INDIA REAL ESTATE APPRAISAL USING MACHINE LEARNING

A PREPRINT

Agata Kopyt
Faculty of Mathematics and Information Science
Warsaw University of Technology
Warsaw, Poland

Zuzanna Kotlińska
Faculty of Mathematics and Information Science
Warsaw University of Technology
Warsaw, Poland

Szymon Matuszewski
Faculty of Mathematics and Information Science
Warsaw University of Technology
Warsaw, Poland

Patryk Słowakiewicz
Faculty of Mathematics and Information Science
Warsaw University of Technology
Warsaw, Poland

June 4, 2022

ABSTRACT

Nowadays machine learning is a really important area of computational science. It focuses on analysing and interpreting patterns and structures in data to enable learning and decision making outside of human interaction. Such techniques can be especially used for real estate appraisal. This paper focuses on predicting property prices in Indian metropolitan areas using four tree-based models: Random Forest, XGBoost, Catboost and LightGBM. Firstly, we perform exploratory data analysis to examine the most important variable distributions and interesting relations between columns. Our next step is modeling and hyperparameter tuning and then choosing top three models based on their RMSE scores. Finally, we analyse these models using explainable artificial intelligence methods, such as creating Break Down and Ceteris Paribus profiles. We also measure the importance of features and their contribution to predicted price value.

Keywords Machine Learning · Real Estate Appraisal · India · Tree-Based Models

1 Introduction

Data is the lifeblood of all business. Every decision based on data analysis is crucial - it can either make company successful or make it fall further behind the competition. Machine learning algorithms are helping industries like financial services or healthcare to gain more customers, improve their experience and reduce costs. With greater access to data and computation power, machine learning is becoming more and more present in our lives. The most popular example is products or services recommendations. We face it everyday, while choosing a movie to watch online, listening to music on streaming services or online shopping.

This paper describes another important usage of machine learning - real estate appraisal. This powerful tool can help to predict property prices based on various features, such as area, location, number of bedrooms or nearby facilities. For our prediction we concentrate on tree-based models: Random Forest, XGBoost, Catboost and LightGBM to perform the regression task. Each of them has different advantages and disadvantages and can be suitable for different purpose. This is why we create a considerable number of experimental models with and without optimal hyperparameters to choose the best five for each algorithm. To compare these models of regression of prices in India's metropolitan area we used mainly Root Mean Square Error (RMSE). After the analysis, we can choose the best three models which are going to be furtherly examined using explainable AI methods (XAI). For this task we are using DALEX package to

make visualisations that can help us understand the variables' impact on the final price prediction. For example, we can analyse price breakdown for particular observation, measure feature importance and examine partial dependence profile for our most important variable - *Area*.

2 Data

The data we have analysed originates from Kaggle and concerns properties in six Indian cities – Bangalore, Chennai, Delhi, Kolkata, Hyderabad and Mumbai. It contains information about their prices, as well as other factors (location, area, number of bedrooms, among others) that affect them. The data set (all six data frames combined) consists of 32963 rows and 41 columns.

Firstly, we checked some basic information – the type of our data and the number of NaN values. The majority of the variables is numeric type, which is certainly conductive to further analysis. At first glance, we have encountered about 7000 of NaN values in all columns except for *Price*, *Area*, *Location*, *No. of Bedrooms* and *Resale* which have no NaNs. However, in the dataset description there is an information that such values are also marked as '9' so taking that into consideration it turns out that the majority of columns have as many as 22870 missing data. For continuous variables such as *Price*, *Area* and *Location* there are 4924, 2452 and 1776 unique records.

We looked at what the distributions of the continuous variables contained in our dataset, such as *Price* and *Area*, look like. As they appear to be exponential they may need to be transposed.

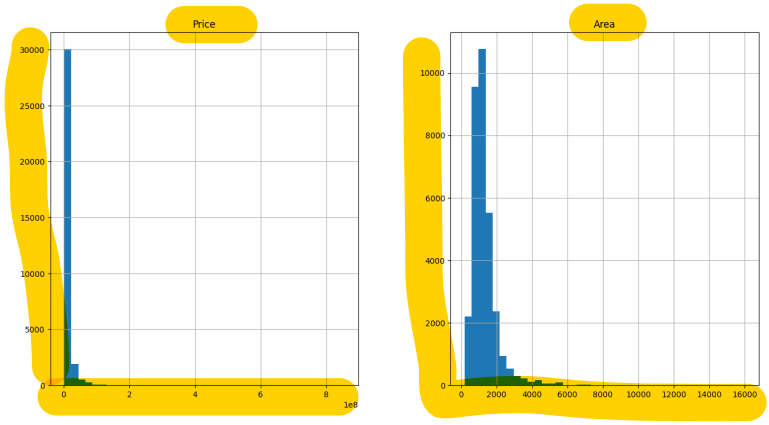
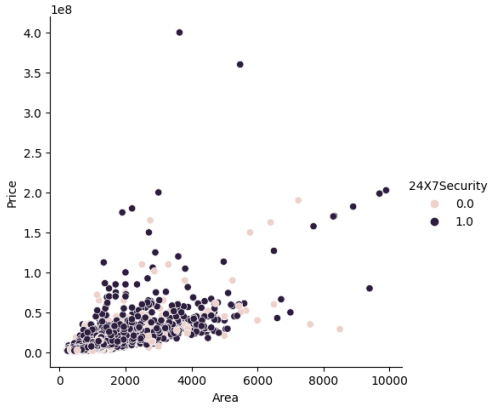


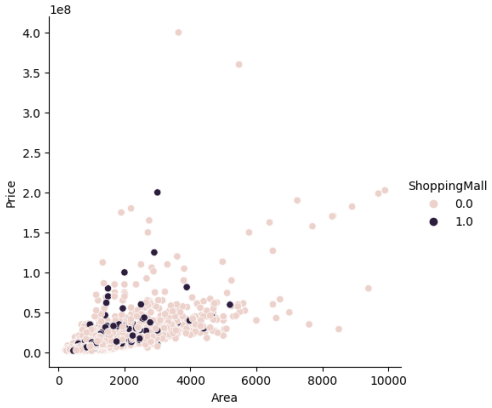
Figure 1: Price and Area distribution

We have also checked Cramer V correlation between our variables (firstly we have grouped *Price* and *Area* into categories and created *PriceGroup* and *AreaGroup* variables) and searched for strong correlations (>0.75) so we could drop redundant columns. There is of course a really strong correlation (nearly 1.00) between *Location* and *City* - but it is no surprise as every city has generally its unique district names. Of course in our analysis we want to concentrate mainly on price so we have checked correlations for *PriceGroup* but there don't seem to be any. Finally, we have checked some most important variable distributions and interesting relations between columns. Here are our conclusions:

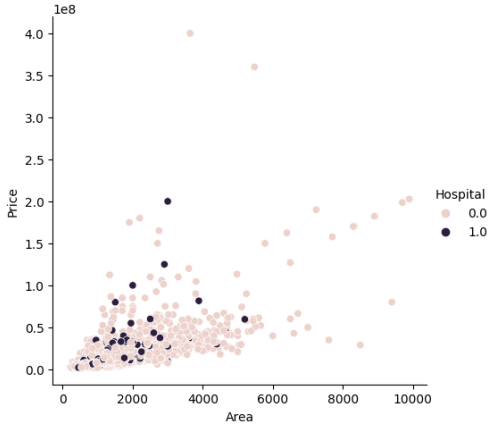
- Top property locations in India are Noida (Delhi), New Town (Kolkata) and Khargar (Mumbai). It's worth noticing that Noida is a whole city (in our dataset Delhi is a union territory), so that's probably why it has the biggest number of properties comparing to other locations which are mainly just districts.
- The most common size of the property is about 1600 sq feet (150 sq metres).
- Most of the properties in India have 24/7 security.
- The biggest (and generally most expensive) properties are far from public facilities like shopping mall or hospital (as they are probably located on the suburbs, where plots are bigger).
- The majority of properties have no gas connection.
- Indias's most expensive areas are Sunder Nagar and Marsarovar garden (both in Delhi) and MG Road (Mumbai). The leader is Sunder Nagar with average house prices over 800000000.0 INR (10M EUR!). This small city is called the ultimate address of the uber-rich in Delhi.



