Project Report

“House Price Prediction”

Submitted by:-

Abhisek Srichandan

Darshan Daga

Madhu Varun

- # Introduction

This project has been done as part of my course for the Great Learning we subscribed online. We had 1 month to fulfill the requirements in order to succeed the module. Every week a meeting was arranged amongst out team members to discuss the progress and fix the next objectives.

- # The Project

  - **Current Situation** Predicting the house price in the current economic market has no real/accurate measures. House price is often driven by the brokers taking in liu the unawareness of customers giving them a huge advantage, fooling/cheating the said customer. If only there was a way that could take into consideration all the factors that make the house price and setting this as a standard everywhere around the country/ world would help customers not being cheated on providing them with more transparency. The present method is that the customer approaches a real estate agent to manage his/her investments and suggest suitable estates for his investments. But this method is risky as the agent might predict wrong estates and thus leading to loss of the customer's investments. Generally, the property values rise with respect to time and its appraised value need to be calculated. This appraised value is required during the sale of property or while applying for the loan and for the marketability of the property. These appraised values are determined by the professional appraisers. However, drawback of this practice is that these appraisers could be biased due to bestowed interests from buyers, sellers

or mortgages. Thus, we require an automated prediction model that can help to predict the property values without any bias. This automated model can help the first time, buyers and less experienced customers to understand whether the property rates are overrated or underrated

- **Opportunity for improvement:** The land prices are predicted with a new set of parameters with a different technique. Also we predicted the compensation for the settlement of the property. Mathematical relationships help us to understand many aspects of everyday life. When such relationships are expressed with exact numbers, we gain additional clarity Regression is concerned with specifying the relationship between a single numeric dependent variable and one or more numeric independent variables. House prices increase every year, so there is a need for a system to predict house prices in the future. House price prediction can help the developer determine the selling price of a house and can help the customer to arrange the right time to purchase a house. Machine learning tools and algorithms can become a great tool in solving the issue. We can take advantage of all the available features and use it to analyse and predict house prices. We can benefit both client house buyer and client house seller.

- **Data Requirement:** We need a huge database of previously sold houses along with the prices that they were sold in and the various features related to the house. Source of data and challenges: Data can be acquired from a lot of sources like the websites that rent or sell houses such Magic bricks, makaan.com, 99acres.com, and Kaggle. Some companies even have their Facebook pages mentioning the details of the house and prices. We can also refer to government repositories. However these data might not be clean and organised. Therefore it might need a lot of pre-processing.

- **Size of the data:** Size of the data that we need is at least 1000 data points. We expect a house price on all of these. Since the data can come various formats such as MS Word, PPT, Paper etc. we need sufficient storage space and computation requirements to pre-process the data and bring it to a consistent state before modelling.

- **Features**

The following features and their brief descriptions were provided in the raw data:

1. cid: a notation for a house
2. dayhours: Date house was sold
3. price: Price is prediction target
4. room_bed: Number of Bedrooms/House
5. room_bath: Number of bathrooms/bedrooms
6. living_measure: square footage of the home
7. lot_measure: quare footage of the lot
8. ceil: Total floors (levels) in house
9. coast: House which has a view to a waterfront
10. sight: Has been viewed
11. condition: How good the condition is (Overall)
12. quality: grade given to the housing unit, based on grading system
13. ceil_measure: square footage of house apart from basement
14. basement_measure: square footage of the basement
15. yr_built: Built Year
16. yr_renovated: Year when house was renovated

17. zipcode: zip

18. lat: Latitude coordinate

19. long: Longitude coordinate

20. living_measure15: Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area

21. lot_measure15: lotSize area in 2015(implies-- some renovations)

22. furnished: Based on the quality of room 23: total_area: Measure of both living and lot

We see that there are 22 features. Price is the dependent/target variable that we hope to predict at the end of the project.

Steps in our approach

- ## EDA and pre-processing

1. We start by importing the required packages that we think will be useful currently, like numpy, pandas etc. However as we proceed we add in to the imports as per the requirement.
2. We import the dataset “innercity.csv” and have an overlook on its features and with the type of attributes in mind, we used basic statistical measures on my data such as the mean, the mode, the standard deviation, etc.
3. We infer the following from our observations

    i. There were no missing values.

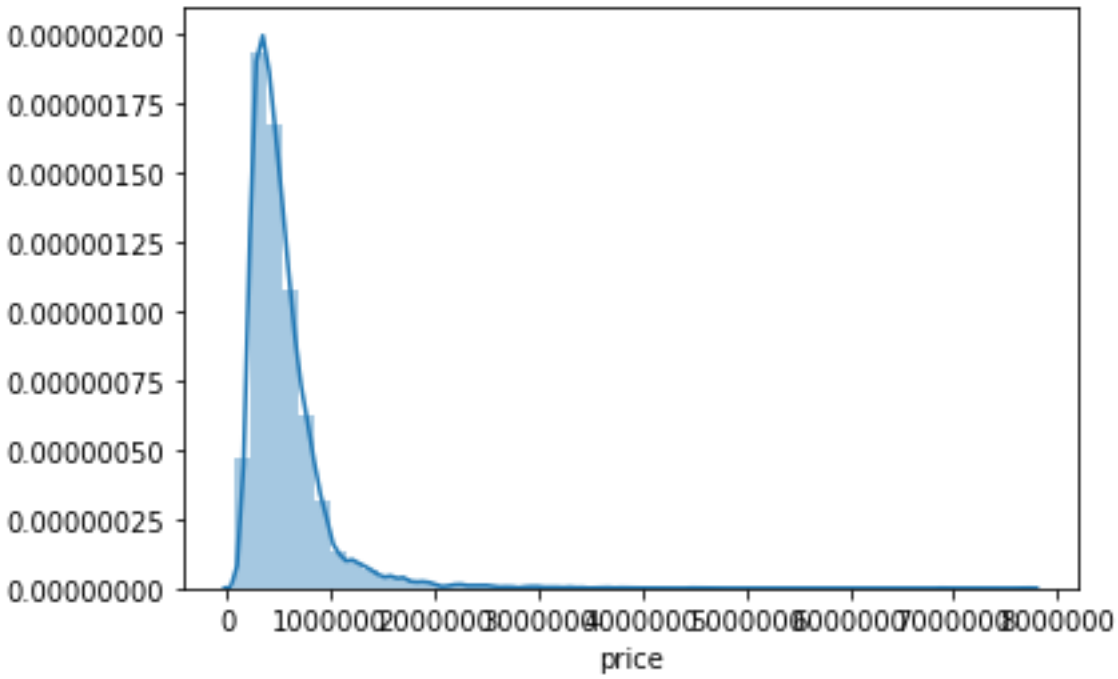
    ii. The Q1, Q2, Q3 (25%, 50%, 100%) are good indicators of the shape of the different attributes.

    iii. There are presence outliers in many attributes.

iv. Attributes cid, yr_built, dayhours provide no relevant insight and will be removed from the dataset in further analysis.

