Bilkent University

Industrial Engineering

IE 490

Real Estate Value Control and Reference System Project Using Neural Nets and Other Statistical Learning Methods

---

Author: Kazım Sanlav

Supervisor: Dr. Savaş Dayanık

May 7, 2018

Abstract

The aim of our project is to create a system for predicting real estate prices for use by valuation agencies, banks or the government. This work will help evaluate the fair market value of real estate. For this purpose, we have examined the relationship between the fair market values and attributes of the properties that have been subject to real valuation reports in the past periods. Our dataset includes information about properties in districts of Ankara that were subject to valuation reports. However, the dataset does not have a proper structure and includes lots of missing values. We have dealt with these issues and proposed a Random Forest model for price prediction. Currently, we are working on building Neural Network model for better price prediction.

1.Problem Definition

Real estate values are determined by a group of experts by getting the average of price suggestions. In this paper, we aim to create an autonomous and unbiased expert system that imitates the decision making ability of human experts. It can potentially be used by valuation agencies, banks and governments by determining fair market values of real estate.

2.Preparing Data

To train the system, we used previous real estate valuation reports by examining the fair market values and their relations with the attributes of the real estates. Our dataset includes information about real estate in districts of Ankara that were subject to validation reports. The features we used include number of floors, surface area, legal gross area, current area, legal fair price, age of the title, area, duration. New variable called area is created using the following formula, (pay*yüzölçümü / payda) on existing columns of row data. Similarly, duration is created using the columns ay and yil in order to calculate the age of the building.

Initially, the dataset did not have a proper structure and it contained lots of missing values. To be able to use the data for training a model, the data had to be prepared. Generally, data is cleaned by converting string values to integers. For example, in the number of floors column, ‘zemin’ was converted to 0, ‘birinci kat’ was converted to 1 and so on. Date column

was also made consistent, for example August 2009 or 1.09.2009 was converted to 01/09/2009. Similar conversions were applied to most of the columns by using regular expressions. In addition, logical ranges were applied to columns. For example, year of the construction ought to range from 1930 to 2017 therefore; dates of construction outside this range were removed from the data. Finally, districts that contained less than 50 houses were removed because these do not provide sufficient data for training the models to predict real estate prices of different districts. After all the cleaning steps mentioned above, rows that had missing values were removed from the data. All this pre-processing led to loss of more than half of the data.

Columns

- *tapunun_tarihi* has **51%**
- *bagimsiz_bolum_arsa_pay* has **30%**
- *bagimsiz_bolum_arsa_payda* has **30%**
- *bagimsiz_bolum_kat* has **15%**
- *adil_piyasa_degeri_mevcutdurumdegeri* has **17%**
- *yuzolcumu* has **13.4 %**

missing values

Table 2.1 Highest percentages of missing values for columns.

Our dataset consists of lots of regions and within regions there are districts where houses are located. As houses prices vary between locations and we do not have any features about a location information such as latitude and longitude, we decided to focus on regions separately instead of training the model on the whole dataset which consisted of about 100000 samples. There are two columns, mahalle_kod and ilce_kod, in dataset that help us separate houses by regions. To select the region for training the model, price distributions and number of samples in each region was considered. The 79th region was selected for the model training because distribution of prices was more balanced as compared to other regions and it is in top 15 most crowded regions.

	bagimsiz_bolum_kat	yuzolcumu	yasal_burut_alani	mevcut_alani	adil_piyasa_degeri_yasal_durum	tapunun_yasi	area	durat
count	537.000000	537.000000	537.000000	537.000000	5.370000e+02	537.000000	537.000000	537.000(
mean	2.277467	2310.871918	124.515829	125.274674	1.273855e+05	4.398510	71.110976	31.0353(
std	1.609010	6120.240907	62.341053	62.385877	5.758241e+04	4.284369	250.035279	12.6744(
min	-1.000000	61.000000	12.000000	58.000000	3.000000e+04	0.000000	0.172940	0.000000(
25%	1.000000	701.000000	108.000000	108.000000	1.000000e+05	1.000000	50.000000	22.0000(
50%	2.000000	964.000000	120.000000	120.000000	1.250000e+05	3.000000	57.000000	35.0000(
75%	4.000000	1346.000000	135.000000	135.000000	1.500000e+05	6.000000	66.603910	42.0000(
max	10.000000	113400.000000	1452.000000	1452.000000	1.160000e+06	23.000000	5800.000000	47.0000(

Table 2.2 Statistics about the houses in 79. region.

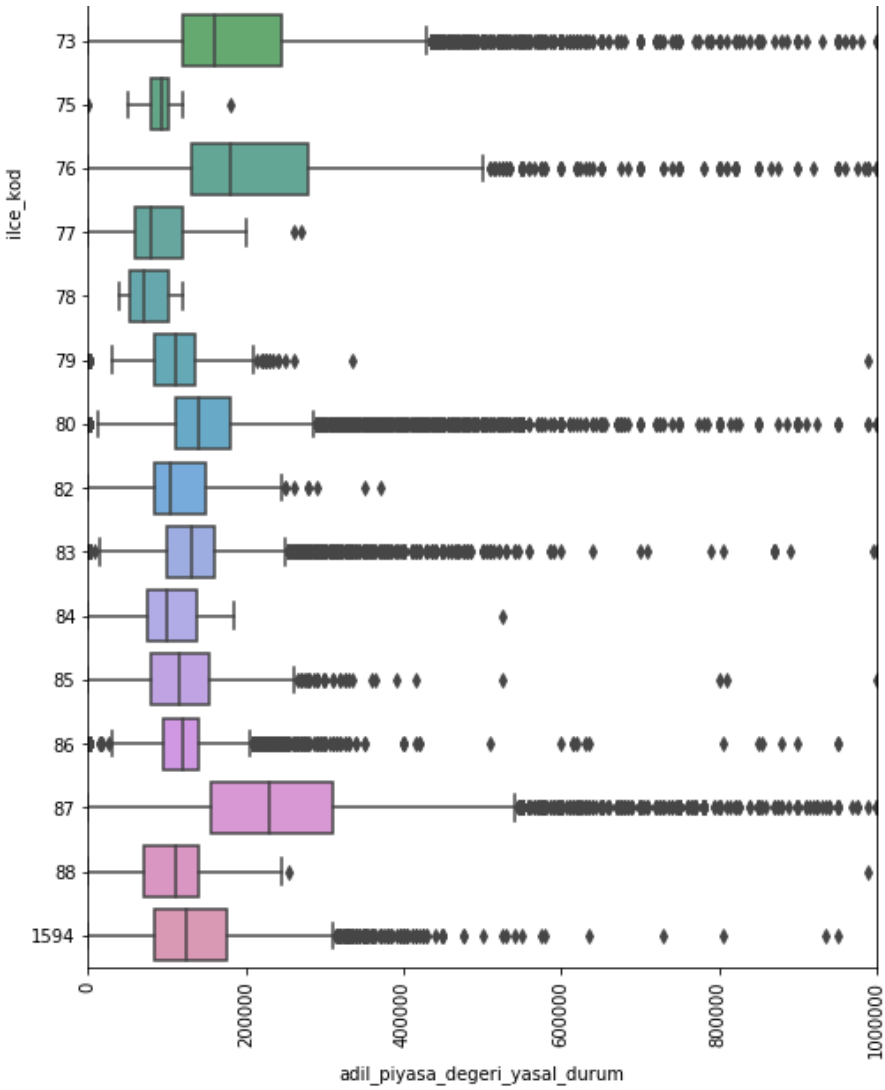


Figure 2.1 Price distributions of houses in regions

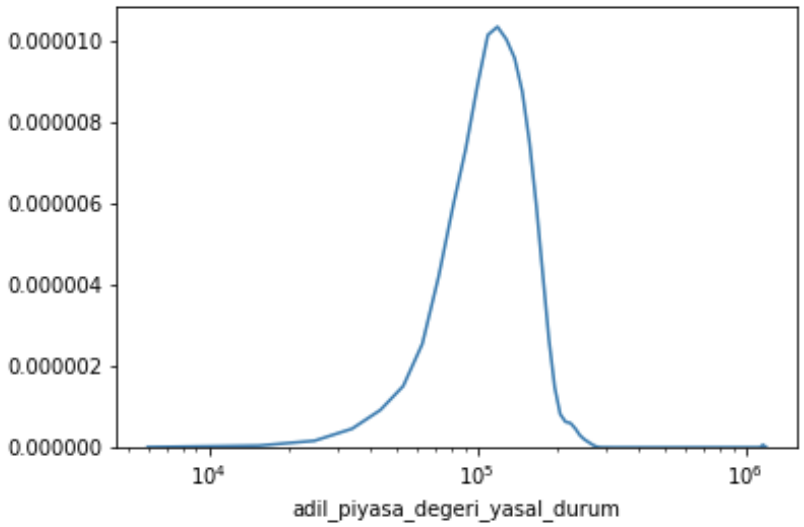
```
df.ilce_kod.value_counts().head(10)
```

```
80      11608
70       7780
83       7707
73       7526
86       6167
87       5990
65       3908
1594     1785
85       1068
76        680
Name: ilce_kod, dtype: int64
```

Table 2.3 Number of houses in region in ascending order (top 10 region)

3.Analyze and Explore the Data

After selecting the 79th region selected and determining the training data, price distribution was analyzed.

