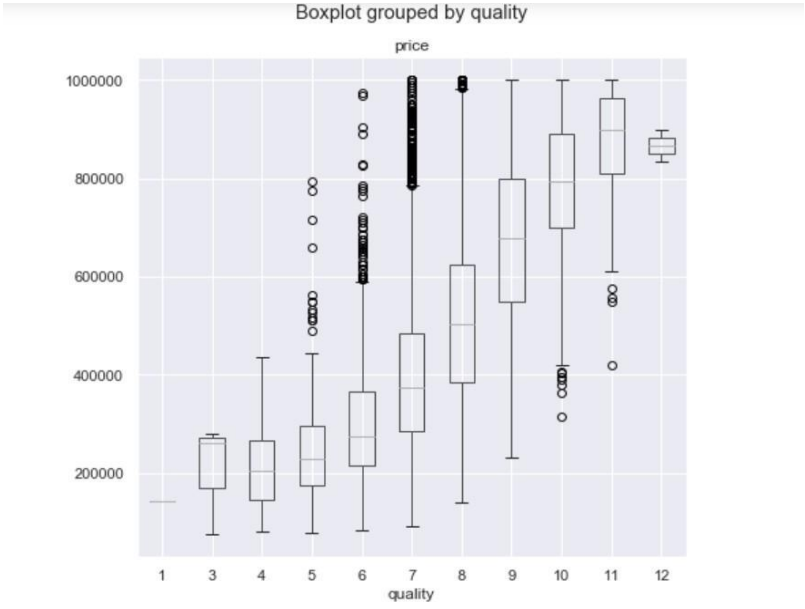


Fig. 5.4.1

Inference from 5.4.1: Maximum number of houses have quality 7 as grade which means excellent quality frame houses.

Below is the image shows price and ceil distribution:

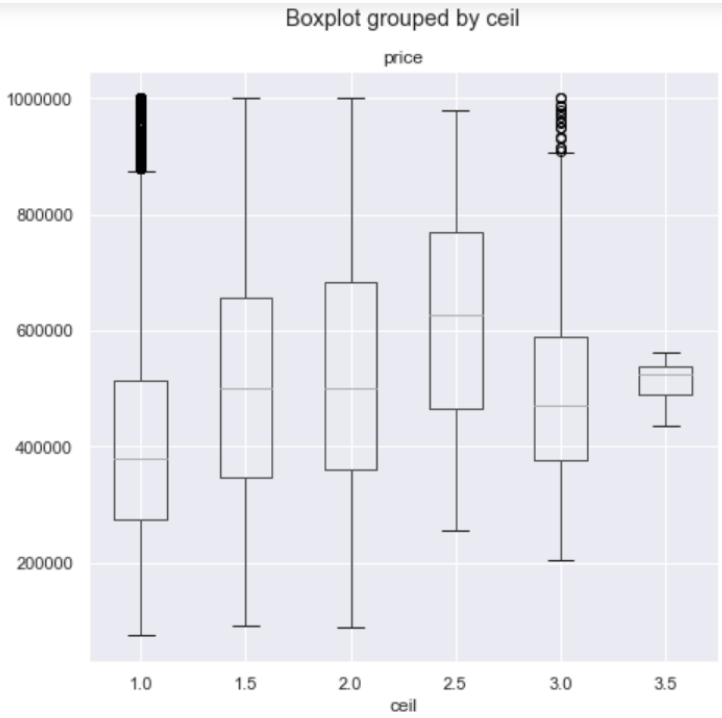


Fig. 5.4.2

Inference from Fig. 5.4.1: Most of the houses have ceil ranging between 2 to 2.

Below is the image that shows price depending on the room bed attribute:

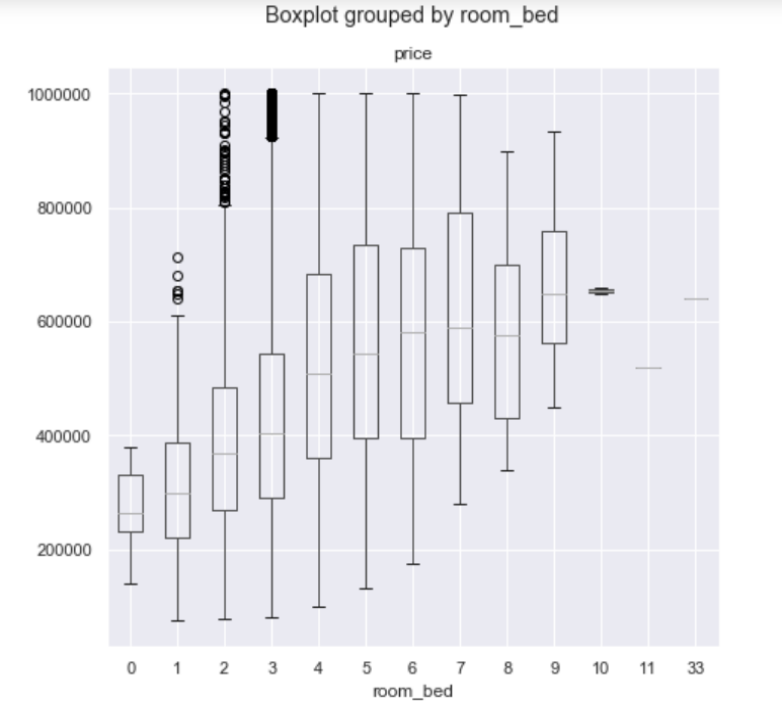


Fig. 5.4.3

Inference from 5.4.3: Average number of bedrooms range from 2.5 to 3.