4. Performing EDA on our Dataset we find that our Dataset has 21613 rows and 23 columns, with no missing values. Following are other observations :

- Price is right skewed. Seeing the distribution, we can say that a lot of houses are below the 2000000 range.



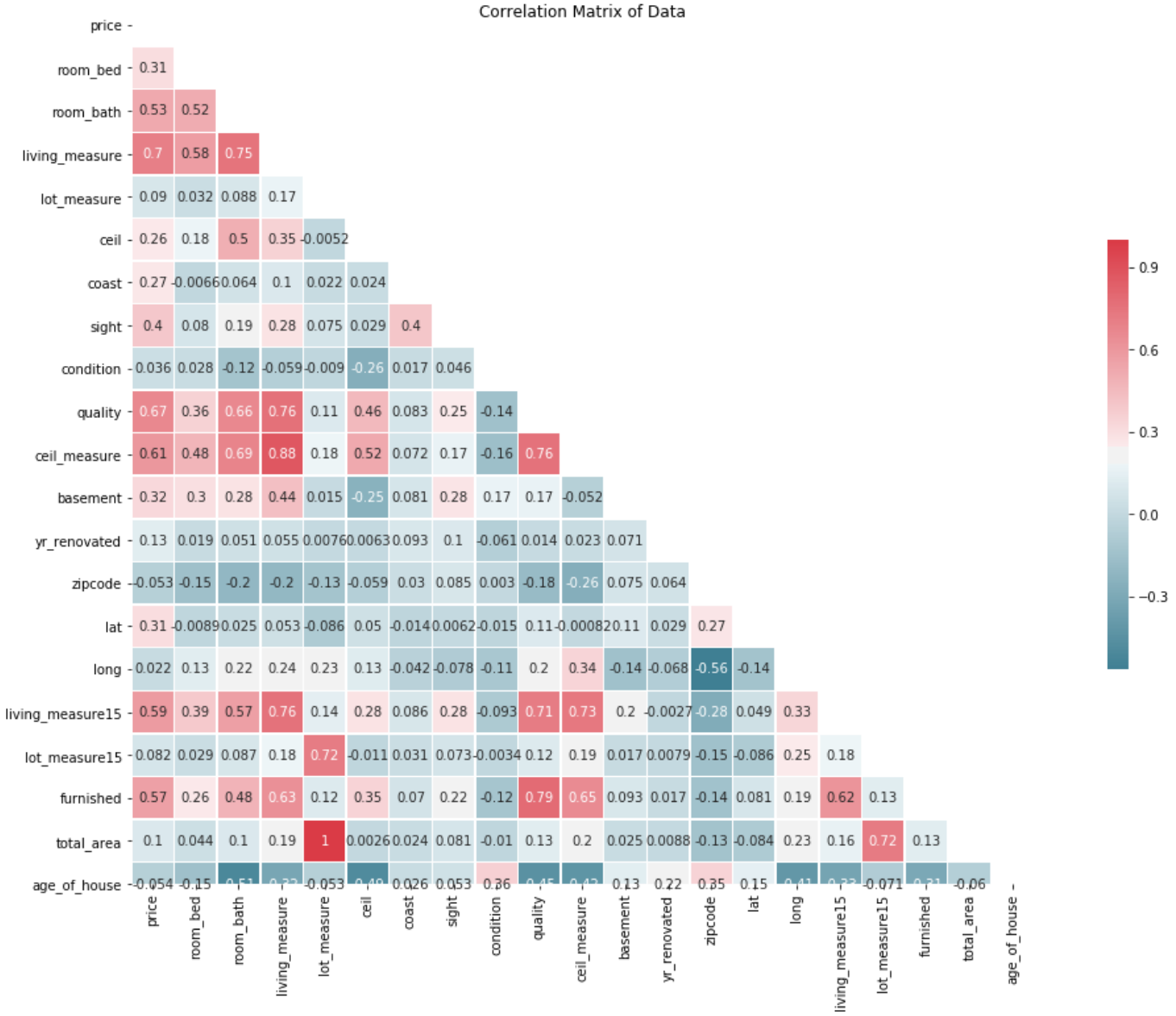
- From dayhours column we extracted the year in which the house was sold and the made this a categorical column with 2 categories(2015 and 2016). Then we used this column and subtracted year_built from it which gave us the age of the house. We found a correlation between year of house and price which states that the older house doesn't go for much price.
- The we created groups inside each of the zipcode, room_bath and room_bed columns as individually all these have a lot of distinct values which if

taken as it is may lead to creation of a huge number of columns while encoding.

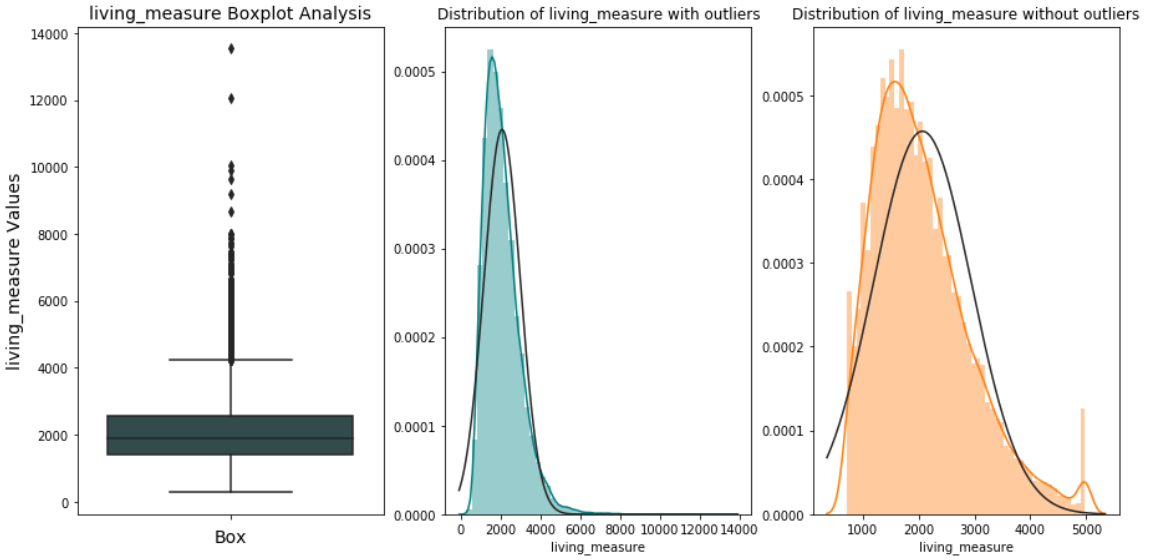
- This ends the preparation of all the columns and starts our outlier detection and outlier treatment pipeline.

