CE6051: Machine Learning in Civil Engineering

(Jan-May 2021)

Project Report

'Housing Sale Price Prediction using Machine Learning Models'

Course Instructor:

Dr. Benny Raphael

Submitted by:

Group 8

CE17B040	Kedar Phadnis
CE17B110	Bennabhaktula Ritesh
CE17B116	Hrishikesh Gadekar
CE17B131	Suraj Meena
NA17B110	M S Nikhil Sen

Department of Civil Engineering

Indian Institute of Technology, Madras

April, 2021

1. Abstract

The real estate and housing prices are linked with the economy of any country. Despite this, we rarely do have accurate tools for predicting housing prices even from the large amount of data available over the years. The dataset used was obtained from 'The Center for Spatial Data Science', University of California website that contains housing sale prices for the years 2014 and 2015 from a city in USA. The aim is to predict the prices for the year 2016 and also identify any relevant patterns between the multiple features available which can help assess the factors affecting housing prices. The Real estate accounts for over $1.13 trillion in the US economy, and homes form one of the most significant investments. The ability to accurately forecast prices based on previous sales can be a valuable tool for ensuring that the buyer or seller is making an informed decision.

The analysis done consists of- using various libraries from Python viz. Pandas, NumPy, Seaborn, SciKit Learn etc., to summarize, pre-process the dataset to ensure the suitability of the dataset for various Machine Learning Models, visualizing the distributions in the dataset using graphical and statistical methods. Towards the end, we train various models on the dataset and assess their performance using multiple parameters such as R-squared, Residual plots etc. to figure out the best model which could be used for the purpose of prediction.

The general flow of the steps used for the analysis and prediction is as shown below:

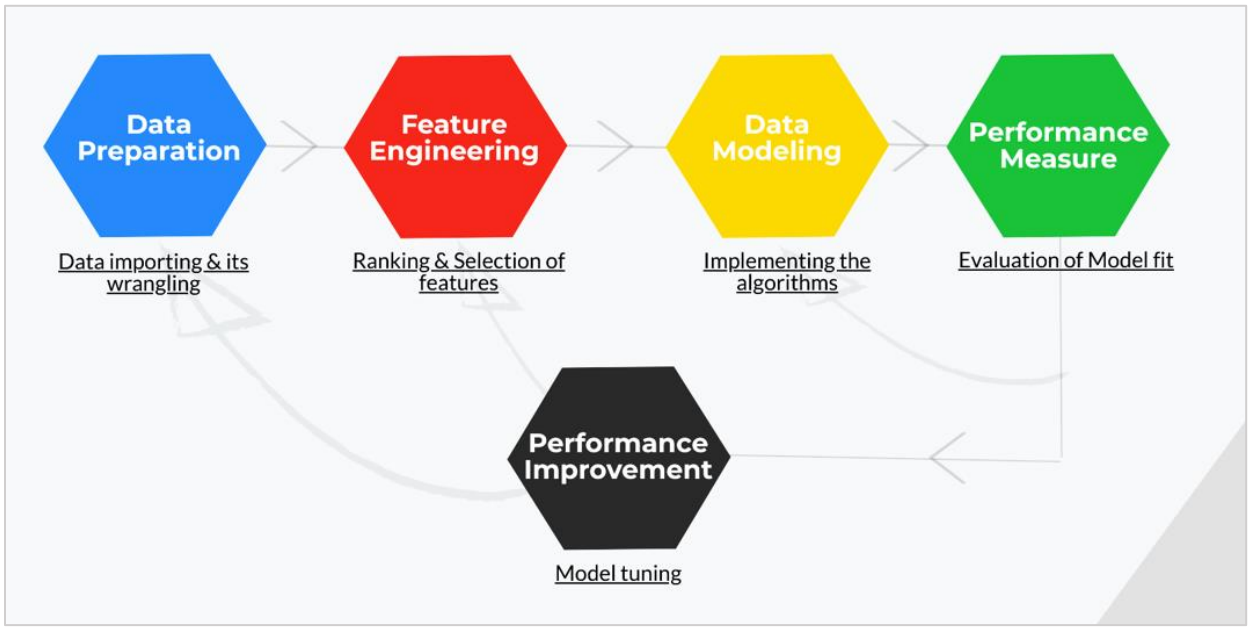


Figure 1: ML Model building process

In the upcoming sections, we will now go over each of the above points in detail, starting off with the data preparation. The Jupyter Notebook for the project can be found here:

GitHub repository Google drive

2. Data Pre-processing

It is important to clean and prepare the dataset in order to make it suitable for applying Machine learning models which generally ascertain various assumptions about the data. We import our dataset in the Jupyter Notebook as a Pandas DataFrame 'df'.

2.1 Data Description

The dataset consists of 20 independent features and the 'price' as the target feature forming a total of **21613 rows** as described in the table. There is a total of 17 Numerical & 4 Categorical features.

Sr No	Features	Significance
1	id	Unique ID for each home sold
2	date	Date of the home sale
3	bedrooms	Number of bedrooms
4	bathrooms	Number of bathrooms
5	sqft_living	Square footage of the apartments interior living space
6	sqft_lot	Square footage of the land space
7	floors	Number of floors
8	waterfront	A variable for whether the apartment is overlooking waterfront or not
9	view	An index from 0 to 4 of how good the view of the property was
10	condition	An index from 1 to 5 on the condition of the apartment,
11	grade	An index from 1 to 13
12	sqft_above	The sqft of the interior housing space that is above ground level
13	sqft_basement	The sqft of the interior housing space that is below ground level
14	yr_built	The year the house was initially built
15	yr_renovated	The year of the house's last renovation
16	zipcode	What zipcode area the house is in
17	lat	Lattitude of the location of the house
18	long	Longitude of the location of the house
19	sqft_living15	The sqft of interior housing living space for the nearest 15 neighbors
20	sqft_lot15	The square footage of the land lots of the nearest 15 neighbors
21	price	Price of each home sold

Based on the type of data present, we have broadly classified our features as follows:

- Continuous features: `['price', 'sqft_living', 'sqft_lot', 'lat', 'long', 'sqft_above', 'sqft_basement']`
- Discrete features: `['bedrooms, 'bathrooms', 'floors', 'waterfront', 'view', 'grade']`

The number of unique values in each of the feature were found using `df.nunique()` method.