Below is the image that shows price of the house depending on number of bathrooms:

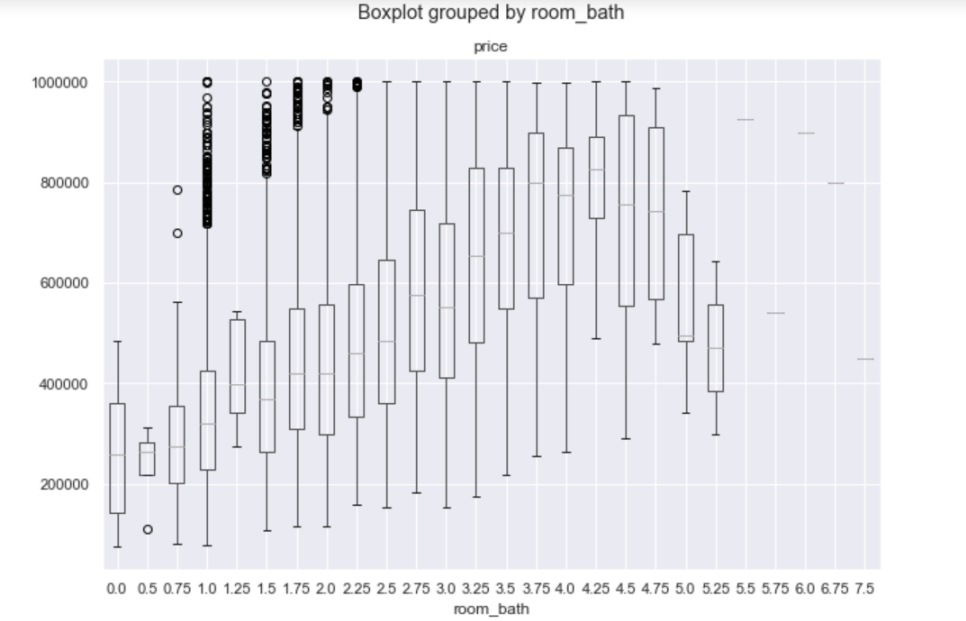


Fig.5.4.4

Inference from 5.4.4: Number of bathrooms ranges between 2.5 to 4.

Below is the image which shows the dataset contains houses built from year 1900 to 2015:

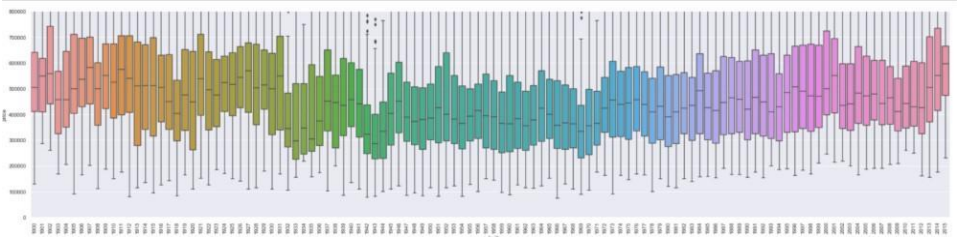


Fig.5.4.5

Inference from 5.4.5: Generally, the year of build of houses should have a linear relationship with house price. But data is not showing that.

Below is the image which shows data distribution for all the variables in data set:

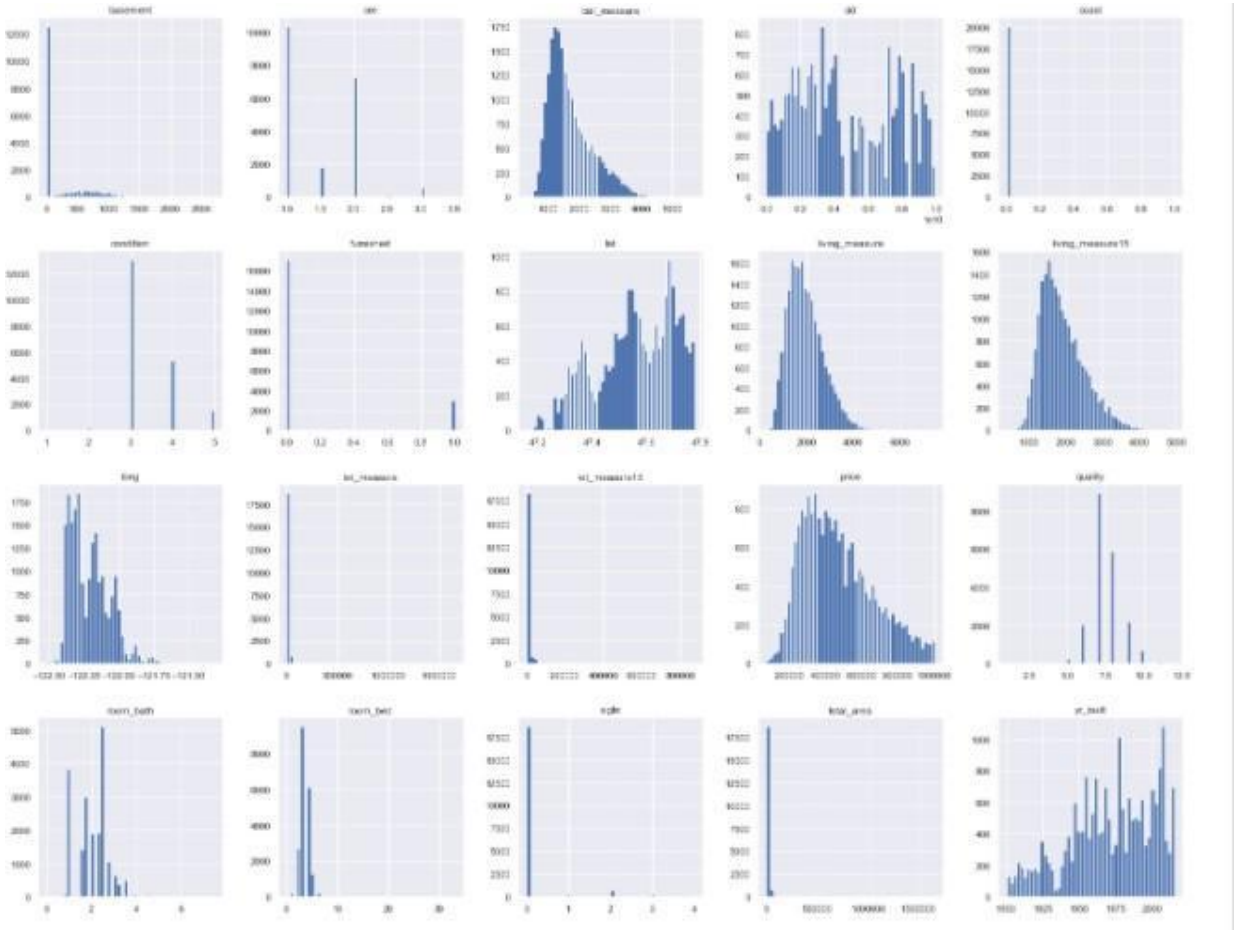


Fig. 5.4.6

Inference from Fig. 5.4.6:

- Most of the houses have no basement.
- Most of the houses have only 1 floor as mentioned in ceil.
- Ceil measure ranges between 1000-2000 for maximum number of houses.
- No house has a waterfront view.
- Condition mostly has the range 3.
- A smaller number of houses were furnished.
- Quality of the houses range at 7 the most.
- The histograms of Price and living_measure is right-skewed.

Below image shows Correlation between all the variables:

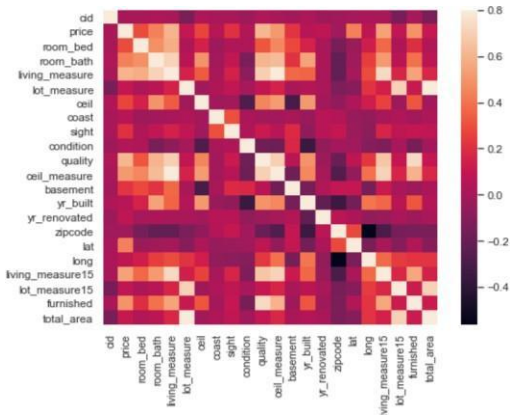


Fig. 5.4.7

Below is the image showing correlation price matrix:

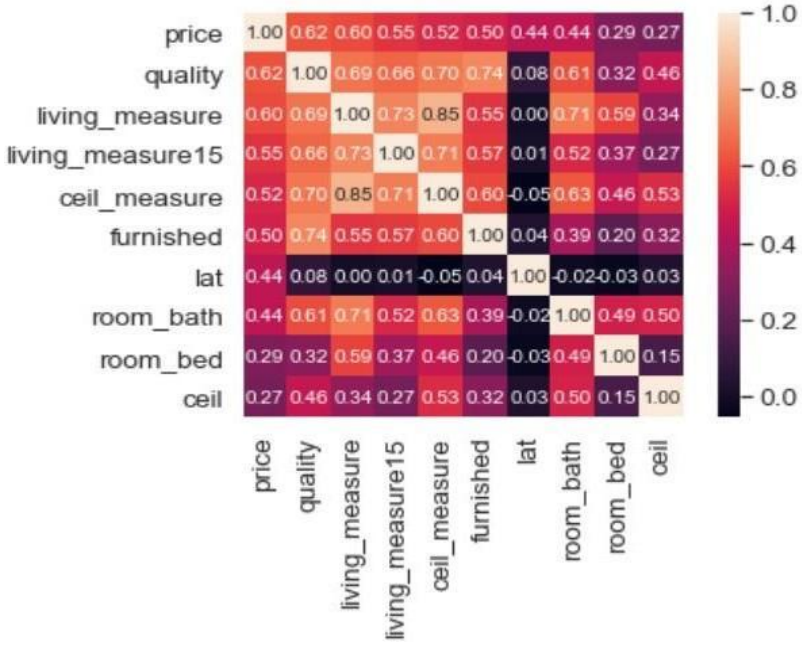


Fig. 5.4.8

Inferences from Fig. 5.4.8:

The features quality, living_measure, living_measure15, ceil_measure displayed the highest correlation with the price of the house. Moreover, there is a high correlation of living_measure with e.g., number of bathrooms and grade. This is common sense, as the square feet increase, so does the number of floors and bathrooms.

Below is the image that shows high correlation between the attributes, we will keep them:

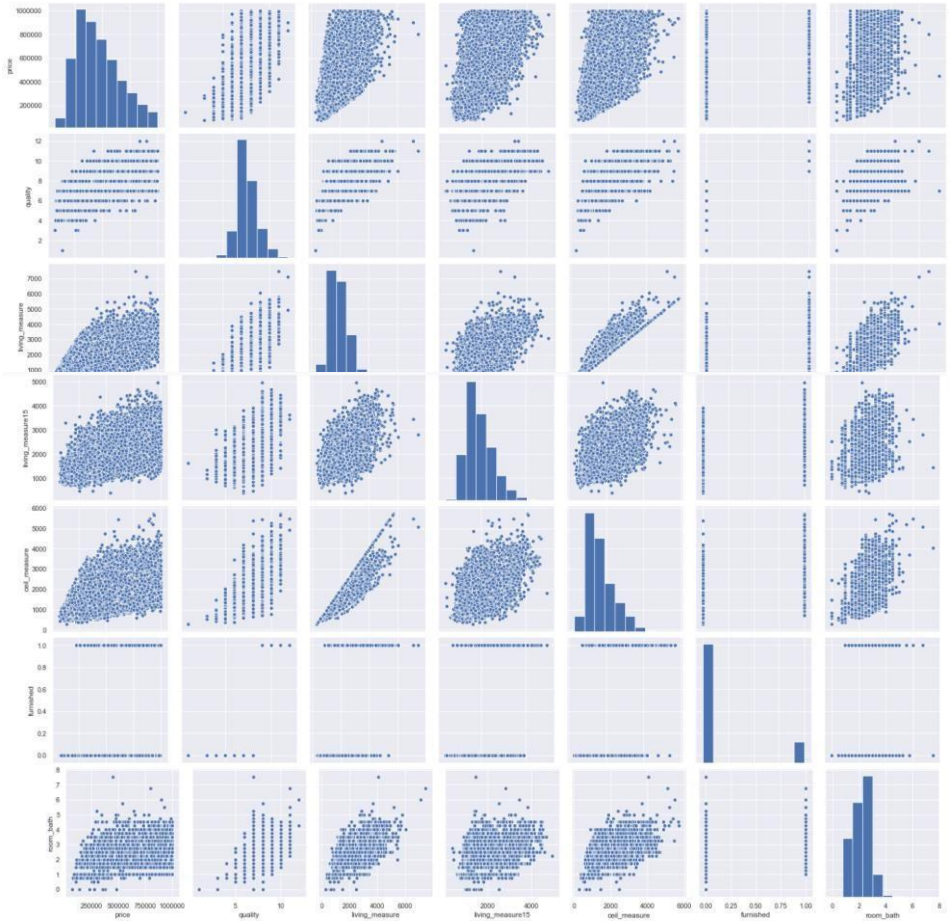


Fig.5.4.9

Below is the image that shows price with living_measure and ceil_measure attributes:

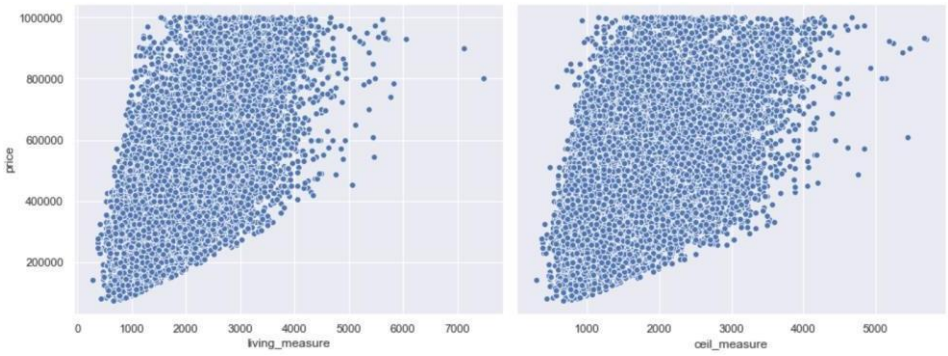


Fig.5.4.10

Inference from Fig. 5.4.10: Living measure area and ceil measure area impacts the price in higher level.