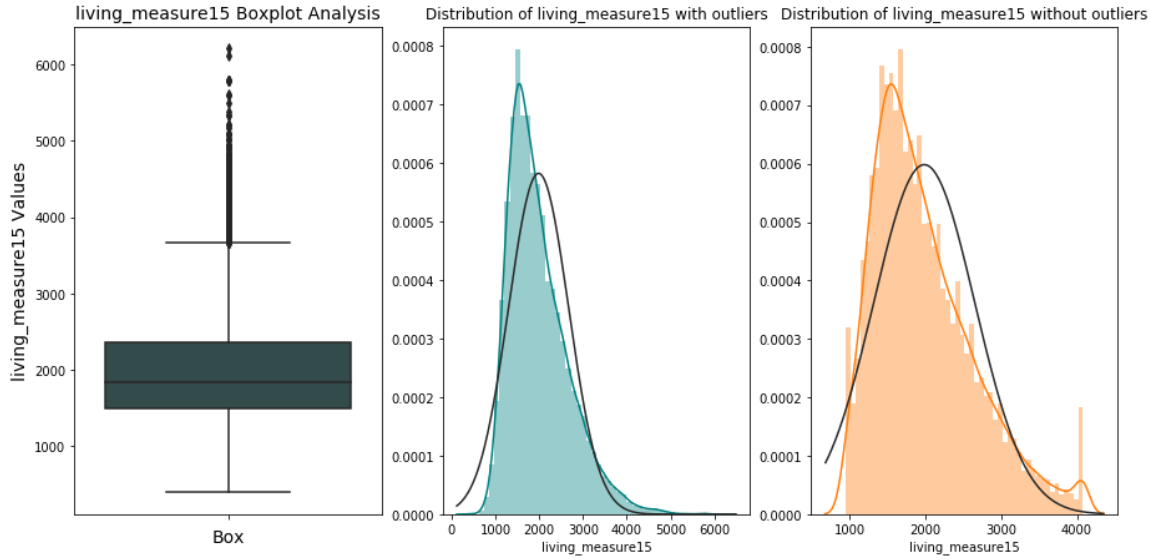
I. Outlier detection : Used the interquartile method to detect the outliers. Anything above q3 + 1.5*IQR range and anything below q1 – 1.5*IQR was considered as an outlier.

II. Outlier removal : detected outliers were replace with the safest mean value. Because replacing with mean did not change the distribution by much.

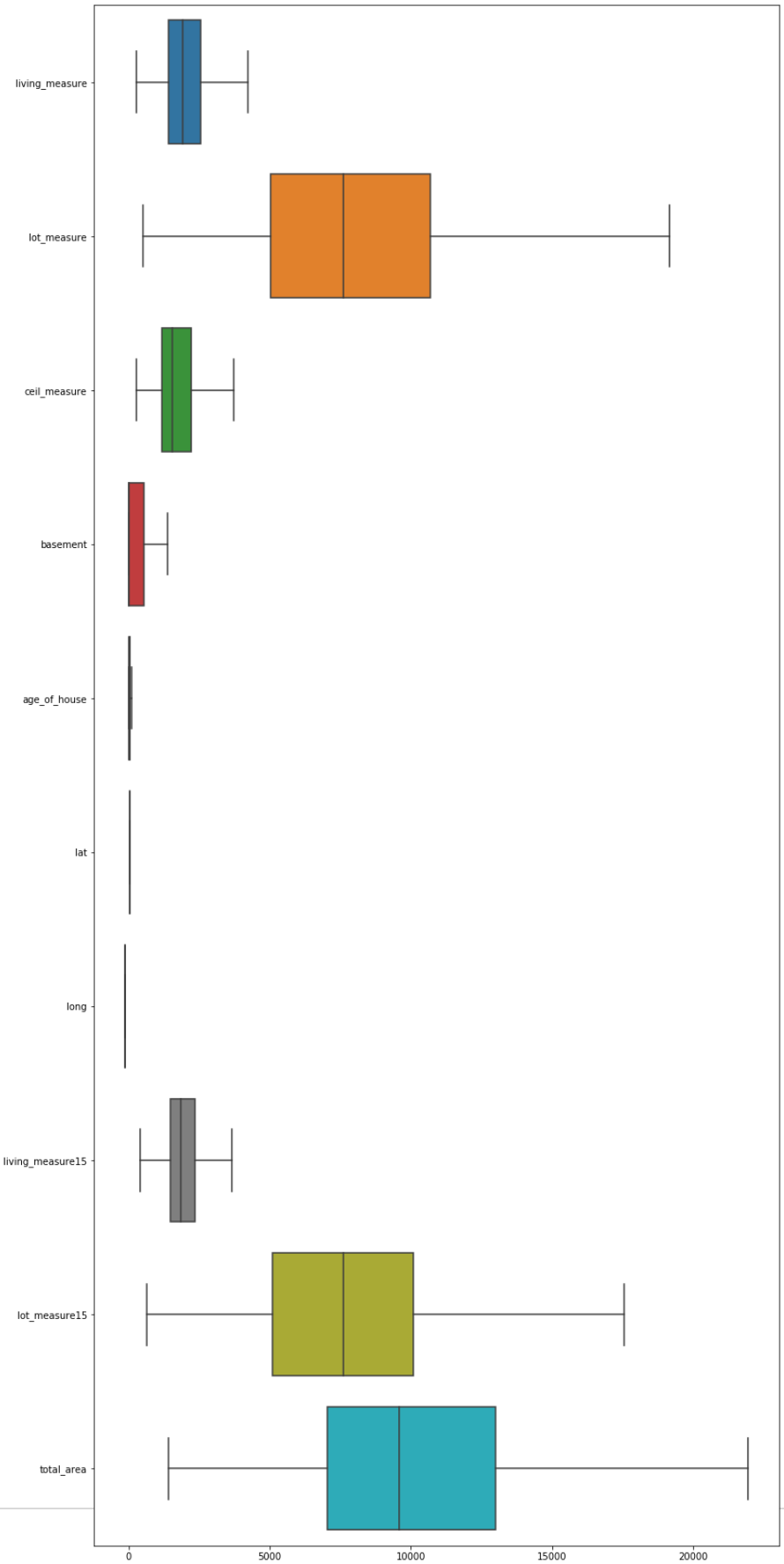


5. Looking at the pairplot and the correlation matrix, we realize that there is absolute similarity between lot_measure and total_area and machine learning models do not tend to do well on linearly correlated columns thus we dropped one of the two.
6. The following outlier distributions were observed:



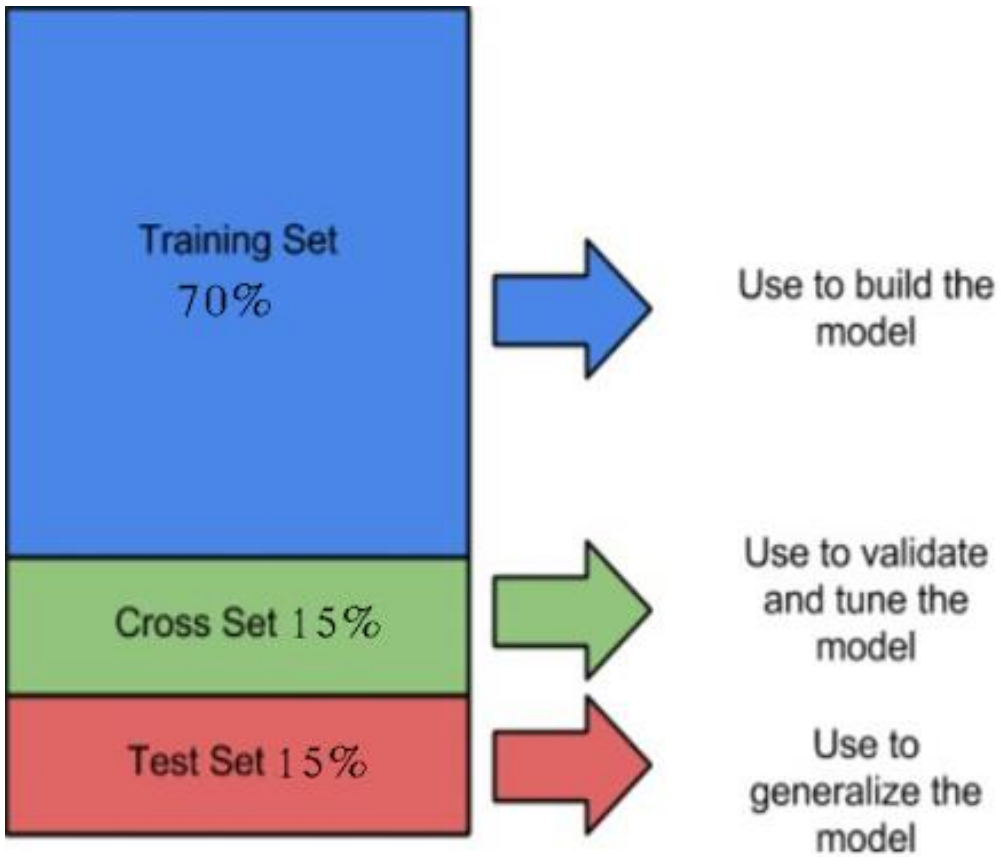


7. Looking at the distribution after outlier treatment:


living_measure
lot_measure
ceil_measure
basement
age_of_house
lat
long
living_measure15
lot_measure15
total_area
0
5000
10000
15000
20000

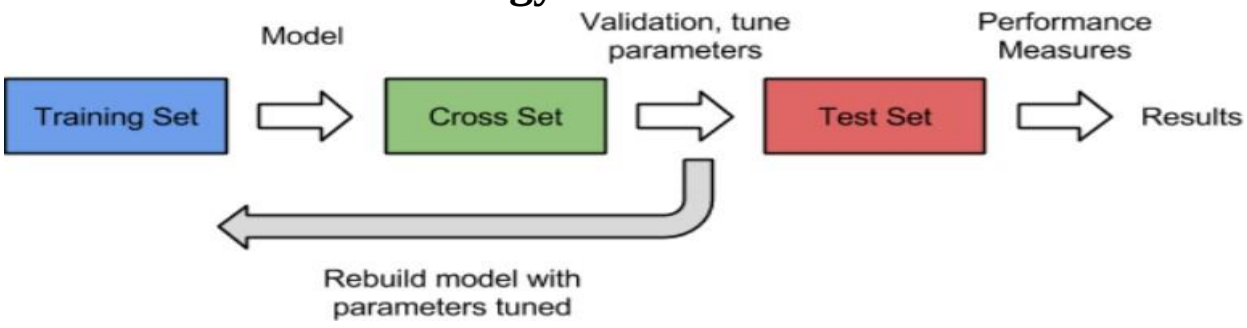
- **Sampling the dataset and model building**

1. Sampling the dataset: Given our dataset, we applied a sampling technique in order to divide it into three different subsets having each its own utility. It is assumed that more data we have to build a model the more it will have tend to give good results. We divided the data into train and test set in the ratio of 70:30. We divided the test set again into validation and test set in equal ratio. Therefore the ratio of train, test and validation sets were 70:15:15. The validation set would be kept untouched and would be used during hyper-parameter tuning.



2. The training set as its name indicates it is used to train the learning algorithm. The cross validation set is used to validate the model and make optimizations. Indeed, because the model is built from the training set, we can't check it on the same set because the result would be overly optimistic. It can be also useful for optimization as I will show in the following sections. The test set is used only to see how well the learning algorithm is generalized, meaning how it performs with unknown data.

- **Problem Methodology**



A regression problem requires the prediction of a quantity. Since we have to predict the quantity 'price', the problem is a regression problem. A problem with multiple input variables is often called a multivariate regression problem.

We would be trying out different regression machine learning algorithms.

- **Initial Benchmark**

We are expecting a winner of 85-90% model performance.

- **Metrics Used**

There are many different evaluation metrics out there but only some of them are suitable to be used for regression.

There are 3 main metrics for model evaluation in regression:

1. R Square/Adjusted R Square

2. Mean Square Error(MSE)/Root Mean Square Error(RMSE)

3. Mean Absolute Error(MAE)

R-Squared

R Square measures how much of variability in dependent variable can be explained by the model. It is square of Correlation Coefficient(R) and that is why it is called R Square.

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

R square formula

R Square is calculated by the sum of squared of prediction error divided by the total sum of square which replace the calculated prediction with mean. R Square value is between 0 to 1 and bigger value indicates a better fit between prediction and actual value.

Here, we are going to use R Squared/ Adjusted R Squared technique for evaluation since they are better in explaining the model to other people because you can explain the number as a percentage of the output variability.

R squared

```
print("R-squared of Linear Regression",metrics.r2_score(y_test, y_pred_lr))
print("R-squared of Linear Regression",metrics.r2_score(y_test,y_pred_lasso))
print("R-squared of Ridge Regression",metrics.r2_score(y_test, y_pred_ridge))
```

```
R-squared of Linear Regression 0.7247307987615725
R-squared of Linear Regression 0.7197590033185964
R-squared of Ridge Regression 0.7227733063918524
```

We can represent the accuracy of Linear Regression in the form of 72.4 %.

Mean Absolute Error

```
print("MAE of Linear Regression",metrics.mean_absolute_error(y_test, y_pred_lr))
print("MAE of Lasso Regression",metrics.mean_absolute_error(y_test, y_pred_lasso))
print("MAE of Ridge Regression",metrics.mean_absolute_error(y_test, y_pred_ridge))
```

```
MAE of Linear Regression 118677.95278909412
MAE of Lasso Regression 118763.56922040529
MAE of Ridge Regression 118748.03369857592
```

RMSE metric

```
print("RMSE of Linear Regression",rmsle(y_test, y_pred_lr))
print("RMSE of Lasso Regression",rmsle(y_test, y_pred_lasso))
print("RMSE of Ridge Regression",rmsle(y_test, y_pred_ridge))
```

```
RMSE of Linear Regression 193480.92836758387
RMSE of Lasso Regression 195220.39433395094
RMSE of Ridge Regression 194167.64977336323
```

MSE, RMSE or MAE are better to be used to compare performance between different regression models.