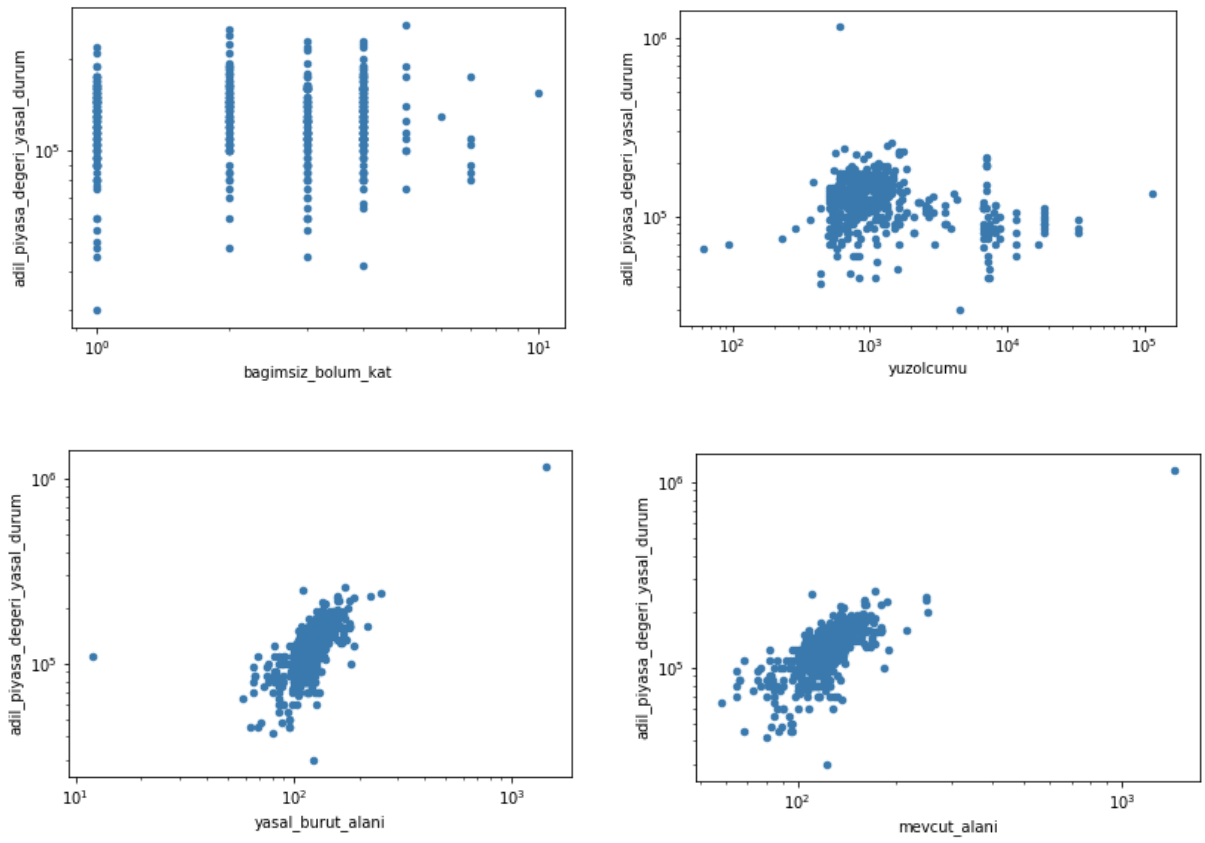


- Deviate from the normal distribution.
- Have big positive skewness.

Figure 3.1 Price distribution for 79. region

Figure 3.1 shows that price distribution has a positive skew and deviates from the normal distribution. There are less houses that have price more than 10^5 compared to houses that have price less than 10^5. This skewed distribution can affect the learning capability of the model.

After price distribution analysis, correlation of features with houses prices was analyzed. From figure 3.2, it can be seen that ‘mevcut alan’ has a positive correlation with price as points are on the diagonal axis. Other numerical features do not show any interesting relationships.



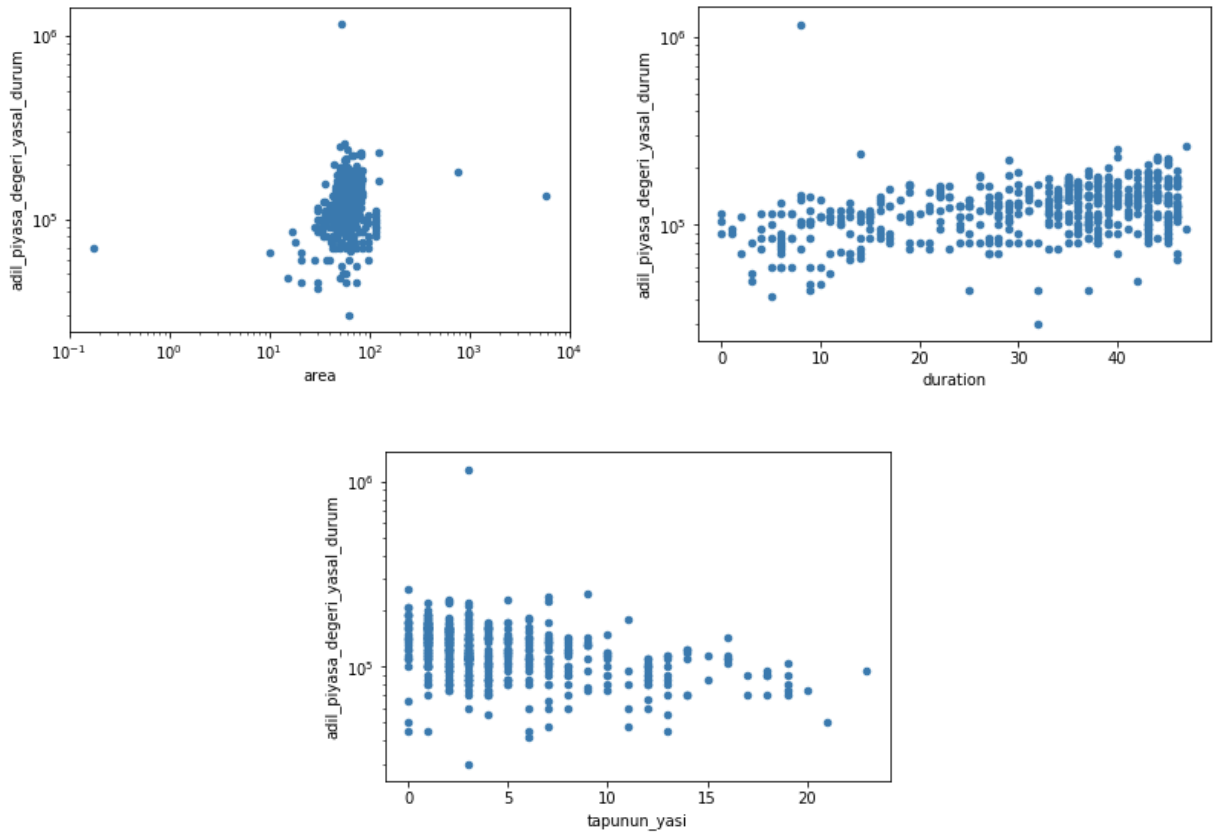


Figure 3.2 Correlation of Numerical features with house prices

Figure 3.3 Scatter plot of the data

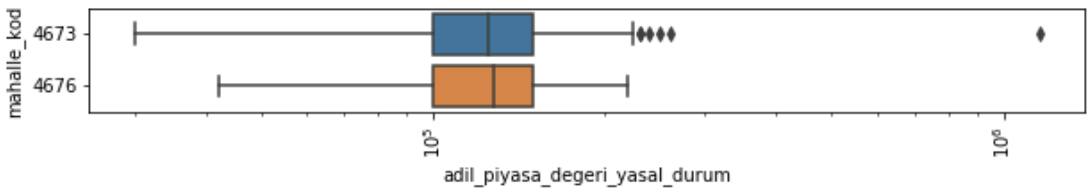


Figure 3.4 Correlation of Categorical features with house prices

When we look at the distribution of house prices by districts, 4673th district has a very far away outlier and some closer ones. This will be an issue for the model training and we will discuss on it later.