Below is the image that shows scatter plot with price and living_measure attribute:

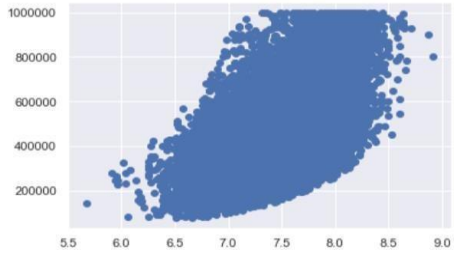
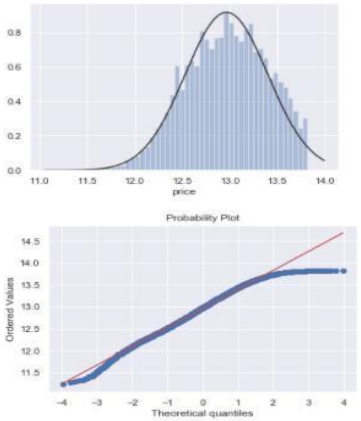


Fig.5.4.11

Inference from Fig. 5.4.11: The two points on the bottom right can be outliers. We can choose to drop the rows (or instances) associated with these two points.

Below is the image that shows log fitted to a normal distribution curve applying log transformation:



5.5 DATA SPLIT:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	basement	yr_built	yr_renov
0	3034200666	20141107T000000	13.602441	4	3.25	8.013012	13457	1.0	0	0	...	0	1956	
1	8731981640	20141204T000000	12.533576	4	2.50	7.843849	7500	1.0	0	0	...	800	1976	
2	5104530220	20150420T000000	12.909170	3	2.50	7.770645	4324	2.0	0	0	...	0	2006	
3	6145600285	20140529T000000	12.611538	2	1.00	6.709304	3844	1.0	0	0	...	0	1916	
4	8924100111	20150424T000000	13.457406	2	1.50	7.244228	4050	1.0	0	0	...	0	1954	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	5137800030	20140701T000000	12.611538	4	2.50	7.741968	3826	2.0	0	0	...	0	2006	
21609	8562890910	20140619T000000	12.676076	4	2.50	8.157657	5000	2.0	0	0	...	0	2003	
21610	1442880160	20140627T000000	13.088709	4	2.75	7.933797	5527	2.0	0	0	...	0	2014	
21611	622100130	20140917T000000	12.807653	2	2.00	7.272398	15000	1.0	0	0	...	0	1985	
21612	6413600276	20150324T000000	12.779732	3	1.00	6.877296	5922	1.5	0	0	...	0	1949	

20153 rows × 23 columns

```
X = Data.drop('price',axis =1).values
y = Data['price'].values
#splitting Train and Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
Data
```

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	basement	yr_built	yr_renov
0	3034200666	20141107T000000	13.602441	4	3.25	8.013012	13457	1.0	0	0	...	0	1956	
1	8731981640	20141204T000000	12.533576	4	2.50	7.843849	7500	1.0	0	0	...	800	1976	
2	5104530220	20150420T000000	12.909170	3	2.50	7.770645	4324	2.0	0	0	...	0	2006	
3	6145600285	20140529T000000	12.611538	2	1.00	6.709304	3844	1.0	0	0	...	0	1916	
4	8924100111	20150424T000000	13.457406	2	1.50	7.244228	4050	1.0	0	0	...	0	1954	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	5137800030	20140701T000000	12.611538	4	2.50	7.741968	3826	2.0	0	0	...	0	2006	
21609	8562890910	20140619T000000	12.676076	4	2.50	8.157657	5000	2.0	0	0	...	0	2003	
21610	1442880160	20140627T000000	13.088709	4	2.75	7.933797	5527	2.0	0	0	...	0	2014	
21611	622100130	20140917T000000	12.807653	2	2.00	7.272398	15000	1.0	0	0	...	0	1985	
21612	6413600276	20150324T000000	12.779732	3	1.00	6.877296	5922	1.5	0	0	...	0	1949	

20153 rows × 23 columns

Inference from Fig. 5.5.1: We have univariate, bi-variate analysis, heat map analysis Cid and dayhours columns were removed during model creation as from the EDA we checked it was not having relation with the target variable.

As a part of data split for the machine learning algorithms as train data and test data, we have split the entire data here as 80% of the records as train data for model training and rest 20% of the records as test data for performance evaluation.

6. MODEL BUILDING:

For the regression problem in the given data set, we have implemented 5 different algorithms-

- **Multiple linear regression**- It is a statistical technique. It can use several variables to predict the outcome of a different variable. The goal of **multiple** regression is to model the linear relationship between your independent variables and your dependent variable.

