House Price Predictor

Team 7
Jaren, Louden, Keng, Kevin

Goal:

The goal of our project is to predict the price of houses in India depending on factors such as:

- Posted by (Owner, Dealer)
- Under construction
- RERA (Real Estate Regulatory Authority) approved
- Number of rooms (BHK)
- Type (house or apartment)
- Square feet
- Ready to move
- Resale
- Address
- Longitude
- Latitude

Challenges:

The challenges we faced during this project:

- Outlier removal and how to handle preprocessing.
- The use of the address attribute (there were simply too many unique observations for it to be a useful predictor).
- It is very difficult to map out all the locations on a tree to predict the price.
- Looking through many potential algorithms and deciding which one to use.
- The large data sets had some issues when running through RapidMiner.
- Our dataset had a bad feature-to-observation ratio, with 29451 observations in the training dataset and 68720 observations in the test dataset. The training data set had 12 features while the test data set had 11 features.
- We had predictor variables with perfect multicollinearity, meaning two or more variables shared the same pattern and their effect on the dependent variables was imperceptible.

Our Data:

What we had to work with:

- Train.csv [29451 Observations, 12 Features]
- Test.csv [68720 Observations, 11 Features]

Sample of training data set:

	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT	READY_TO_MOVE	RESALE	ADDRESS	LONGITUDE	LATITUDE	TARGET.PRICE_IN_LACS.
1	Owner	0	0	2	BHK	1300.2364070	1	1	Ksfc Layout,Bangalore	12.969910	77.5979600	55.0
2	Dealer	0	0	2	BHK	1275.0000000	1	1	Vishweshwara Nagar,Mysore	12.274538	76.6446050	51.0
3	Owner	0	0	2	BHK	933.1597222	1	1	Jigani,Bangalore	12.778033	77.6321910	43.0
4	Owner	0	1	2	BHK	929.9211427	1	1	Sector-1 Vaishali,Ghaziabad	28.642300	77.3445000	62.5
5	Dealer	1	0	2	BHK	999.0092470	0	1	New Town,Kolkata	22.592200	88.4849110	60.5
6	Owner	0	0	3	BHK	1250.0000000	1	1	South Chittoor,Kochi	10.033280	76.2825710	42.0
7	Dealer	0	0	3	BHK	1495.0539570	1	1	Sodala,Jaipur	26.916347	75.7956000	66.5
8	Owner	0	1	3	BHK	1181.0129460	1	1	Kharar,Mohali	30.740000	76.6500000	52.0
9	Dealer	0	1	2	BHK	1040.0000000	1	1	Bileshivale,Bangalore	13.054202	77.6740020	41.6
10	Owner	0	1	2	BHK	879.1208791	1	1	Chromepet,Chennai	12.951610	80.1409700	36.0
11	Owner	0	0	3	BHK	1350.3086420	1	1	Deshbandhu Para,Siliguri	26.724219	88.3263570	35.0
12	Dealer	0	0	2	BHK	1333.0101790	1	1	Hebbal,Bangalore	13.040340	77.5913470	110.0
13	Owner	0	0	2	BHK	927.1779023	1	1	Garebhavipalya,Bangalore	12.969910	77.5979600	48.0
14	Owner	0	1	2	BHK	1122.1719460	1	1	Sector-119 Noida,Noida	28.587711	77.4031230	62.0
15	Owner	0	0	1	BHK	649.9837504	1	1	sanjay nagar,Raigad	13.035200	77.5772000	20.0
16	Dealer	1	1	3	BHK	1394.1176470	0	1	Sector-150 Noida,Noida	28.429614	77.4817110	71.1
17	Owner	0	0	3	BHK	1800.0847100	1	1	Jharapada,Bhubaneswar	20.275267	85.8624070	85.0
18	Dealer	1	1	3	BHK	2124.8967060	0	1	Konanakunte,Bangalore	12.885500	77.5638000	180.0
19	Owner	0	0	2	BHK	1100.0000000	1	1	Nagpur Road,Wardha	21.153890	79.0830600	22.0
20	Dealer	0	1	3	BHK	2178.6492370	1	1	Kogilu,Bangalore	13.092356	77.6134680	120.0

Libraries Used:

- rpart - builds classification or regression models
- rpart.plot and ggplot2 - map variables to visualization of plots
- Fselector - attribute subset selection (information gain)
- caret - functions for training and plotting classification and regression models
- dplyr - data manipulation
- party - train model using decision tree
- Amelia - multivariate incomplete data
- dials - create and manage values of tuning parameters

Other Things Considered:

- We considered using the KNN algorithm we used to complete Homework 2 assignment to create a prediction.

Preprocessing

Data cleaning:

- Removing outliers, with specific focus on square feet and price

Data reduction:

- Removed address column
- Ideas: Both of these were scrapped as they yielded lower accuracy values
    - We had also considered removing latitude and longitude (either both or just one) to see if we had multicollinearity issues here.
    - We had also considered removing square feet to reduce the complexity of the plot and because we were unsure of its use as a predictor

The Problem With Using Address

More Locations

When taking in each location an the price there was high information gain, but there were also locations in the test set that were not in the training set.

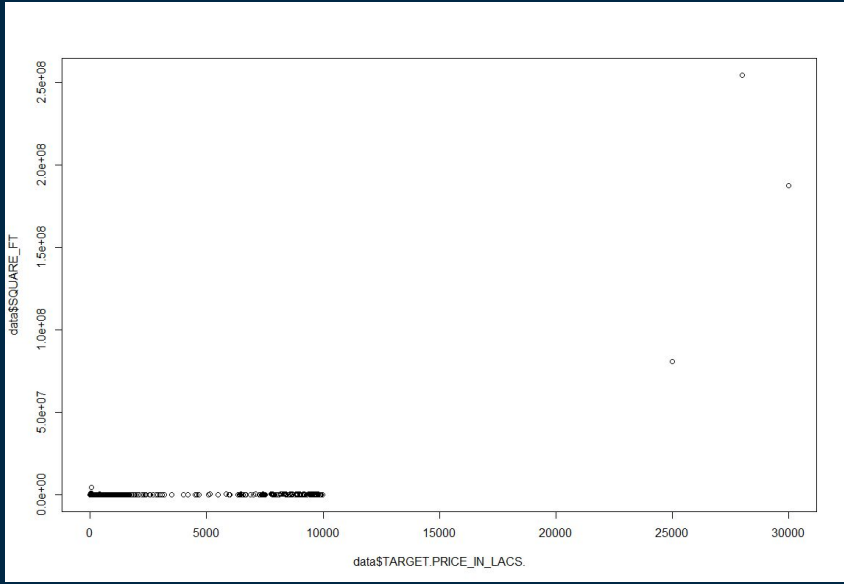
Address was a bad predictor due to its lack of usability and unique non-numeric values

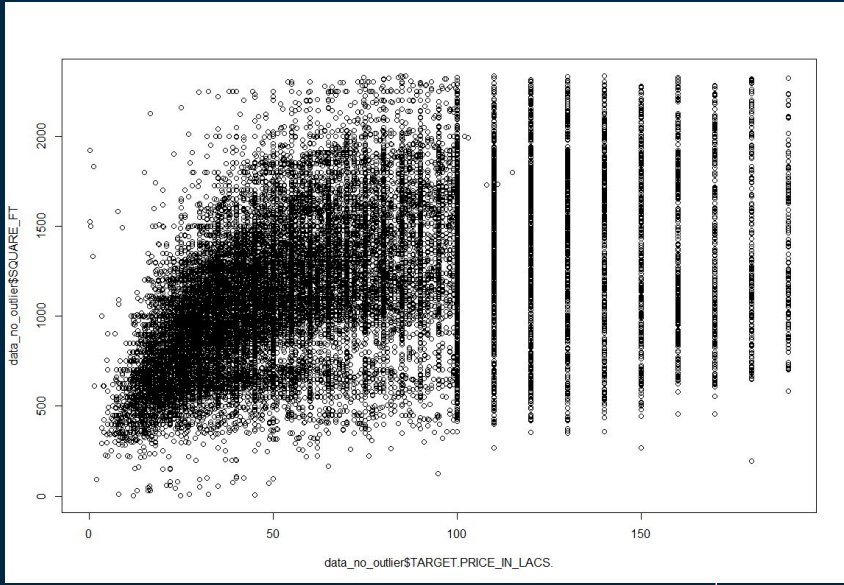
Results

When trying to create the tree using location the data within the tree was too big to take into account, so we removed address from the dataset entirely.

```
> information.gain(data)
                        attr_importance
UNDER_CONSTRUCTION      0.0382996080622
RERA                    0.0432583505984
BHK_NO.                 0.0089062483029
BHK_OR_RK               0.0004315323029
SQUARE_FT               0.0199802775425
READY_TO_MOVE           0.0382996080622
RESALE                  0.0772354222942
ADDRESS                 0.4211684547515
LONGITUDE               0.1601368152464
LATITUDE                0.1397330036632
TARGET.PRICE_IN_LACS.   0.1205277163631
is_outlier              0.0303238778748
```

Outlier Removal





How We Removed Them

```r
#get outliers from boxplot data
outliers <- boxplot(data$TARGET.PRICE_IN_LACS., plot = FALSE)$out

#add column to indicate of a row is an outlier
data$is_outlier <- ifelse(data$TARGET.PRICE_IN_LACS. %in% boxplot.stats(data$TARGET.PRICE_IN_LACS.)$out, 1, 0)

#remove rows with outliers
data_no_outlier <- subset(data,is_outlier==0)

#finds and removes sqft outliers
outliers_sqft <- boxplot(data_no_outlier$SQUARE_FT, plot = FALSE)$out
data_no_outlier$is_outlier <- ifelse(data_no_outlier$SQUARE_FT %in% boxplot.stats(data_no_outlier$SQUARE_FT)$out, 1, 0)
data_no_outlier <- subset(data_no_outlier,is_outlier==0)

#clean data
clean_data <- data_no_outlier[ , !names(data_no_outlier) %in% c("is_outlier", "ADDRESS")]
```

Data After Preprocessing

What we had to work with:

- Train.csv [29451 observations, 12 features], without outliers [25575 observations, 11 features]
- Test.csv [68720 observations, 11 features]

The data ...

	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT	READY_TO_MOVE	RESALE	LONGITUDE	LATITUDE	TARGET.PRICE_IN_LACS.
1	Owner	0	0	2	BHK	1300.2364070	1	1	12.96991000	77.59796000	55.0
2	Dealer	0	0	2	BHK	1275.0000000	1	1	12.27453800	76.64460500	51.0
3	Owner	0	0	2	BHK	933.1597222	1	1	12.77803300	77.63219100	43.0
4	Owner	0	1	2	BHK	929.9211427	1	1	28.64230000	77.34450000	62.5
5	Dealer	1	0	2	BHK	999.0092470	0	1	22.59220000	88.48491100	60.5
6	Owner	0	0	3	BHK	1250.0000000	1	1	10.03328000	76.28257100	42.0
7	Dealer	0	0	3	BHK	1495.0539570	1	1	26.91634700	75.79560000	66.5
8	Owner	0	1	3	BHK	1181.0129460	1	1	30.74000000	76.65000000	52.0
9	Dealer	0	1	2	BHK	1040.0000000	1	1	13.05420200	77.67400200	41.6
10	Owner	0	1	2	BHK	879.1208791	1	1	12.95161000	80.14097000	36.0
11	Owner	0	0	3	BHK	1350.3086420	1	1	26.72421900	88.32635700	35.0
12	Dealer	0	0	2	BHK	1333.0101790	1	1	13.04034000	77.59134700	110.0
13	Owner	0	0	2	BHK	927.1779023	1	1	12.96991000	77.59796000	48.0
14	Owner	0	1	2	BHK	1122.1719460	1	1	28.58771100	77.40312300	62.0
15	Owner	0	0	1	BHK	649.9837504	1	1	13.03520000	77.57720000	20.0
16	Dealer	1	1	3	BHK	1394.1176470	0	1	28.42961400	77.48171100	71.1
17	Owner	0	0	3	BHK	1800.0847100	1	1	20.27526700	85.86240700	85.0
18	Dealer	1	1	3	BHK	2124.8967060	0	1	12.88550000	77.56380000	180.0
19	Owner	0	0	2	BHK	1100.0000000	1	1	21.15389000	79.08306000	22.0
20	Dealer	0	1	3	BHK	2178.6492370	1	1	13.09235600	77.61346800	120.0
21	Owner	0	0	2	BHK	881.1435285	1	1	13.05000000	80.11000000	45.0

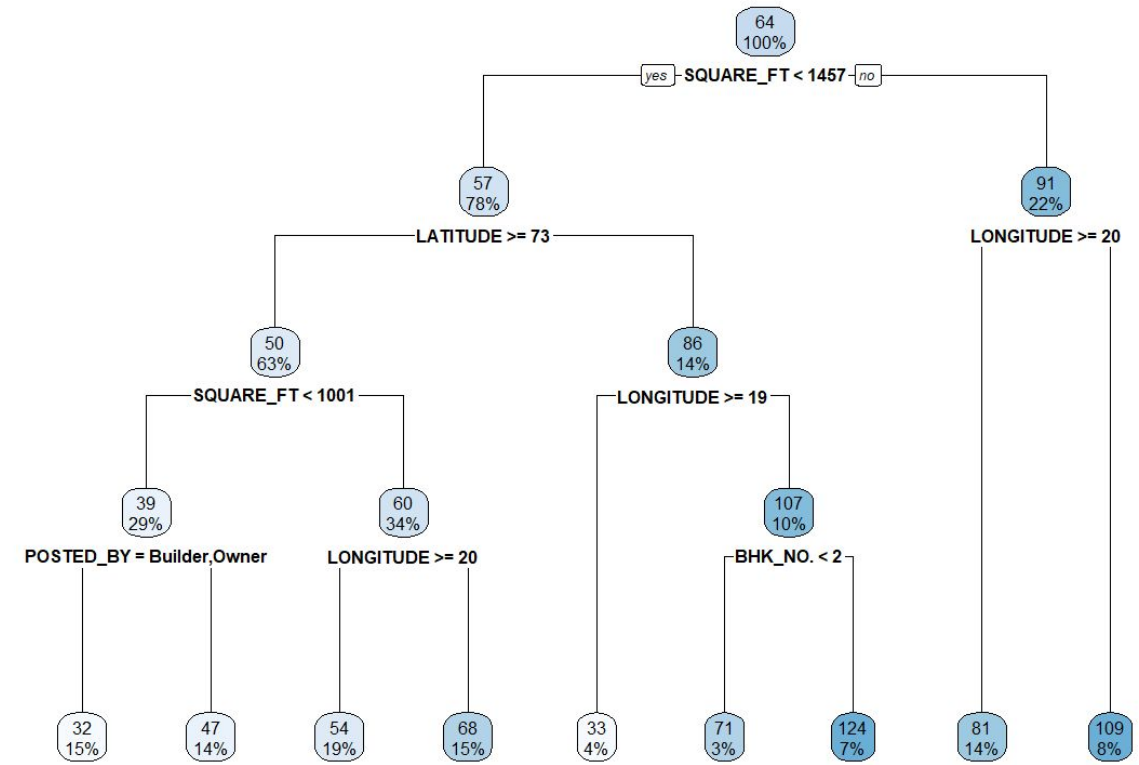
Making the Training Set and Test Set

The actual "test set" that came with our training set had no values for target price to compare our model with, so we opted for the Holdout method and split our original training set into 80% train, 20% test.

```
create_train_test <- function(data, size = 0.8, train = TRUE) {
  n_row = nrow(data)
  total_row = size * n_row
  train_sample <- 1: total_row
  if (train == TRUE) {
    return (data[train_sample, ])
  } else {
    return (data[-train_sample, ])
  }
}
```

```
#create test and train out of accurate data
data_train <- create_train_test(clean_data, 0.8, train = TRUE)
data_test <- create_train_test(clean_data, 0.8, train = FALSE)
```

Our Solution: Regression Tree!




```
#create the basic decision tree
tree <- rpart(TARGET.PRICE_IN_LACS. ~., data=data_train, method = 'anova')
rpart.plot(tree)
```

```
> print("Accuracy for tree 1, no settings tweaks: ")
[1] "Accuracy for tree 1, no settings tweaks: "
> print(acc)
[1] 0.4443792766
```

```
> print("Accuracy for training set: ")
[1] "Accuracy for training set: "
> print(acc)
[1] 0.4575268817
```

Determining Accuracy

```
inc <- function(x) {
  eval.parent(substitute(x <- x + 1))
}
```

```
#compare predictions with actual
counter <- 0
for(i in 1:dim(data_test)[1]) {
  if(data_test$PREDICTED.PRICE_IN_LACS.[i] <= (data_test$TARGET.PRICE_IN_LACS.[i] + 15) & data_test$PREDICTED.PRICE_IN_LACS.[i] >= (data_test$TARGET.PRICE_IN_LACS.[i] - 15)) {
    inc(counter)
  }
}
```

```
#get accuracy
acc <- counter/nrow(data_test)
```

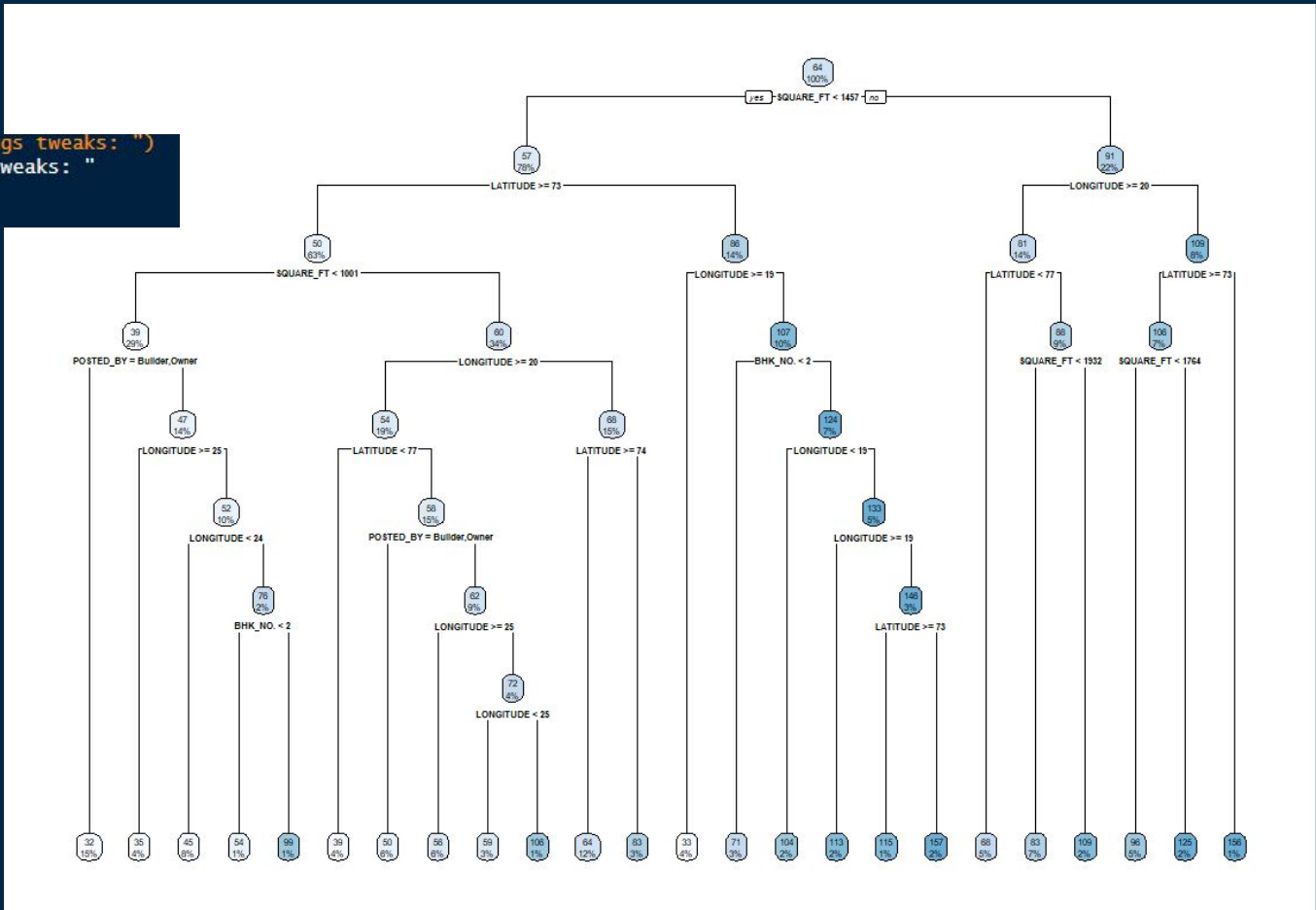
Changing CP Value

CP = Complexity parameter, saves computing time by pruning off splits that are obviously not worthwhile. Essentially, the user informs the program that any split which does not improve the fit by ‘cp’ will likely be pruned off by cross-validation, and that hence the program need not pursue it.
From our testing, changing the other values, like max depth and min buckets, did not change the tree enough to actually help the accuracy.

```
#tweak tree values
control <- rpart.control(cp = 0.005)
```

Second Tree

```
> print("Accuracy for tree 2, with settings tweaks: ")
[1] "Accuracy for tree 2, with settings tweaks: "
> print(acc)
[1] 0.5112414467
```



Random Forest

PREDICTED.PRICE_IN_LACS.
16.66708
73.60660
67.38003
46.69424
14.35676
15.75401
53.72335
80.34434
40.10073
135.53892
112.14196
92.94567
95.51399
56.55541
50.01430
95.84984
67.88688
19.54026
53.74343
96.08037
62.11572

```
Random Forest

25575 samples
   10 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 20458, 20461, 20460, 20460, 20461
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared   MAE
   2    24.87075  0.6381165  18.37184
   6    19.21336  0.7597914  12.72764
  11    19.48986  0.7528482  12.79744

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 6.
```

mtry = number of predictors that will be randomly sampled at each split
Rsquared = how much variation of a dependent variable is explained by the independent variable(s)

Random Forest Results Explained

mtry	RMSE	Rsquared	MAE
2	24.87075	0.6381165	18.37184
6	19.21336	0.7597914	12.72764
11	19.48986	0.7528482	12.79744

A mid-range mtry (randomly sampled predictors at each split) led to the best results for RMSE, Rsquared, and MAE

- For **RMSE** (root-mean-square error), the values sit around a somewhat low range. Though, it would have been preferable to have lower values since the range of prices is 6.79 LACS to 186.81 LACS.
- For **Rsquared** (AKA, coefficient of determination), this is a percentage indicating how well the predictors predicted the dependent variable. Sitting at around 76% on its best sample.
- For **MAE**, it isn't too dissimilar from RSME. Mean Absolute Error measures the average absolute difference between the predicted and actual values in the dataset.
    - RSME has a tendency to be swayed heavily by a single (or a few) major errors, so MAE would be a better indication of accuracy for this model.

*Note: LACS is equivalent to 100,000 in the Indian numbering system

Why Did We Use a Random Forest?

- The accuracy of a standard regression tree is not the greatest due to restrictions of the model alone. The restriction that got us was its sensitivity to overfitting.
    - We have a lot of features with unique values for each observation. So much so, that our tree ends up being gigantic, especially without modification of the complexity parameter.
- Random forests are essentially just a bunch of regression trees pooled together into one final set of predictions.
- They have every single benefit that we need:
    - Works well with nonlinear data
    - Has a lower risk of overfitting
    - Runs efficiently on a large dataset
    - Better accuracy than a majority of other classification algorithms
    - Handles any outliers we couldn't catch numerically
- We also can use it parallel to k-fold cross-validation, which can allow us to handle the overfitting issues even more effectively.

Conclusions:

What we learned from the project:

Regression Trees:

- Using a Regression Tree isn't the most optimal way of predicting values.
- Regression Trees can produce large trees, which reduces interpretability
- Regression Tree is a Basic algorithm is reasonably simple to implement
- Splitting criteria tend to be sensitive, and small changes in the data can result in large changes in the tree structure

Predicting house prices are hard to do due to many different factors such as locations and addresses.

Random Forests:

- Takes the simplicity of regression trees and boosts the prediction accuracy

RESOURCES

[Dataset]: https://www.kaggle.com/datasets/anmolkumar/house-price-prediction-challenge

[Decision tree tutorial]: https://www.guru99.com/r-decision-trees.html

[Handling Rank-Deficient Data]:
https://www.statology.org/prediction-from-rank-deficient-fit-may-be-misleading/

[R package documentation]: https://www.rdocumentation.org/