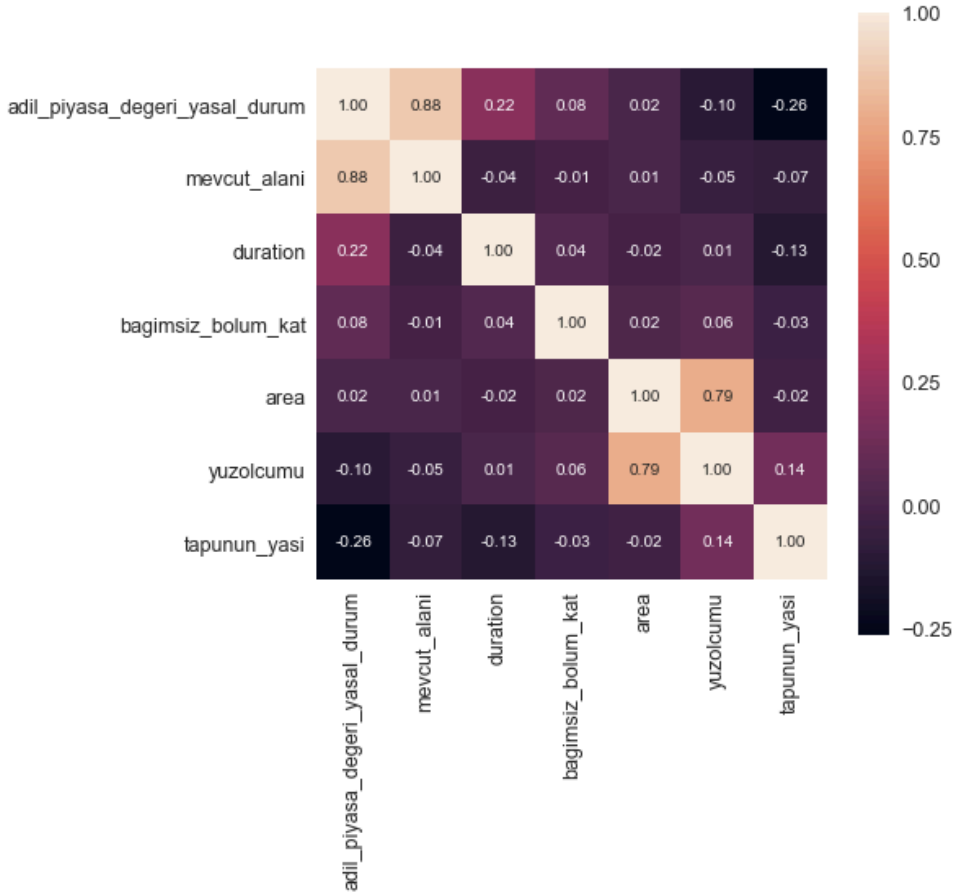


Figure 3.5 Correlation Matrix for Numerical features with house prices

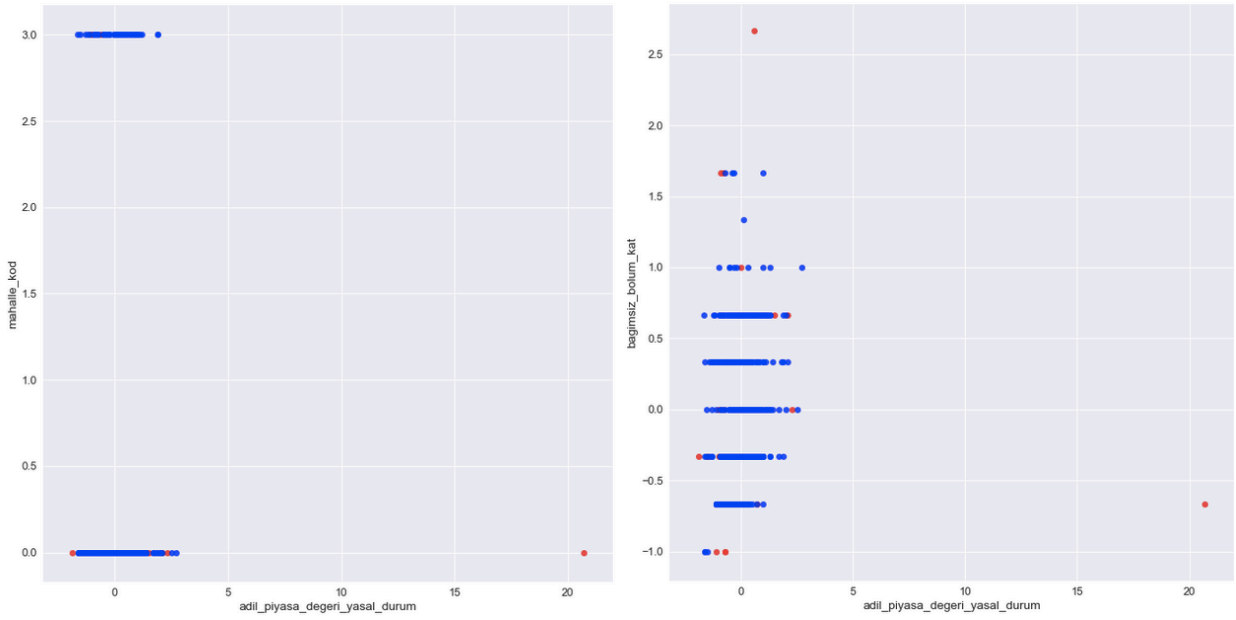
Correlation matrixes are really helpful tools to analyze correlations between target variable and also between features. From the figure above, we can see that *mevcut_alani* is 88% positively correlated with the price. It will help model to train on data more than other features. Other than that, *tapunun_yasi* and *duration* is correlated with the price. *Area* and *yuzolcumu* is also highly correlated, which should not be in the model training phase together because it will impair the

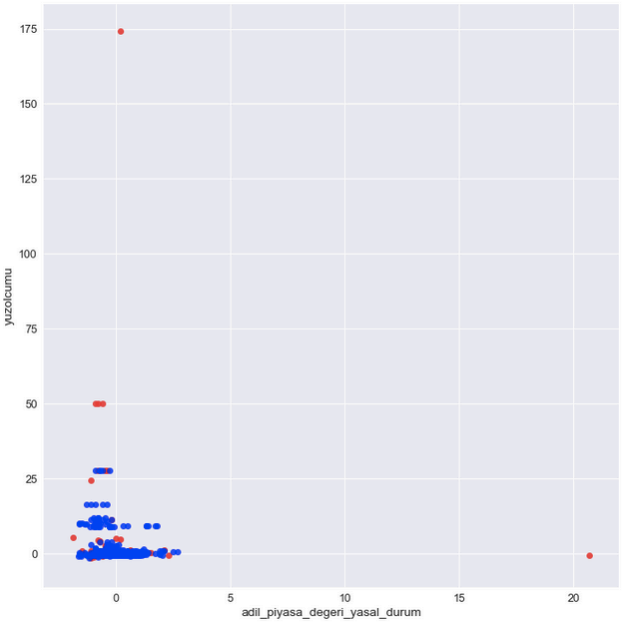
interpretability of the model and will also increase the dimension which should be avoided in unnecessary cases. *Yuzolcumu* was dropped from the data for these reasons.

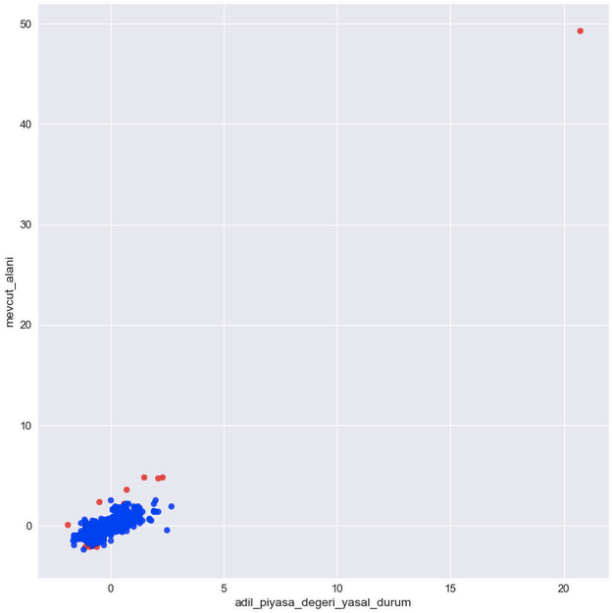
After analyzing the features and prices, we wanted to detect outliers and compare the accuracy of model with and without outlier removal. Form Figure 2.3, it can be seen that there are some outliers and generally outliers harm the model because they result in overfitting. It is also likely that outliers in this data are just typing errors. This is probable in our case because, in the earlier stages of data cleaning, we noticed a number of illogical construction year that we had to remove.

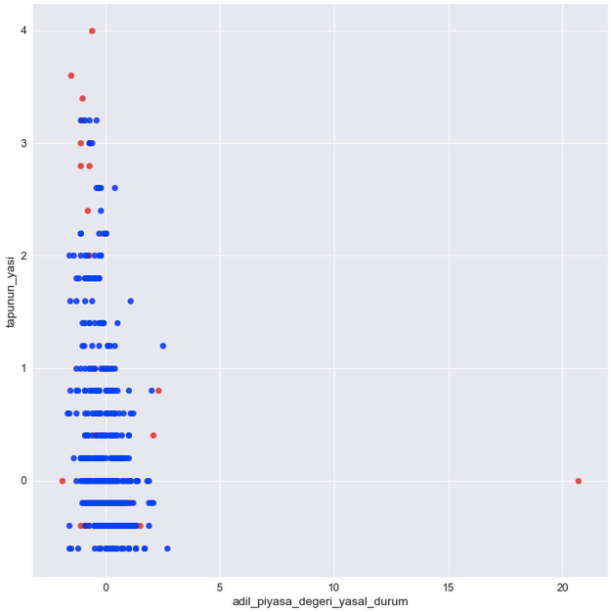
To detect outliers, *DBSCAN* is used. It is a density based clustering algorithm focused on finding neighbors by density on an *n-dimensional sphere* with radius *e*, parameter of the DBSCAN. A cluster can be defined as the maximal set of *density connected points* in the feature space. (Ester 226)

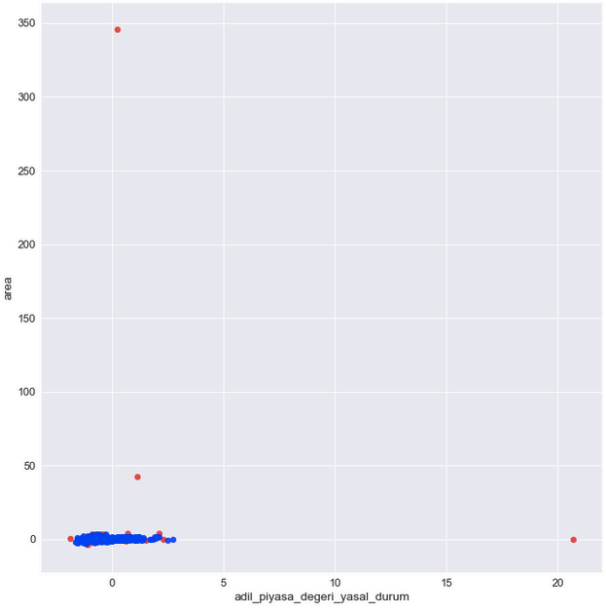
Parameter *e* is selected such that ratio of outliers suggested by the DBSCAN is less than 1% of the data. But before using it, data has to be transformed by *RobustScaler,* as scale of features can affect DBSCAN highly.