- **KNN regression-** It 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 neighborhood.

- **Decision tree regression**- It is one of the predictive modelling approaches used in machine learning. These are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions and are a non-parametric supervised learning method used for both classification and regression tasks.

- **Random forest regression-** Random forest regression is a supervised learning algorithm that uses ensemble learning methods for regression. It operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.

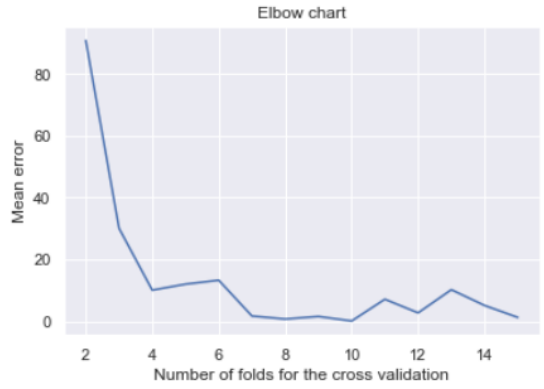
- **Gradient boost regression-** A Gradient Boosting regression combines the predictions from multiple decision trees to generate the final predictions. The main idea is to set the target outcomes for this next model in order to minimize the error.

6.1 MULTIPLE LINEAR REGRESSION:

As a starting process of modelling, we decided to implement a naive algorithm in the data i.e., multiple linear regression because it is always recommended to test the data with basic algorithms for checking the worst-case accuracy and adaptability of the data to the higher dimensions before moving into the advanced algorithms.

Train ratio used was 80:20 since it was a good practice to give more data for training purposes as it is a basic algorithm.
We have used the number of folds in the cross validations as 4 that we inferred from the elbow method. The following image shows the implementation of the elbow method for the determination of the number of folds in the cross validation.

```
Cross validation with 2 folds : 90.87114271286742
Cross validation with 3 folds : 30.086564460555604
Cross validation with 4 folds : 10.039149370613586
Cross validation with 5 folds : 11.991900127764165
Cross validation with 6 folds : 13.260615566242242
Cross validation with 7 folds : 1.6718067046071414
Cross validation with 8 folds : 0.7053461548245252
Cross validation with 9 folds : 1.5333668272438483
Cross validation with 10 folds : 0.06668560134328913
Cross validation with 11 folds : 7.104131424234589
Cross validation with 12 folds : 2.694542143437841
Cross validation with 13 folds : 10.194423224265751
Cross validation with 14 folds : 5.091733837039272
Cross validation with 15 folds : 1.2263477996592163
```



According to the above chart, we can assume a value of 4 for K

The following image shows the conclusions that we got from the multiple linear regression.

	R^2(train)	R^2(test)	Adjusted R^2(train)	Adjusted R^2(test)	4-Fold Cross Validation
Multiple linear regression	0.70702	0.699851	0.706743	0.699568	0.705082

6.1.1 INFERENCE FROM THE MULTIPLE LINEAR REGRESSION:

- Model was giving poor accuracy in the test data with a R2 score of 0.69.
- Hence, the model was not reliable and there is a scope for improvement.

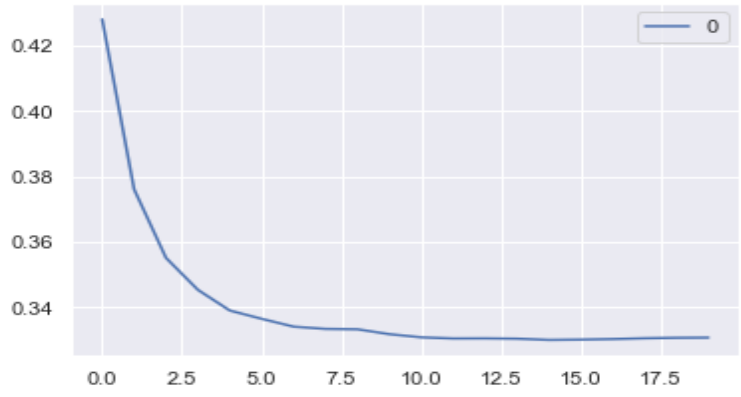
6.2 KNN REGRESSION:

The second model was also a multidimensional algorithm i.e., K Nearest neighbor algorithm. KNN has a history of giving good accuracy in linear regression problems since it takes the mean value of the nearest neighbors. Hence, KNN was selected as the second algorithm.