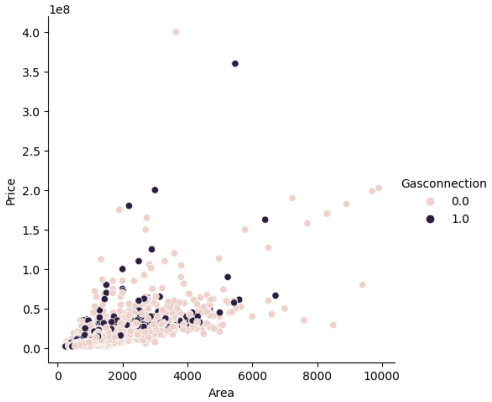
(a) 24x7 security



(b) Shopping mall access



(c) Hospital access



(d) Gas connection

Figure 2: Relation between various variables

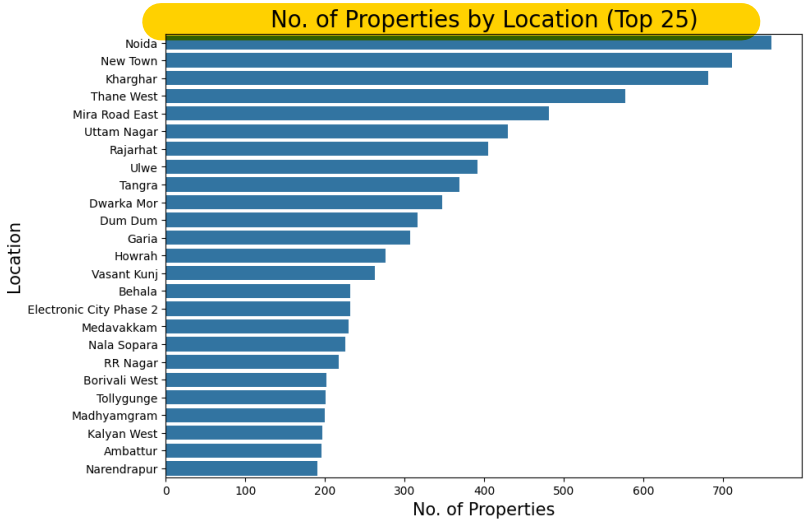


Figure 3: Top property Locations in India

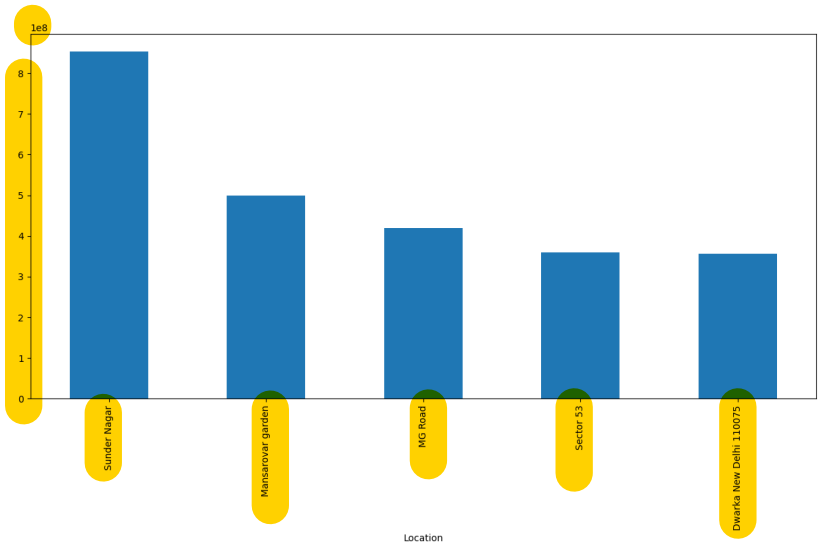


Figure 4: India's most expensive areas

For business purposes we decided to create our models only for one of six data frames described earlier - Bangalore. As housing prices and standards in every Indian district differ, the best solution is to create a separate prediction model for each of India's metropolitan cities. The model we are going to create for Bangalore will work for other data sets as well. Therefore, below we presented some highlights of Bangalore exploratory data analysis.

Bangalore data set consists of 6207 rows and 40 columns. Apart from columns such as *Price*, *Area*, *Location*, *No. of Bedrooms* and *Resale* there are 4256 NaN values for every other variable.

- Top property locations in Bangalore are Electronic City Phase 2, RR Nagar and Begur with over 150 properties located in each of them.
- The most common size of the property is 1445 sq feet (approx 133 sq metres). Properties over 3960 sq feet (huge - 365 sq metres) and below 500 sq feet are rare to find.
- The majority of houses costs less than 15000000.0 INR (178k EUR).
- It goes without saying that the bigger the area, the higher the price and number of bedrooms. Surprisingly, there happen to be properties with big footage and the highest number of bedrooms that cost less than average. On the other hand, there are also expensive, small houses with one or two bedrooms.
- Properties located close to sports facilities, gymnasium or jogging track tend to be more expensive. There are not many properties near school or shopping mall but if there are any, they are usually smaller (which implies lower price).
- Bangalore's most expensive areas are Richmond Town, Bannerghatta Road Jigani and Kalkere with average property prices higher than 55000000.0 INR (653k EUR).

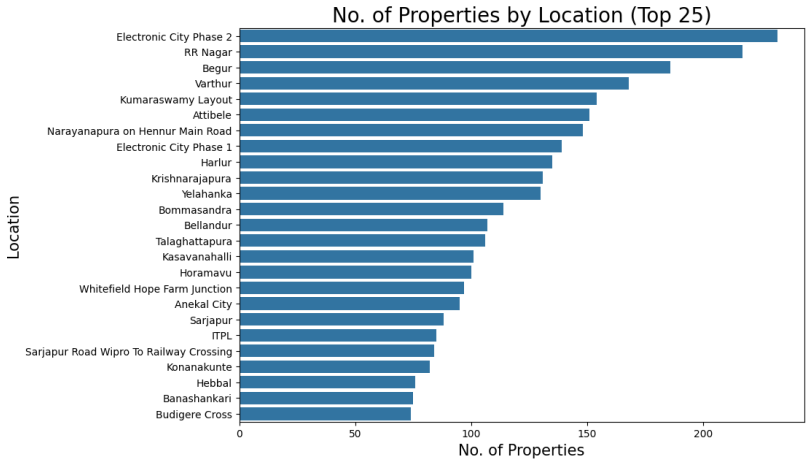


Figure 5: Top property Locations in Bangalore

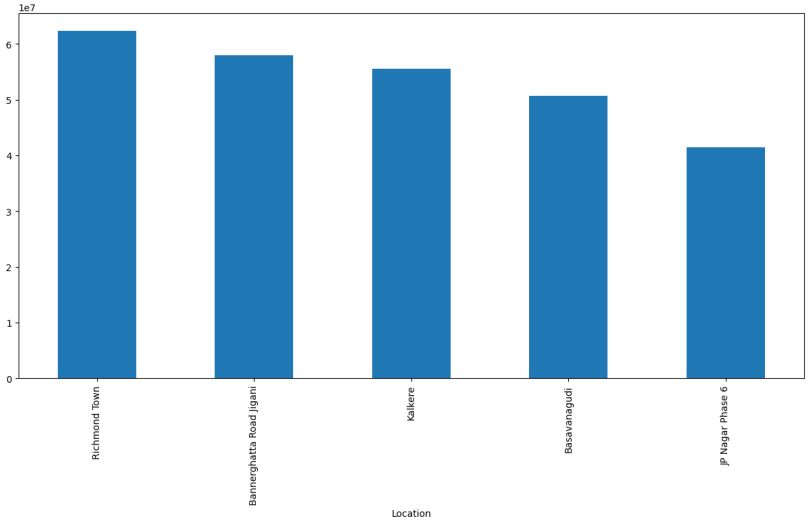


Figure 6: Bangalore's most expensive areas

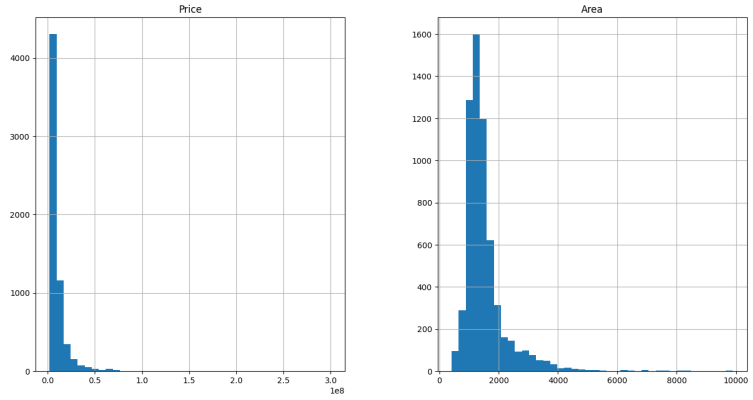
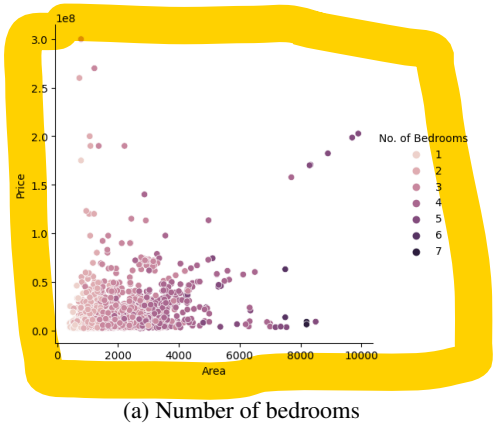
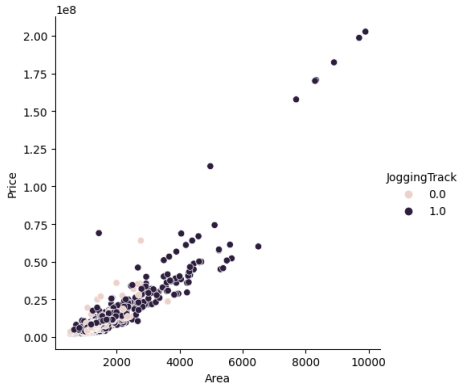


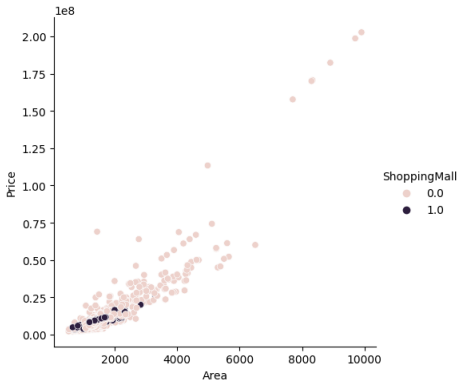
Figure 7: Area distribution



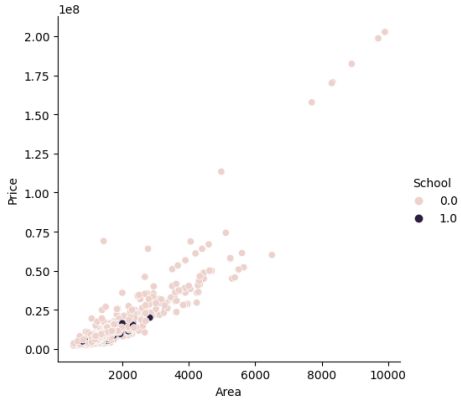
(a) Number of bedrooms



(b) Jogging track access



(c) Shopping mall access



(d) School access

Figure 8: Relation between various variables

3 First step modeling

Our fundamental interest of research are tree-based models, so we selected four for later exploration. The ones we wanted to examine are: Random Forest, XGBoost, LightGBM and Catboost. For each of them we created a considerable number of experimental models with and without optimized hyperparameters to choose the best five models. To compare these algorithms for regression of prices in India's city Bangalore we used several metrics of calculating the error of predictions: Root Mean Square Error (RMSE), Mean Square Error (MSE), Mean Absolute Error (MAE).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - y_i)^2}{N}} \tag{1}$$

$$MSE = (RMSE)^2 \tag{2}$$

$$MAE = \frac{\sum_{i=1}^{N}|x_i - y_i|}{N} \tag{3}$$

Where:

- N = sample size
- x_i = predicted value
- y_i = observed value

Our next move was to select the best three models of the remaining twenty (five from each algorithm). Here are the descriptions and results of the first step modeling.

3.1 Random Forest Models

3.1.1 Method description

The Random Forest model for regression task is an ensemble algorithm that combines multiple Regression Trees (RTs). Each RT is trained using a random subset of features and the output is the average of the individual RTs. The sum of squared errors for a tree T is calculated by:

$$S = \sum_{c \in leaves(T)} \sum_{i \in C} (y_i - m_c)^2 \tag{4}$$

Where $m_c = \frac{1}{n_c} \sum_{i \in C} y_i$ is the prediction for leaf c.

The main aim of each split in RT is to minimize the S.

3.1.2 Hyperparameters to optimize

Typical Random Forest implementations have five hyperparameters to consider into in-depth optimization. These are:

- **number of trees in model (num.trees)** - the more rows in the data, the more trees are needed but there is a danger of overfitting the model.
- **maximal tree depth (max.depth)** - the deeper the tree, the more splits it has and it captures more information about the data.
- **minimal node size (min.node.size)** - setting this number larger causes smaller trees to be grown (and thus take less time).
- **number of variables to possibly split at in each node (mtry)** - the sum of the carnality of each variable. To illustrate: in case three variables A, B, C are selected at a node i, with A being a binary feature, B an integer in the interval [1,100] and C a continuous variable taking on 200 distinct values in the current sample, the number of possible splits is 2+100+200=302. The bigger the mtry, the bigger randomness of splits in RT.
- **splitting rule (splitrule)** – regression tasks require one of four possible splitting rules: estimation by variance ('variance'), by maximally selected rank statistics ('maxstat'), by random choice of a certain number of attributes ('extratrees') or by beta distributions of variables ('beta').

There are four classic approaches to tune hyperparameters: *manual* – based on our intuition, experience or guessing, *grid search* – creating a grid of parameters combinations and iterating through it to find the best combination, *random search* - randomly searching hyperparameters using a grid and *automated hyperparameter tuning* – based on gradient descent algorithm.

3.1.3 Libraries used and optimizing methods

In R programming language the popular package for creating Random Forest (RF) models is called **'ranger'**. The default parameters of the RF implementation in this package are:

- num.trees = 500
- max.depth = 0 (means unlimited depth)
- min.node.size = 5 (for regression)
- mtry = (rounded down) square root of the number variables
- splitrule = 'variance'

At the early stage of modeling, we decided to compute model with all default parameters to see what the basic RMSE, MSE and MAE metrics are for the Bangalore dataset. Then we started changing *splitrule* parameter to see if any is better than the 'variance' rule. Nevertheless, the default *splitrule* proved to be the best one for the regression task. Our next move with exploring Random Forest in R language was tuning parameters with **'mlr3'** library, which is responsible for efficient, object-oriented programming on the building blocks of machine learning. To tune the RF algorithm, we had to specify:

- the search space: *num.trees* from 10 to floor of 10% of training observations at the first attempt and to 1000 at the second attempt, *mtry* from 2 to 15 at the first attempt and to 100% of training observations at the second attempt, *max.depth* from the set of 5,15 and 100.
- maximum of iterations in each tuning: we set it to 10.
- control object defining tuning type: we chose random search in this case.

3.1.4 Best models presentation

NR	SEED	NUM.TREE	M.TRY	MAX.DEPTH	SPLITRULE	RMSE	MSE	MAE
1	123	Def.	Def.	Def.	Def.	10548491	1.112707e+14	4482309
2	123	321	15	Def.	Def.	10590541	1.21596e+14	4588262
3	123	85	15	100	Def.	10625789	1.129074e+14	4585146
4	123	321	15	100	Def.	10663314	1.137063e+14	4627616
5	123	Def.	Def.	Def.	extratrees	10680223	1.140672e+14	4563912

Table 1: Scores of test sample

3.2 XGBoost Models

3.2.1 Method description

XGBoost is an optimized gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

3.2.2 Hyperparameters to optimize

The XGBoost hyperparametres we decided to consider into optimization are:

- **eta (learning rate)** - step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
- **max.depth** - step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
- **min.child.weight** - step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
- **nrounds** – the number of decision tress in the final model

3.2.3 Libraries used and optimizing methods

The default values of the hyperparameters presented above are:

- eta = 0.3
- max.depth = 6
- min.child.weight = 1
- nrounds (needs to be specified by the user himself)

Unfortunately the XGBoost R package does not have any inbuilt feature for doing grid/random search. To overcome it, we used **'mlr'** library to perform the extensive parametric search and try to obtain optimal accuracy. Another approach was using **xgb.cv** function for calculating best nrounds value for the model and at the same time following the rules presented below:

- If RMSE for xgboost model created using default parameters is not satisfying, decrease *eta* value.
- Don't introduce *gamma* until you can observe a significant difference in train and test error.
- Tune the regularization parameters *(alpha, lambda)* if required using the best parameters obtained so far.
- At last, carefully increase/decrease eta and follow the procedure.

3.2.4 Best models presentation

NR	SEED	ETA	NROUNDS	MAX. DEPTH	MIN.CHILD. WEIGHT	RMSE TEST	RMSE TRAIN	MAE TEST	MAE TRAIN
1	101	0.07	100	7	Def.	9391780	9460934	4248611	3874707
2	101	0.1	79	7	Def.	9615621	9319090	4260425	3829237
3	101	0.1	100	7	2	9503901	9716940	4275710	3879759
4	101	0.05	91	7	Def.	9268299	9732675	4227089	3962150
5	101	0.06	115.	7.	Def.	9540686	9418837	4256814	3875439

Table 2: Scores of test and train sample

3.3 Catboost models

3.3.1 Method description

Catboost is another algorithm that was used in this project. It's relatively new as the initial release took place in 2017 and stable release occurred at the beginning of April this year (2022). Nevertheless, as for year 2021 it was listed the top-7 most frequently used ML framework in the 2021 survey.
CatBoost is based on the theory of decision trees and gradient boosting. The core idea of boosting is to sequentially combine many weak models (models performing slightly better than random chance) in order to create a strong competitive predictive model. The sequential fitting of decision trees by gradient boosting, results in fitted trees learning from the mistakes of former trees and hence reduced errors. This process of adding a new function to existing ones is continued until the selected loss function is no longer minimized.

3.3.2 Hyperparameters to optimize

Due to our research, the key parameters to optimize in this case are:

- **iterations (num_trees)** - the maximum number of trees that can be built. To speed up the training, the number of iterations can be decreased. When the number of iterations decreases, the learning rate needs to be increased.
- **depth (max_depth)** – the maximal tree depth. The higher the value, the more complex a model will become, which indicates increased performance on the training set, but might cause overfitting. In most cases, the optimal depth ranges from 4 to 10.
- **learning_rate (eta)** - the parameter, which controls how much to change the model, while moving toward a minimum of a loss function. The lower the value, the better the accuracy but slower the model learns. On the contrary, higher value may result in unstable training process.

3.3.3 Libraries used and optimizing methods

The default values of the hyperparameters presented above are:

- iterations = 1000
- depth = 6
- learning_rate = 0.3

In order to avoid long computing time we decided to optimize the hyperparameters using random search. The Caret library provides random search hyperparameters tunning for Catboost in R. Unfortunately it doesn't cooperate well

with regression models yet, as they occur to the function as classification. Catboost is steadily increasing in popularity, thus not all methods are supported for it in R yet. Therefore, the tunning was conducted manually. The choice of the values for parameter grid was inspired by the research on hyperparameter tunning performed in assembled articles. The first model was built using the default parameters so as to compare whether the tunning was successful. The metrics used for comparison were RMSE and MAE – for both of them the lower the score, the better. In total of the 20 models built, we managed to achieve better scores then the default model.

3.3.4 Best models presentation

Models were built on Bangalore dataset. The metrics scores are presented in Indian rupees. As for the day 10.05.2022, 1 000 000 INR was equivalent to approx. 12 300 EUR.

NR	SEED	DEPTH	LEARNING. RATE	ITERATIONS	RMSE TEST	RMSE TRAIN	MAE TEST	MAE TRAIN
1	123	4	0.1	Def.	1741564	1042481	838151.6	479544.5
2	123	4	Def.	Def.	1733489	838735.2	673060.5	212610.5
3	123	8	Def.	Def.	1540132	810712.7	564952.2	86457.45
4	123	8	0.1	Def.	1617599	822247.7	624959.3	163346.1
5	123	8	Def.	80	1651593	1148300	841683.4	556736

Table 3: Scores calculated on test and train samples

3.4 LightGBM models

3.4.1 Method description

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages: Faster training speed and higher efficiency. Lower memory usage. Better accuracy. Support of parallel, distributed, and GPU learning. Capable of handling large-scale data. While other algorithms trees grow horizontally, the LightGBM algorithm grows vertically meaning it grows leaf-wise and other algorithms grow level-wise. LightGBM chooses the leaf with a large loss to grow. It can lower more loss than a level-wise algorithm when growing the same leaf.

3.4.2 Hyperparameters to optimize

Core parameters used during tuning of hyperparameters:

- **num_leaves** - LightGBM adds nodes to trees based on the gain from adding that node, regardless of depth. Because of this growth strategy, it isn't straightforward to use max_depth alone to limit the complexity of trees. The num_leaves parameter sets the maximum number of nodes per tree. Decrease num_leaves to reduce training time.
- **max_depth** - limit the max depth for the tree model. This is used to deal with over-fitting when a dataset is small. The tree still grows leaf-wise
- **learning_rate** - the size of the "step" that the algorithm takes. learning_rate will not have any impact on training time, but it will impact the training accuracy. As a general rule, if you reduce num_iterations, you should increase learning_rate.
- **num_iterations** - The num_iterations parameter controls the number of boosting rounds that will be performed. Since LightGBM uses decision trees as the learners, this can also be thought of as the "number of trees". If you try changing num_iterations, change the learning_rate as well.

LightGBM library has its function that consumes data set and gives wanted shape of data, that works the best with LightGBM models, so Bangalore cleared data we prepared with that function too. During modeling, besides generic LightGBM parameters we tried, F(y) - function was used on the *Price* column. Since values of the *Price* column are from a wide range, we tried to examine models with a logarithm of Price but models with that change did not reach a better score.

3.4.3 Libraries used and optimizing methods

The default values of the hyperparameters presented above are:

- num_leaves = 31
- max_depth = -1
- learning_rate = 0.1
- num_iterations = 100

Hyperparameters were tuned with the use self-written grid search where the covered grid was:

- max_depth = [2,4,5, 10, 15, 20],
- learning_rate = [0.01, 0.1, 1],
- num_iterations = [500, 1000, 1500]

as so models with self-chosen parameters were examined. Hyperparameters were chosen with respect to RMSE. What was unexpected is that the self-chosen parameters appeared to be the most relevant ones.

3.4.4 Best models presentation

NR	N.LEAVES	DEPTH	F(Y)	NUM.ITERATIONS	LEARNING.RATE	RMSE TEST	MAE TEST
1	Def.	5	line	1000	0.1	2355046	872573.8
2	Def.	5	line	500	0.1	2554069	986036.0
3	100	10	line	1000	0.1	2355046	872573.8
4	Def.	20	line	1000	0.1	2695366	853137.2
5	Def.	4	line	5000	0.1	2797650	1160256.7

Table 4: Scores of test sample

3.5 Conclusion

After exchanging the best parameter values with other research teams, we chose top three models, presented below. The obtained scores for models suggest that the best algorithm for inspected case is Catboost. As shown in the plot, for all models, the predicted values are relatively close to the real ones with only a few outliers.

	depth	n_estimators	learning_rate	l2_leaf_reg	rsm	RMSE train	RMSE test
Model 1	6	1200	0,1	0,01	0,9	814 684.3	1 546 002
Model 2	7	1000	0.06	10	1	1 131 753	1 573 945
Model 3	5	800	0,1	0,05	0,75	859 914.2	1 662 501

Table 5: Best models' parameters and results

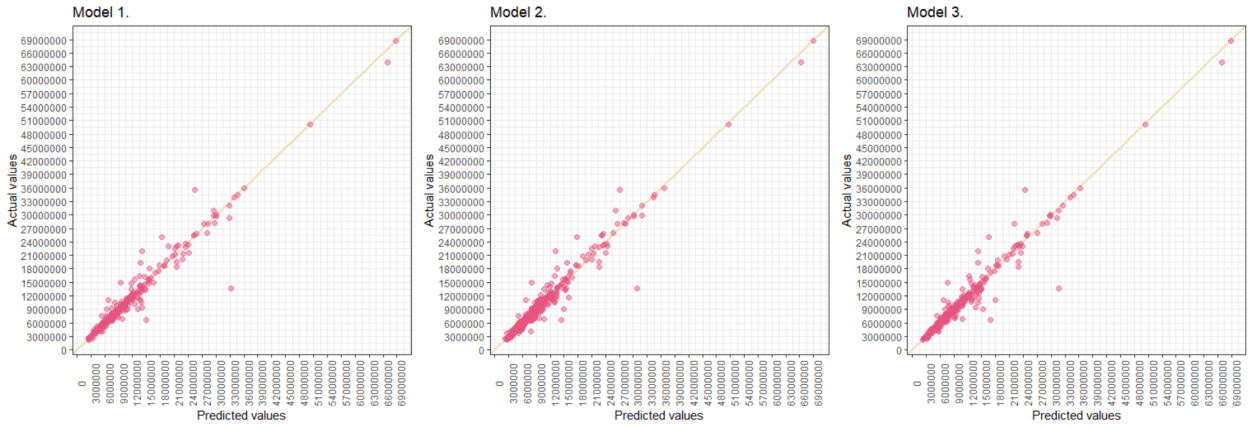


Figure 9: Top 3 models

4 Model analysis using XAI methods

After choosing the best models, our next step is to analyse them using Explainable AI (XAI) methods. Firstly, we are going to check model's prediciton decomposition using Break Down approach and Ceteris Paribus profile. Then we are going to present Permutation Feature Importance for each model. The final step will be creating Partial Dependence Profile and Accumulated Local Dependence and comparing the results.

Below are top three models including their hyperparameters:

- **Model 1 - Catboost model**
  - depth=6
  - iterations=1200
  - learning_rate=0.1
  - l2_leaf _reg=0.01
  - rsm=0.9
  - loss_function='RMSE'
- **Model 2 - Catboost model**
  - depth=5
  - iterations=800
  - learning_rate=0.1
  - l2_leaf _reg=0.05
  - rsm=0.75
  - loss_function='RMSE'
- **Model 3 - LightGBM model**
  - num_iterations=5000
  - learning_rate=0.0001

4.1 Break Down profile

Break-down plots show how the contributions attributed to individual explanatory variables change the mean model's prediction to yield the actual prediction for a particular single observation. The green bars indicate positive changes in the mean predictions, whereas the red ones - negative changes. Below are Break Down visualisations for a chosen observation for each model, compared with the results from the another, different observation.

4.1.1 Model 1

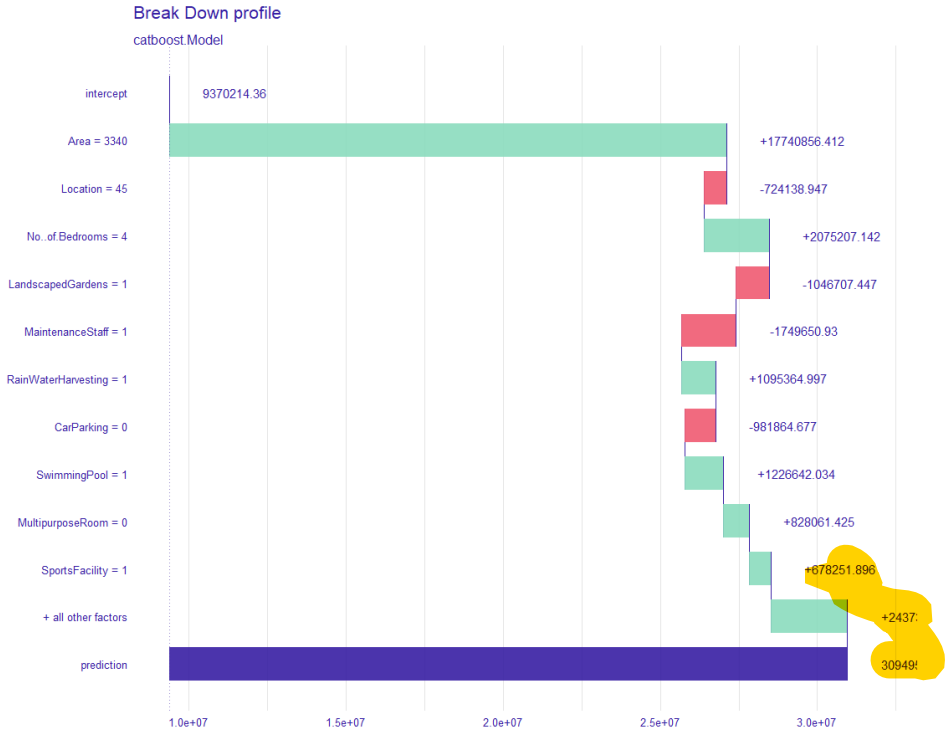


Figure 10: Break Down Profile for observation 1

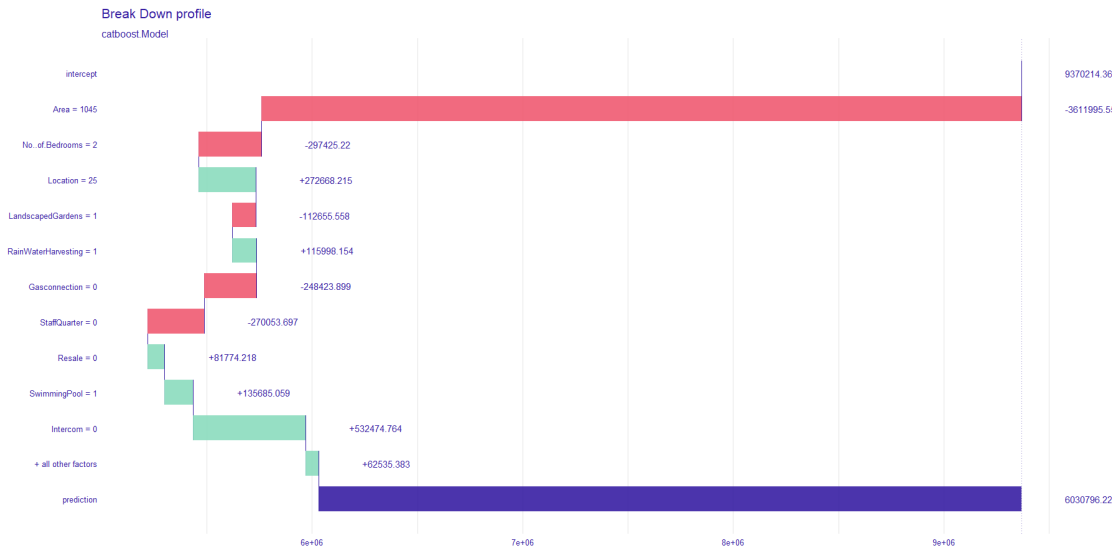


Figure 11: Break Down Profile for observation 2

As we can see above, chosen observations significantly differ. In the first one, *Area* and *No.of.Bedrooms* have positive impact on the price - which seems reasonable as the property has four bedrooms and as much as 3340 square feet. However its location may not be preferable as it decreases the price. The property in the second observation has more than three times smaller area and only two bedrooms which has of course negative impact on the value. On the other hand, it may be closer to the city centre as *Location* increases the price.

4.1.2 Model 2

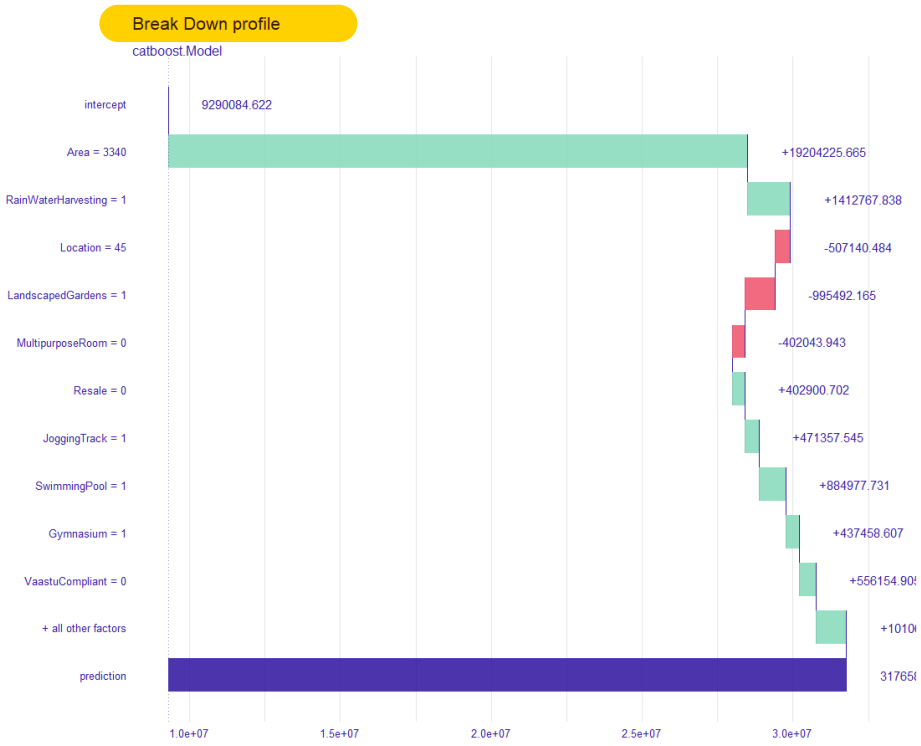


Figure 12: Break Down Profile for observation 1

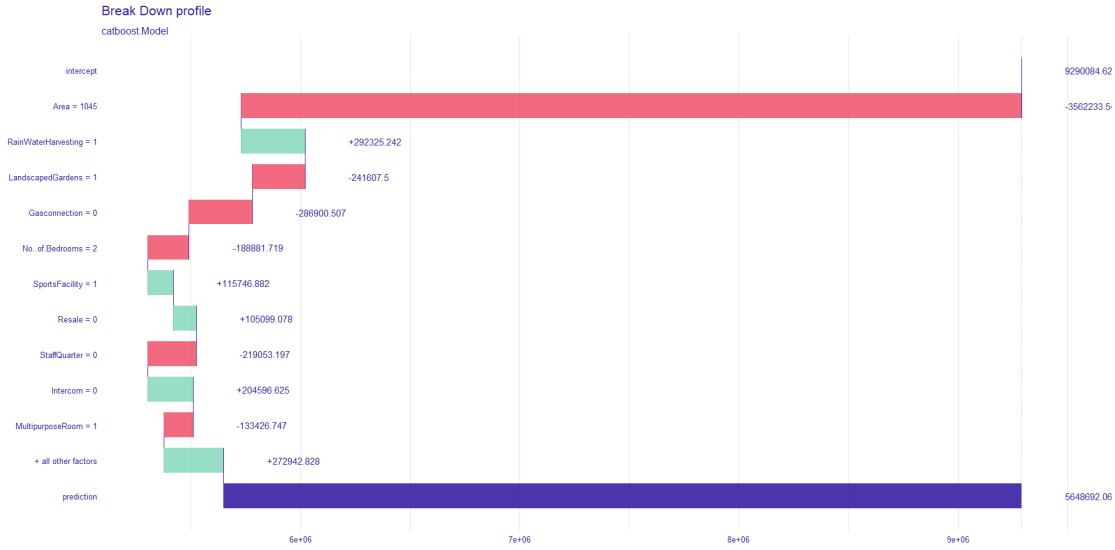


Figure 13: Break Down Profile for observation 2

For the first observation in the second model we can also notice positive impact of *Area* on the property value. Surprisingly *RainWaterHarvesting* increases the price by nearly 1.5 million INR. Again, there is a negative impact of *Location* and *LandscapedGardens* on the price. The reason of such decrease may be long distance from the city centre and for the presence of landscaped gardens - higher maintenance costs. The situation slightly differs for the second observation where we have much smaller property - here we can notice huge negative impact of *Area* variable on the price.

4.1.3 Model 3

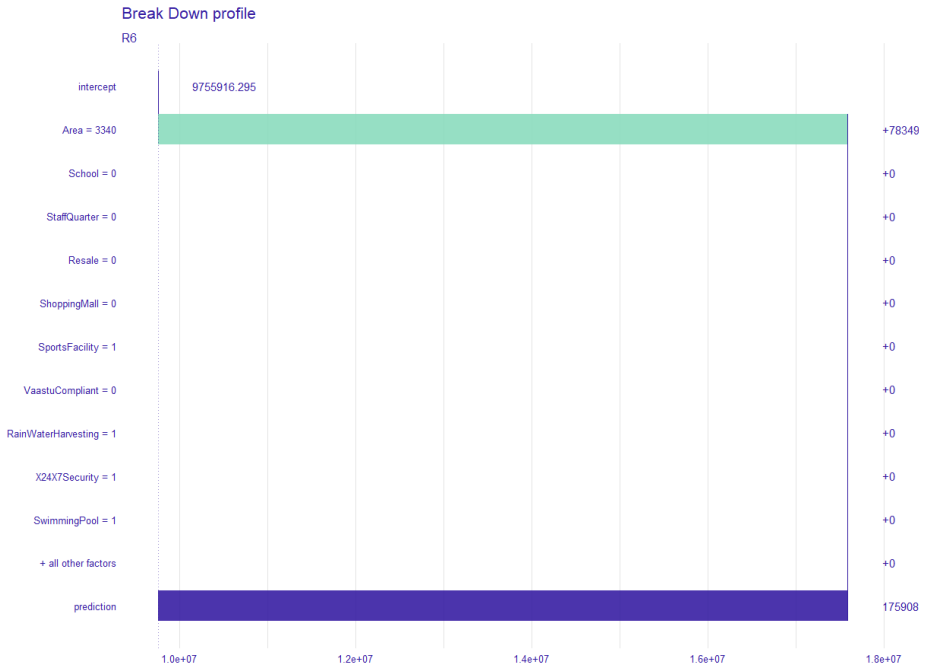


Figure 14: Break Down Profile for observation 1

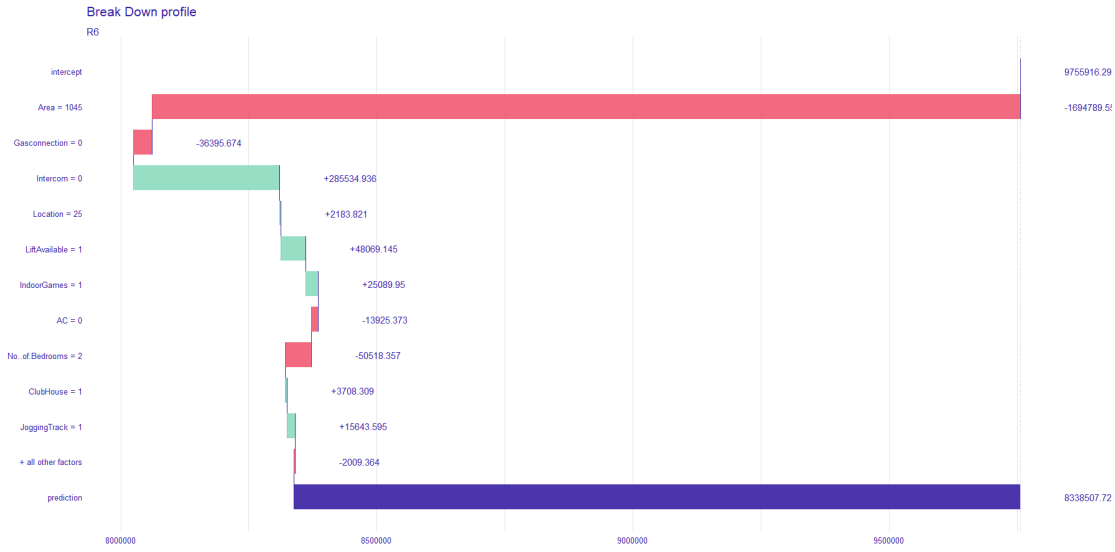


Figure 15: Break Down Profile for observation 2

The results for the first observation in the third model are surprising as only *Area* has any impact (positive in this particular example) on the price. For the second observation the results are definitely more varied. Similarly to the previous models, the area has negative impact on the property value as it is smaller than average. Also lack of gas connection decreases the price which seems reasonable. Surprisingly, no intercom has a positive impact on the property value but it may indicate that the property is not a flat - and as we know houses are usually more expensive than flats.

4.2 Ceteris Paribus profile

Ceteris-paribus (CP) profiles show how a model's prediction would change if the value of a single exploratory variable changed, whereas other variables remained unchanged. Below are CP visualisations for a chosen observation for each model, compared with the results from the another, different observation.

4.2.1 Model 1

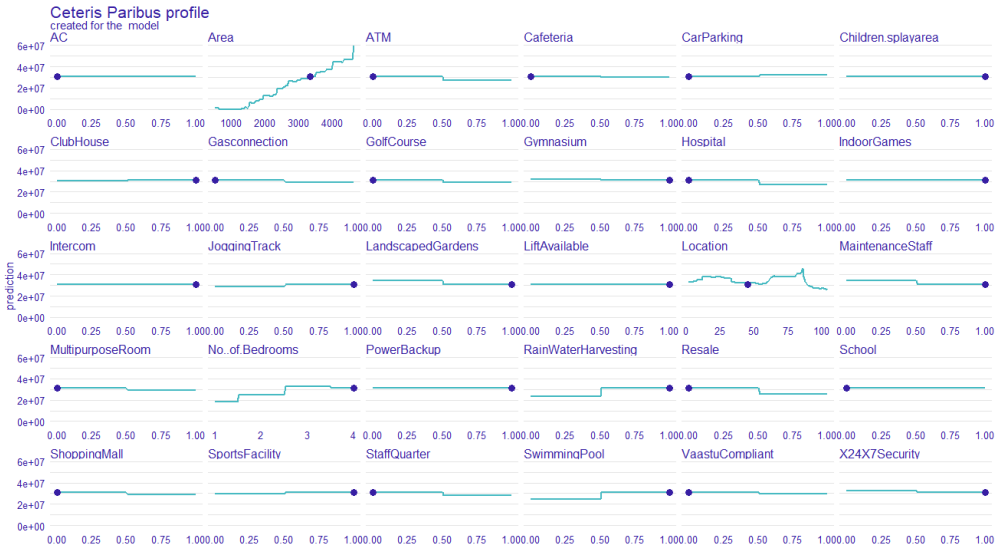


Figure 16: Ceteris Paribus Profile for observation 1

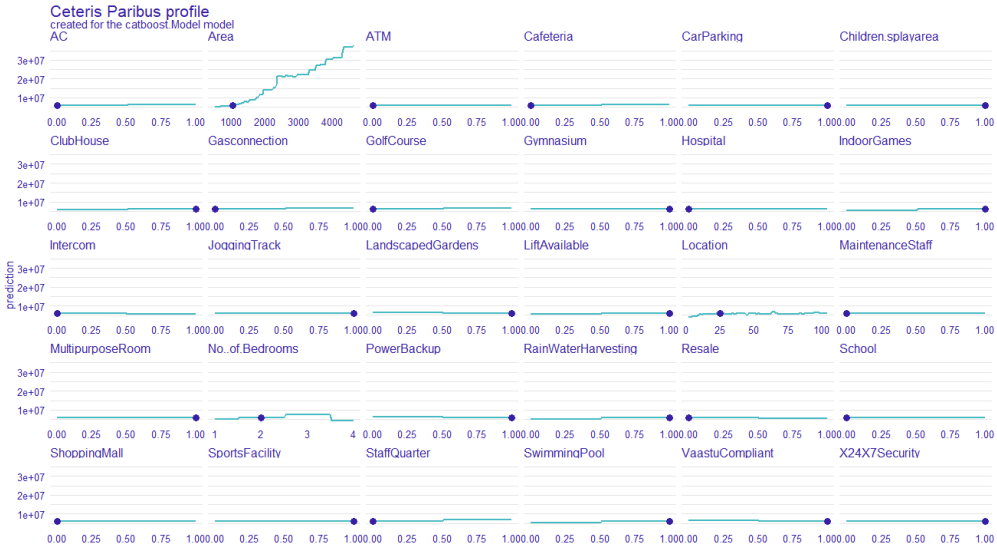


Figure 17: Ceteris Paribus Profile for observation 2

The general shape of the two Ceteris Paribus profiles for the first model is similar and the results seem to be reasonable. For example, for the variable *Area*, for both observations we can see that if property's size was bigger, while keeping values of all other explanatory variables unchanged, the price would increase significantly. What is interesting for the first observation is the fact that if the number of bedrooms was reduced to three, the price wouldn't decrease - actually it would get a bit higher. The reason may be the fact that property's area remained unchanged so the average bedroom size would be bigger and that would be more convenient. However, for the second observation the relationship between these two variables is inverse - if the number of bedrooms changed from two to three, the price would increase. On the other hand if it changed to four, we would observe a drop in property's value.

4.2.2 Model 2

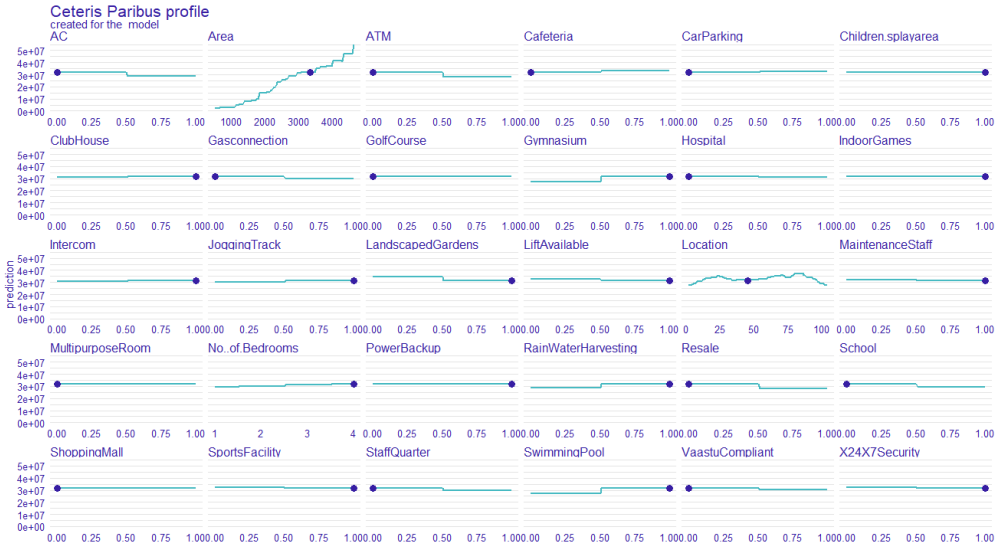


Figure 18: Ceteris Paribus Profile for observation 1

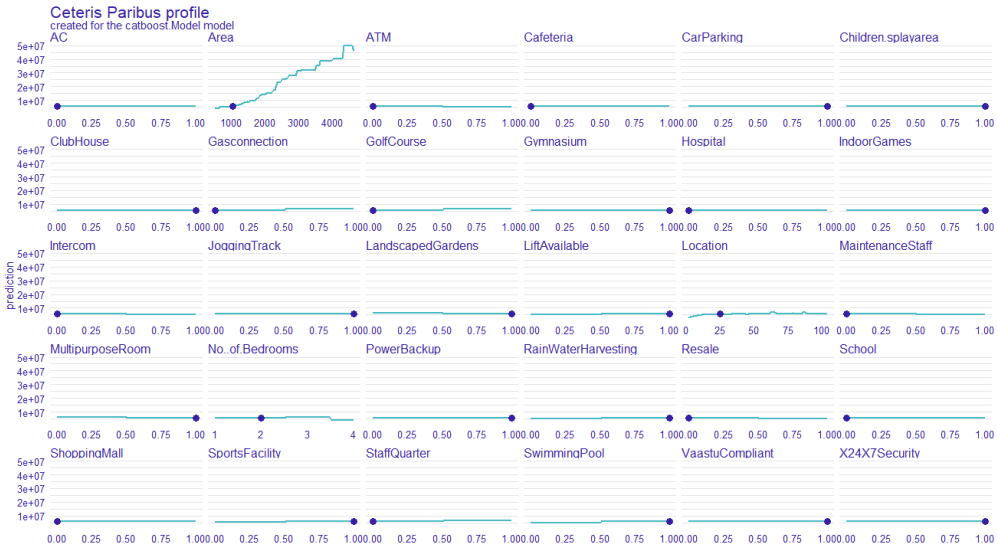


Figure 19: Ceteris Paribus Profile for observation 2

In the second model Ceteris Paribus profiles for both observations again look similar. We can observe high impact of the variable *Area* on the price - of course the bigger the area, the higher the price. For the first observation we can notice that if the property had air conditioning, gas connection or landscaped gardens, the price would decrease. It is surprising as all of these facilities seem convenient. Also for the first observation the price varies depending on the variable *Location* whereas for the second observation it remains nearly unchanged.

4.2.3 Model 3

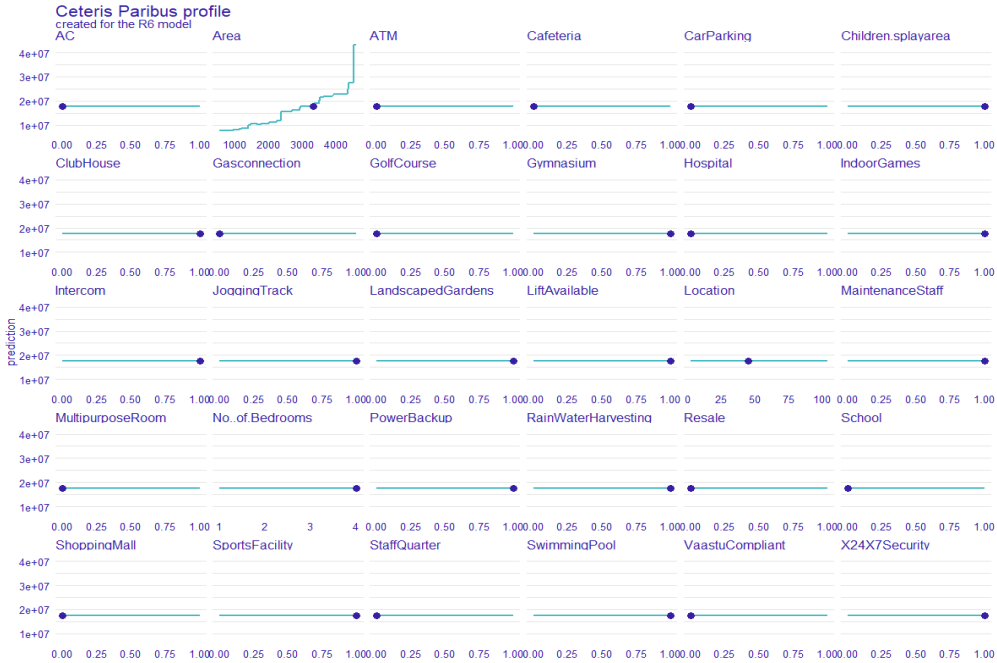


Figure 20: Ceteris Paribus Profile for observation 1

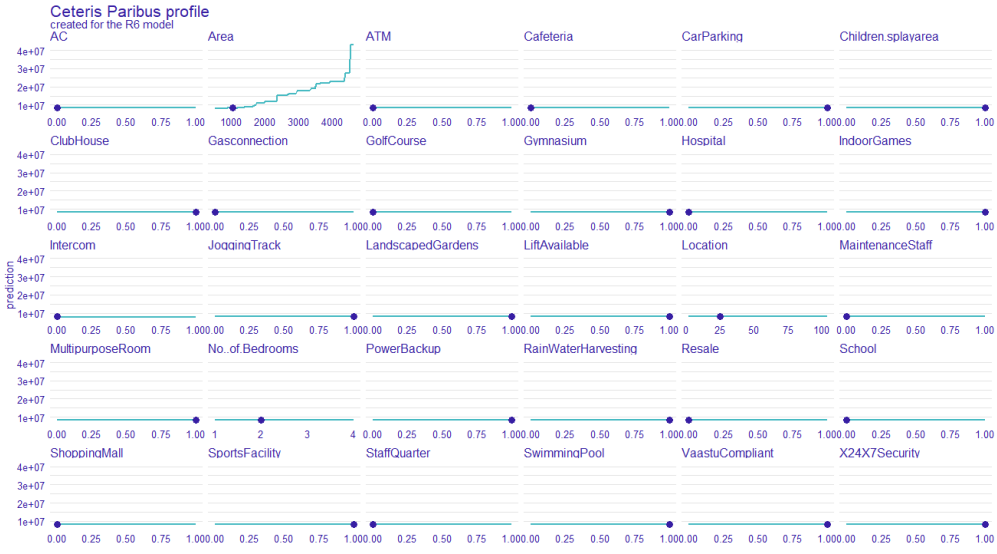


Figure 21: Ceteris Paribus Profile for observation 2

In the LightGBM model the only significant variable for both observations is of course *Area*. While in previous models the price varied linearly, here it grows exponentially, depending on the property's area. Surprisingly, the number of bedrooms and the location seem to have no effect on the price. The results don't look as reasonable as in Catboost models.

4.3 Permutation Feature Importance

The concept of permutation feature importance is to measure the importance of a feature by calculating the increase in the model's prediction error after permuting the feature. A feature is considered important if shuffling its values increases the model error because it implies that it has a significant impact on the prediction. Below are Feature Importance visualisations for each model.

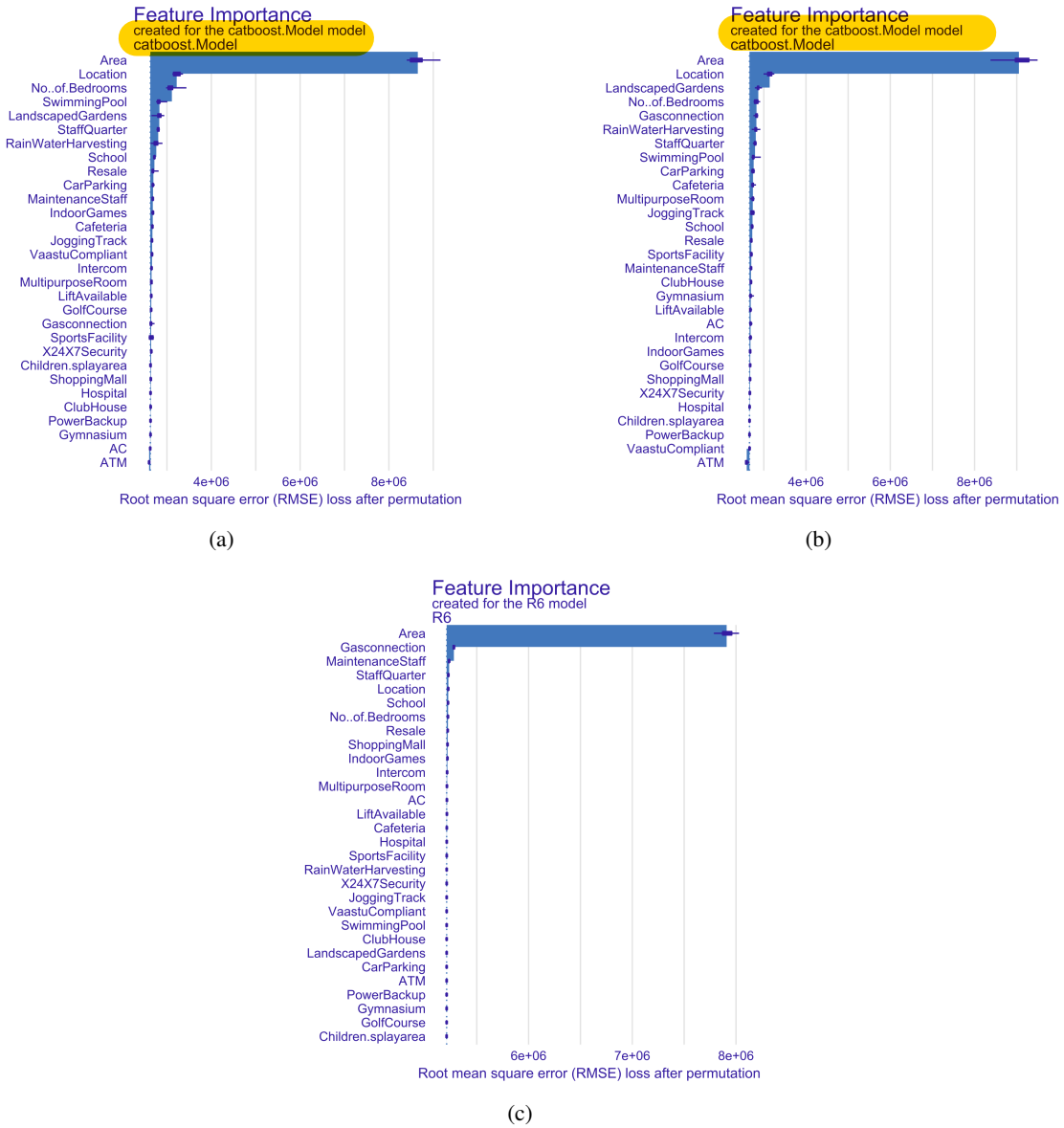


Figure 22: Permutation Feature Importance for all models

As we can notice above *Area* is undoubtedly the most important feature in our prediction. For Catboost models the other important features include *Location* and *No.of.Bedrooms* which seems reasonable. On the other hand for LightGBM model the results differ as apart from *Area*, *Gasconnection* and *MaintenanceStaff* were classified as most important. To sum up Catboost models results seem more intuitive which is not a big surprise as Catboost models performed better in our metric scores analysis.

4.4 Partial Dependence Profile & Accumulated Local Dependence

The general idea underlying the construction of Patrial Dependence profiles is to show how does the expected value of model prediction behave as a function of a selected explanatory variable. A PD profile is estimated by the mean of the CP profiles for all instances (observations) from a dataset. However, such profiles may be misleading if explanatory variables are correlated. Thus, it not really makes sense to consider for example properties with small area and large number of bedrooms. That is why it is essential to check the Accumulated Local Dependence profile as it eliminates the effect of correlated variables. Below we are going to present comparison of PD and AL plots for each model. All of the plots are constructed for the variable *Area* as it has the highest impact on the price.

4.4.1 Model 1

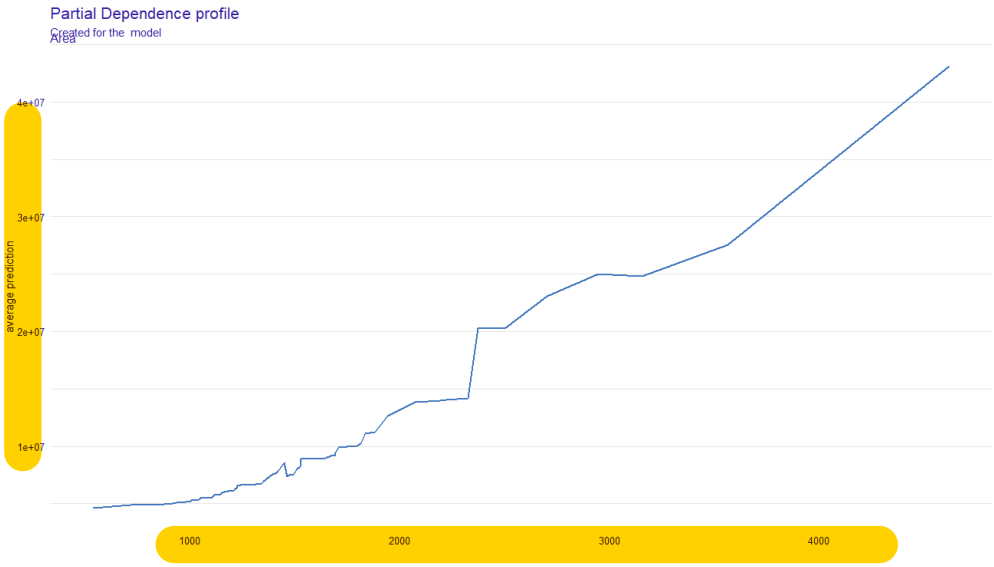


Figure 23: Partial Dependence Profile for Area in model 1

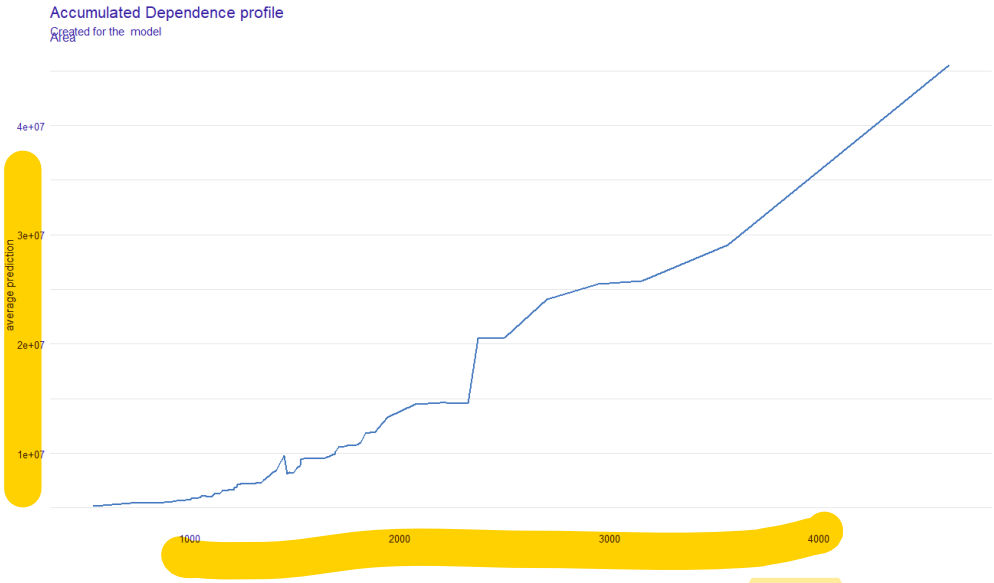


Figure 24: Accumulated Local Dependence for Area in model 1

4.4.2 Model 2

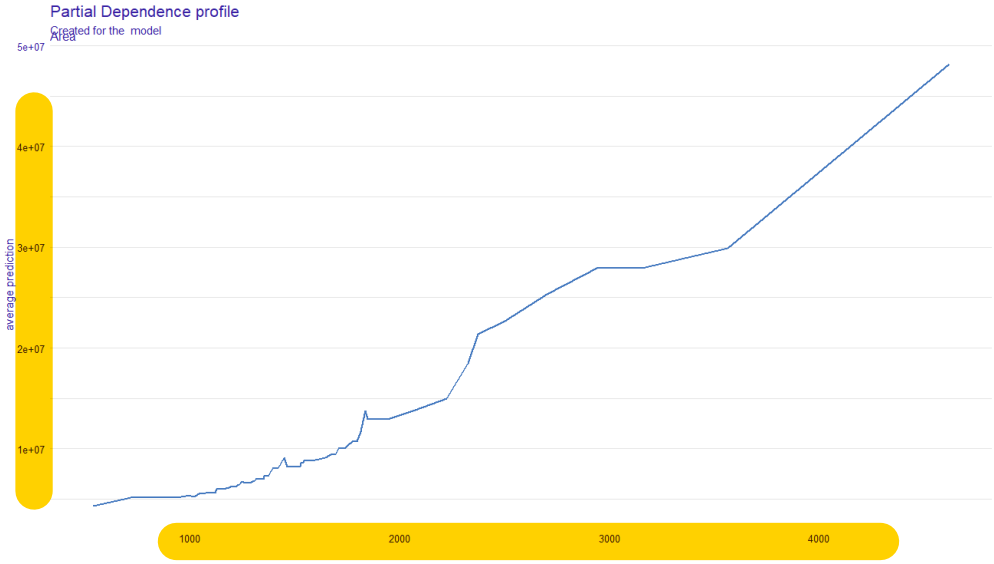


Figure 25: Partial Dependence Profile for Area in model 2

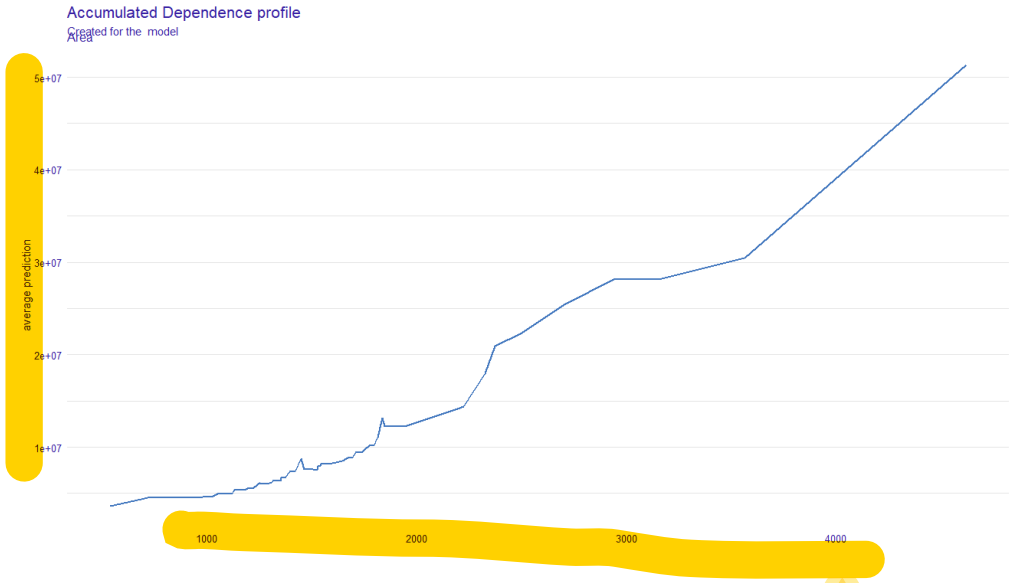


Figure 26: Accumulated Local Dependence for Area in model 2

4.4.3 Model 3

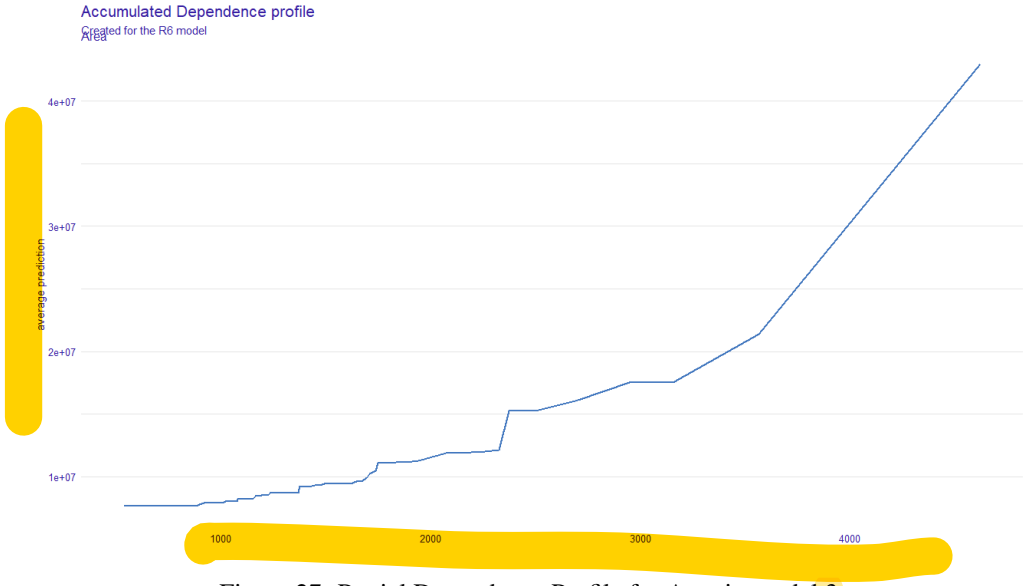


Figure 27: Partial Dependence Profile for Area in model 3

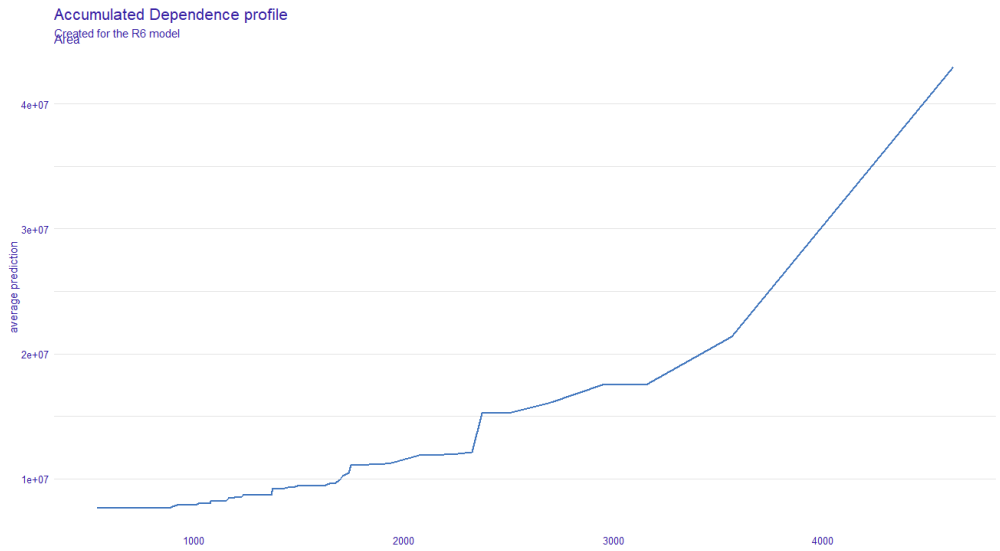


Figure 28: Accumulated Local Dependence for Area in model 3

To sum up, Partial Dependence profiles and Accumulated Local Dependence profiles are similar for all three models. Furthermore, we can notice that ALE and PDP plots for each models are the same. This may be due to the fact that the explanatory variables in our data set are not strongly correlated.

4.5 Conclusions

To conclude, after analysing our top three models with Explainable AI methods we got satisfactory results. As in the previous chapter, the Catboost models performed the best. The variables contributions obtained from Break Down plots and Ceteris Paribus profile analysis gave reasonable, intuitive effects. From all of these methods it turned out that the variable *Area* has the biggest impact on the price - as we expected.

References

To be done.