yuzolcumu
adil_piyasa_degeri_yasal_durum


mevcut_alani
adil_piyasa_degeri_yasal_durum


tapunun_yasi
adil_piyasa_degeri_yasal_durum


area
adil_piyasa_degeri_yasal_durum

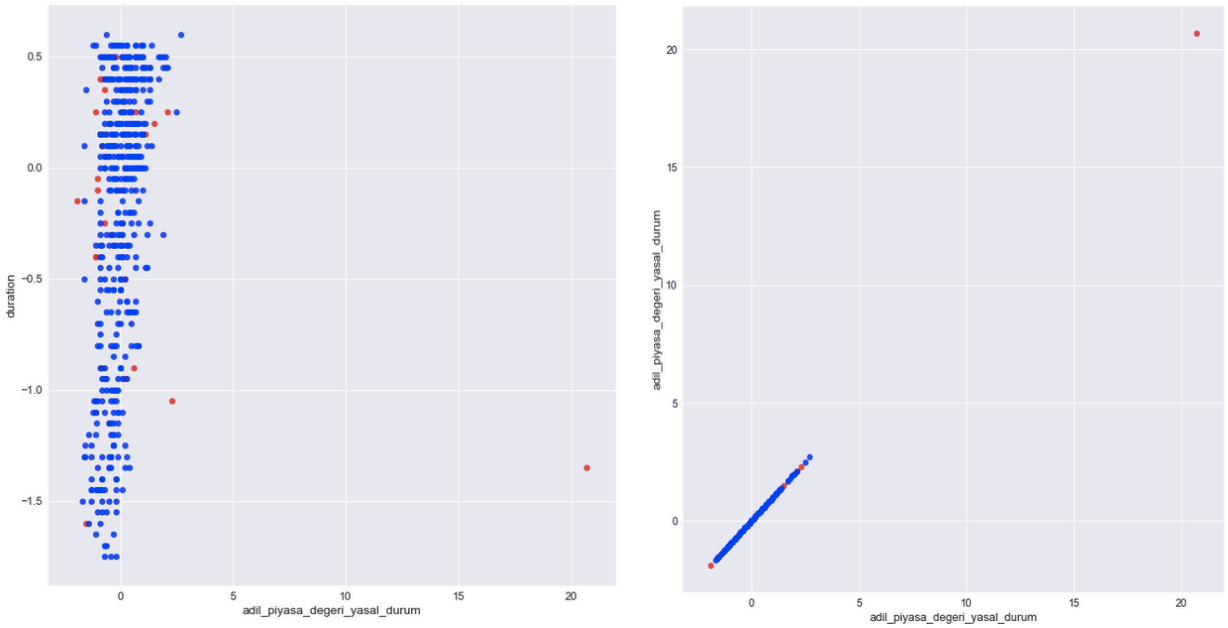


Figure 3.6 Suggested outliers for DBSCAN in red color.

The main characteristic of RobustScaler and DBSCAN is that both of them are robust to outliers, so that we can use them for outlier detection.

Lastly, before creating the models and starting training, we need to categorize the *mahalle_kodu* in our dataset. As they are integers in our data, they will be interpreted as numerical features by the models and things like mahalle_kodu 100 > mahalle_kodu 99 will lead wrong results. *One Hot Encoding* is used for this purpose, each unique mahalle kodu is turned into a separate feature in the data and each sample assigned to its mahalle_kodu by having 1 for the true mahalle_kodu and 0 for others.

yuzolcumu	mevcut_alani	tapunun_yasi	area	duration	adil_piyasa_degeri_yasal_durum	mahalle_kod_4673.0	mahalle_kod_4676.0
875.0	125.0	4.0	60.0	4.0	115000.0	1	0
585.0	125.0	3.0	60.0	13.0	120000.0	1	0
629.0	136.0	4.0	50.0	0.0	115000.0	1	0
1252.0	133.0	7.0	60.0	1.0	90000.0	1	0
1137.0	167.0	3.0	75.0	6.0	130000.0	1	0

Table 3.1 One Hot Encoding to categorical features

4.Model Training and Evaluation

4.1 Random Forest

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. (Ho 278). The reason Random Forest was chosen over other models is its ability to find interactions between features well and the fact that it is robust to overfitting / outliers. Also, it has Out-Of-Bag estimation functionality which directly gives out of sample estimates where model predict on samples which it did not see on training. This yields generalization on measure of model performance.

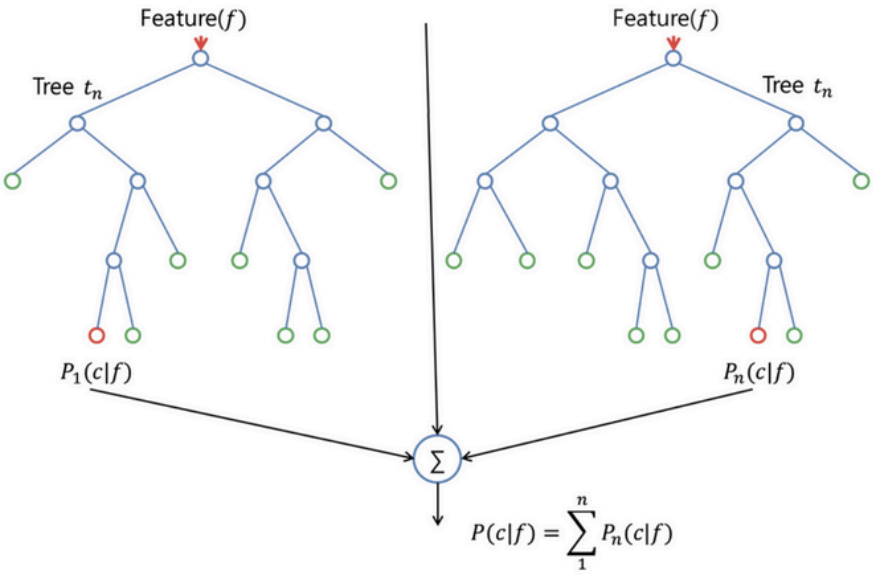


Figure 4.1 Random Forest Model

4.2 Cross-Validation for Parameter Optimization

Cross-validation was used for finding optimal parameters for Random Forest model. Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. (Geisser)

4.2.1 Randomized Search CV

Alternatives for parameters we want to optimize were supplied and Randomized Search CV tried 30 combinations and returned the best set. This may not be the optimal solution, however we used it since it is fast and decreases the optimal region.

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 600, num = 3)]
# Number of features to consider at every split
max_features = ['auto', 2,3,4,'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 80, num = 8)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 4,8]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,8]
```

Figure 4.2.1.1 Randomized Search CV supplied parameters

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:   21.7s finished
```

```
rf_random.best_params_
```

```
{'max_depth': 10,
 'max_features': 4,
 'min_samples_leaf': 2,
 'min_samples_split': 4,
 'n_estimators': 400}
```

Figure 4.2.1.2 Randomized Search CV optimal parameters

4.2.2. Grid Search CV

Grid Search CV gets the output of Randomized Search CV and some points near them to look at every possibility in this region. It fit almost 500 times and returned the optimal parameters, which yielded best accuracy in cross-validation estimates. Parallel computing was used in these search algorithms to decrease the duration. Optimal parameters are supplied in the figure 4.2.2.1.

```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [10, 20],
    'max_features': [ 4,5,6],
    'min_samples_leaf': [1, 2,3],
    'min_samples_split': [2,4,6],
    'n_estimators': [300,400,500]
}
```

Figure 4.2.2.1 Grid Search CV supplied parameters

```
Fitting 3 folds for each of 162 candidates, totalling 486 fits
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   25.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 486 out of 486 | elapsed:  4.3min finished
```

```
grid_search.best_params_
```

```
{'bootstrap': True,
 'max_depth': 20,
 'max_features': 5,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 300}
```

Figure 4.2.2.2 Grid Search CV optimal parameters

4.3 Error Statistics

After finding best parameters, the model was trained on the data and predictions are compared with the real values to get errors. Mean Absolute Error used for error calculations. Below there are statistics about the distributions of the errors and its visualization. We can see that mean of MAPE is 11% and standard deviation is 17%. This means that on average; the model predicts a house price in range 89%-111% of its true price.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y}_i|}{Y_i}$$

```
df.error.describe()
```

```
count    537.000000
mean       0.112545
std        0.175808
min        0.000000
25%        0.030033
50%        0.064663
75%        0.133133
max        2.694152
Name: error, dtype: float64
```

Table 4.3.1 Error statistics

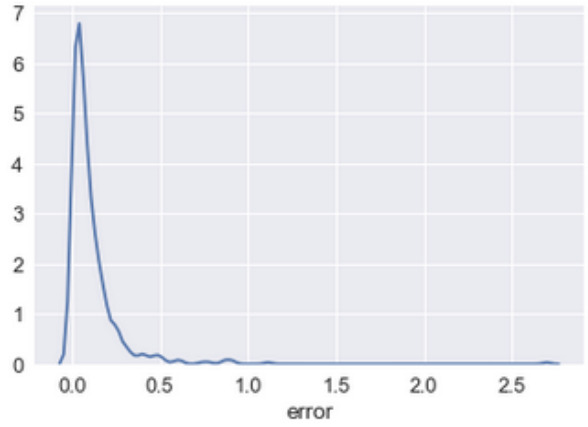


Figure 4.3.1 Error distribution

Below, predictions of Random Forest model are plotted against the true price values. It is imported to note that we did not exclude the DBSCAN's suggested outliers from the dataset. We kept them for examination in this stage. In the plot, samples colored green are suggested outliers. Normal samples are blue, and there are two lines fitted on blue and green points. They show the direction of distributions of this two group. From prediction – real target value plot we want to see samples on diagonal axes which indicates low error. We can see that blue line is close the diagonal axis and the green deviates from the axes. This shows that the model is not doing a good job on estimation of prices of the samples suggested by DBSCAN to be outliers.

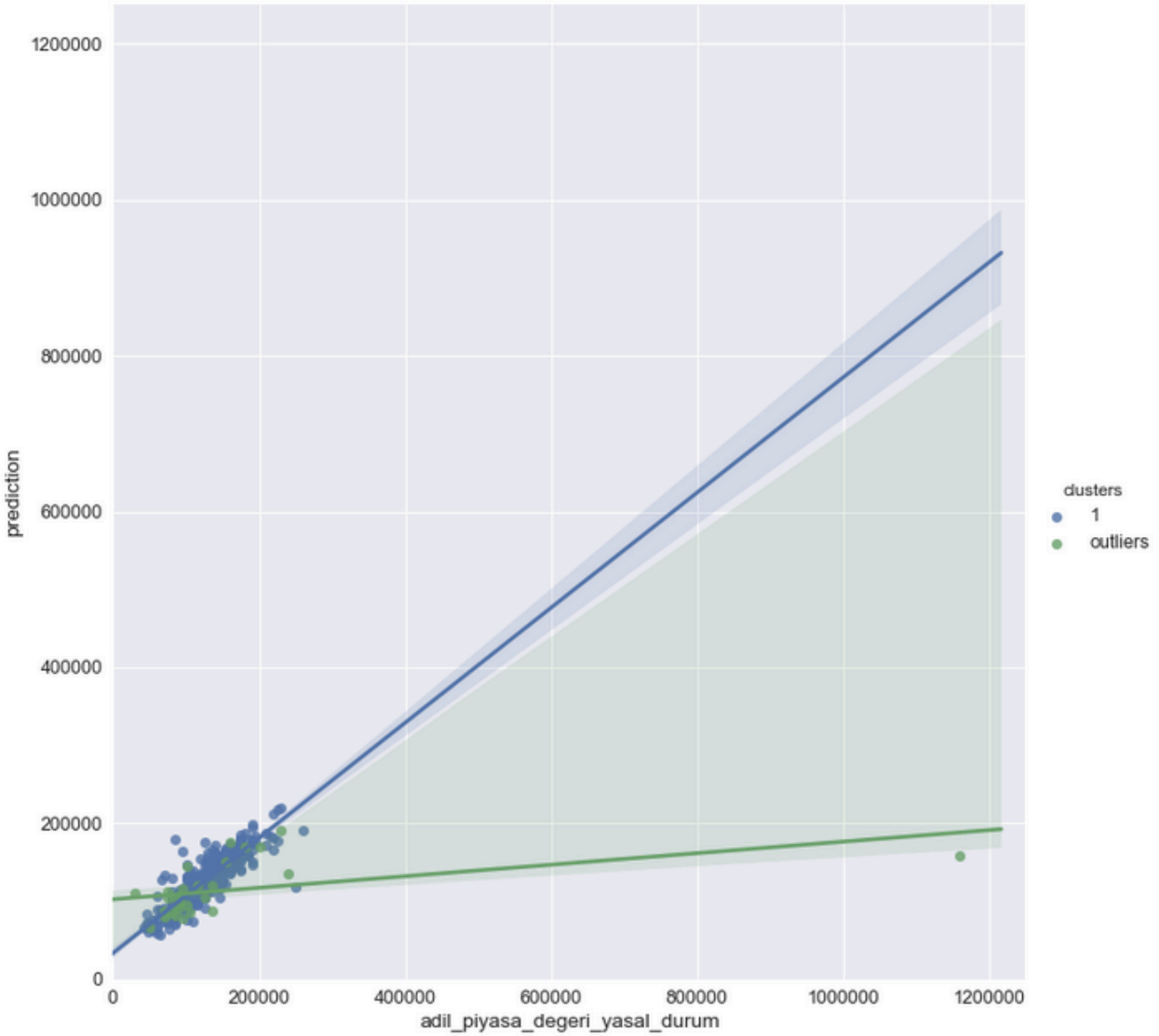


Figure 4.3.2 Prediction – Target Value Plot

Rate of houses below 20% MAPE quantile is 87% as can be seen from below figure.

Rate of houses that have less than 20% prediction error

```
(df[df.error<=0.2].shape[0])/df.shape[0]
```

```
0.8715083798882681
```

87% of the houses contain less than 20% error rate.

Figure 4.3.3 20% MAPE quantile

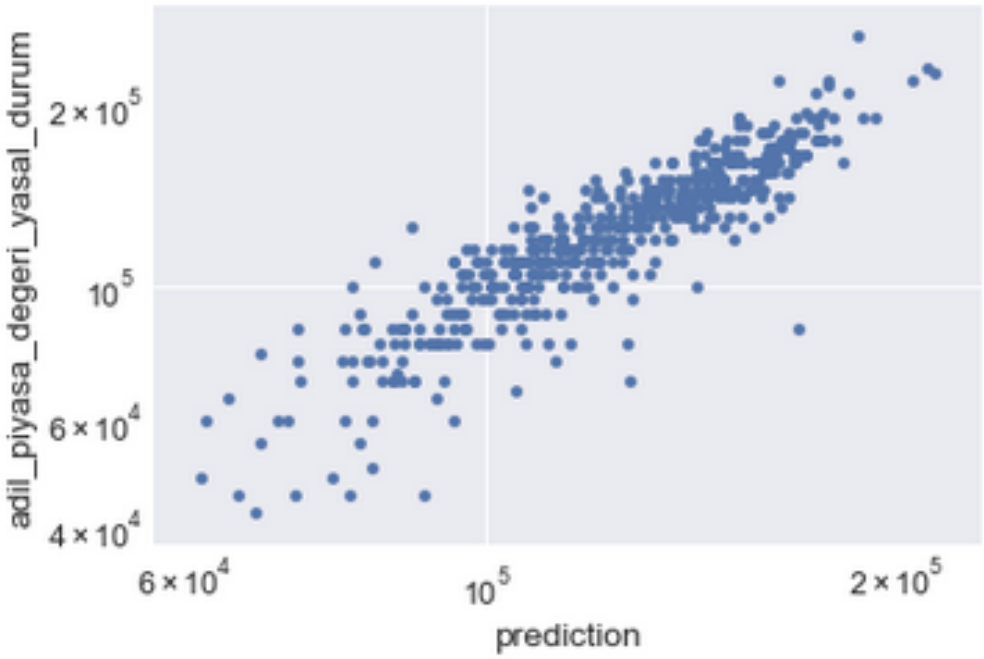


Figure 4.3.4 Prediction – Target Value Plot (without outliers)

The above plot shows again the prediction – real estate prices. However, before training the model and predicting prices, the outliers suggested by DBSCAN were removed from the data. Doing so increased the model accuracy to 89% as can be seen from the figure below.

```
(df[df.error<=0.2].shape[0])/df.shape[0]
```

```
0.8895463510848126
```

89% of the houses contain less than 20% error rate.

Figure 4.3.5 20% MAPE quantile

4.4 Cross Validated Random Forest Model Confidence Interval of Accuracy

To be certain about the model accuracy, different seeds were tried for random number generator and confidence intervals derived from them are shown in the table below. It can be seen that we get the (0.883-0.833) accuracy confidence interval.

```
seed = 11
oob error: 0.19
R^2: 0.96
20% error quantile: 0.888
------------------------------
seed = 23
oob error: 0.19
R^2: 0.96
20% error quantile: 0.884
------------------------------
seed = 42
oob error: 0.19
R^2: 0.96
20% error quantile: 0.890
------------------------------
seed = 68
oob error: 0.19
R^2: 0.96
20% error quantile: 0.884
------------------------------
seed = 81
oob error: 0.19
R^2: 0.96
20% error quantile: 0.884
------------------------------


20% Error Quantile CI: (0.883 - 0.883)
```

20% Error Quantile CI: (0.883 - 0.883)

Table 4.4.1 Cross Validated Random Forest Model Confidence Interval of Accuracy

4.5 Main Effects Graph

To be able to understand how the model uses each feature for prediction, main effects graphs were created. In each plot, every sample has mean values for every feature except one feature that we want to examine. For this feature, a grid was created from the minimum to the maximum value of a given feature, and predictions were made on this grid. It can be seen from below figures that model predictions behave mostly linear with the feature values.

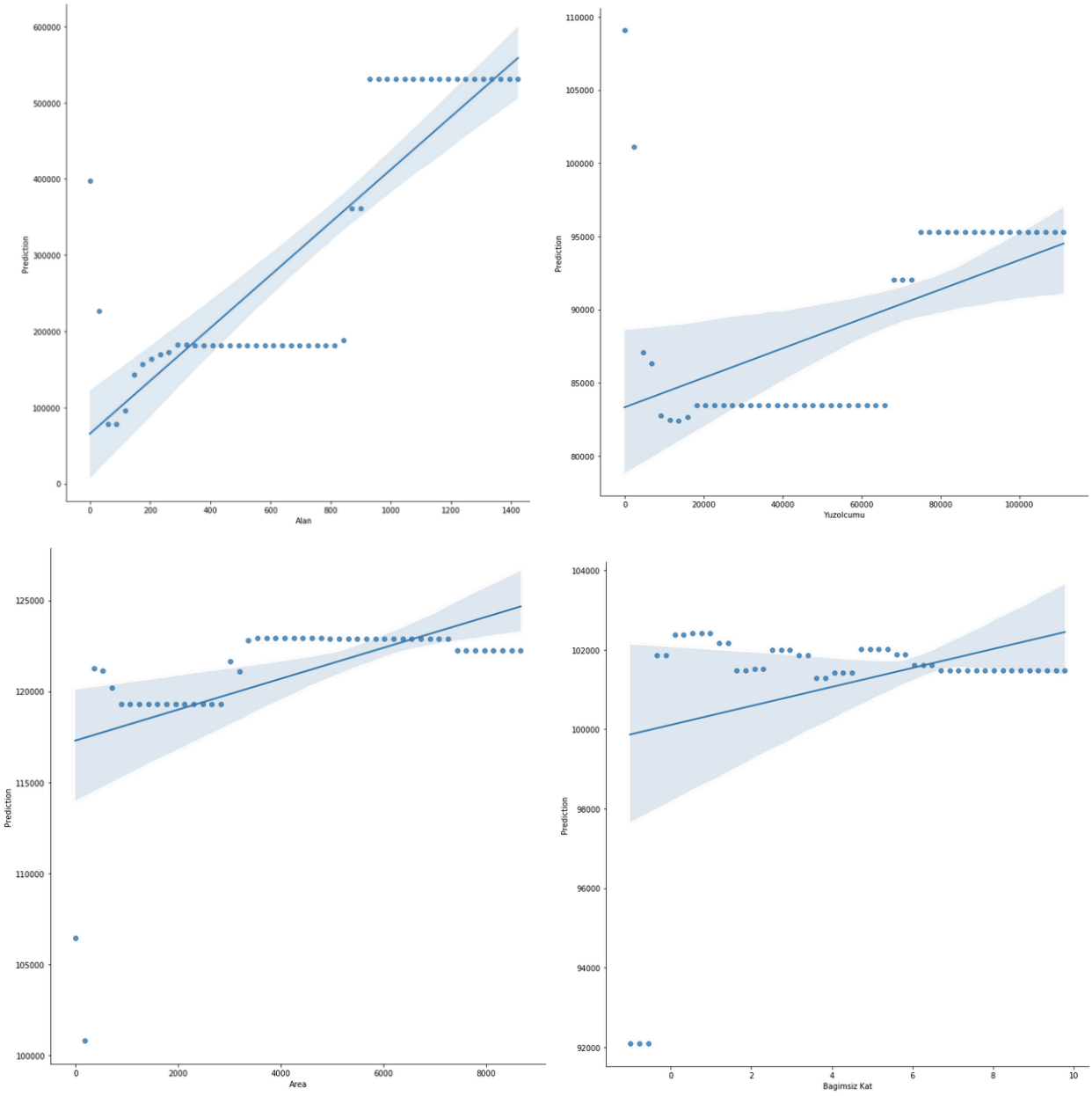


Figure 4.5.1 Main Effects Graphs

4.6 Focus on 80. Region

In this stage of the research, we wanted to see the model's accuracy on most crowded region, the 80th region.

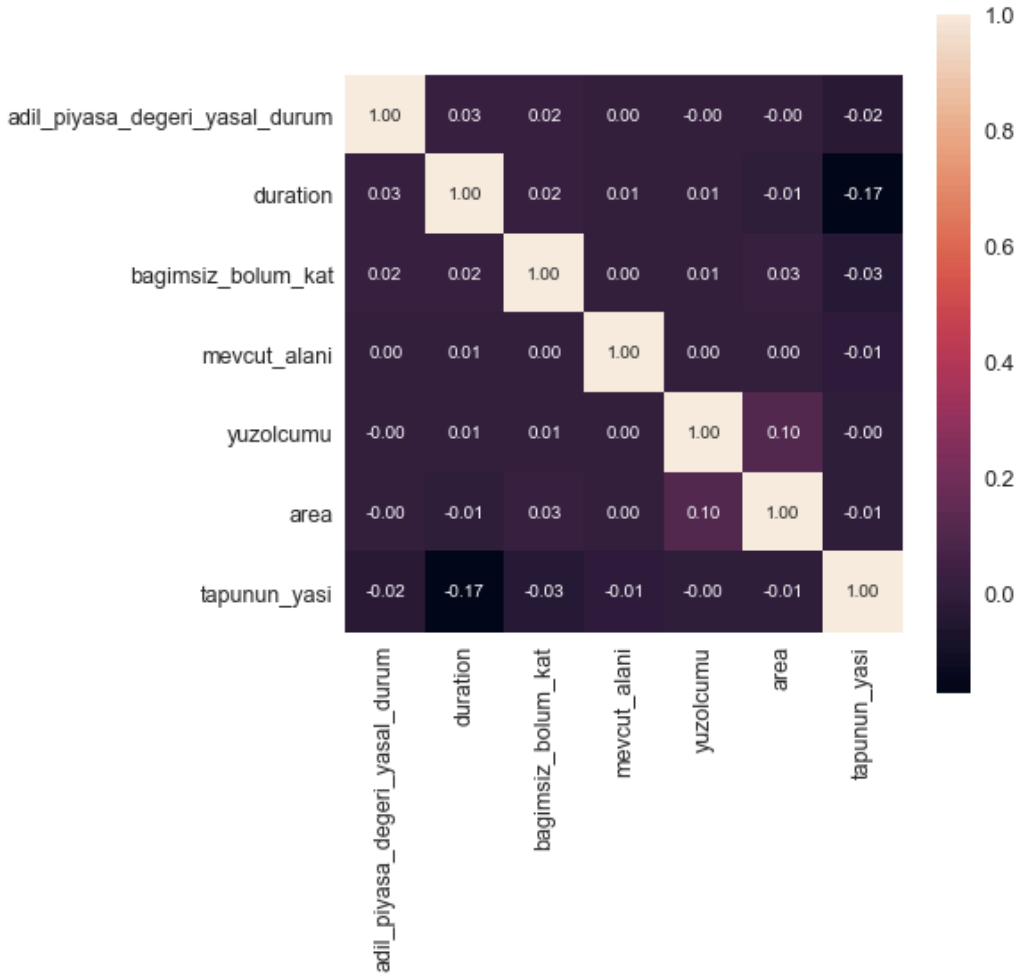


Figure 4.6.1 Correlation Matrix for 80. Region

From the correlation matrix, we see no interesting features. Also, from figure 4.6.2 it can be seen that price distribution is very unbalanced for this region.

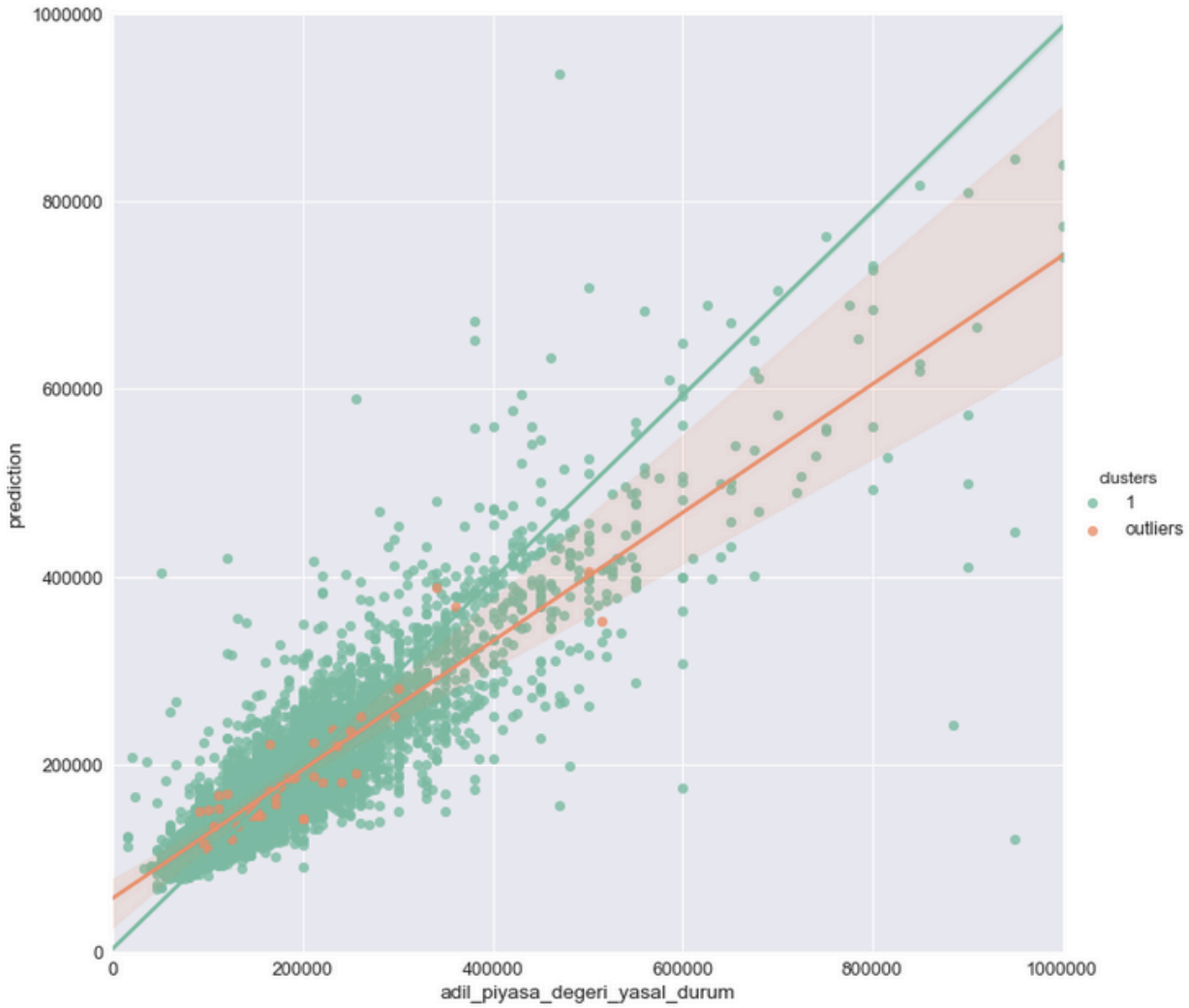


Figure 4.6.2 Prediction – Target Value Plot for 80. Region

From the plot above, it can be seen that now there are many points that deviate from the diagonal axis and the suggested outliers do not actually fit on a line away from diagonal axis. The model accuracy has decreased to around 70%. Now we wanted to understand source of these errors.

4.7. A closer look at houses outside of 20% error quantile

The plot below shows the distribution of houses that are outside the 20% MAPE quantile by districts. Note that the 80th region contains more districts. Most of the districts have unbalanced distributions for errors. These samples were also examined by distribution of other features in the

following plots. The model's accuracy does not change by value of features; all lines are parallel to x-axis.

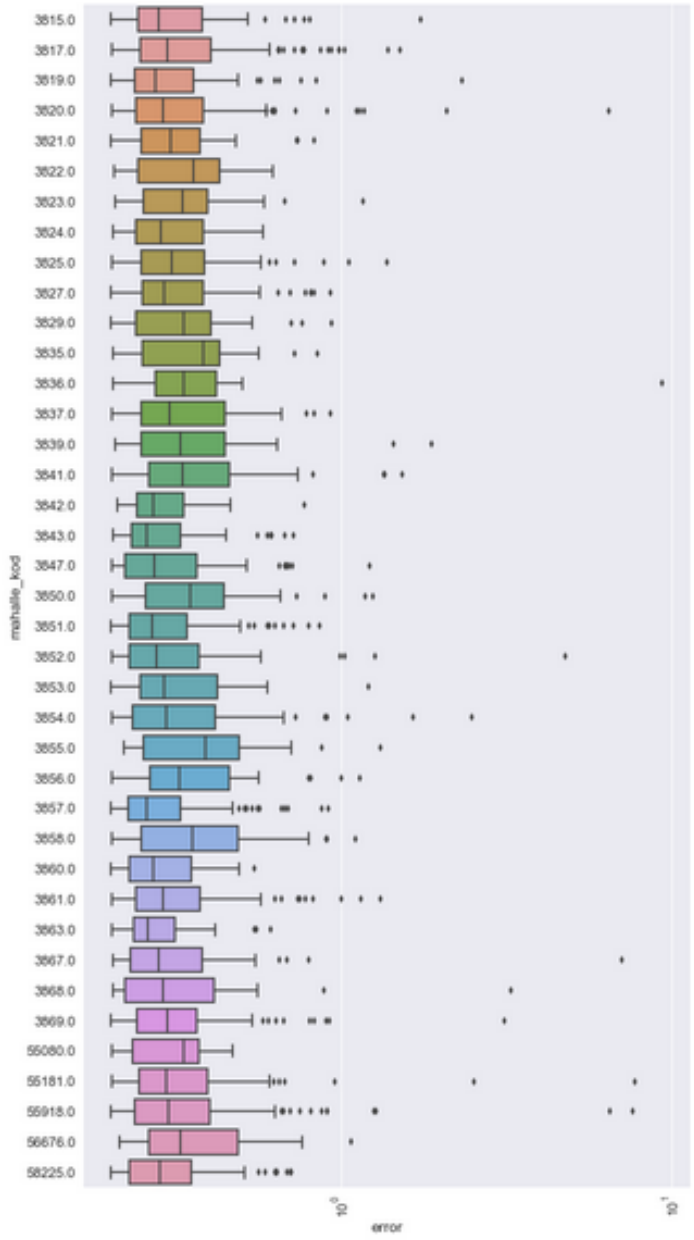
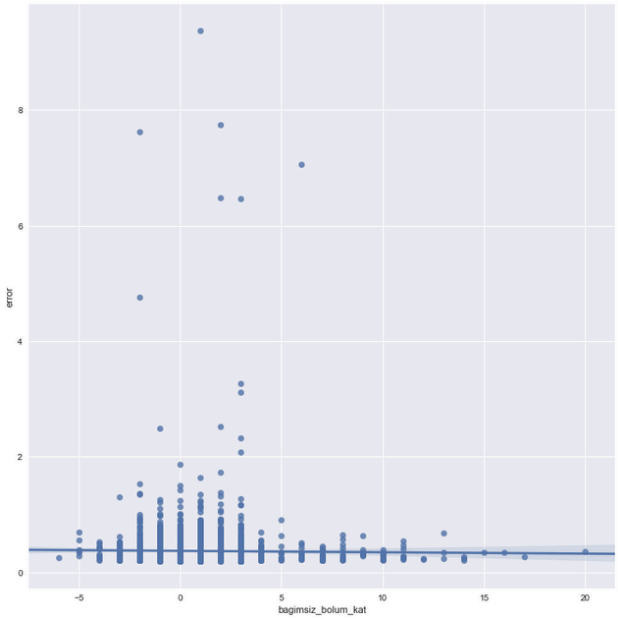
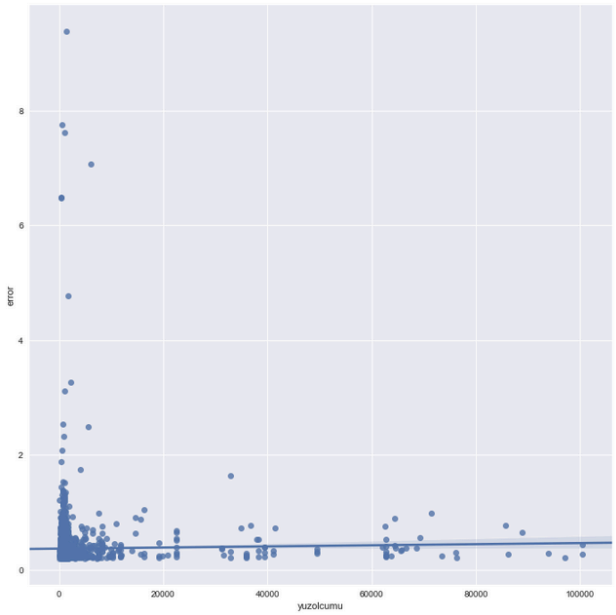
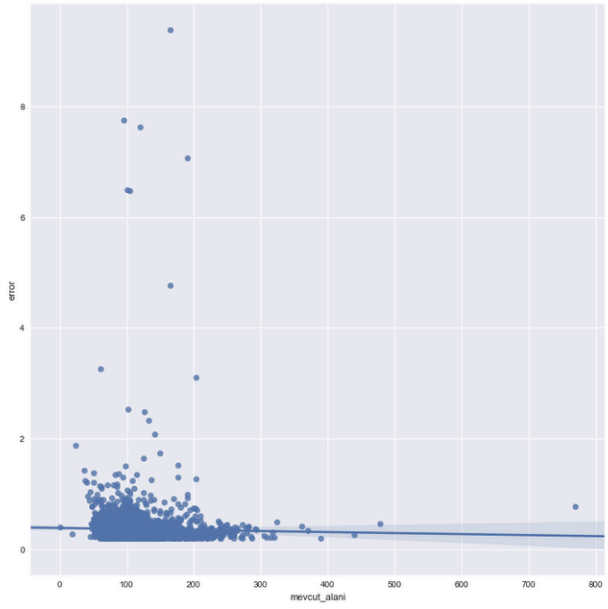
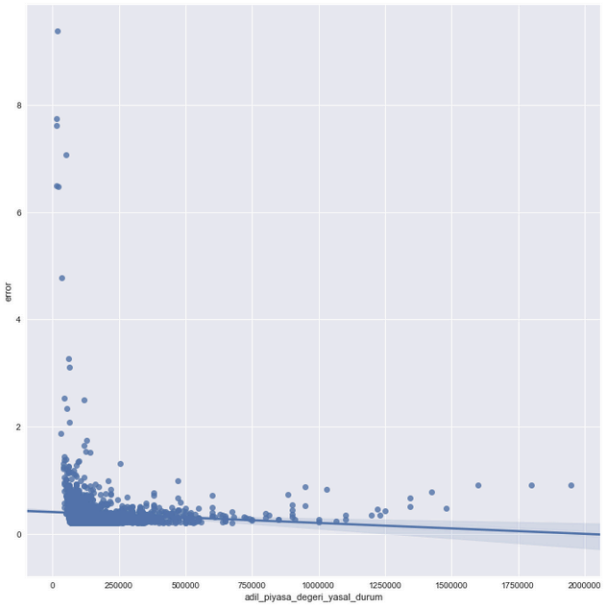


Figure 4.7.1 Error distribution of houses outside 20% error quantile by districts


error
bagimsiz_bolum_kat


error
yuzolcumu


error
mevcut_alani


error
adil_piyasa_degeri_yasal_durum

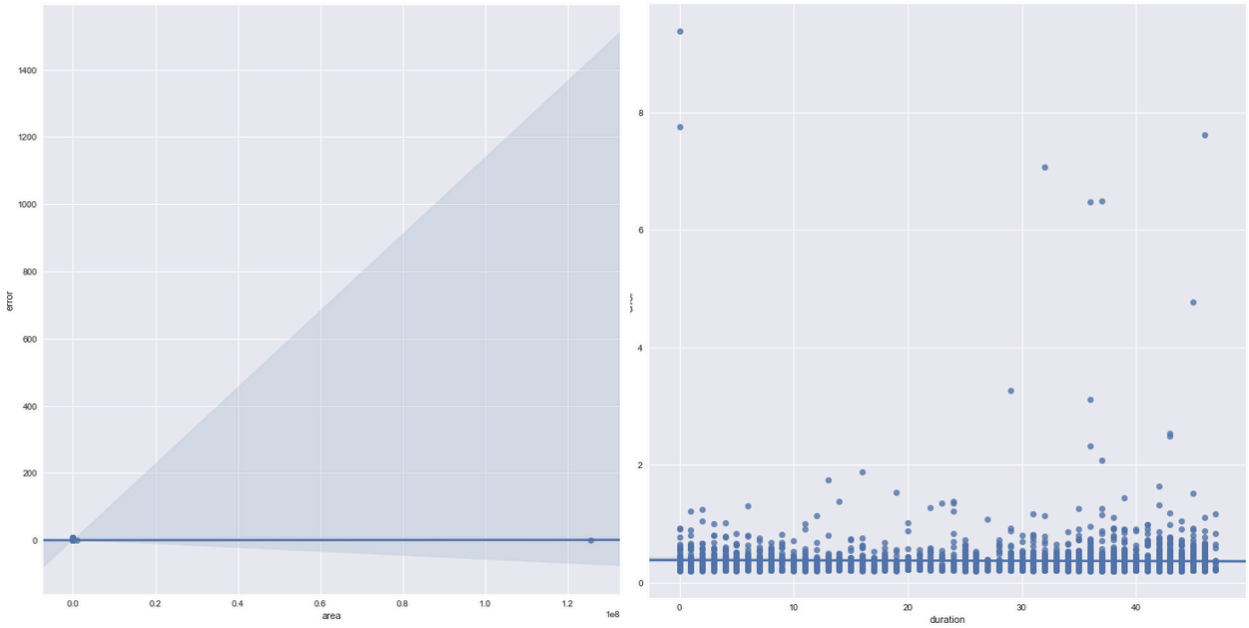


Figure 4.7.2 Error of houses outside 20% error quantile plotted by features

4.8 Neural Network

As the second model, a Neural Network model was proposed and trained. Neural Network is a model where layers of neuron are connected to each other and by biases and weights between them define the output of the model. Keras library was used to create the model. Keras wrappers require a function as an argument. This function that we must define is responsible for creating the neural network model to be evaluated. We defined the function to create the baseline model to be evaluated. It is a simple model that has a single fully connected hidden layer with the same number of neurons as input attributes-9. The network uses rectifier activation function (relu) for the hidden layer. No activation function is used for the output layer because it is a regression problem and we want to predict numerical values directly without transformation.