To find the number of neighbors, we implemented the elbow method, and we got the output as 7.

```
RMSE value for k=  1 is: 0.42813952223202806
RMSE value for k=  2 is: 0.3761513970874899
RMSE value for k=  3 is: 0.35516108401762014
RMSE value for k=  4 is: 0.34538128484576924
RMSE value for k=  5 is: 0.33905528145610425
RMSE value for k=  6 is: 0.3365137692922571
RMSE value for k=  7 is: 0.3341231112356584
RMSE value for k=  8 is: 0.33343799556957976
RMSE value for k=  9 is: 0.3333185246649571
RMSE value for k=  10 is: 0.33182343295840633
RMSE value for k=  11 is: 0.33085893012128154
RMSE value for k=  12 is: 0.3305056621462902
RMSE value for k=  13 is: 0.33053724848468796
RMSE value for k=  14 is: 0.3304423646216414
RMSE value for k=  15 is: 0.3300666577740836
RMSE value for k=  16 is: 0.3301583739510934
RMSE value for k=  17 is: 0.3303244164371235
RMSE value for k=  18 is: 0.3305711645068022
RMSE value for k=  19 is: 0.33071902220641314
RMSE value for k=  20 is: 0.330779266561709
```

]: `<matplotlib.axes._subplots.AxesSubplot at 0x214474192b0>`



According to the above chart, we can assume a value of 7 for K

We have used the number of cross validations as 4 that also we inferred from the elbow method.

The following image shows the conclusions that we got from the KNN regression:

	R^2(train)	R^2(test)	Adjusted R^2(train)	Adjusted R^2(test)	4-Fold Cross Validation
KNN regression	0.562155	0.400118	0.561878	0.399835	0.40261

6.2.1 INFERENCE FROM THE KNN REGRESSION:

- Unfortunately, we got very poor results from the model that had much lower accuracy than linear regression. This model was giving poor accuracy in the test data with a R2 score of 0.4.
- Hence, the model was not reliable and there is a scope for improvement.

6.3 DECISION TREE REGRESSION:

Since, multidimensional algorithms were not giving expected results, we moved to tree based algorithms. Hence, the next algorithm chosen was decision tree regression. The hyper parameters were optimized using a random search tuning process.

We got the following conclusions because of hyper parameter tuning:

Hyperparameter tuning and modelling using pipelining

```
# Instantiate the decision tree regressior
from sklearn.tree import DecisionTreeRegressor


# Define the train and test sets
X_train = train_set.loc[:, train_set.columns != 'price']
y_train = train_set['price']
X_test = test_set.loc[:, train_set.columns != 'price']
y_test = test_set['price']

from sklearn.pipeline import make_pipeline
# Create a pipeline
pipe = make_pipeline((DecisionTreeRegressor()))
# Create dictionary with candidate learning algorithms and their hyperparameters
grid_param = [
                {"decisiontreeregressor": [DecisionTreeRegressor()],
                 "decisiontreeregressor__max_depth":[10,20,20,40,50,60,70,80,90,100],
                 "decisiontreeregressor__min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
                 "decisiontreeregressor__max_leaf_nodes": [1,2,3,4,5,6,7,8,9,10]}]



# create a gridsearch of the pipeline, the fit the best model
rf_random = RandomizedSearchCV(pipe, grid_param, cv=4, verbose=0,n_jobs=-1)
best_model = rf_random.fit(X_train, y_train)
#best_model.score(X_test,y_test)
rf_random.best_params_
```

```
{'decisiontreeregressor__min_samples_leaf': 7,
 'decisiontreeregressor__max_leaf_nodes': 10,
 'decisiontreeregressor__max_depth': 20,
 'decisiontreeregressor': DecisionTreeRegressor(max_depth=20, max_leaf_nodes=10, min_samples_leaf=7)}
```

The following image shows the conclusions that we got from the decision tree regression.

	R^2(train)	R^2(test)	Adjusted R^2(train)	Adjusted R^2(test)	4-Fold Cross Validation
Decision tree regression	0.659566	0.648411	0.65929	0.648128	0.651276

6.3.1 INFERENCE FROM THE DECISION TREE REGRESSION:

There were no considerable improvements in the prediction accuracy.

- Model was giving poor accuracy in the test data with a R2 score of 0.64.
- Hence, the model was not reliable and there is a scope for improvement.

6.4 RANDOM FOREST REGRESSION:

Since, the decision tree is quite an unstable model, we moved to a bagging model i.e., random forest regression.

The hyper parameters were optimized using a random search tuning process. We got the following conclusions because of hyper parameter tuning:

Hyperparameter tuning and modelling using pipelining ¶

```
: from sklearn.pipeline import make_pipeline
  from sklearn.ensemble import RandomForestRegressor
  # Create a pipeline
  pipe = make_pipeline((RandomForestRegressor()))
  # Create dictionary with candidate learning algorithms and their hyperparameters
  grid_param = [
                {"randomforestregressor": [RandomForestRegressor()],
                 "randomforestregressor__n_estimators": [5,10,15,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100],
                 "randomforestregressor__max_depth":[5,8,15,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,105,110,115,120,125,
                 "randomforestregressor__min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
                 "randomforestregressor__max_leaf_nodes": [2, 5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,110,12

  # create a gridsearch of the pipeline, the fit the best model
  rf_random = RandomizedSearchCV(pipe, grid_param, cv=4, verbose=0,n_jobs=-1)
  best_model = rf_random.fit(X_train, y_train)
  #best_model.score(X_test,y_test)
  rf_random.best_params_
```

```
: {'randomforestregressor__n_estimators': 55,
   'randomforestregressor__min_samples_leaf': 7,
   'randomforestregressor__max_leaf_nodes': 290,
   'randomforestregressor__max_depth': 115,
   'randomforestregressor': RandomForestRegressor(max_depth=115, max_leaf_nodes=290, min_samples_leaf=7,
                         n_estimators=55)}
```