- # Model Fitting and testing

1. We create an empty dictionary results{} to store the accuracies of our models. It will be used to compare our models for analysis later.
2. We start with simple linear regression algorithm

- **Learning Algorithm**

Among the large choice of learning algorithm we chose to use Linear Regression because my dependent variable is continuous. The goal was to model a relationship between y , the dependent variable and x , .. , x , the independent variable where is the number of 1 . p p them. The general form of this is given by the following equation

$$y = f(x) + \varepsilon$$

Since I had several independent variable (features) to predict the dependent variable (Price),I was in the case of multiple linear regression (the ^ just means estimation), given by

: $y (x) \frown = f \frown + \varepsilon\ f(x)\ x .. x \frown = \beta 0 + \beta 1\ 1 + . + \beta p\ p$

Formula :Linear Regression with multiple variables

This learning method is parametric because I assumed that the shape of my model (function) was linear that have simplified greatly the problem because I only needed to estimate a set of parameters. Nonetheless, the disadvantage of using a parametric method is that the model chosen will usually not match the true unknown form of f (function connecting the input variable x to the output variable y). This problem can be addressed by turning the model into something more flexible that can fit many different functional forms for f but can lean to other problems such as overfitting because we need to estimate more parameters. A precision about the error term. Actually it exists two

kind of error, the reducible error also known as residual term (distance between the estimated regression line and the data points), and the irreducible error also known as the error term (distance between the true regression line and the data points). The reducible error is what I tried to minimize as much as possible and it is actually the goal of the optimization algorithm used to find the unknown parameters

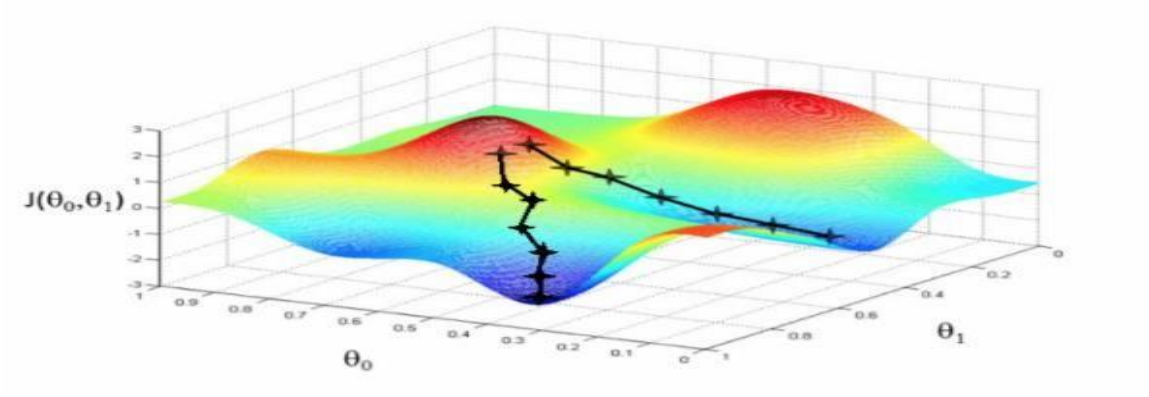
- **Cost Function**

The cost function is what we want to minimize as much as possible. In regression problem the commonly cost function is the mean squared error that quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation, given the following formula

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{f}(x_i))^2$$

- **Gradient Descent**

After having defined the model as linear and the cost function to evaluate its accuracy, we had to find a way to automatically determine the best parameters that minimize the cost function and consequently improve the performance of my model. A well-known optimization algorithm is the gradient descent. Basically it tries to find the local optima (or global optima if the function is convex) of the function by taking "steps" until it hopefully converges. The "step" is actually done by taking the derivative (the line tangent to a function) of the cost function. The slope of the tangent is the derivative at that point and gives the direction to move towards.



Gradient descent using two parameters

where θ0 and θ1 are the parameters respectively on the x axis and the y axis and J(θ ,) 0 θ1 the cost function on the z axis. For simplicity, the value of the parameters are given. The crosses represent each step taken by the gradient descent. Here the gradient descent has been run twice with different starting values for parameters θ0, θ1 The gradient descent is composed by two parameters that can be tuned, the learning rate α allowing to converge (or not) quickly by taking

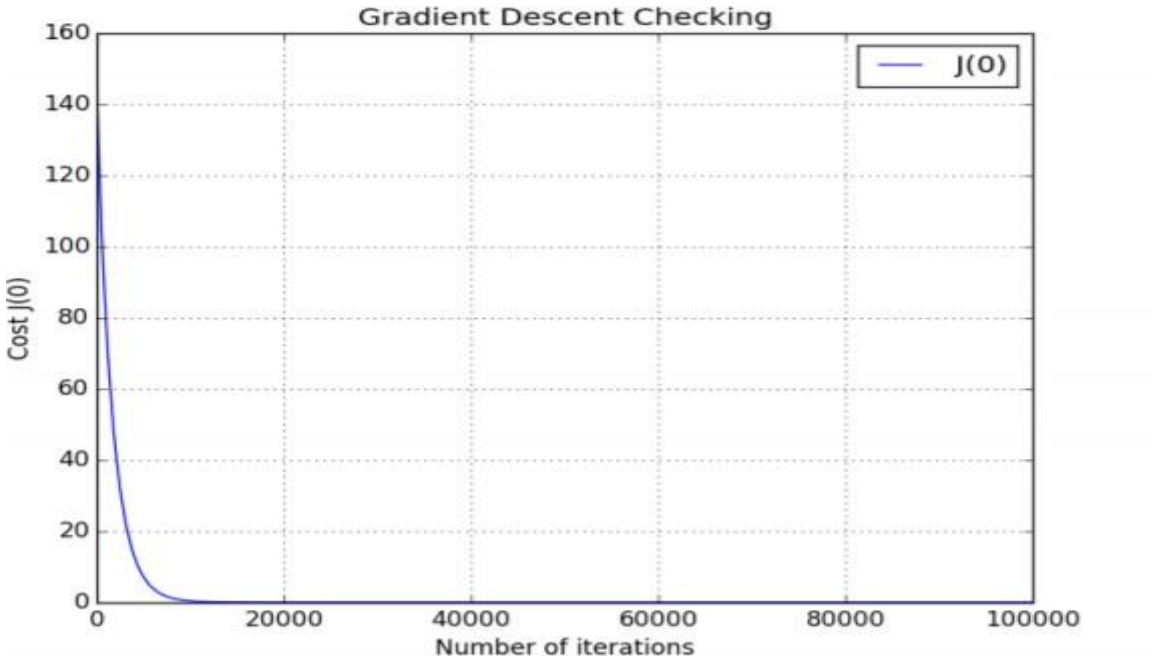
large values, and the number of iterations, meaning the number of times the gradient descent will iterate on the observations to approach the local optima. The learning rate has to be chosen carefully because if it is too large, the gradient descent will

fail to converge. Usually it takes values such as, 0.001, 0.003, 0.01, and so on

In the case of linear regression, the concrete algorithm of gradient as following :

$$\begin{aligned} &\textit{Repeat until convergence } \{ \\ &\quad \textit{for } j = 0, \ldots, p \\ &\quad\quad \beta_j := \beta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{f}(x_i) x_i^j) \\ &\} \end{aligned}$$

where p is the number of parameters, x is the value of the independent variable j i j for the i th observation with a special consideration for β0 since it corresponds to the y intercept in that case and has no independent variable x , its x is a constant equals to . j i 1 Due of that, I had to add a constant column equal to 1 in first position for all of my sets. It is important to understand that the parameters have to be updated simultaneously.



A way to check that the gradient descent works well with the given is by plotting the cost function over the number iterations. Indeed, since the gradient descent take a step toward the global optima and the function is convex, the cost function

should decrease after each iteration until convergence where the decreasing will be smoother.

- **Regularization**

The regularization is a technique to address overfitting / high variance problem. Overfitting arise when the model fits the data point too precisely and catch also noisy or outlier points. The model does not generalize well with new unknown data. This problem can be due to a complex model with many features. The regularization will penalize the parameter β by reducing them.

Initial Cost function

$$\frac{1}{n}\sum_{i=1}^{n}(y_i-\widehat{f}(x_i))^2$$

Regularized it becomes

$$\frac{1}{n}\sum_{i=1}^{n}(y_i-\widehat{f}(x_i))^2+\lambda\sum_{j=1}^{p}\beta_p^2$$

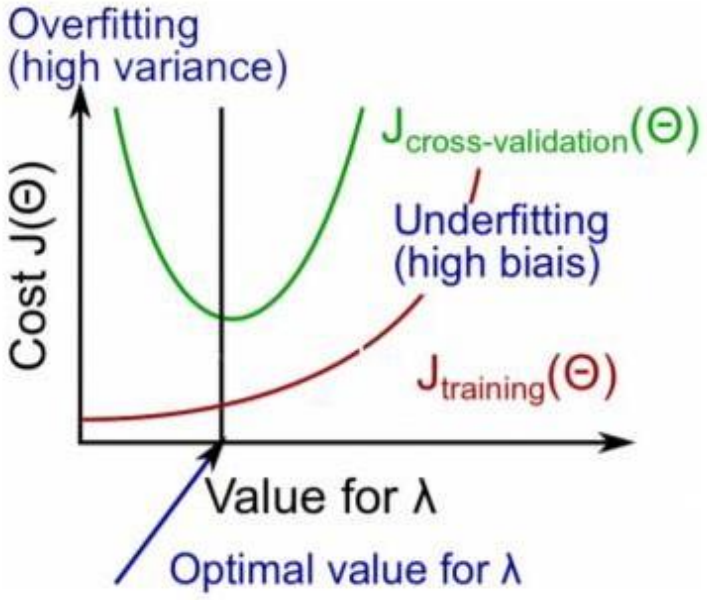
where λ is the regularization term determining how much the cost of the parameters will be inflated. Larger λ is, more the function will be smoothed, but a value too large will cause underfitting. Underfitting is when the model fails to catch the trend of the data (to fit the data), and so produce a large MSE. Note that we don't penalize β . 0 The regularization is also applied to the gradient descent and becomes as

follow :

$$
\begin{aligned}
&\textit{Repeat until convergence } \{ \\
&\quad \textit{for } j = 0 \\
&\qquad \beta_0 := \beta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{f}(x_i) x_i^0) \\
&\quad \textit{for } j = 1, \ldots, p \\
&\qquad \beta_j := \beta_j - \alpha \left[\left(\frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{f}(x_i) x_i^j \right) + \frac{\lambda}{n} \beta_j \right] \\
&\}
\end{aligned}
$$

One question could be how to choose the regularization term λ . To answer this, a way is to

plot the cost function over a set of λ using the training set and the cross validation set. Concretely, we learn a model using a given regularization term and the training set and then we plot the cost function for the cross validation set and the training set using the model learnt without regularization. This should give this kind of plot



3. We fit the linear regression model and the accuracy scores were noted. This was done again for the regularization techniques lasso and ridge.

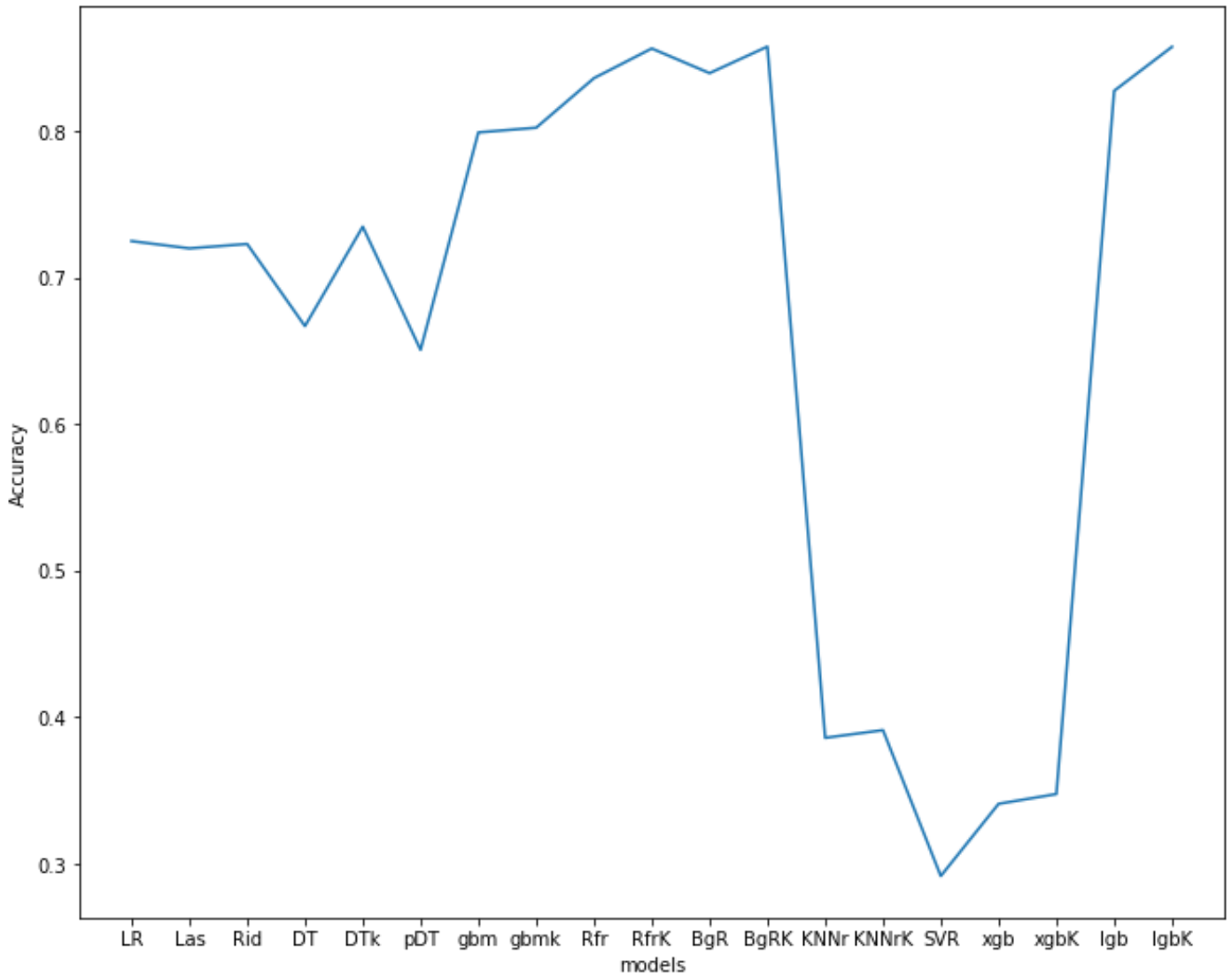
4. Next we trained and tested several models. The following models were put in use:

- Decision Tree Regressor along with its K-fold
- Pruned decision tree to reduce over fitting
- Ensemble techniques
  - Gradient Boosting Regressor along with its K-fold cross-validation
  - Random Forest Regressor along with its K-fold cross-validation
  - Bagging Regressor along with its K-fold cross-validation
  - KNN regressor along with its K-fold cross-validation
  - Support vector regressor along with its K-fold cross-validation
  - XGBoost along with its K-fold cross-validation
  - Lightgbm along with its K-fold cross-validation

- **Results**

The dictionary containing the accuracy scores of the models was plotted and we get the following plot:

```
{'LR': 0.7247307987615725,
 'Las': 0.7197590033185964,
 'Rid': 0.7227733063918524,
 'DT': 0.6667228986717038,
 'DTk': 0.7345491070305965,
 'pDT': 0.6505248843687457,
 'gbm': 0.7988685318955611,
 'gbmk': 0.8020711603980658,
 'Rfr': 0.8358859671175011,
 'RfrK': 0.8561301649684522,
 'BgR': 0.8393062332285418,
 'BgRK': 0.8574035102302331,
 'KNNr': 0.38582851166453636,
 'KNNrK': 0.3910702009222057,
 'SVR': 0.29155015233821324,
 'xgb': 0.34081585176046525,
 'xgbK': 0.3474919038895837,
 'lgb': 0.827260448120759,
 'lgbK': 0.8572474085231252}
```



The scores of the different models are plotted above where the names have their usual meaning

- We see that :

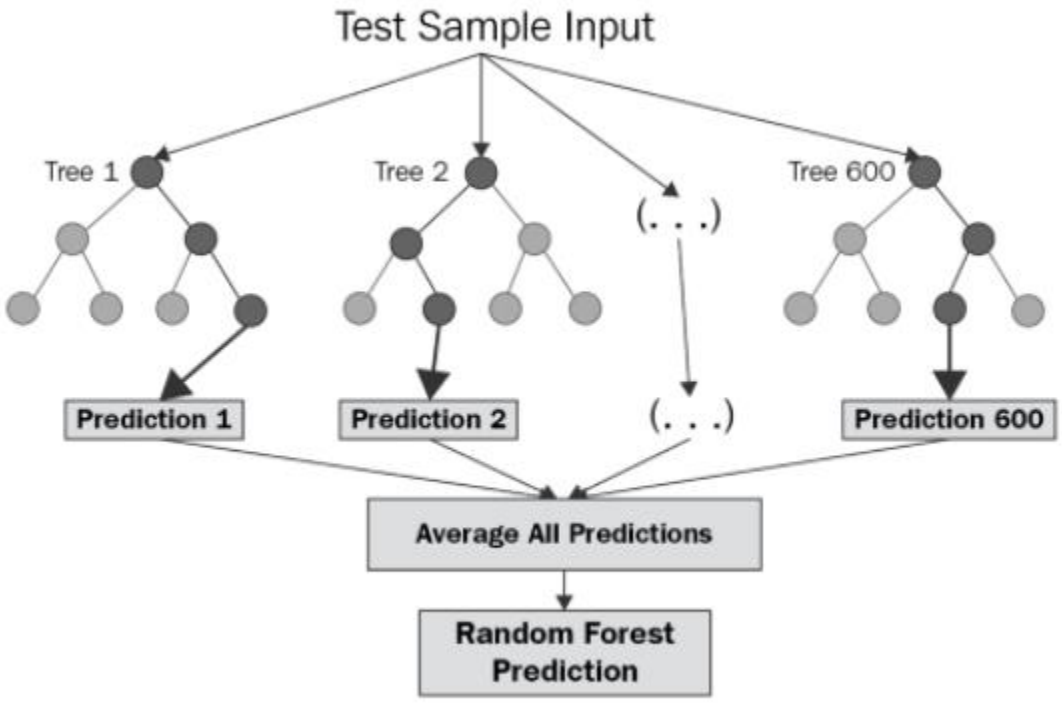
➢ Random forest with K-fold closs validation, bagging regressor with K-fold cross validation and lightgbm with K-fold cross-validation are the best contenders as

they are giving higher accuracies and are in par with each other.

- However ,it is important to note that in Random forest without K-fold cross validation, the accuracy score was higher than the other models and the difference between K-fold results and the accuracy of only Random forest trained once was lesser.

- Hence, we decide to go select the ensemble method Random forest as our model that we would want to go ahead with.
- However, to compare the results we decided to tune the hyper-parameters of lightgbm to see if we were right.

- **Final Model: Random Forest Regressor**



Random Forest Structure

Random forest is a Supervised Learning algorithm which uses ensemble learning method for classification and regression.

Random forest is a bagging technique. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees.

It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees, with some helpful modifications:

1. The number of features that can be split on at each node is limited to some percentage of the total (which is known as the hyper-parameter). This ensures that the ensemble model does not rely too heavily on any individual feature, and makes fair use of all potentially predictive features.
2. Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents overfitting.

- **Hyper-parameter Tuning**

We need to introduce the validation set during hyperparameter tuning. This is done so as to see how the model generalizes on unseen data. We combine both the training and validation sets and check the results on the test set.

```
X_merged=X_train.append(X_val)
```

```
y_merged=y_train.append(y_val)
```

Merging the training and validation sets

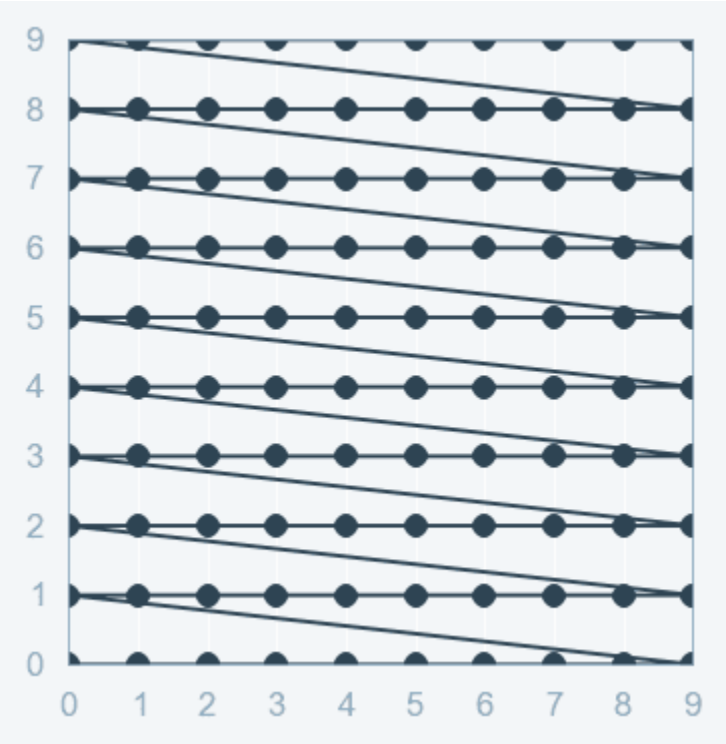
- We tuned the hyper parameters of the model lightgbm using RandomSearchCV and we got the best parameter combination. We used these parameters in the model for the test data and we got accuracy of around 72%.

- **GridSearchCV**

In this tuning technique, we simply build a model for every combination of various hyperparameters and evaluate each model. The model which gives the highest accuracy wins. The pattern followed here is similar to the grid, where all the values are placed in the form of a matrix. Each set of parameters is taken into consideration and the

accuracy is noted. Once all the combinations are evaluated, the model with the set of parameters which give the top accuracy is considered to be the best. Below is a visual description of uniform search pattern of the grid search.

One of the drawbacks of grid search is that when it comes to dimensionality, it suffers when evaluating the number of hyperparameters grows exponentially. However, there is no guarantee



that the search will produce the perfect solution, as it usually finds one by aliasing around the right set.

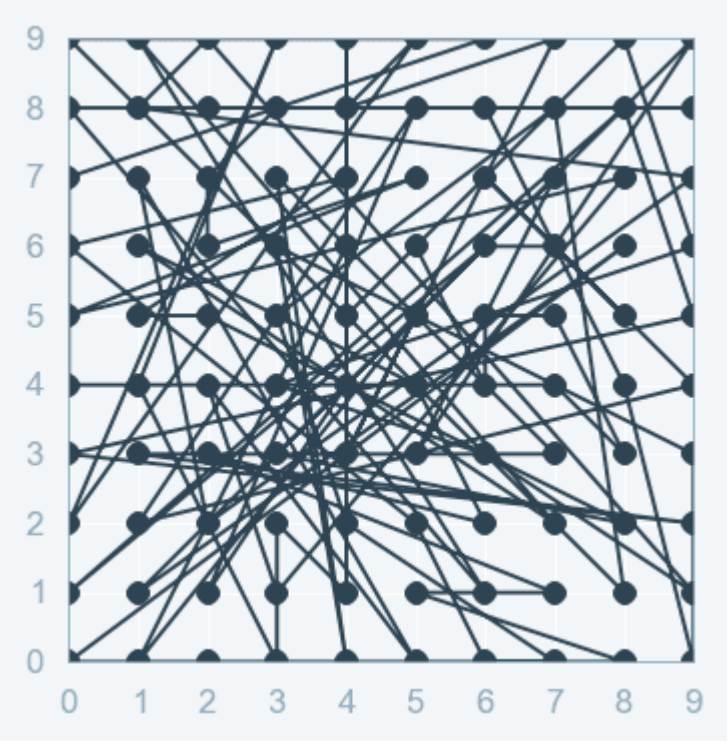
- When we performed the gridsearchCV and used the best parameters to train the random forest, we came up with an accuracy of 52.39%.

- he accuracy is relatively lower as we have severly restricted the hyper parameter ranges. This was done to minimize execution time. The Girdsearch has lower probability of finding the best combination than the randomsearch..

- ## RandomSearchCV

  Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. It is similar to grid search, and yet it has proven to yield better results comparatively. The drawback of random search is that it yields high variance during computing. Since the selection of parameters is completely random; and since no intelligence is used to sample these combinations, luck plays its part.

Below is a visual description of search pattern of the random search:



As random values are selected at each instance, it is highly likely that the whole of action space has been reached because of the randomness, which takes a huge amount of time to cover every aspect of the combination during grid search. This works best under the assumption that not all hyper-parameters are equally important. In this search pattern, random combinations of parameters are considered in every iteration. The chances of finding the optimal parameter are comparatively higher in random

search because of the random search pattern where the model might end up being trained on the optimised parameters without any aliasing.

- We get the following best parameters after hyper-parameter tuning using RandomsearchCV.

```
rf_random.best_params_
```

```
{'n_estimators': 15,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'log2',
 'max_depth': None,
 'bootstrap': False}
```

- When we test the model using these best hyper-parameters we get an accuracy score of 81.17%.

- **Implications**

We can say that our model will give good accuracy for house price predictions mostly 80-90% of the time. This prediction can be beneficial for first time customers or less experienced customers from falling prey into the hands of the agents who might be biased

or might have personal motive for profits. The customers can have a pre-informed notion on what the price could be and they can negotiate on that basis.

- **Limitations**

The most important limitations are the dearth of better features

- Better features can greatly influence a model's predicting performance. For example, we could have a feature called 'location' or 'neighbourhood'. One might search for the proximity to work, the charm of the neighborhood, how the home is situated on the lot, ease of access, noise from neighbors, traffic, and pets, as well as access to parks, shopping, schools, and public transportation. We could therefore benefit a lot from this feature. Sometimes some houses could be overpriced because they are situated near specific locations.

- People might be affected by a house's exterior appearance. They might get influenced by its colour, features etc. We could do good by adding such a feature.

- Sometimes people tend to reject houses based on the past stories regarding the house and their past owners. Although the price of these houses might be too low, the facilities provided might not be less. People tend to make an emotional decision regarding these houses. This creates problem in predicting the prices of these houses in machine learning algorithms.

- Sometimes other problems are inherent. Categorical features can be clearly understood by the human mind and common sense. But sometimes the algorithms

approach to deal with these categorical variables might not be exactly right.

- ❖ We can stack different ensemble methods to have a more sturdy accuracy scores.

- ❖ Better features could be provided to us for better prediction.

• Conclusion

We learned how to deal with categorical values and outliers. We observed and analysed different regression algorithms. We came up with one algorithm Random forest regressor and tuned its hyper-parameters. We got an accuracy of 81.17% after hyper-parameter tuning. Besides technical improvement, we also improved socially as a team. We improved upon collaboration and division of labour. The project exposed us to think about real life problem and how to deal with it.

We would take more time in feature engineering and EDA. We would like to stack our ensemble models to get a more robust accuracy.