Capstone Project

Machine Learning Engineer Nanodegree

Dmytro Illarionov

January 7th. 2019

1. Definition

1.1 Project overview

The real estate world deals with a lot of money and the calculation of house prices is clearly a much needed and important part of it. Both the seller and the buyer are interested in the high accuracy of the prices. The price itself depends on a lot of different factors and parameters such as type of real estate, total area, number of rooms and bedrooms, distance from downtown etc.
Building a model which will calculate and predict the prices of real estate items based on different information about them is a classical supervised regression problem in machine learning. *Supervised learning* is one kind of machine learning algorithms that automates decision-making processes by generalizing from known examples. In this case the machine learning algorithms learns from input/output pairs because a "teacher" provides supervision to the algorithms in the form of the desired outputs for each example that they learn from and the algorithm finds a way to produce the desired output given an input [1].
Predicting of house prices is not a brand new topic. For example, one very popular data set is describing the sale of individual residential property in Ames, Iowa from 2006 to 2010. Dean De Cock from Truman State University has compiled, explored and analyzed the data set. Published results of his work can be found in web [2]. This data set is also an ongoing competition on kaggle.com [3]. More than 4500 teams take part on it and publish their results of exploring the data using all possible algorithms and techniques, from linear regression, over XGBoost and PCA to neural networks.
The input data, which will be explored in this project, comes from kaggle.com [4] and contains information about different apartments in Melbourne. This information was collected based on publicly available results posted on domain.com.au.

1.2 Project statement

The challenge of the project is to solve a regression problem namely to estimate the best model which will predict the price of different real estate items based on explanatory features provided in the dataset. It means that the model will obtain as input a certain number of explanatory variables which describe quality and quantity of many physical attributes of the property and the output of the model is going to be the price that mostly fits a property with this parameters. Most of the variables are type of information that a typical home buyer would want to know about a potential property. This input variables from used data set are described in detail in the section *2.1 Data exploration*.
The following two approaches will be considered to solve the problem. The first approach consist in evaluating the regression models from *scikit-learn* library of python. The second approach is to build the neural network using *Keras* library.

1.3 Metrics

To evaluate the performance of regression models we will be using the *root-mean-square-error* (RMSE) [5] and *r2_score* also known as coefficient of determination [6].
Root-mean-square-error is a frequently used measure of the differences between values predicted by a model and the values observed.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

n - number of observations,
$\hat{y}_i$ - predicted values,
y_i - true values.

RMSE is always non-negative, and a value of 0 would indicate a perfect fit to the data. I have chosen to use RMSE because it is a suitable metrics for regression problems since the aim is to predict a certain number and not categorize a samples. Very important is that RMSE is in the same units as the data outcome. It means it will be in units of output variable (in our case in dollars). It will allow me personally estimate how good or how bad my model is in terms of performance and accuracy and also explain the result to persons, who are not interested in details of how the algorithm works but want to have a quick measurable result, which they can easy estimate and understand. Low RMSE values are desired.

r2_score is the second type of metrics for evaluations in this project. It describes the proportion of the variance in the dependent variable that is predictable from the independent variable(s). Best possible score is 1.0. r2_score is needed to measure how well the regression predictions approximate the real data points. Low r2_scores of training sets by high r2_score of training set will indicate that the model “remembers” the data rather than generalize the dependencies between features and target variable.

2. Analysis

2.1 Data Exploration

The data for the project provides 20 explanatory variables describing different aspects of 13580 apartments. The data set represents mixed data with 13 numerical and 7 categorical values. There are also several missing values in some attributes. These missing values can be treated using deletion or imputation techniques [7].

Input variables[1]:

1. Suburb: (categorical) Names of Suburbs
2. Address: (categorical) Address of real estate item
3. Rooms: (numerical) Number of Rooms
4. Type: (categorical) br - bedroom(s); h – house, cottage, villa, semi, terrace; u - unit, duplex; t - townhouse; dev site - development site; o res - other residential.
5. Method: (categorical) S - property sold; SP - property sold prior; PI - property passed in; PN - sold prior not disclosed; SN - sold not disclosed; NB - no bid; VB - vendor bid; W - withdrawn prior to auction; SA - sold after auction; SS - sold after auction price not disclosed. N/A - price or highest bid not available.

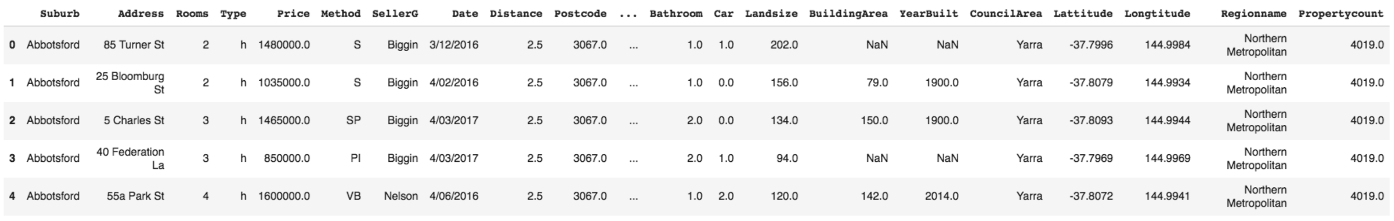
[1] In proposal was provided an excerpt of used input data

6. SellerG: (categorical) Real Estate Agent
7. Date: (numerical) Date sold
8. Distance: (numerical) Distance from CBD
9. Postcode: (numerical) Zip Code
10. Bedroom : (numerical) Number of Bedrooms
11. Bathroom: (numerical) Number of Bathrooms
12. Car: (numerical) Number of carspots
13. Landsize: (numerical) Land Size
14. BuildingArea: (numerical) Building Size
15. YearBuilt: (numerical) Year of construction
16. Lattitude: (numerical) Latitude
17. Longtitude: (numerical) Longitude
18. CouncilArea: (categorical) Governing council for the area
19. Regionname: (categorical) General Region
20. Propertycount: (numerical) Number of properties that exist in the suburb

Output variable:

Price: (numerical) Price in dollars

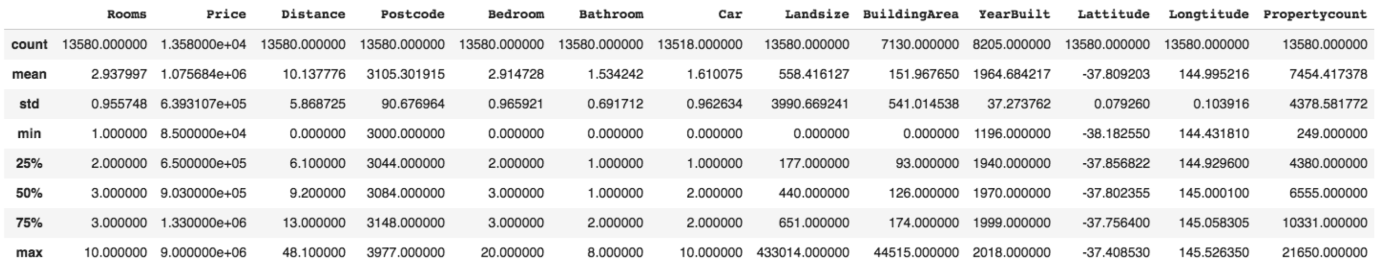
As mentioned above the entire data has 21 rows and 13580 columns. Table 1 provides an insight in the data which will be explored and analyzed.



	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Lattitude	Longtitude	Regionname	Propertycount
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0	NaN	NaN	Yarra	-37.7996	144.9984	Northern Metropolitan	4019.0
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0	79.0	1900.0	Yarra	-37.8079	144.9934	Northern Metropolitan	4019.0
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0	150.0	1900.0	Yarra	-37.8093	144.9944	Northern Metropolitan	4019.0
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	3067.0	...	2.0	1.0	94.0	NaN	NaN	Yarra	-37.7969	144.9969	Northern Metropolitan	4019.0
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	3067.0	...	1.0	2.0	120.0	142.0	2014.0	Yarra	-37.8072	144.9941	Northern Metropolitan	4019.0

Tab.1: Excerpt of data with numerical and categorical features

Table 2 provides a statistical information about the data. Important to mention that the values of numerical data are completely in different ranges. For instance, range of column *Rooms* is 1...10, but the target variable *Price* $8.5*10^4...9*10^6$. This is important to know for future feature engineering and preparing inputs for regression models neural net.