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
21436	372	3625	13	30	1038	9782	6	2	5

condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
5	12	946	306	116	70	70	5034	752	777	8689

2.2 Feature Transformation

A majority of columns in the dataset were deemed to be in an acceptable form for applying ML algorithms. However, some of the columns required some transformation by converting them to new columns. Pandas library methods such as `to_datetime()` were extensively used for this purpose. These transformations and new features are listed below:

1) Dropping the 'id' column

```
# Dropping the 'id' column
df.drop('id', axis=1, inplace=True)
```

2) Converting the 'date' column to useful format

```
# Convert to date object
df['date'] = pd.to_datetime(df['date'])

df['year'] = pd.DatetimeIndex(df['date']).year
df['month'] = pd.DatetimeIndex(df['date']).month

df.drop('date', axis=1, inplace=True)
```

3) Creating a new feature 'age' of the house

```
df['age'] = df['year']-df['yr_built']
```

Figure 2: Using Pandas methods for feature transformations

1. `Year`: A new nominal feature extracted from the date feature as shown in the code.
2. `Age`: This continuous type feature indicates how old the house was at the time of sale.
3. `Renovated`: A binary feature indicating whether the house was renovated or not.

Apart from this, the dataset has no missing values. Thus, no imputation was required.

2.3 Outliers

Outliers are unusual values in a dataset, that usually fall a long way apart from the other observations. The boxplots shown in fig. 3 for all the features are a clear indicative of the outliers present in almost all the features of our dataset. The corresponding distributions indicate a near normal distribution of all these features with no skewness observed.

Now, it is almost always advisable to remove the outliers present in the dataset in order to ensure effectiveness of our model. For this, several methods such IQR method can be used.

However, it was concluded that these outliers correspond to large values of number of bathrooms, number of bedrooms, condition, grade, and view which may be from luxurious homes. Since this is possible in an ideal scenario, we decided to retain these outliers.

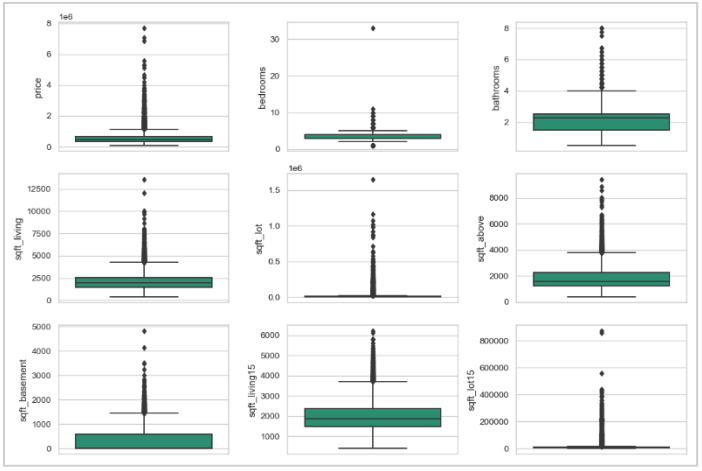


Figure 3: Outliers in the dataset

2.4 Inconsistency in the data

Some of the features in our data had anomalies. Over 800 rows of data had to be removed which was a significant loss. However, since this was clearly an erroneous data, we were bound to do so. These were addressed as follows:

```
# Checking for houses with either 0 bedrooms or 0 bathrooms
questionable1 = df[(df.bedrooms == 0) | (df.bathrooms == 0)]
questionable1
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
875	1100000.0	0	0.00	3064	4764	3.5	0
1149	75000.0	1	0.00	670	43377	1.0	0
3119	380000.0	0	0.00	1470	979	3.0	0
3467	288000.0	0	1.50	1430	1650	3.0	0
4868	228000.0	0	1.00	390	5900	1.0	0
5832	280000.0	1	0.00	600	24501	1.0	0
6994	1300000.0	0	0.00	4810	28008	2.0	0
8477	339950.0	0	2.50	2290	8319	2.0	0
8484	240000.0	0	2.50	1810	5669	2.0	0
9773	355000.0	0	0.00	2460	8049	2.0	0
9854	235000.0	0	0.00	1470	4800	2.0	0
10481	484000.0	1	0.00	690	23244	1.0	0
12653	320000.0	0	2.50	1490	7111	2.0	0
14423	139950.0	0	0.00	844	4269	1.0	0
18379	265000.0	0	0.75	384	213444	1.0	0
19452	142000.0	0	0.00	290	20875	1.0	0

16 rows × 22 columns

Figure 4: Using Pandas library, we identified all the rows having zero bedrooms or zero bathrooms. Since this is not an ideal scenario for conventional homes, we omitted all such rows. As seen near the left bottom, a total of 16 rows were deleted.

```
# Checking for houses with greater living area than their lot area
questionable2 = df[df['sqft_living'] > df['sqft_lot']]
questionable2
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	co
63	549000.0	3	1.75	1540	1044	3.0	0	0	
116	518500.0	3	3.50	1590	1102	3.0	0	0	
175	425000.0	3	2.50	1120	1100	2.0	0	0	
406	450000.0	3	2.00	1290	1213	3.0	0	0	
547	259950.0	2	2.00	1070	649	2.0	0	0	
...	...	...	...	...	...	...	...	...	
21595	520000.0	2	2.25	1530	981	3.0	0	0	
21601	467000.0	3	2.50	1425	1179	3.0	0	0	
21604	429000.0	3	2.00	1490	1126	3.0	0	0	
21607	475000.0	3	2.50	1310	1294	2.0	0	0	
21608	360000.0	3	2.50	1530	1131	3.0	0	0	

788 rows × 22 columns

Figure 5: Using Pandas library, we identified all the rows having interior living space of the house greater than its lot area. Since lot area of a house cannot be smaller than the living space, a total of 788 such rows were deleted.

3. Exploratory Data Analysis (EDA)

In the Exploratory Data Analysis, we use data visualization techniques in order to gain some more clear insights and observations about the distribution of the features in our dataset. The relationship between our features helped us understand the important features to use for the final modelling and some more interesting observations to help improve prediction accuracy.

3.1 General Observations

Some of the interesting observations from the different plots (available to view in the Notebook) are mentioned below-

- Price increases as the values of number of bathrooms, number of bedrooms, of sqft_living and sqft_above increase.
- As intuitive and also observed from the graphs, the price increases with increase in the Grade, Condition and View.
- The average price of a renovated house is greater than a house which was not renovated.
- For both the types of loans, the average annual income of the customers with default loan is higher than the ones with non-default loans.
- A waterfront significantly increases the price of the house.

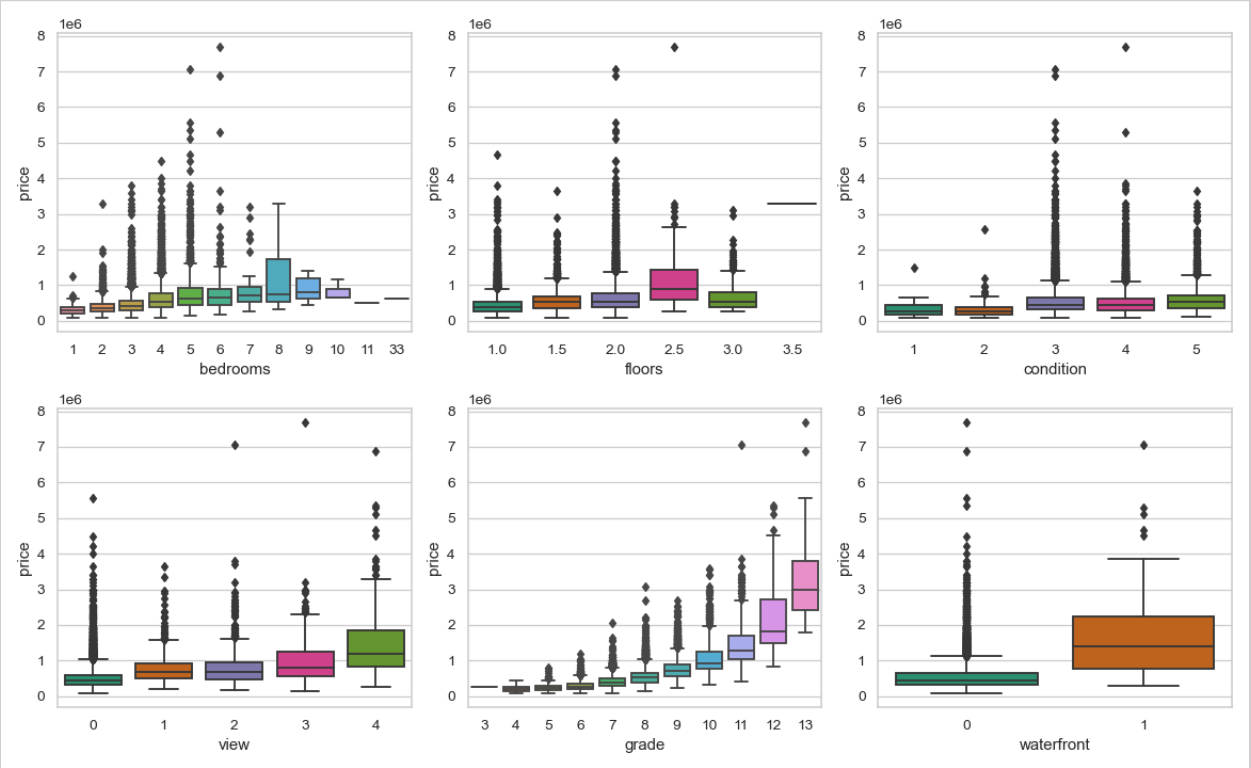


Figure 6: Boxplot showing variation between the discrete variables and the target variable **price** on the Y-axis

3.2 Some more Plots

We used Python Seaborn library in order to plot variables against each other as well as observe univariate distributions. The Seaborn library offers a variety of univariate, bivariate and multivariate (with the 'hue' parameter) plots. Some of the plots and their interpretations are shown below:

i) The variables sqft_living and sqft_above indicate a linear relationship with 'price'

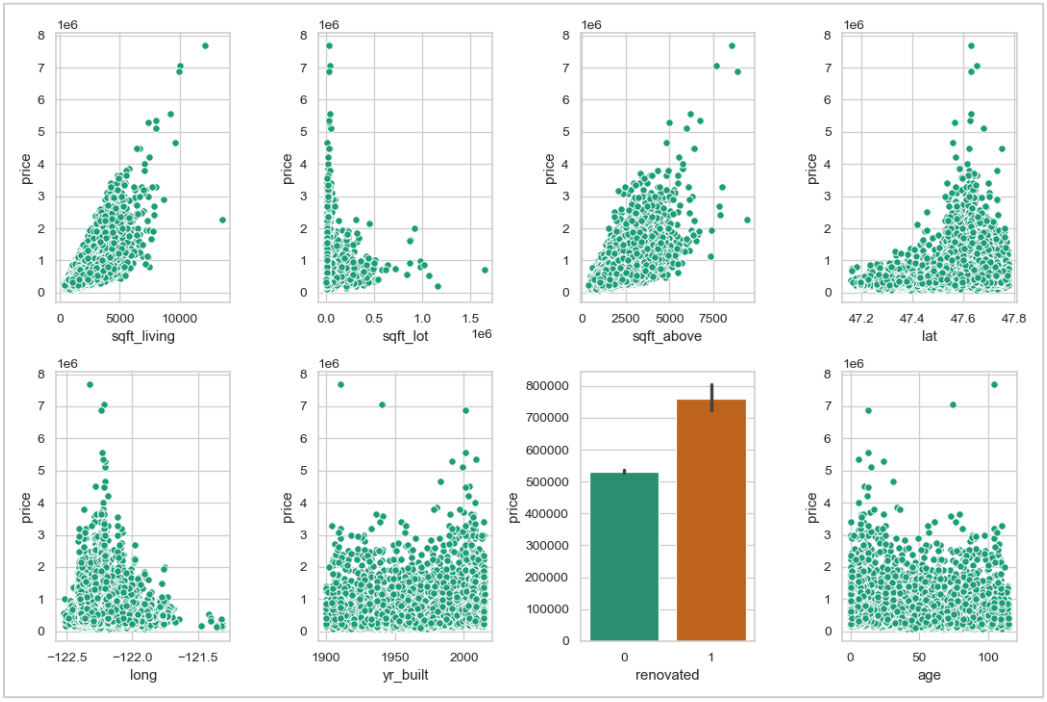


Figure 7 Scatterplot and barplot showing variation with the target variable 'price'

ii) The number of bathrooms or bedrooms increase with lot and living space area.