The following image shows the conclusions that we got from the random forest regression-

	R^2(train)	R^2(test)	Adjusted R^2(train)	Adjusted R^2(test)	4-Fold Cross Validation
Random forest regression	0.883838	0.832056	0.883562	0.831773	0.83616

6.4.1 INFERENCE FROM THE RANDOM FOREST REGRESSION:

- Fortunately, we got the expected results with a good prediction accuracy.
- This model generated a R2 score of 0.88 for the train data and 0.83 for the test data.
- The model was reliable but for a second opinion, we decided to try a boosting model also.

6.5 GRADIENT BOOSTING REGRESSION:

Gradient boosting will always try to reduce the bias hence it was expected that the R2 score of train data will increase or stay intact.

We used the random search tuning process for finding the perfect hyper parameters. We got the following conclusions because of hyper parameter tuning-

Hyperparameter tuning and modelling using pipelining

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.pipeline import make_pipeline
# Create a pipeline
pipe = make_pipeline((GradientBoostingRegressor()))
# Create dictionary with candidate learning algorithms and their hyperparameters
grid_param = [
                {"gradientboostingregressor": [GradientBoostingRegressor()],
                 "gradientboostingregressor__n_estimators":  [5,10,15,25,30,35,40,45,50],
                 "gradientboostingregressor__max_depth":[5,8,15,25,30,35,40,45,50],
                 "gradientboostingregressor__min_samples_leaf":[3,4,5,6,7,8,9,10],
                 "gradientboostingregressor__max_leaf_nodes": [2, 5,10,15,20,25,30,35,40,45]}]


# create a gridsearch of the pipeline, the fit the best model
rf_random = RandomizedSearchCV(pipe, grid_param, cv=4, verbose=0,n_jobs=-1)
best_model = rf_random.fit(X_train, y_train)
#best_model.score(X_test,y_test)
rf_random.best_params_
```

```
{'gradientboostingregressor__n_estimators': 40,
 'gradientboostingregressor__min_samples_leaf': 8,
 'gradientboostingregressor__max_leaf_nodes': 45,
 'gradientboostingregressor__max_depth': 40,
 'gradientboostingregressor': GradientBoostingRegressor(max_depth=40, max_leaf_nodes=45, min_samples_leaf=8,
                           n_estimators=40)}
```

The following image shows the conclusions that we got from the gradient boosting regression:

	R^2(train)	R^2(test)	Adjusted R^2(train)	Adjusted R^2(test)	4-Fold Cross Validation
Gradient boost regression	0.894128	0.857921	0.893839	0.85762	0.865816

6.5.1 INFERENCE FROM THE GRADIENT BOOSTING REGRESSION:

- As per the expectation, we got a high R2 score for train data intact with a value of 0.88.

- However, the R2 score of test data increased little bit to 0.85.

6.6 CHALLENGES FACED DURING THE MODELLING PROCESS:

- The main challenge was tuning the hyper parameters to reach at an optimum level by neither under fitting nor overfitting the model.
- Implementation of certain techniques like elbow method and cross validation needed some browsing and understanding of the concepts.
- Understanding the theory behind advanced techniques like bagging and boosting was a good learning attempt.

7. COMPARISON OF ALGORITHMS:

Model	R^2(train)	R^2(test)	Adjusted R^2(train)	Adjusted R^2(test)	4-fold cross Validation
Multiple Linear Regression	0.706574	0.703192	0.706297	0.702912	0.705036
K-Nearest Neighbor Regression	0.562155	0.400118	0.561878	0.399835	0.40261
Decision tree Regression	0.659566	0.648411	0.65929	0.648128	0.651276
Random Forest Regression	0.883838	0.832056	0.883562	0.831773	0.83616
Gradient boosting Regression	0.888899	0.855202	0.888622	0.854918	0.853326

- Clearly, multiple linear regression, KNN regression and Decision tree regression are not fit for this data.
- As a part of advanced modelling, we tried the data with a bagging model (Random forest). It performed well in the test data with a R2 score of 0.83.
- For a second opinion, we tried the data with a boosting model (Gradient boosting). It also performed well in the test data with a R2 score of 0.85.
- There is no considerable change in the R2 score for both the models with respect to the train data but there is a small increase in the R2 score of test data in the gradient boosting model. Hence, we can select gradient boosting as the finalized algorithm for this regression problem.

8. DEPLOYMENT:

9. SUMMARY:

9.1 IMPLICATIONS:

9.2 LIMITATIONS:

9.3 FUTURE ENHANCEMENT: