PROJECT REPORT

HOUSE PRICE PREDICTION

Using Multiple regression,Decision Tree and Random Forrest

By,

Vilas R

Sai kartheek

Sakshya M M

Monish B Jain

SUMMARY

Finding the price of a house is more than just the site value and how big it is. In this project we make use of 19 parameters to determine the price of similar houses using machine learning algorithms. We used a total of 21000 data items to train the model to achieve this goal.

In order to find the best model for this we tried out three algorithms such as Decision tree, Random forest and multiple linear regression, and based on this we found the best model for this kind of data.

We found out that Multiple linear regression paired with Gradient Boosting regressor gave the most accurate pricing at 89.5%, while Random forest and Decision tree gave a accuray of 85% and 71% respectively.

By this we can imply that for problems with variable parameters which affect the result both positively and negatively Multiple Linear regression with Gradient boosting regressor is the best algorithm to make use of.

OVERVIEW

Since we had a lot of options of algorithms to choose from we decided to choose three which seemed the most relevant and optimal for this problem.

This led us to use the three above mentioned algorithms and compared the results.

The main code is written in python because that's the language our team is most familiar with and its flexibility when compared to other languages.

First we separated data into Dependable variables which was "price" and independent variables "room_bed","condition","lat" to name a few.

Then since the Independent variables are all in different units and scales we used a scaling function to standardise the features by subtracting the mean and then scaling to unit variance.

We plotted multiple graphs against price to determine the dependency using various python modules. Next we divided the given data into train and test parts by using train_test_split which randomly splits the data, for this we split 80% of data for training and 20% for testing, compared to 90/10% as to avoid overfitting.

The three algorithms used were Multiple linear Regression, Decision tree and Random forest , we used multiple linear regression combined with gradient Boosting regressor which gives a prediction model in the form of ensemble of weak prediction models, this usually outperforms random forest which is why we implemented the following algorithm in our problem solution.

STEP BY SOLUTION

We start by importing all the required modules and libraries for this project and importing the data into the main file

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import explained_variance_score
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
from sklearn import ensemble
from sklearn import linear_model

dataset = pd.read_csv("/content/drive/MyDrive/hackathon 26-11/innercity.csv")
```

We later defined our variables as x,y where y was the “price” and x rest of the parameters.

We don't require the column cid and dayhours as they are in string format and don't affect the price in any form, so we will be dropping them from the dataset.

We use data(info) to check if there are any null values and we use data cleaning to remove the null values which in this case there were no null values so we skipped the process

```
dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
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
```

We then used describe() to make general implications on the data and see what we were working on and later going to see some visualization to see

how and what we can infer from visualization, which has been explained in the visualization part of the report.

```
dataset.describe()
```

	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition	quality	ceil_measure
count	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
mean	5.401822e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691
std	3.673622e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978
min	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000

We use train data and test data , train data to train our machine and test data to see if it has learnt the data well or not.

Instead of using the traditional method of splitting the first 80 or 90% of the data for training and 10/20% for testing, we made use of the train_test_split function with which you don't need to divide data set manually.By default sklearn train_test_split will make random partitions for the two subsets and this leads to better accuracy of the model.

```
X = dataset.iloc[:,1:].values
y = dataset.iloc[:,0].values

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

Decision Tree algorithm

We implemented the decision tree algorithm defined in sklearn and got the accuracy of the model.

```
tr_regressor = DecisionTreeRegressor(random_state=0)
tr_regressor.fit(X_train,y_train)
tr_regressor.score(X_test,y_test)
pred_tr = tr_regressor.predict(X_test)
decision_score=tr_regressor.score(X_test,y_test)
expl_tr = explained_variance_score(pred_tr,y_test)
tr_regressor.score(X_test,y_test)
```

We got a very low accuracy from this model therefore we decided to try more models to get better accuracy which led us to use the following models

Random Forest

This gave a better accuracy than the Decision tree model but we still weren't satisfied with the results.

```
rf_regressor = RandomForestRegressor(n_estimators=28,random_state=0)
rf_regressor.fit(X_train,y_train)
rf_regressor.score(X_test,y_test)
rf_pred =rf_regressor.predict(X_test)
rf_score=rf_regressor.score(X_test,y_test)
expl_rf = explained_variance_score(rf_pred,y_test)
rf_regressor.score(X_test,y_test)
```

```
0.8481266628220434
```

Multiple Linear Regression combined with Gradient boosting regressor

At first even this model gave a very low accuracy but with the help of gradient boosting regressor we were able to obtain the most accurate result compared to other models used.

```
[29] regr = linear_model.LinearRegression()
     a = dataset[["room_bed","room_bath","living_measure","lot_measure","ceil","coast","sight","condition",
     b = dataset[["price"]]
     sx = scale.fit_transform(a)
     a_train , a_test , b_train , b_test = train_test_split(sx, b , test_size = 0.10,random_state =2)
     regr.fit(a_train,b_train)
     regr.score(a_test,b_test)

     0.7137651604703148
```

Here we define the boosted model

```
bstregr = ensemble.GradientBoostingRegressor(n_estimators = 500, max_depth = 5, min_samples_split = 2,
bstregr.fit(a_train,b_train)
bstregr.score(a_test,b_test)

/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConversionWarning: A column-vec
  y = column_or_1d(y, warn=True)
0.8916249685365928
```

Model Evaluation and Comparison

At beginning we weren't sure about which models or algorithm we should implement for our project. And after some research we decided to go with three models including Random forest and after that decision tree and at last we tried out Multiple linear regression. We wanted to get the best accuracy and results for our project and after comparing these 3 models we concluded that Multiple Linear Regression gave out the most accurate results.

```
print("Decision tree  Regression Model Score is: ",round(tr_regressor.score(X_test,y_test)*100))
print("Random Forest Regression Model Score is: ",round(rf_regressor.score(X_test,y_test)*100))
print("Multiple Linear Regression Model Score is: ",round(bstregr.score(a_test,b_test)*100))
```

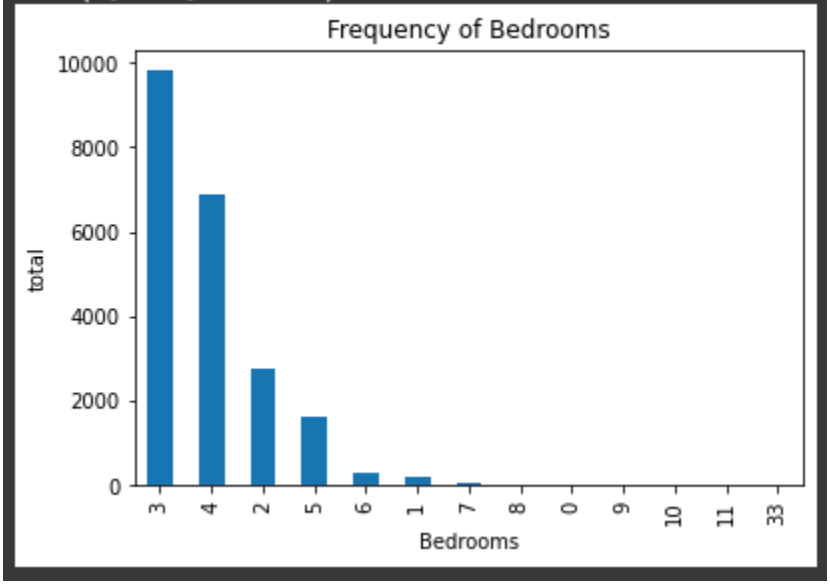
```
Decision tree  Regression Model Score is:  71
Random Forest Regression Model Score is:  85
Multiple Linear Regression Model Score is:  89
```

From the above it is clear that random forest accuracy is 88% and also explained variance score is 0.84 . So Random Forest is a suitable model for predicting the price of the house.

Though there remains other regression model which can bring out the best of the dataset.

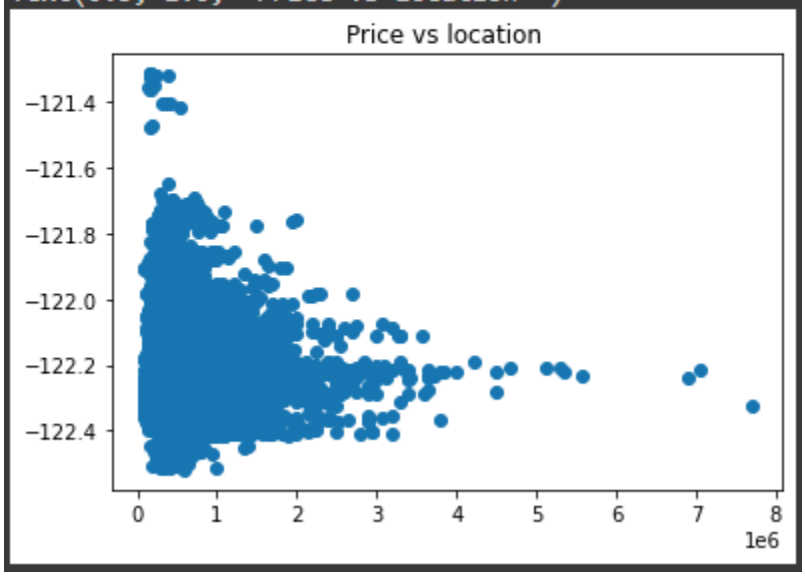
Visualisation

```
Text(0, 0.5, 'total')
```



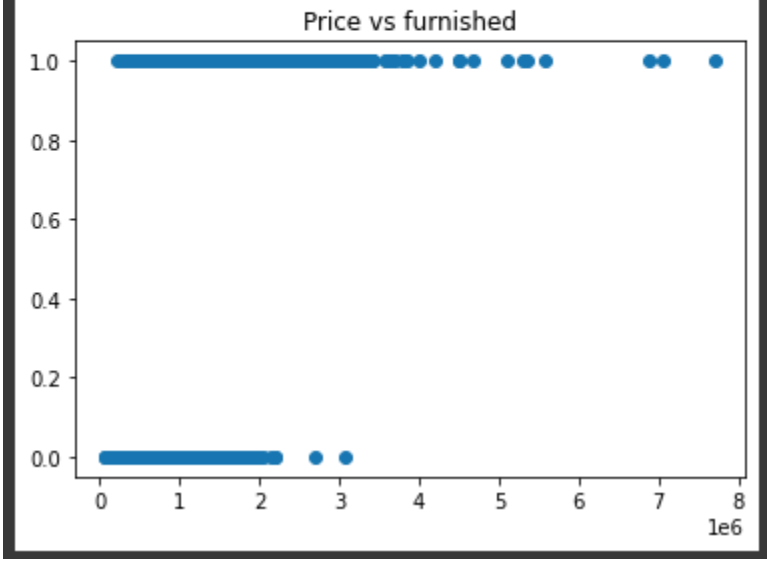

```
plt.scatter(dataset.price,dataset.long)
plt.title("Price vs location ")
```

```
Text(0.5, 1.0, 'Price vs location ')
```



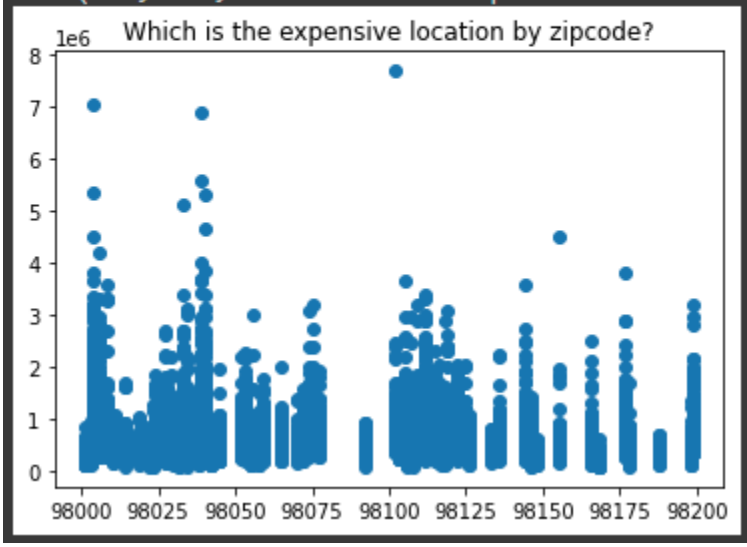
```
plt.scatter(dataset.price,dataset.furnished)
plt.title("Price vs furnished")
```

```
Text(0.5, 1.0, 'Price vs furnished')
```




```
plt.scatter(dataset.zipcode,dataset.price)
plt.title("Which is the expensive location by zipcode?")
```

```
Text(0.5, 1.0, 'Which is the expensive location by zipcode?')
```

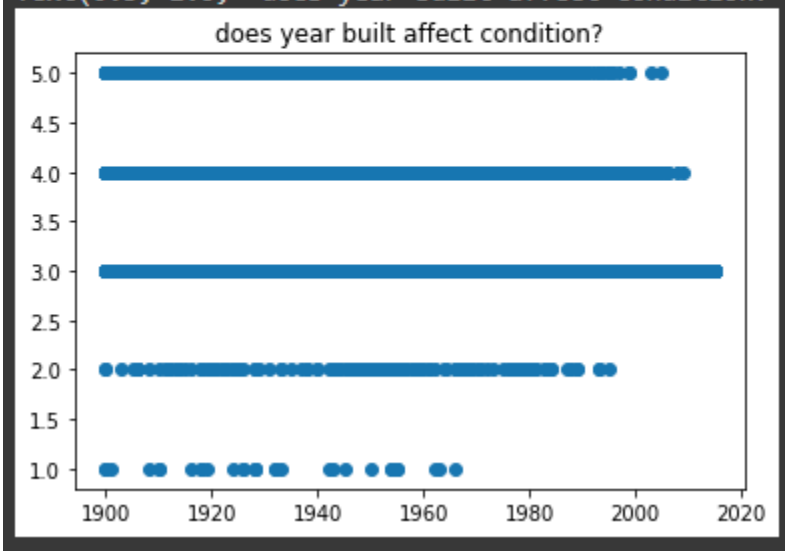


```
splt.scatter(dataset.price,dataset.quality)
plt.title("How does price vary with quality?")
```

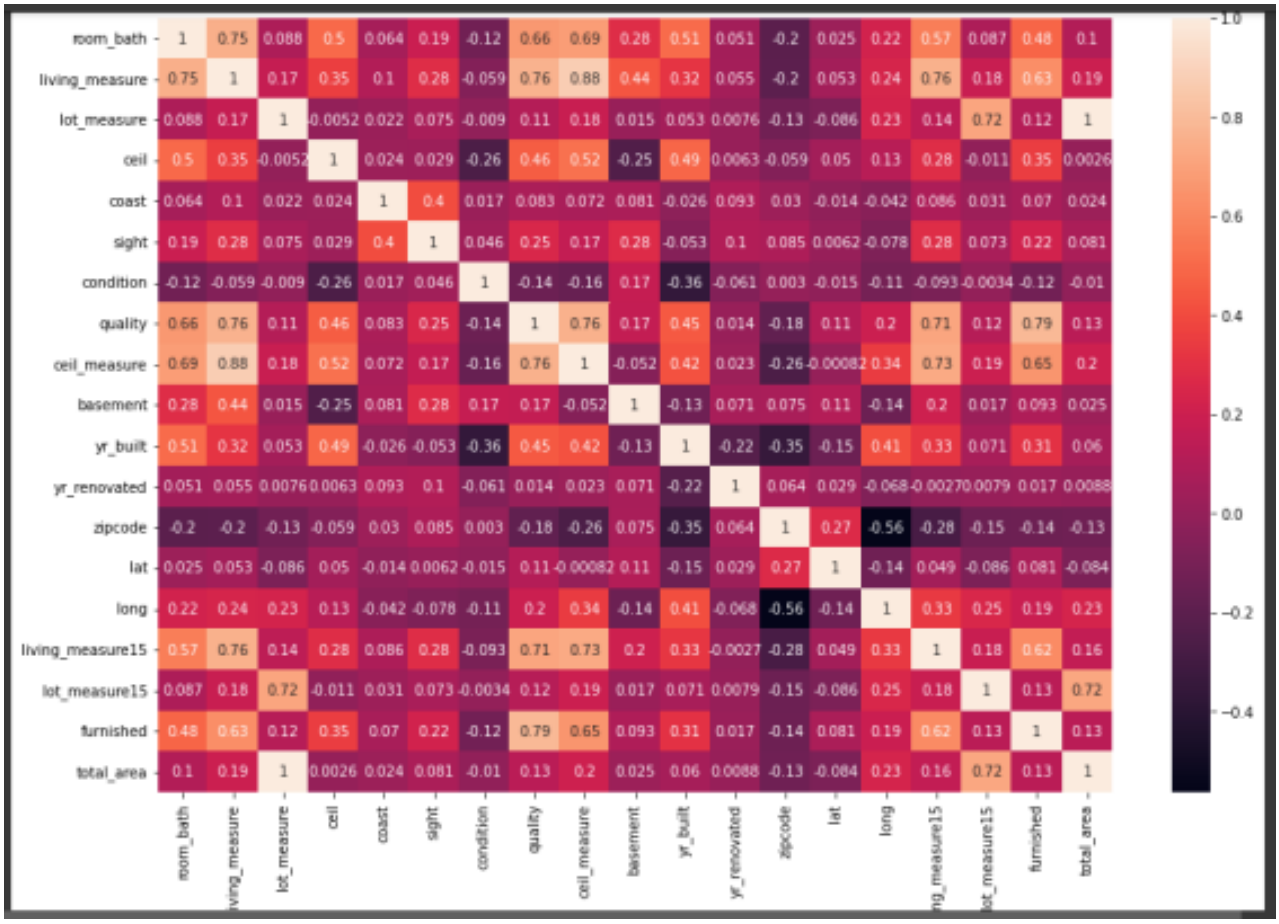
```
Text(0.5, 1.0, 'How does price vary with quality?')
```


```
plt.scatter(dataset.yr_built,dataset.condition)
plt.title("does year built affect condition?")
```

```
Text(0.5, 1.0, 'does year built affect condition?')
```



We will use heatmap to view the co relation between variables.



Implications

House price forecasting is an important topic of real estate. Machine learning techniques are applied to analyze historical property transactions in India to discover useful models for house buyers and sellers. This attempts to reveal the high discrepancy between house prices in the most expensive and most affordable suburbs in the cities.Moreover, experiments demonstrate that the Multiple Linear Regression that is based on mean squared error measurement is a competitive approach

Limitations

The collection of data will be a difficult task and the data collected from historical dataset transactions are not very accurate and hence not reliable.

For newly developed areas we can't get historical dataset to train models.

Closing Reflections

This study employs machine learning techniques to develop a price prediction model for housing problems. It uses a rather large publicly available dataset of historical real estate transactions . The regression model performances of the models are compared with one another and with the benchmark model. The empirical results show that the multiple regression model is the most accurate of all models selected. The developed model may facilitate the prediction of future housing prices and the establishment of policies for the real estate market. Particularly, the sellers and buyers of properties can benefit from this study and make better-informed decisions regarding the property evaluation. In addition, property agents can focus on the seasonality effects, especially during the summer season, when most of the people buy their properties, and on the clear preference for two- or three-bedroom properties. The financial organizations and mortgage lenders may also find the study beneficial and identify more accurate real estate property value, risk analysis, and lending decisions. The study can be enlarged in a subsequent research by increasing the dataset size so potentially uncovered details and features of the dataset and of this study can be addressed. An increased dataset would potentially be good enough for employing deep neural networks, which can assure that more in-depth analysis on the housing price prediction can be performed. Then, the enlarged housing price prediction problem can be tackled as a classification problem.