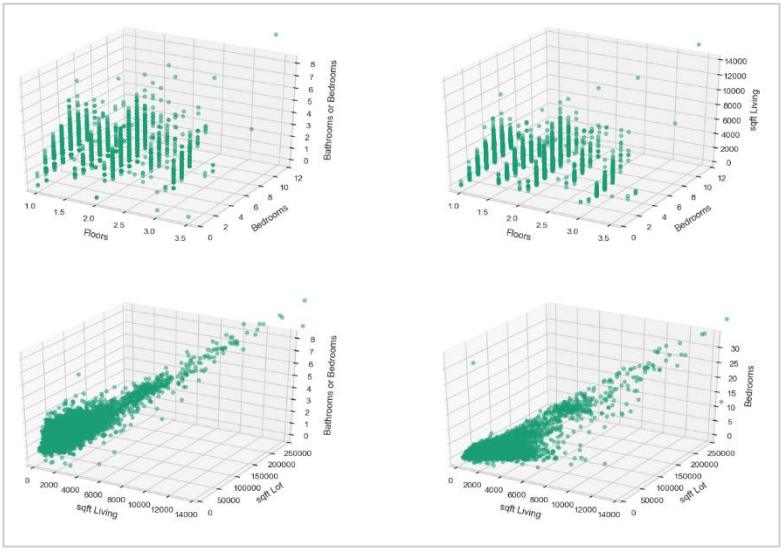


Figure 8: 3D plots showing pattern between 3 different features

iii) The subplot below shows variation between two different features with ‘hue’ set on the waterfront feature. The target variable ‘price’ is on the Y-axis.

Figure 9: Bivariate plots with 'hue' set on waterfront feature

4. Feature Engineering

At times, the features available in the dataset can be readily used for analysis and are sufficient for the purpose of modelling. However, it's often necessary to create additional engineered features for an enhanced dataset. The process of creating new features from raw data to increase the predictive power of the learning algorithm is called as 'Feature Engineering'. Engineered features usually help capture additional information that is not easily apparent in the original feature set. Several types of feature engineering techniques include imputation of missing values, handling outliers, log transformations, binning, one-hot encoding and scaling etc.

In our dataset, we have applied **binning** on the feature 'age' and 'yr_renovated'. The houses in the dataset are as old as 115 years and as new as less than a year old. Since the range of this feature is vast, we decided to create several interval type features out of these features using binning.

The code snippet shows the code used for the same:

```
df_dm = df.copy()
```

```
# just take the year of sale
df_dm['sales_yr']=df_dm['year']

# add the age of the buildings when the houses were sold as a new column
df_dm['age']=df_dm['sales_yr'].astype(int)-df_dm['yr_built']
# add the age of the renovation when the houses were sold as a new column
df_dm['age_rnv']=0
df_dm['age_rnv']=df_dm['sales_yr'][df_dm['yr_renovated']!=0].astype(int)-df_dm['yr_renovated'][df_dm['yr_renovated']!=0]
df_dm['age_rnv'][df_dm['age_rnv'].isnull()]=0

# partition the age into bins
bins = [-2,0,5,10,25,50,75,100,100000]
labels = ['l_1','1_5','6_10','11_25','26_50','51_75','76_100','g_100']
df_dm['age_binned'] = pd.cut(df_dm['age'], bins=bins, labels=labels)
# partition the age_rnv into bins
bins = [-2,0,5,10,25,50,75,100000]
labels = ['l_1','1_5','6_10','11_25','26_50','51_75','g_75']
df_dm['age_rnv_binned'] = pd.cut(df_dm['age_rnv'], bins=bins, labels=labels)
```

Figure 10: Code for binning the 'age' feature

The newly created or engineered features are:

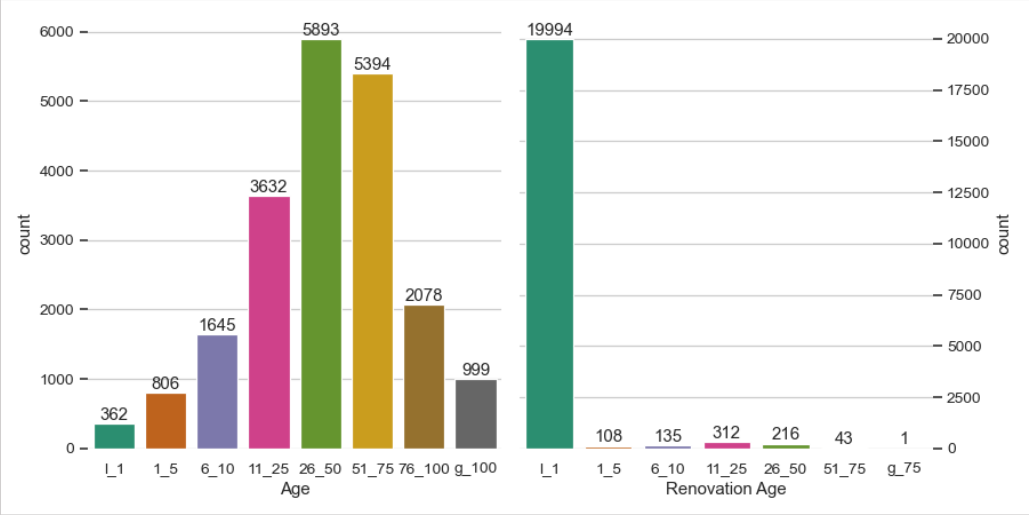


Figure 11: The newly engineered features show that majority of houses weer 26-50 years old

5. Feature Selection

Feature selection refers to the decision about which predictor variables should be included in a model. It involves a combination of common sense, theory, and testing the effectiveness of different combinations of features in a predictive model. It mainly eliminates the effects of the curse of dimensionality. Besides, this technique reduces the problem of overfitting by enhancing the generalisation in the model. Thus, it helps in better understanding of data, improves prediction performance and accuracy by removing irrelevant features, and reducing the computational time as well as space which is required to run the algorithm.