	Rooms	Price	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216	7454.417378
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916	4378.581772
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810	249.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600	4380.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100	6555.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305	10331.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530	145.526350	21650.000000

Tab.2: Statistics of numerical features

The data must be also examined if there are some duplicates (entire rows) and missing values to prevent models learning the same data many times or none data. There are no duplicates in this data set but a few columns have missing values. *Table 3* provides detailed information about columns with missing values.

Column	**Missing values**	**Missing values, %**
Car	62	0.5
BuildingArea	6450	47.5
YearBuilt	5375	39.6
CouncilArea	1369	10.1

Tab. 3: Statistics about missing values

Strategy for handling missing values:

- The column *Car* has only 0.5% of missing values. Also the values in the column are numerical nature. The 62 missing values will be replaced with the mean value of the columns namely 2.
- The other three columns have much more missing values. If they will be filled with mean values, randomly or using regression imputation, this can distort the output compared if the missing values were filled with real data. These columns will be dropped and not considered by future calculations.

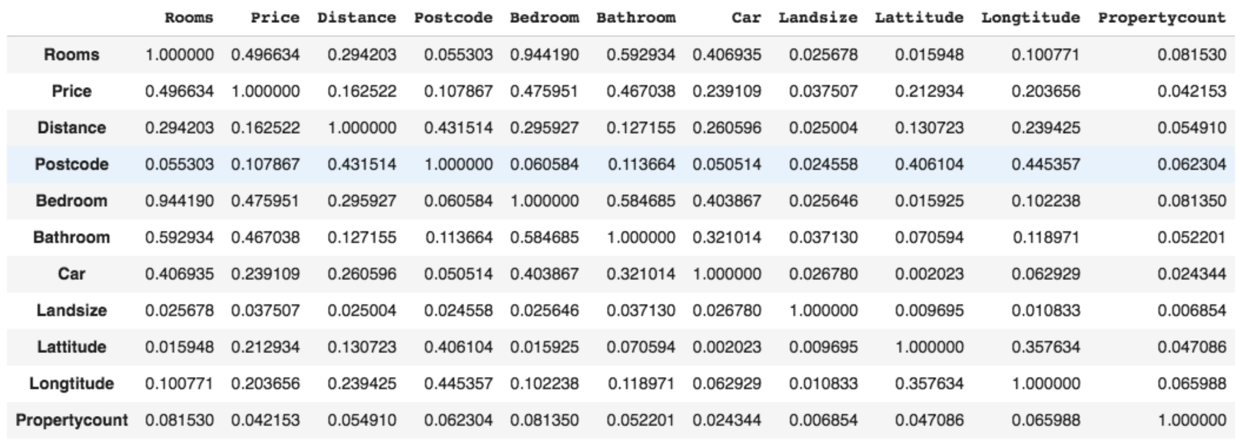
It is also important to know how many unique values contains each column with categorical values. Categorical values cannot be used as input to the algorithms. So they must be preprocessed.

Suburb	**Address**	**Type**	**Method**	**SellerG**	**Regionname**	**Postcode**	**Propertycount**
314	13378	3	5	268	8	198	311

Tab. 4: Number of unique values in columns with categorical data

The most common way to represent categorical variables is *one-hot-encoding*. The idea behind dummy variables is to replace a categorical variable with one or more new features that can have values 0 and 1. The values 0 and 1 make sense for many models in *scikit-learn*, and any number of categories can be represented by introducing one new feature per category [1].
By one-hot-encoding of *Suburb, Adress, SellerG* the amount of features will increase enormously. This fact will definitely decrease the calculation performance of the model. In this way only columns *Type, Method and Regionname* will be preprocessed using one-hot-encoding technique and the rest of categorical columns will be dropped. For this purpose *pandas.get_dummies()* function was applied [9].

Table 3 represents the cross-correlation between numerical variables. Pearson correlation measures the linear association between continuous variables. In other words, this coefficient quantifies the degree to which a relationship between two variables can be described by a line. The closer correlation is to 1, the more an increase in one variable associates with an increase in the other [8].



	Rooms	Price	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	Lattitude	Longtitude	Propertycount
Rooms	1.000000	0.496634	0.294203	0.055303	0.944190	0.592934	0.406935	0.025678	0.015948	0.100771	0.081530
Price	0.496634	1.000000	0.162522	0.107867	0.475951	0.467038	0.239109	0.037507	0.212934	0.203656	0.042153
Distance	0.294203	0.162522	1.000000	0.431514	0.295927	0.127155	0.260596	0.025004	0.130723	0.239425	0.054910
Postcode	0.055303	0.107867	0.431514	1.000000	0.060584	0.113664	0.050514	0.024558	0.406104	0.445357	0.062304
Bedroom	0.944190	0.475951	0.295927	0.060584	1.000000	0.584685	0.403867	0.025646	0.015925	0.102238	0.081350
Bathroom	0.592934	0.467038	0.127155	0.113664	0.584685	1.000000	0.321014	0.037130	0.070594	0.118971	0.052201
Car	0.406935	0.239109	0.260596	0.050514	0.403867	0.321014	1.000000	0.026780	0.002023	0.062929	0.024344
Landsize	0.025678	0.037507	0.025004	0.024558	0.025646	0.037130	0.026780	1.000000	0.009695	0.010833	0.006854
Lattitude	0.015948	0.212934	0.130723	0.406104	0.015925	0.070594	0.002023	0.009695	1.000000	0.357634	0.047086
Longtitude	0.100771	0.203656	0.239425	0.445357	0.102238	0.118971	0.062929	0.010833	0.357634	1.000000	0.065988
Propertycount	0.081530	0.042153	0.054910	0.062304	0.081350	0.052201	0.024344	0.006854	0.047086	0.065988	1.000000

Table 3: Cross-correlation between variables

There are two columns with very high correlation factor (0.94) between each other namely *Rooms* and *Bedrooms*. For now we will take it in consideration and continue to explore the data.

2.2 Exploratory Visualization

Based on *Figure 1* and *Figure 2* let's examine the dependencies between input variables and target variable. One the one hand the more rooms, bedrooms, bathrooms, number of car spots the higher is the price (till definite amount of each item). In case of bedrooms there is one outlier with value 20 as well as *Landsize* with 433014.
We can also determine inverse dependency between price and distance to central business district. By examining the longitude and latitude we can establish cirtain areas in the city with much higher price levels.
The same considerations affects the postcode (zip code). *Postcode* and *Propertycount* represent numerical categories. These two variables also must be preprocessed. For this purpose we will use the *Label Encoder* class. Label Encoder converts the categorical labels into value between 0 and n_classes-1 [10]. The problem here is, since there are different numbers in the same column, the model will misunderstand the data to be in some kind of order, 0 < 1 < 2. One hot encoding will also not solve this problem because of a huge amount of unique values in this two columns (Table 4).
The price depends also on type of the apartment and method how it was sold.
Figure 3 provides an insight in the distribution of values. On this plot the outliers were already eliminated. For further calculations the *Landsize* will be limited to 0...1500 and the Price to 0.2*10^6...2.2*10^6. For comparison the plot has the distribution of price and land size before and after outlier elimination.

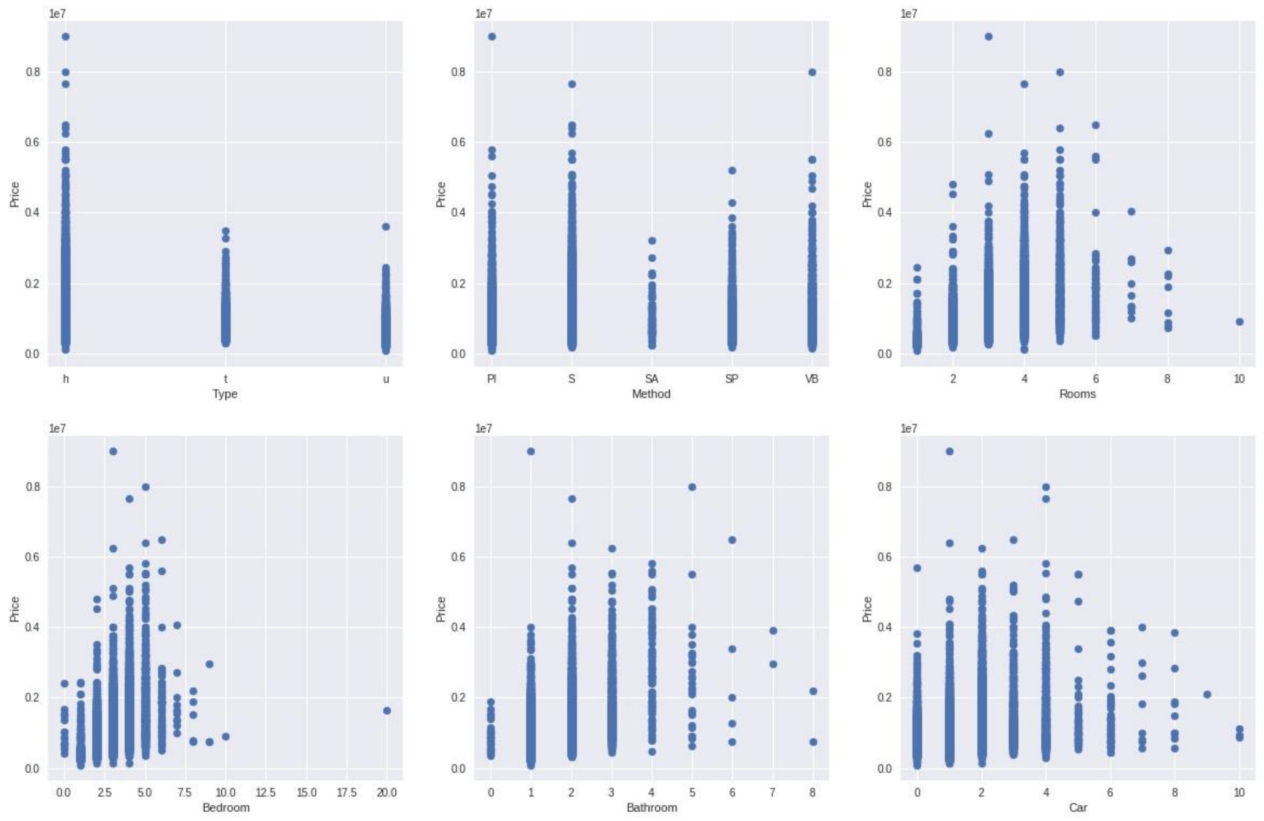


Fig. 1: Dependencies between feature and target variables (group 1)

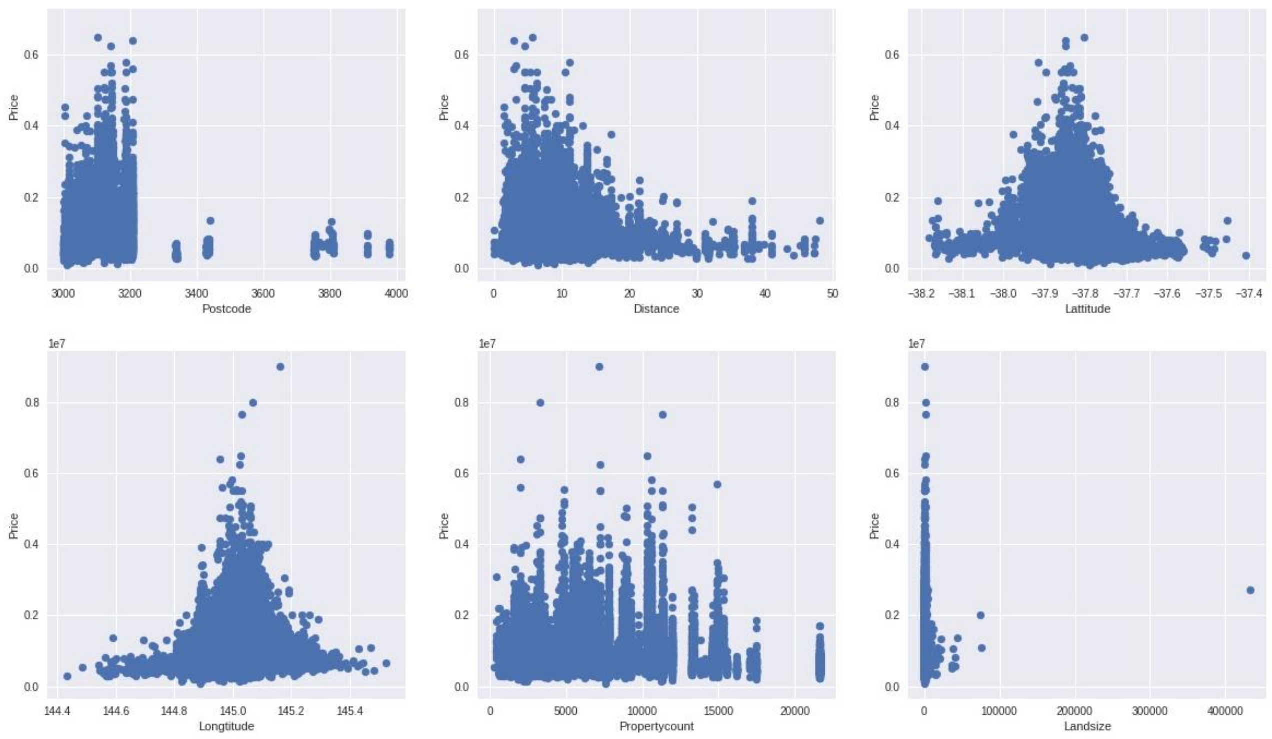


Fig. 2: Dependencies between feature and target variables (group 2)

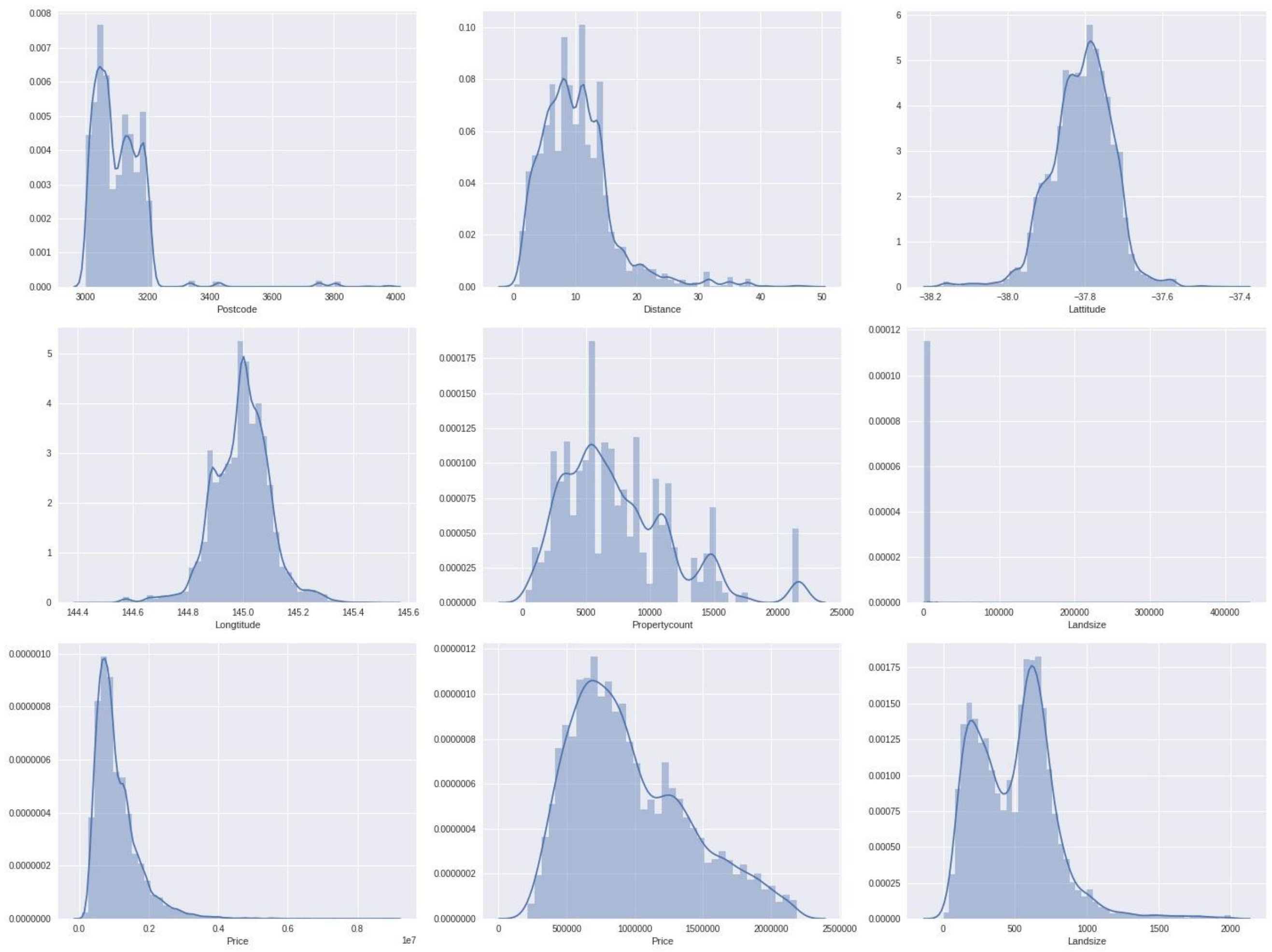


Fig. 3: Values distribution

Not all features have the same impact and are important for calculation. For defining which feature must be included in calculations and which one can be dropped feature selection will be applied. Feature selection, the process of finding and selecting the most useful features in a dataset, is a crucial step of the machine learning pipeline. Unnecessary features decrease training speed, decrease model interpretability, and, most importantly, decrease generalization performance on the test set [11]. With tree-based machine learning models, such as *RandomForestregressor*, we can find *feature importance*. The absolute value of the importance is not as important as the relative values, which we can use to determine the most relevant features for a task [11].

After preprocessing the data there are 26 features left. *Figure 4* and *Figure 5* show clearly how each of them impacts the calculations. There are only eight features which have more than 7% of importance. Rest features have less than 3%. These features will be dropped.

Table 4 shows how dropping columns with importance less 3% impacts the calculation. By decreasing the calculation time on **40%** RMSE increases only on **5.2%** and r2_score decreases on **2.6%**.

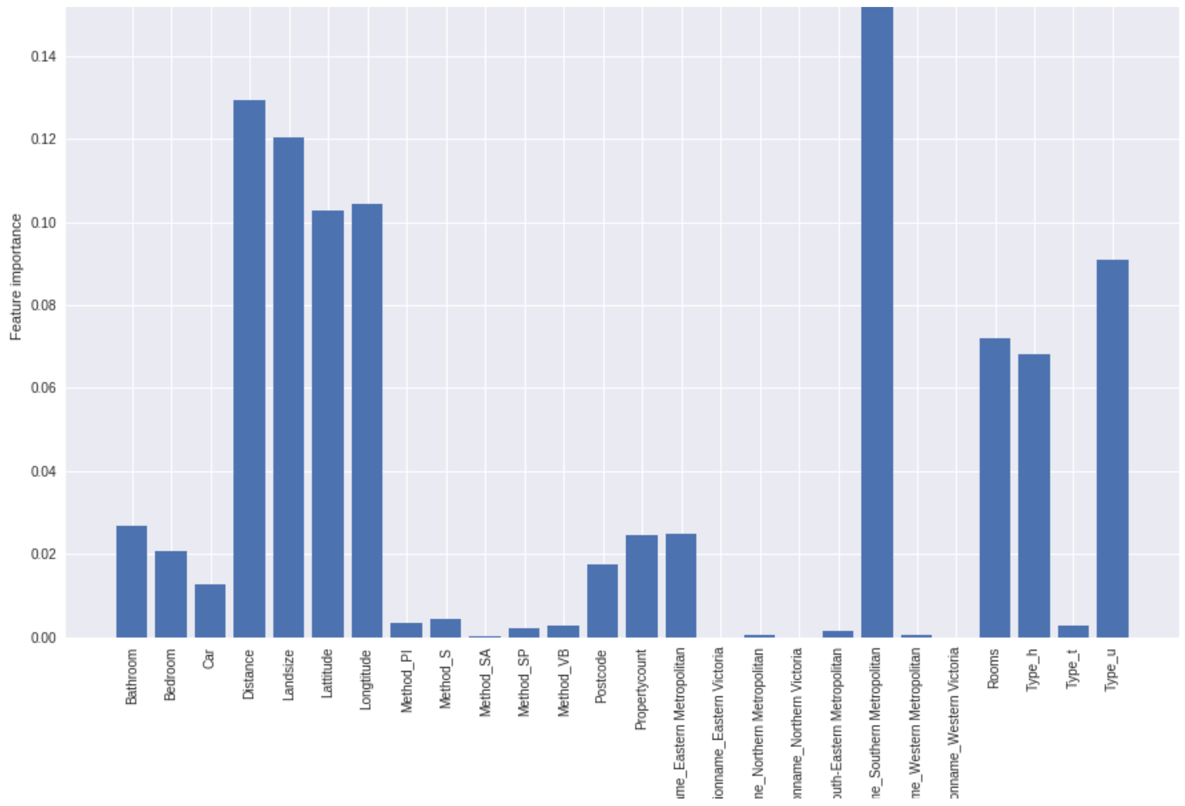


Fig. 4: Feature importance

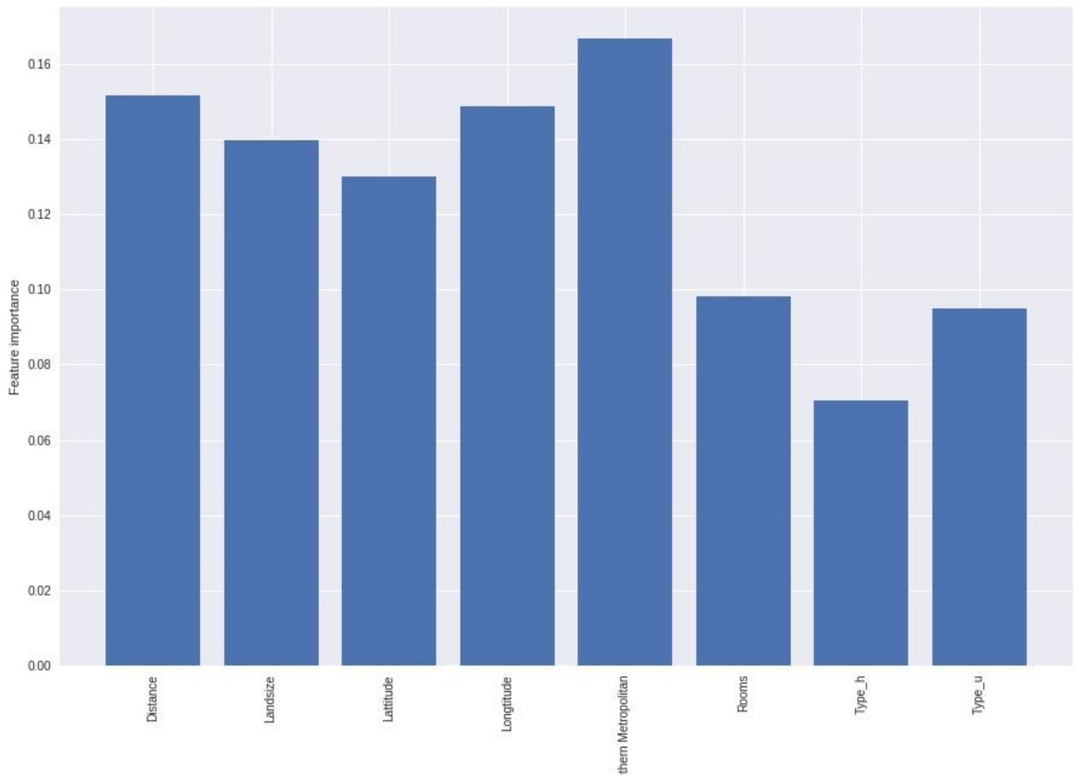
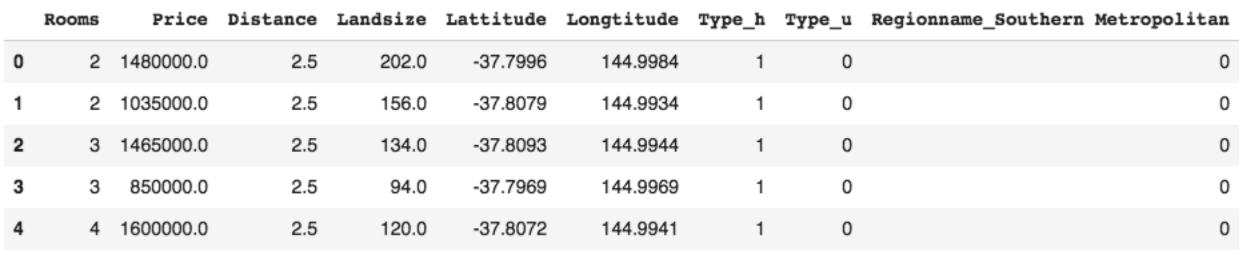


Fig. 5: Feature importance after dropping less features

	Fit-predict time, sec	**delta, %**	**RMSE,$**	**delta, %**	**R2_score, [-]**	**delta, %**
Before dropping columns	5.108	-	178440.788	-	0.816	-
After dropping columns	3.062	↓40	188239.998	↑5,2	0.795	↓2.6

Table 4: Impact of dropping unimportant features

Table 5 consists of the cleaned and preprocessed data before splitting it into target (X) and feature (y).



	Rooms	Price	Distance	Landsize	Lattitude	Longtitude	Type_h	Type_u	Regionname_Southern Metropolitan
0	2	1480000.0	2.5	202.0	-37.7996	144.9984	1	0	0
1	2	1035000.0	2.5	156.0	-37.8079	144.9934	1	0	0
2	3	1465000.0	2.5	134.0	-37.8093	144.9944	1	0	0
3	3	850000.0	2.5	94.0	-37.7969	144.9969	1	0	0
4	4	1600000.0	2.5	120.0	-37.8072	144.9941	1	0	0

Table 5: Cleaned and preprocessed data

2.3 Algorithms and Techniques

The aim of this project is to find a model which will best fit and "work" with given data set. For this purpose I will try different methods and models provided in *scikit-learn* library. Main of them are the following:

1. Linear regression models: simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent and dependent variable [12]
2. Ensemble learning: ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking) [13]
3. Boosting algorithms: boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners - models that are only slightly better than random guessing, such as small decision trees - to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds. The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction [14]
4. Regression trees: uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves) [15].
5. K-Nearest Neighbors: the algorithm uses 'feature similarity' to predict 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 [16]

For each model cross-validation-score with five folds [17], RMSE and calculating time will be estimated. *Cross-validation* is a statistical method of evaluating generalization performance that is more stable and thorough than using a split into a training and a test set. In cross-validation, the data is instead split repeatedly and multiple models are trained [1]. There are several benefits to using cross-validation instead of a single split into a training and a test set. When using cross-validation, each example will be in the training set exactly once: each example is in one of the folds, and each fold is the test set once. Therefore, the model needs to generalize well to all of the samples in the dataset for all of the cross-validation scores. Having multiple splits of the data also provides some information about how sensitive our model is to the selection of the training dataset. Another benefit of cross-validation as compared to using a single split of the data is that we use our data more effectively. The main disadvantage of cross-validation is increased computational cost.

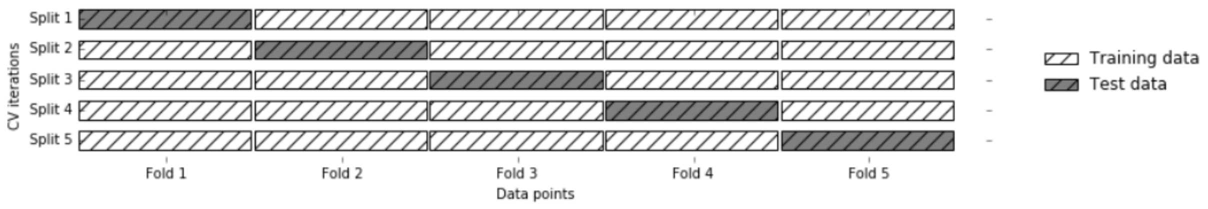


Fig. 6: Data splitting in five-fold cross-validation

Based on results the best few models will be chosen and the hyper parameters of those models will be tuned to obtain better result. The most commonly used method is grid search, which basically means trying all possible combinations of the parameters of interest [1]. In order to decrease the calculation time in this project *RandomizedSearchCV* will be applied [18]. *GridSearchCV* is computationally expensive, especially by searching over a large hyper parameter space and dealing with multiple hyper parameters. A solution to this is to use *RandomizedSearchCV*, in which not all hyper parameter values are tried out. Instead, a fixed number of hyper parameter settings is sampled from specified probability distributions [19].

The second solution in this project consist in developing a neural net using *Keras* library. Neural network is based on a collection of connected nodes called neurons. Each connection can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer) [20].

To build a neural net, the following main steps need to be completed:

- define the type of layers
- define a number of layers
- choose the number of neurons in each layer
- define an activation function
- define a loss function
- define an optimizer
- choose the batch size
- choose the number of epochs

After the neural net is built, it will be trained, if necessary optimized and the performance of the neural net will be checked on training set and finally compared to the benchmark model and regression model from solution #1.

2.4 Benchmark

For the benchmark model was chosen a *LinearRegression* model [21]. After training and evaluating the model on test data following results were achieved.

RMSE, $	264926.625
R2_score, [-]	0.593

Tab. 6: Benchmark model

Figure 7 represents how the model fits the target value.

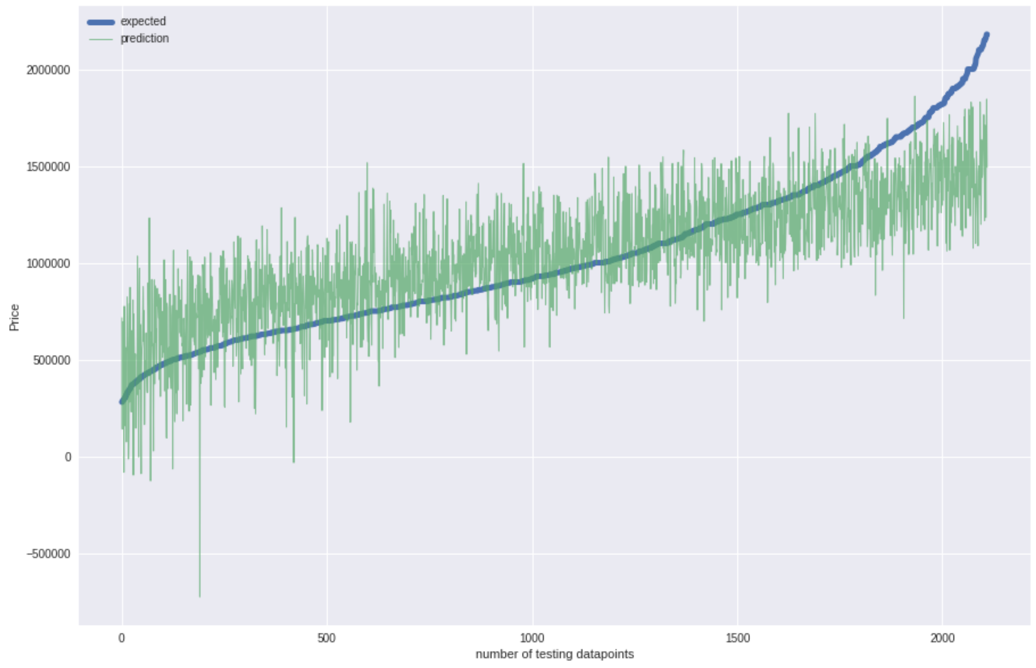
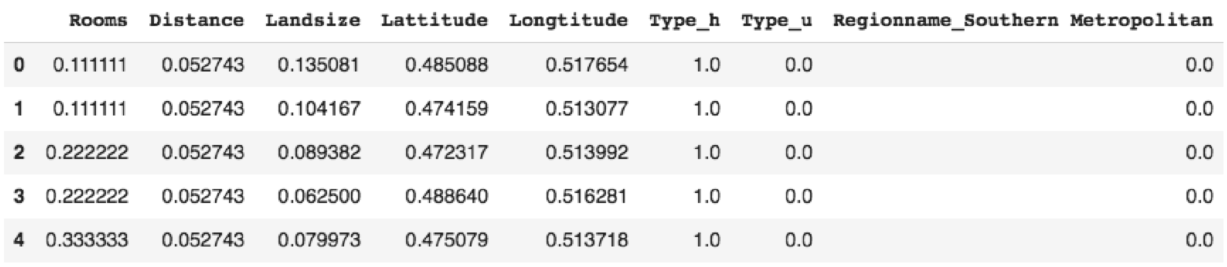


Fig. 7: Target value prediction with benchmark model

3. Methodology

3.1 Data Preprocessing

Main data preprocessing steps, such as handling missing values, outliers, one-hot-encoding, investigating of feature importance, were described in section *2.2 Exploratory Visualization*. As mentioned in section *2.1 Data Exploration*, the data have different ranges. This can negatively affect the calculations. That's why one more preprocessing step is needed namely feature scaling. For this purpose *MinMaxScaler* class will be used [22]. *MinMaxScaler* transforms features by scaling each feature to a given range (typical range 0...1).



	Rooms	Distance	Landsize	Lattitude	Longtitude	Type_h	Type_u	Regionname_Southern Metropolitan
0	0.111111	0.052743	0.135081	0.485088	0.517654	1.0	0.0	0.0
1	0.111111	0.052743	0.104167	0.474159	0.513077	1.0	0.0	0.0
2	0.222222	0.052743	0.089382	0.472317	0.513992	1.0	0.0	0.0
3	0.222222	0.052743	0.062500	0.488640	0.516281	1.0	0.0	0.0
4	0.333333	0.052743	0.079973	0.475079	0.513718	1.0	0.0	0.0

Tab. 7: Input values after feature scaling procedure

3.2 Implementation

Since the aim of the project is to find two solutions of the problem, the implementation process can be split into following stages:

1. Regression models
    - 1.1. Import models from *scikit-learn* and *xgboost* libraries
    - 1.2. Looping over each model[2] and calculating of:
        - 1.2.1. Cross validation score
        - 1.2.2. Root mean square error
        - 1.2.3. Time needed for training and testing each model
    - 1.3. Choosing the model(s) with best result
    - 1.4. Tuning of hyper parameters for chosen models
2. Neural net
    - 2.1. Net architecture
        - 2.1.1. Sequential model with dense layers
        - 2.1.2. Model layers
            - 2.1.2.1. Input layer with 8 inputs for each feature
            - 2.1.2.2. First layer: 240 neurons with “relu” activation
            - 2.1.2.3. Second layer: 120 neurons with “relu” activation
            - 2.1.2.4. Third layer: 60 neurons with “relu” activation
            - 2.1.2.5. Output layer: 1 neuron (because of regression problem) with “linear” activation
        - 2.1.3. For each hidden layer Dropout layer with 10% dropout
        - 2.1.4. Optimizer “Adam”
        - 2.1.5. Loss function “”mean square error”
        - 2.1.6. Learning rate 0.001
        - 2.1.7. Batch size 500
        - 2.1.8. Number of epochs 500
    - 2.2. Splitting the training set into 10 folds using K-Fold cross validation
    - 2.3. Training the model
    - 2.4. Saving the model architecture into *model.yaml* and weights into *model.h5*
    - 2.5. Calculation RMSE for each fold

[2] At this step all models have the standard hyper parameters

Following additional functions were also built and used during implementation process (section *1.2 Helpful functions* in jupyter notebook):

1. model_run(): function was used to find the most important features. Based on attribute *feature_importances_* of *RandomForestRegressor* class
2. missing_median(): Converts all missing values in the specified column to the median
3. plotting_loss(): plots loss learning curve
4. chart_regression(): plots expected target value vs. predicted target value
5. plot_history(): plots training and validation errors
6. model_opt(): tuning of hyper parameters

During implementation and coding process I haven't faced huge difficulties or problems. Before I started to work on this project I have made a plan with main steps and followed this plan during implementation process. On order to make code readable for myself and other persons a tried to keep it as simple as possible, make short notes and comments, divide the code in sections. These two things helped me to avoid big difficulties during coding and implementation.

3.3 Refinement

The architecture of neural net described in the section below has achieved the best results. This was done not in a moment but over many iterations and attempts. By choosing and developing the architecture of neural net following parameters were tried out:

- number of layers: 2, 3, 4
- number of neurons in each layer: (40, 20, 10)...(512, 256, 128)
- activation functions: "relu", "tanh"
- kernel initializer: uniform, lecun_uniform, normal, zero, glorot_normal, glorot_uniform, he_normal, he_uniform
- kernel regularizer: regularizers.l2(0.01)
- activity_regularizer: regularizers.l1(0.01)
- optimizer: RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam
- dropout_rate = 0.0, 0.1, 0.2, 0.3, 0.4
- batch_size = 10, 50, 100, 200, 300, 400, 500
- epochs = 100, 200, 300, 400, 500

To find the best regression model the following hyper parameters were tuned:

- n_estimators: 100, 200, 300
- min_samples_split: 2, 3, 4, 5, 6, 7, 8
- min_samples_leaf: 2, 3, 4, 5, 6, 7, 8
- max_depth: 5, 6, 7, 8, 9, 10
- learning_rate: 0.001, 0.01, 0.1

4. Results

4.1 Model Evaluation and Validation

4.1.1 Regression models

Table 8 and *Figure 8* summarize the results of validating regression models on testing set.

Regressor	Cross valid. score[3], [-]	Cross valid. score[4], [-]	RMSE[3], $	RMSE[4], $
DecisionTree	0.605	-	262938.550	-
GradientBoosting	0.751	0.801	207075.767	185476.095
RandomForest	0.773	0.757	196337.977	204977.020
KNeighbors	0.737	-	209380.641	-
PLSRegression	0.575	-	272572.751	-
BayesianRidge	0.587	-	264948.768	-
HuberRegressor	0.549	-	277665.674	-
ExtraTrees	0.763	0.687	202382.922	232248.944
AdaBoost	0.525	-	297885.404	-
XGBRegressor	0.753	0.799	207075.969	186389.297

Tab. 8: Performance of the regression models with standard hyper parameters

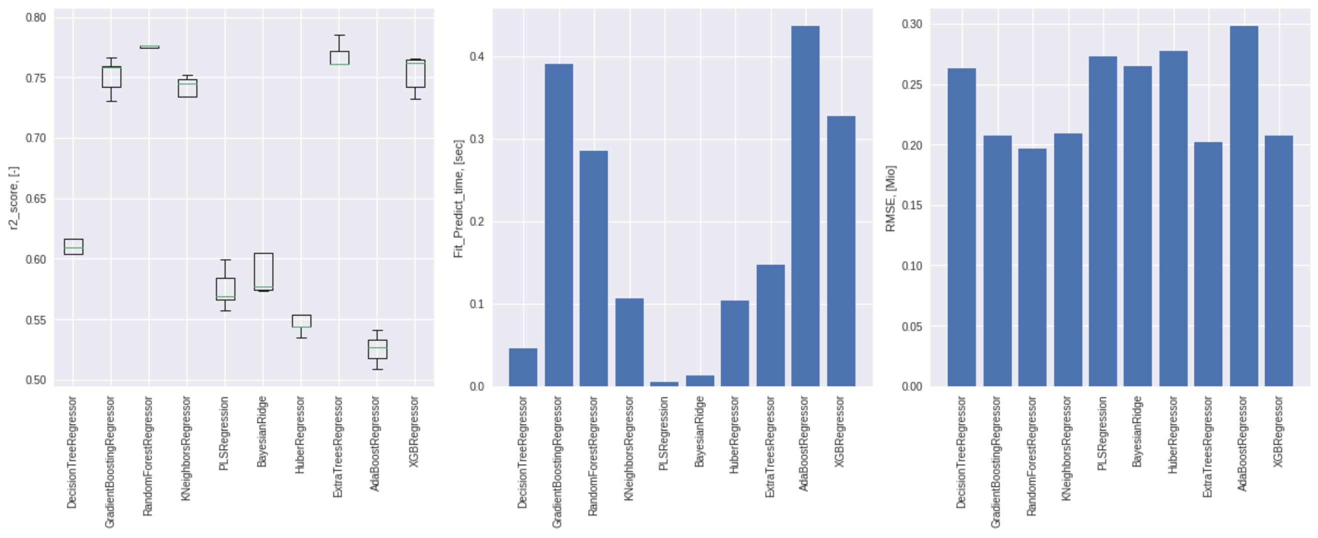


Fig. 8: Performance of the regression models with standard hyper parameters

3 Before tuning of hyper parameters

4 After tuning of hyper parameters

Based on *Table 8* and *Figure 8* four best regression models can be established. These are:

- GradientBoostingRegressor
- RandomForestRegressor
- ExtraTreesRegressor
- XGBRegressor
- KNeighborsRegressor

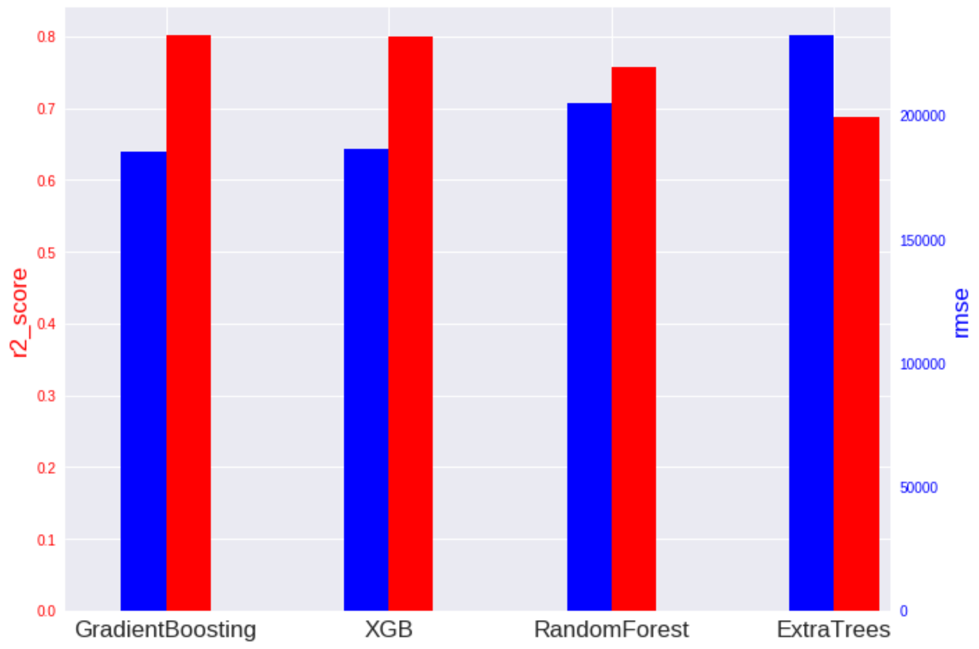


Fig. 9: Regression models with best result after tuning of hyper parameters

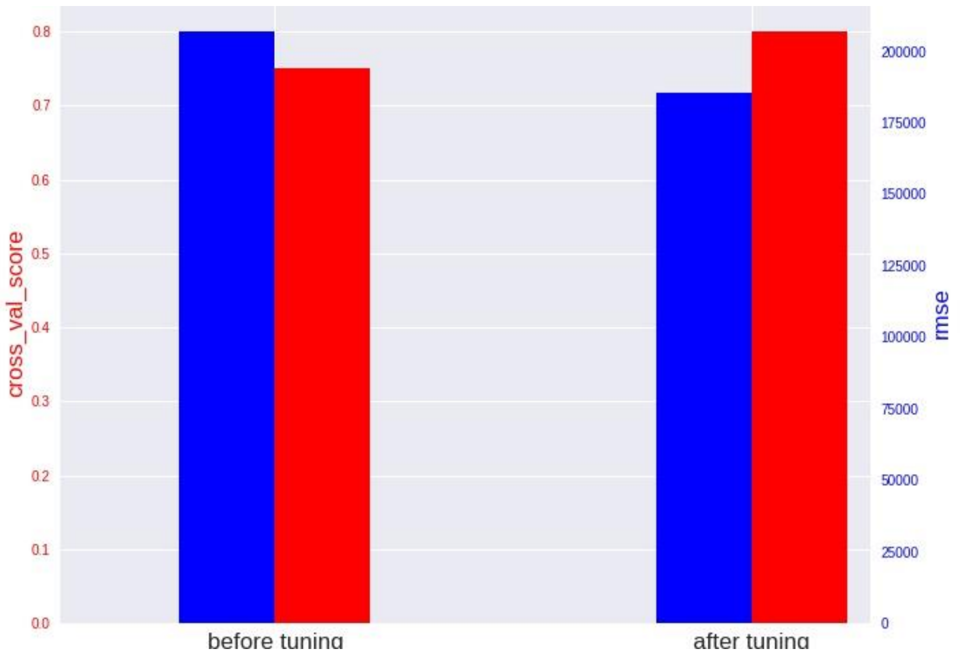


Fig. 10: Improvement of GradientBoostingRegressor after tuning of hyper parameters

GradientBoostingRegressor achieved the best result among all regression models for this particular dataset. After tuning the hyper parameters RMSE has decreased on **10%** and r2_score increased on **5,75%. Compared to the benchmark model RMSE has decreased on 39.4% and r2_score increased on 30,0 %.**

Based on information from *Table 8* and *Figure 9* the conclusion regarding robustness of the *GradientBoostingRegressor* can be made. Before tuning the model the cross-validation-score on training data was **0.751**. r2_score of the model on unseen data (testing set) **0.736**.
After tuning the model's hyper parameters was achieved new r2_score **0.8** on testing (unseen) data.
It means that model was well trained and generalizes the data and dependencies between features and target variable rather than remembers it. It confirms that the model is trustful.

Figure 11 shows how tuned GradientBoostingRegressor fits the target value.

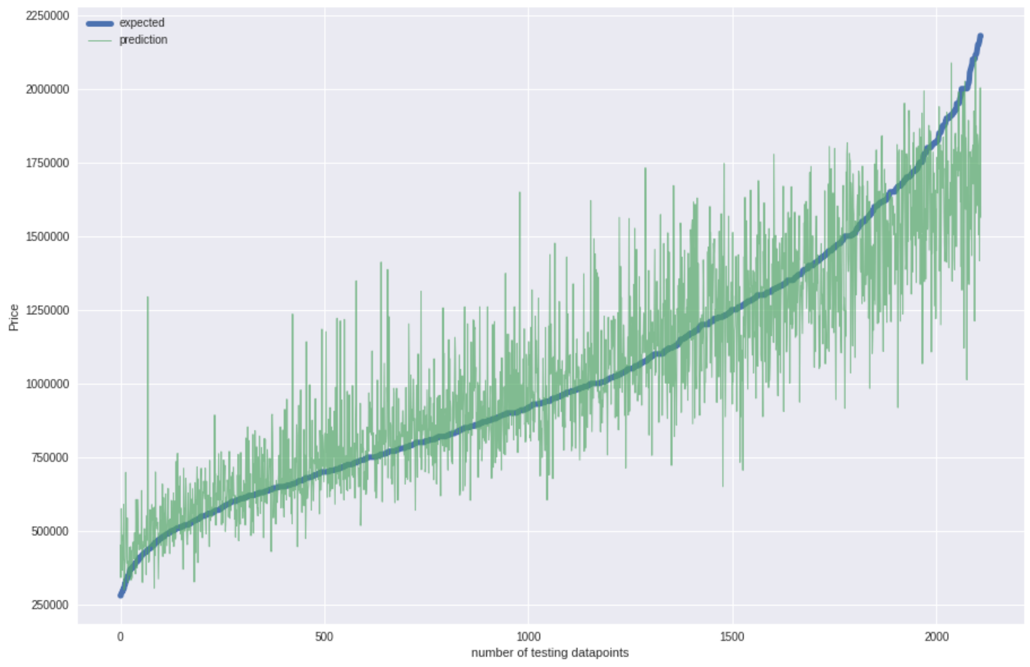


Fig. 11: Target value prediction with GradientBoostingRegressor

4.1.2 Neural net

After training the neural net and evaluating its performance on testing set, following results were achieved:

- RMSE: 235927.672
- r2_score: 0.677

Compared to the benchmark model RMSE by applying the neural net to predict the target value has decreased on 10.9% and r2_score increased on 14,2%.

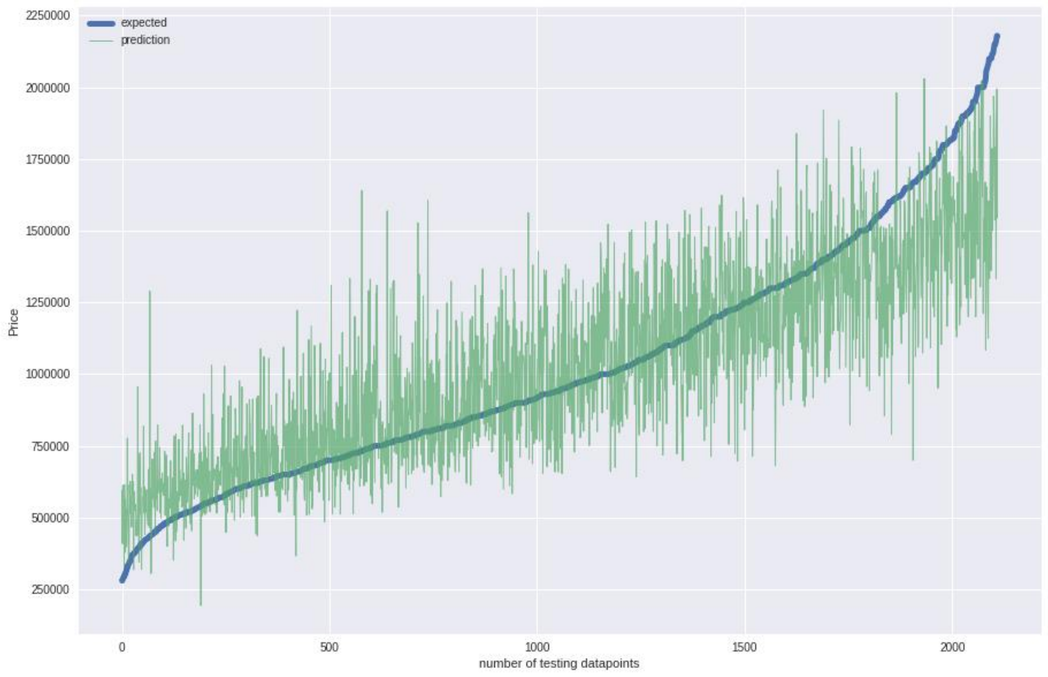


Fig. 12: Target value prediction with neural net

4.2 Justification

Both the neural net and the *GradientBoostingRegressor* perform much better on predicting the price than the benchmark model. In *jupyter notebook* in section *2.3.4 Handling outliers* the range of the *Price* column was limited to 2*10^6. Let's calculate the prediction accuracy of each model.
The prediction accuracy of neural net:

$$\frac{235927.672}{2000000}*100\% = 11.8\%$$

The prediction accuracy of *GradientBoostingRegressor*:

$$\frac{185577.497}{2000000}*100\% = 9.3\%$$

It means that the *GradientBoostingRegressor* is the model we are looking for, even though the neural net has achieved also a gut result and performs only 2.5% worse than the *GradientBoostingRegressor*.
If we compare the RMSE of both models with the mean value of our data set from Table 2 1.076*10^6 the prediction accuracy would be between 17.2% and 21.2%. I would not recommend to use the both models for real world application. Indeed, for building and applying such models for the real world problems much more factors must be taken into account and features supplied for training models.

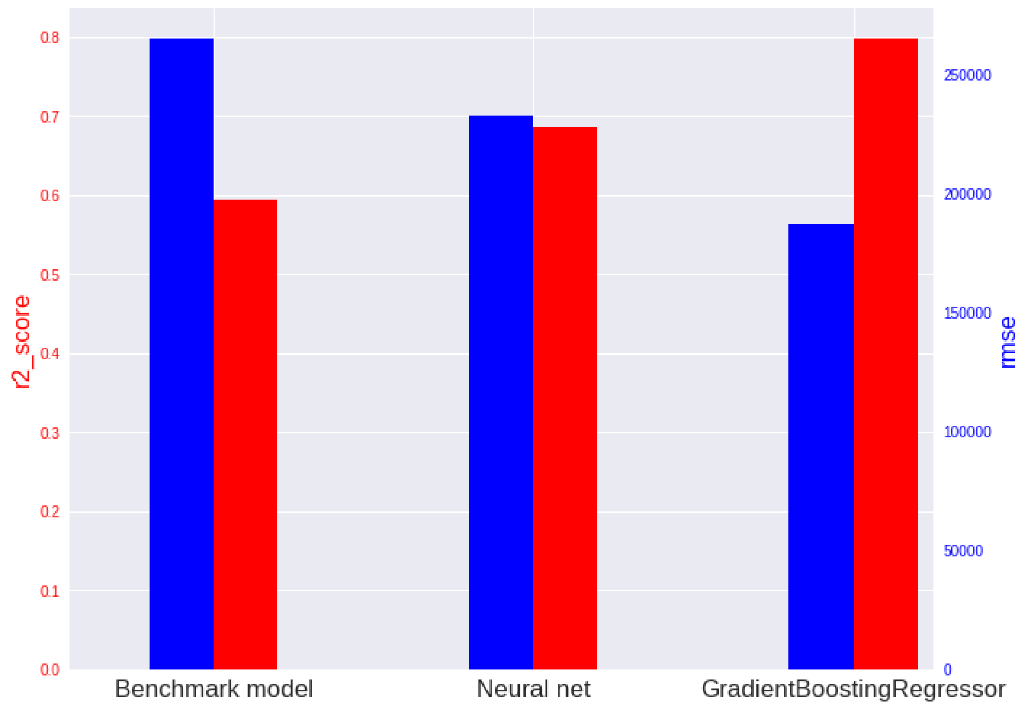


Fig. 13: Comparison of target value prediction with benchmark model, neural net and regression model

5. Conclusion

5.1 Reflection

This project was dedicated to price prediction of the real estate items in Melbourne based on their description and parameters such as number of rooms, bedrooms, car spots, land size etc. Two approaches were chosen to solve the problem. The first one was to find the best regression model and the second one - to build a neural net. The both solutions performed much better as the benchmark model and also achieved average accuracy about 10% of the predicted price range, which in my opinion is a great result.
The main steps of the project were following:

1. Data exploration, cleaning and preprocessing
2. Exploring dependencies between features and target variable
3. Investigating of feature importance
4. Implement a benchmark model
5. Implement various regression models
6. Hyper parameter tuning
7. Building, training and evaluation of neural net performance

The most difficult and time consuming for me were steps 1 and 7. I have spent a lot of time to understand how to clean the data and preprocess different numerical and categorical values. I have noticed that the result and performance of the models strong depend on the quality of input data.

The challenge of implementing a neural net was for me choosing the right architecture and amount of hyper parameters. As described in section *3.3 Refinement* i have tried a lot of architectures of neural net with different hyper parameters to find the neural net which performs well on this data set.

5.2 Improvement

By working on this project i have tried a lot of basic machine learning techniques. Certainly there are several possibilities for improvement:

- Quality improving of input data
- Elevation of time investment in feature engineering
- Finding better parameters of neural net using grid search technique as described in [23]

6. References

[1] “Introduction to Machine Learning with Python” by Andreas C. Müller and Sarah Guido

[2] Dean De Cock, Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project *http://jse.amstat.org/v19n3/decock.pdf*

[3] House Prices: Advanced Regression Techniques *https://www.kaggle.com/c/house-prices-advanced-regression-techniques/kernels*

[4] Input data *https://www.kaggle.com/dansbecker/melbourne-housing-snapshot/data*

[5] Root mean square error *https://en.wikipedia.org/wiki/Root-mean-square_deviation*

[6] Coefficient_of_determination *https://en.wikipedia.org/wiki/Coefficient_of_determination*

[7] Data imputation *https://www.theanalysisfactor.com/seven-ways-to-make-up-data-common-methods-to-imputing-missing-data/*

[8] Correlation *https://www.datascience.com/blog/introduction-to-correlation-learn-data-science-tutorials*

[9] One-hot-encoding *https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html*

[10] Label encoding *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html*

[11] Feature selection *https://towardsdatascience.com/a-feature-selection-tool-for-machine-learning-in-python-b64dd23710f0*

[12] Linear regression models *https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a*

[13] Ensemble learning https://blog.statsbot.co/ensemble-learning-d1dcd548e936

[14] Boosting algorithms *https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a*

[15] Regression trees *https://en.wikipedia.org/wiki/Decision_tree_learning*

[16] K-Nearest Neighbors *https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/*

[17] Cross validation score *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html*

[18] RandomizedSearchCV *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html*

[19] RandomizedSearchCV *https://campus.datacamp.com/courses/supervised-learning-with-scikit-learn/fine-tuning-your-model?ex=11*

[20] Neural net *https://en.wikipedia.org/wiki/Artificial_neural_network*

[21] Linear regression *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html*

[22] MinMax Scaler *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html*

[23] Grid Search Hyperparameters for Deep Learning Models
https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/