The various methods available for feature selection can be broadly classified into 3 main categories. Each of these have their own advantages and disadvantages as described in the table below:

Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.	Embeds (fix) features during **model building process.** Feature selection is done by observing each iteration of model training phase.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features	Sits **between Filter methods and Wrapper methods** in terms of time complexity
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features	Generally used to reduce **over-fitting** by **penalizing** the coefficients of a model being too large.
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.	Examples - **LASSO, Elastic Net, Ridge Regression** etc.

Since we have a huge dataset of 21000 rows with over 20 features with most of them being continuous numerical type, we decided to use the following Wrapper feature selection methods:

1. Forward feature selection technique: It is an iterative method in which we start with having no feature in the model and then start fitting the model with each individual feature one at a time and select the feature with the minimum p-value. Now fit a model with two features by trying combinations of the earlier selected feature with all other remaining features. Again, select the feature with the minimum p-value. Now fit a model with three features by trying combinations of two previously selected features with other remaining features. Repeat this process until we have a set of selected features with a p-value of individual features less than the significance level. We have used in-built Python feature_selection library to implement this technique.

2. Backward feature selection technique: In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features. We used same feature_selection library with forward parameter set to 'False'.
3. Bi-directional Elimination: It is similar to forward selection but the difference is while adding a new feature it also checks the significance of already added features and if it finds any of the already selected features insignificant then it simply removes that particular feature through backward elimination. We choose a significance level to enter and exit the model, Perform the next step of forward selection (newly added feature must have p-value < SL_in to enter). Then perform all steps of backward elimination (any previously added feature with p-value > SL_out is ready to exit the model). Repeat until we get a final optimal set of features.
4. Recursive Feature Elimination: It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

Now, it is also important to identify the optimal number of features. We perform several iterations of forward feature selection and observe the performance. As we can see, after peak performance at n=15, the graph is almost constant. Thus, we will choose top 15 features from the feature selection methods. These features are ranked as shown.

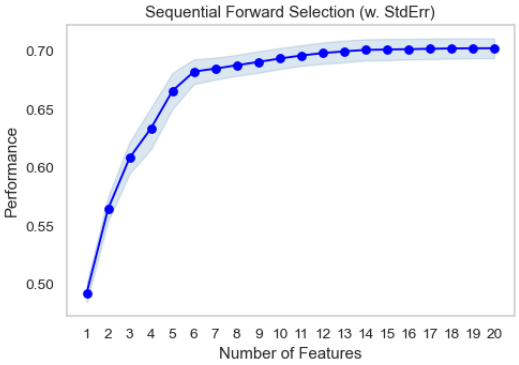


Figure 12: The StdErr graph to identify the optimal number of features.

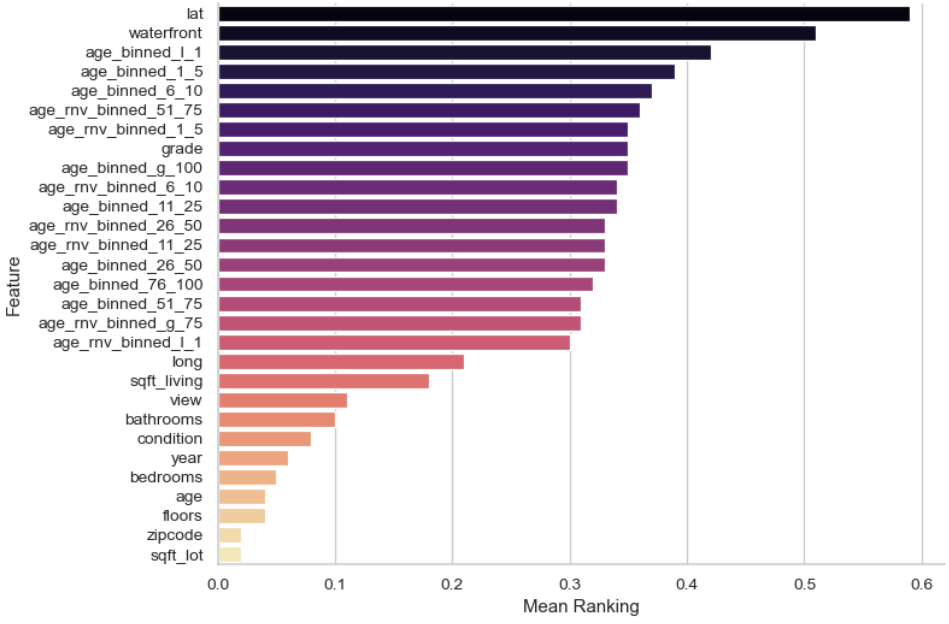


Figure 13: The top 15 features ranked according to their mean importance obtained from feature selection methods

5.1 Correlation Matrix

The correlation present between the variables can be detrimental to our model's performance. We find the correlation using Pandas `df.corr()` method and plot it using Seaborn.

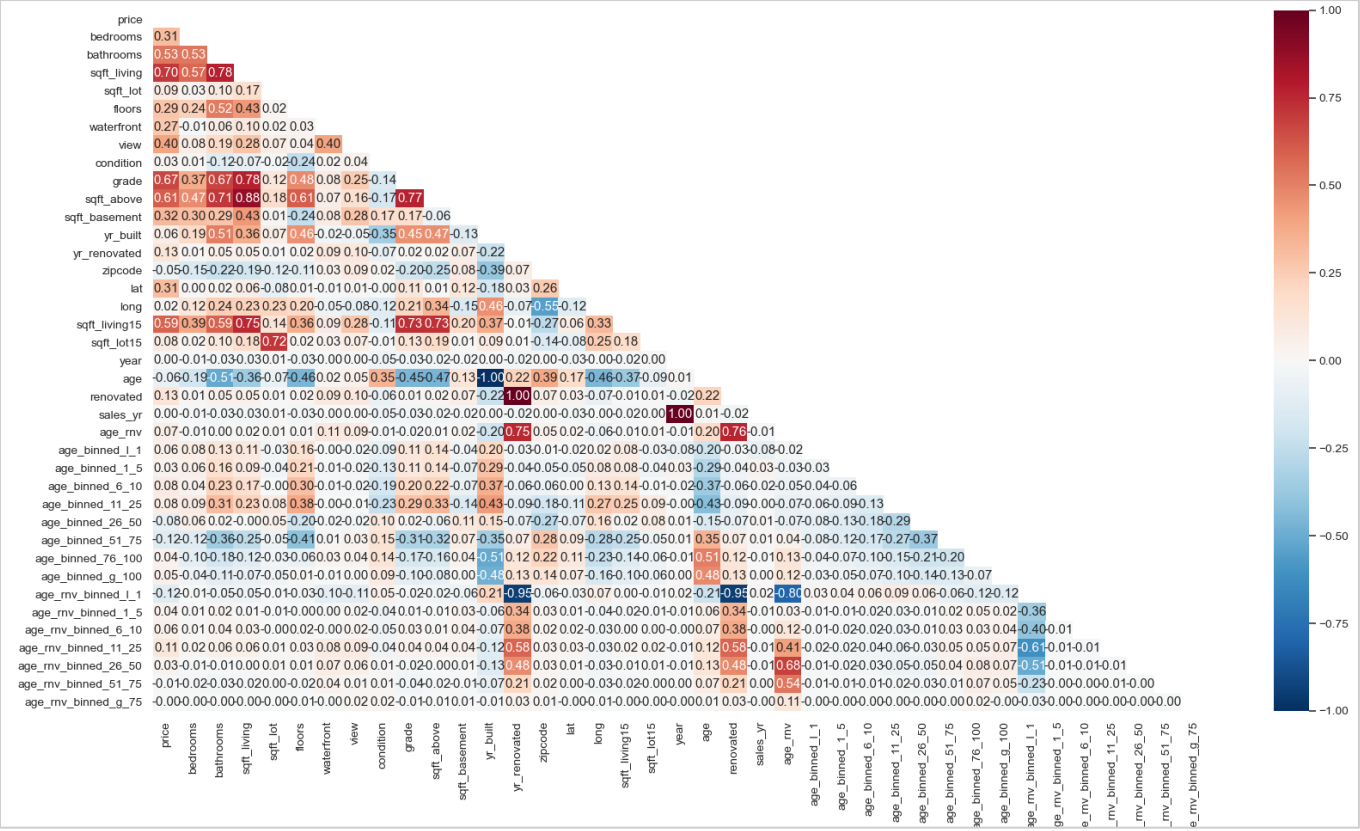


Figure 14: Correlation Heatmap

As we can observe from the above heatmap, Dark Red and Blue hues indicate high (+ve and -ve respectively) correlation. There are a few highly correlated variables as we had expected. The correlated features with a threshold (> 0.6 or < -0.6) are: (age, yr_built), (age_rnv, yr_renovated), (year, sales_yr), (sqft_living, bathrooms), (sqft_lot15, sqft_lot), (sqft_above, grade), (sqft_living15, grade), (sqft_living15, sqft_above). We decided to omit the the features: [`yr_built, yr_renovated, sales_yr, age, age_rnv`]

6. Model Building

In this stage, we tried to fit various models to the data and assessed the fit using **R-squared** value to have a better understanding of the fit. We will go over each of the models and check their performance. The top 15 features obtained from Feature selection methods were used for all the models to maintain consistency. A train-test split of 75-25% ratio was performed on the dataset.

Top 15 features		
bedrooms	waterfront	year
bathrooms	long	age_binned_11_25
sqft_living	age	age_binned_26_50
view	zipcode	age_rnv_binned_1_5
grade	lat	condition

6.1 Multilinear Regression

It is a statistical technique that uses several independent variables to predict the outcome of a dependent variable. The goal of multilinear regression is to model the linear relationship between the independent variables and dependent variables. We used Python built-in OLS and Scikitlearn library for implementation of this algorithm.

The multiple regression model is based on the following assumptions:

- There is a linear relationship between the dependent variables and the independent variables
- The independent variables are not too highly correlated with each other
- y_i observations are selected independently and randomly from the population
- Residuals should be normally distributed with a mean of 0 and variance σ

The coefficient of determination (R-squared) is a statistical metric that is used to measure how much of the variation in outcome can be explained by the variation in the independent variables. R^2 always increases as more predictors are added to the MLR model, even though the predictors may not be related to the outcome variable. R^2 by itself can't be used to identify which predictors should be included in a model and which should be excluded

The results of the Multilinear regression model are as shown in **Figure 15**. Since the R-squared value is close to 0.7, the model is a good fit. Also, the P-value is less than 0.05 for all the coefficients which rejects the Null hypothesis of any of them being zero.

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.698
Model:	OLS	**Adj. R-squared:**	0.697
Method:	Least Squares	**F-statistic:**	4798.
Date:	Wed, 14 Apr 2021	**Prob (F-statistic):**	0.00
Time:	12:56:02	**Log-Likelihood:**	-2.8398e+05
No. Observations:	20809	**AIC:**	5.680e+05
Df Residuals:	20798	**BIC:**	5.681e+05
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.328e+07	2.84e+06	4.675	0.000	7.71e+06	1.88e+07
bedrooms	-3.397e+04	1941.713	-17.496	0.000	-3.78e+04	-3.02e+04
bathrooms	4.333e+04	3251.826	13.323	0.000	3.7e+04	4.97e+04
sqft_living	179.5833	3.211	55.936	0.000	173.290	185.876
waterfront	5.978e+05	1.78e+04	33.668	0.000	5.63e+05	6.33e+05
view	5.233e+04	2143.754	24.413	0.000	4.81e+04	5.65e+04
grade	1.046e+05	2068.845	50.568	0.000	1.01e+05	1.09e+05
zipcode	-650.9593	33.254	-19.576	0.000	-716.139	-585.780
lat	5.879e+05	1.09e+04	54.082	0.000	5.67e+05	6.09e+05
long	-1.788e+05	1.29e+04	-13.857	0.000	-2.04e+05	-1.53e+05
age	3029.1700	67.224	45.061	0.000	2897.406	3160.934

Figure 15: Multilinear Regression Results

6.2 Random Forest Regression

Random forest is an ensemble method which combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model. It is an ensemble of decision trees, usually trained with the bagging method. The general idea of the bagging method is that a combination of learning models increases the overall result. It uses averaging to improve the predictive accuracy and control over-fitting. We used Python built-in Scikitlearn library for implementation of this algorithm.

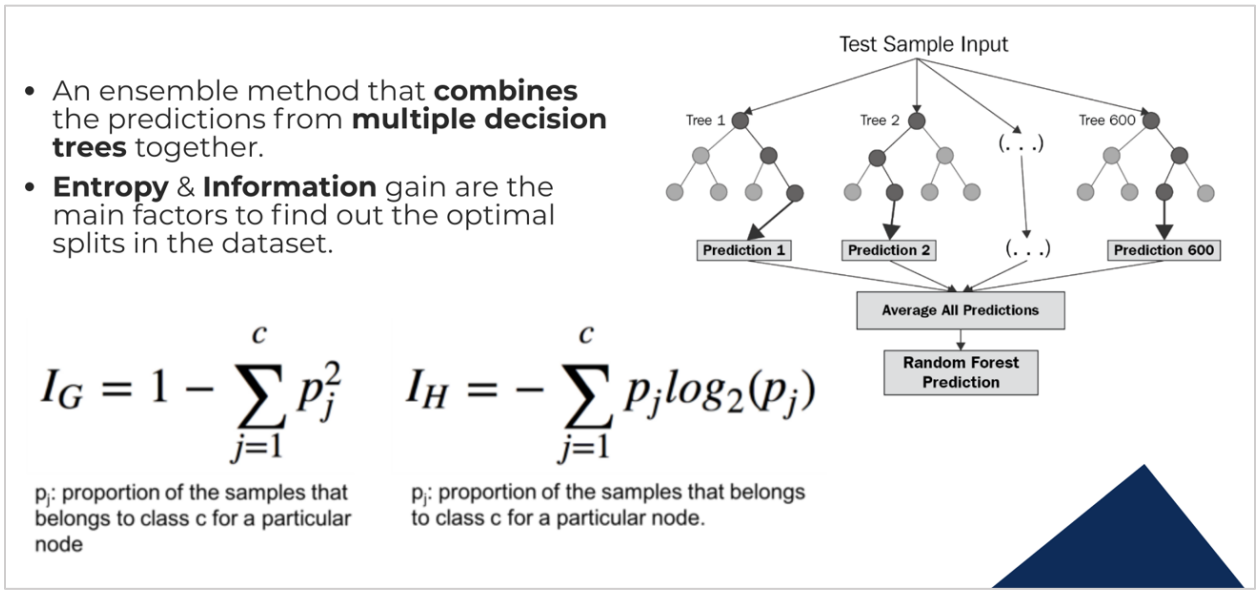


Figure 16: Random Forest Regression

Random Forest provides a good accuracy without any feature scaling required. However, it can sometimes overfit the data and also take longer time for prediction.

6.3 kNN Regression

It is a statistical technique that uses several independent variables to predict the outcome of a KNN regression is a non-parametric method that approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set. We used Python built-in Scikitlearn Kneighbors Regressor library for implementation of this algorithm.

The first step is to calculate the distance between the new point and each training point. There are various methods for calculating this distance, of which the most commonly known methods are – Euclidian, Manhattan (for continuous) and Hamming distance (for categorical). Once the distance of a new observation from the points in our training set has been measured, the next step is to pick the closest points. The number of points to be considered is defined by the value of k.

For a very low value of k (suppose k=1), the model overfits on the training data, which leads to a high error rate on the validation set. On the other hand, for a high value of k, the model performs poorly on both train and validation set. The size of the neighborhood needs can be chosen using cross-validation to select the size that minimizes the mean-squared error. The kNN was the worst performing model on our dataset.

6.4 Gradient Boosting Algorithm

The Gradient boosting algorithm involves three elements:

- A **loss function** to be optimized. e.g: Regression may use a squared error and classification may use logarithmic loss.
- A **weak learner** to make predictions. Allowing subsequent models outputs to be added and correct the residuals in the predictions. Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.
- An **additive model** to add weak learners to minimize the loss function. Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees. After calculating error or loss, the weights are updated to minimize that error.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used. Decision trees are used as the weak learner in gradient boosting. Larger trees can be used generally with 4-to-8 levels. It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes. Instead of parameters, we have weak learner sub-models or more specifically decision trees.

We used Python sklearn.ensemble libray; GradientBoostingRegressor module to implement this algorithm. The code is as shown in **Figure 17.**

```
est = GradientBoostingRegressor(n_estimators=400, max_depth=5,
        loss='ls',min_samples_split=2,learning_rate=0.1)
est.fit(X_train, y_train)

y_predtr = est.predict(X_train)
y_predts = est.predict(X_test)

r_squaretr = est.score(X_train, y_train)
r_squarets = est.score(X_test,y_test)
msetr = metrics.mean_squared_error(y_train, y_predtr)
msets = metrics.mean_squared_error(y_test, y_predts)
maetr = metrics.median_absolute_error(y_train, y_predtr)
maets = metrics.median_absolute_error(y_test, y_predts)
```

Figure 17: Gradient Boosting implementation in Python

6.5 AdaBoost Regression

AdaBoost is one of the first boosting algorithms to be adapted in solving practices. Adaboost helps you combine multiple “weak classifiers” into a single “strong classifier”. The weak learners in AdaBoost are decision trees with a single split, called decision stumps. AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.

The power of ensembling is such that we can still build powerful ensemble models even when the individual models in the ensembles are extremely simple. It helps us capture many of the non-linear relationships, which translates into better prediction accuracy on the problem of interest.

The importance of a feature is computed as the normalized total reduction of the criterion brought by that feature. It is also known as the Gini importance.

The coefficient R2 is defined as (1−u/v), where u is the residual sum of squares ((y_true - y_pred) ** 2).sum() and v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

We used Python sklearn.ensemble libray; AdaBoostRegressor module to implement this algorithm. The code is as shown in **Figure 18.**

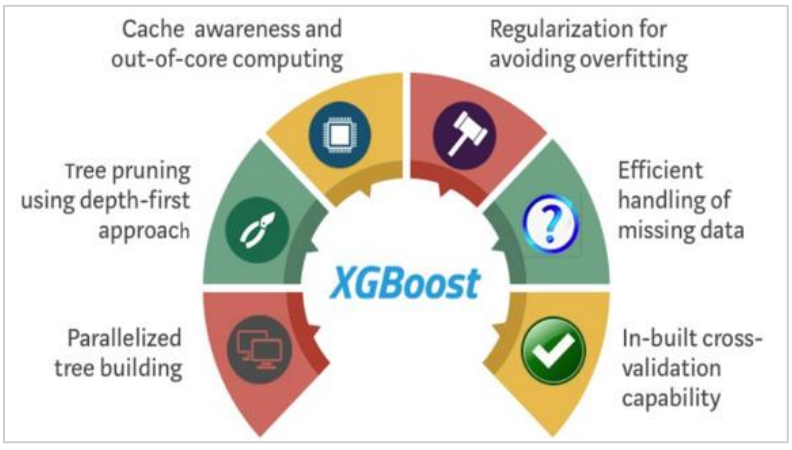
```
ada = AdaBoostRegressor(n_estimators=50,
                        learning_rate=0.2,loss='exponential')
ada.fit(X_train, y_train)

y_predtr = ada.predict(X_train)
y_predts = ada.predict(X_test)

r_squaretr = ada.score(X_train, y_train)
r_squarets = ada.score(X_test,y_test)
msetr = metrics.mean_squared_error(y_train, y_predtr)
msets = metrics.mean_squared_error(y_test, y_predts)
maetr = metrics.median_absolute_error(y_train, y_predtr)
maets = metrics.median_absolute_error(y_test, y_predts)
```

Figure 18: AdaBoost implementation in Python

6.6 XGBoost Regression

Extreme Gradient Boosting is an efficient implementation of the gradient boosting algorithm which is designed to be both computationally efficient and highly effective. The most common loss functions in XgBoost for regression problems is reg:linear and reg:logistics for binary classification. XgBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. Commonly configured hyperparameters are the following:

- **n_estimators**: The number of trees in the ensemble, often increased until no further improvements are seen.
- **max_depth**: The maximum depth of each tree, often values are between 1 and 10.
- **eta**: The learning rate used to weight each model, often set to small values such as 0.3, 0.1, 0.01, or smaller.
- **subsample**: The number of samples (rows) used in each tree, set to a value between 0 and 1, often 1.0 to use all samples.
- **colsample_bytree**: Number of features (columns) used in each tree, set to a value between 0 and 1, often 1.0 to use all features.

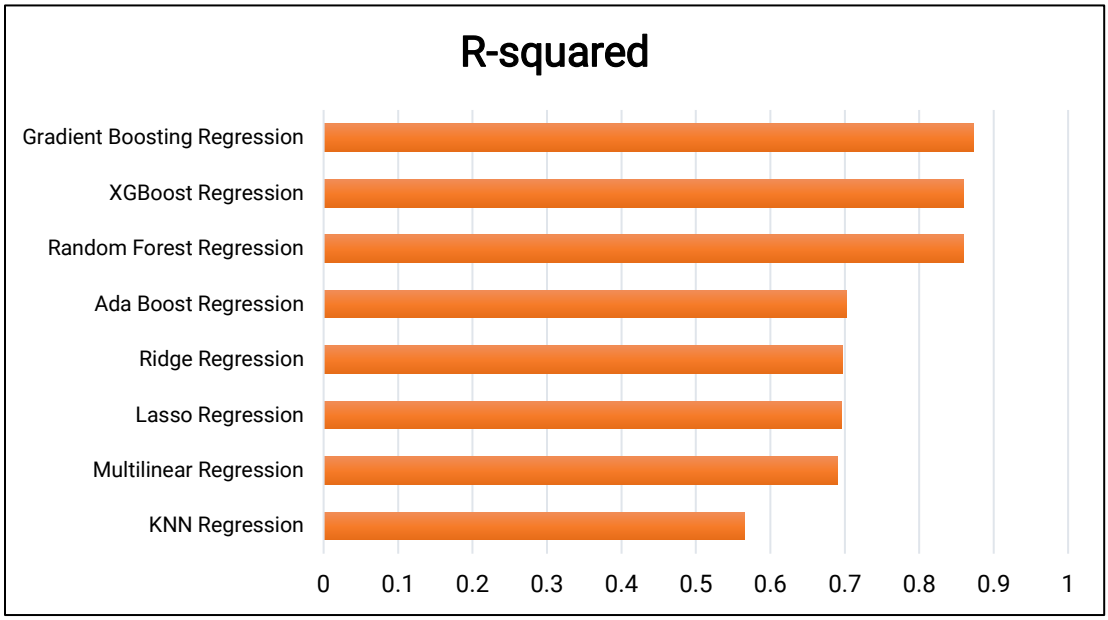


6. Models Summary

The table below summarizes the models with key metrics to assess their performance.

Sr No	Model	R-squared	MSE	RMSE	MAE
1	KNN Regression	0.5657	66930181087.17	258708.67	101363.33
2	Multilinear Regression	0.6903	47720957383.22	218451.27	88028.51
3	Lasso Regression	0.6962	46814515670.86	216366.62	88804.28
4	Ridge Regression	0.6963	46805178564.94	216345.04	88970.50
5	Ada Boost Regression	0.7021	45905830578.73	214256.45	104469.29
6	Random Forest Regression	0.8599	21591546588.24	146940.62	40131.97
7	XGBoost Regression	0.8599	21587066059.22	146925.37	40945.96
8	Gradient Boosting Regression	0.8724	19657214243.94	140204.19	40727.95

Table 1: Summary of models' performance on the test data



7. Final Conclusion & Recommendations

Some of the interpretations are as follows:

- The price will increase every unit increase in sqft_living, no. of bathrooms.
- The price was found to decrease with every unit increase in no. of bedrooms.
- As the latitude increase with the location moving from South to North; price increases. It decreases with increase in longitude as the location moves from West to East.
- For houses with waterfront, the price is higher.
- For houses with grade greater than 8, condition 4 or 5 and View 1 to 4, price is higher.
- The prices were found to be higher in the months of March, April and May. This can be attributed to the seasonality impact.
- Older houses tend to have relatively higher prices.

8. Important Links

Here are the links to the Notebook, Dataset and Report uploaded to a GitHub Repository. Please do check them out!

- GitHub Repository Main Page Link