The efficient ADAM optimization algorithm was used and the mean squared error loss function was optimized.

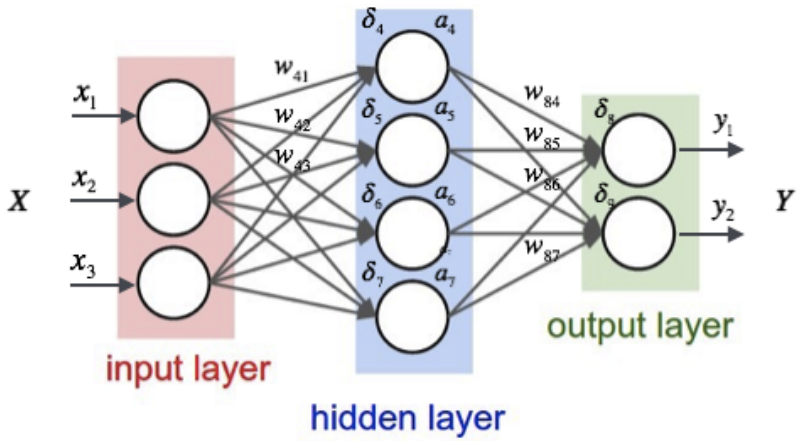


Figure 4.8.1 Neural Network Structure

Train and test data were created to train the model on the training set and evaluate its performance on the test set.

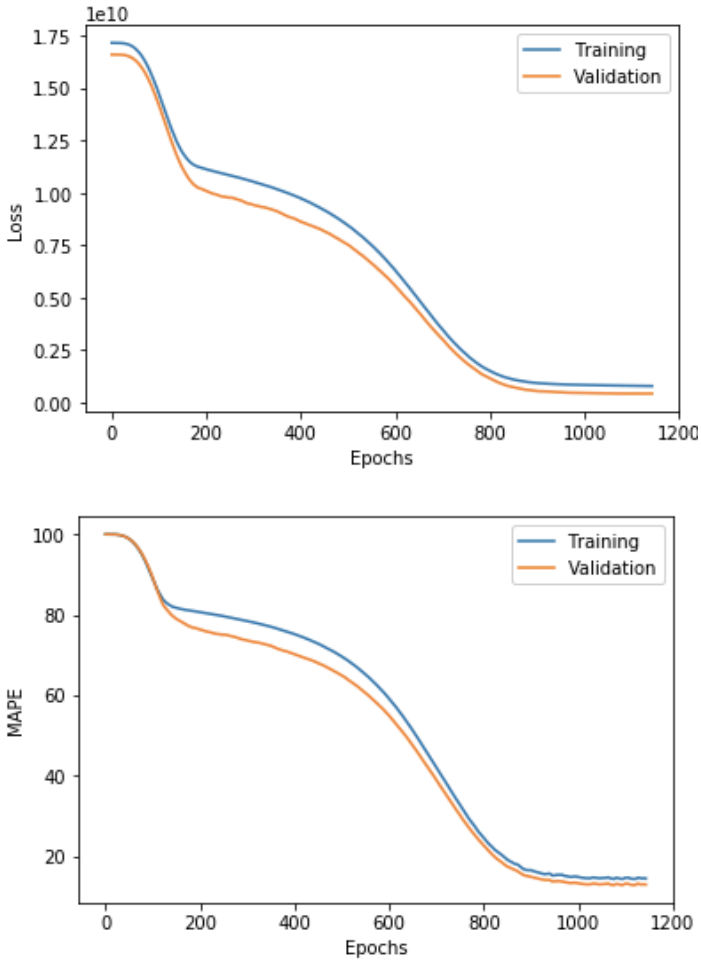


Figure 4.8.2 Base Model training

The above plots show the learning process of the model. Again, we evaluate our model performance using the MAPE metric. It can be seen that model converges after almost 1000 epochs. An epoch is a full forward and backward iteration on the training set to update parameters, bias and weight matrixes. Baseline model has accuracy around 81.3% on test set. Stratified k-fold cross validation was used to derive models performance. Stratified partitioning aimed to save the ratios categorize on unbalanced date when creating sub-samples. Plots below shows that ratios of districts are saved in 3 folds and results of cross-validation.

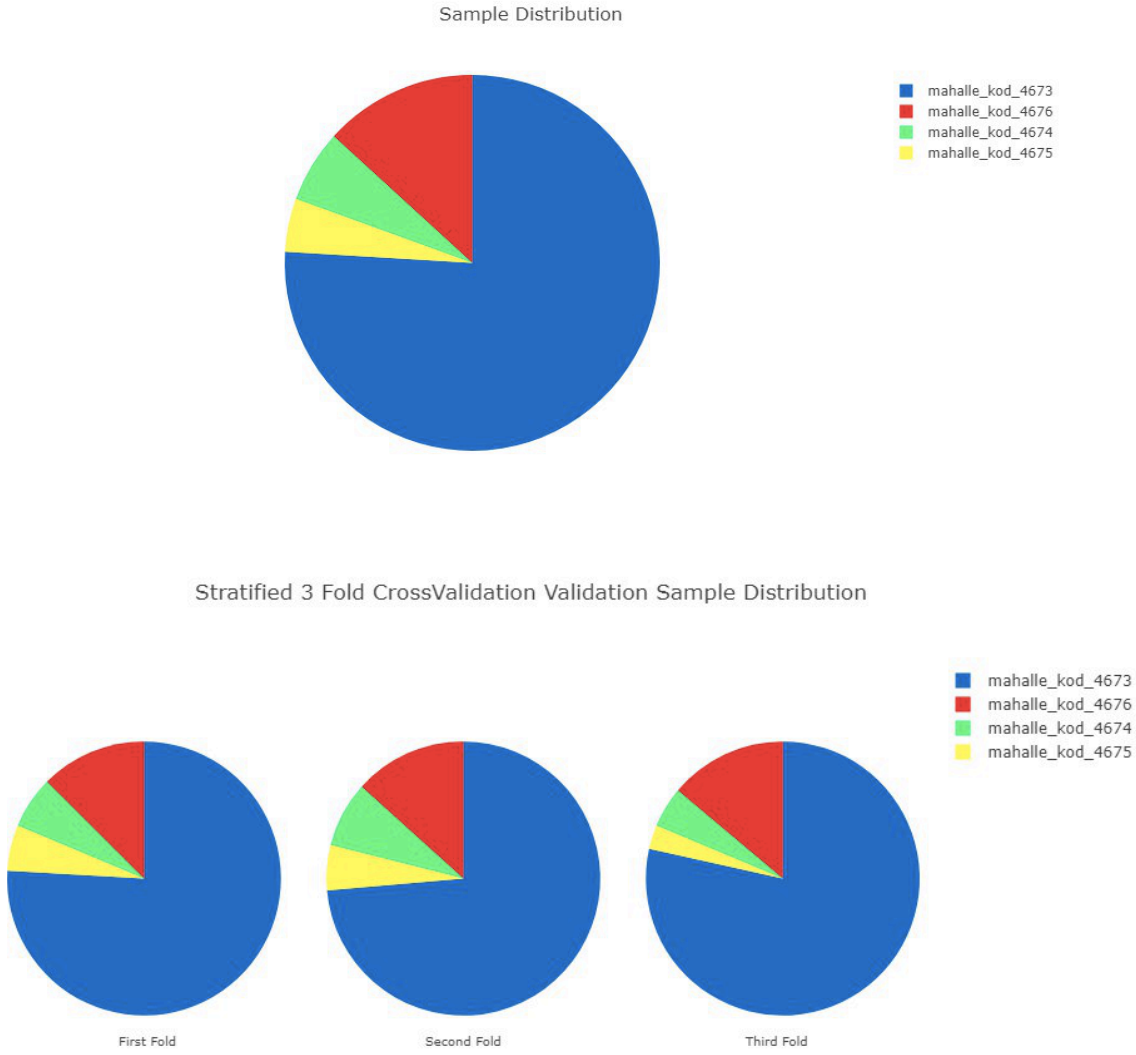


Figure 4.8.3 Stratified k-fold district ratios

```
kfold = StratifiedKFold(n_splits=3, random_state=seed)
estimator2 = KerasRegressor(build_fn=baseline_model, epochs=1200, batch_size=128, verbose=0)
results = np.abs(cross_val_score(estimator2, X, Y, cv=kfold, scoring = mape))
print(results)
print()
print("Cross Validation Mean Absolute Percentage error: %.2f " % (results.mean()))
```

```
[ 14.8476281   15.54637567  69.13683218]

Cross Validation Mean Absolute Percentage error: 33.18
```

Table 4.8.1 Stratified k-fold cross validation results

4.9 Modeling The Standardized Dataset

An important concern with our house price dataset is that the input attributes all vary in their scales because they measure different quantities. It is almost always good practice to prepare the data before modeling it using a neural network model for optimization of the algorithms' efficiency. StandardScaler from Scikit-Learn libraries were used and prices were divided by 1000. The results are shown below.

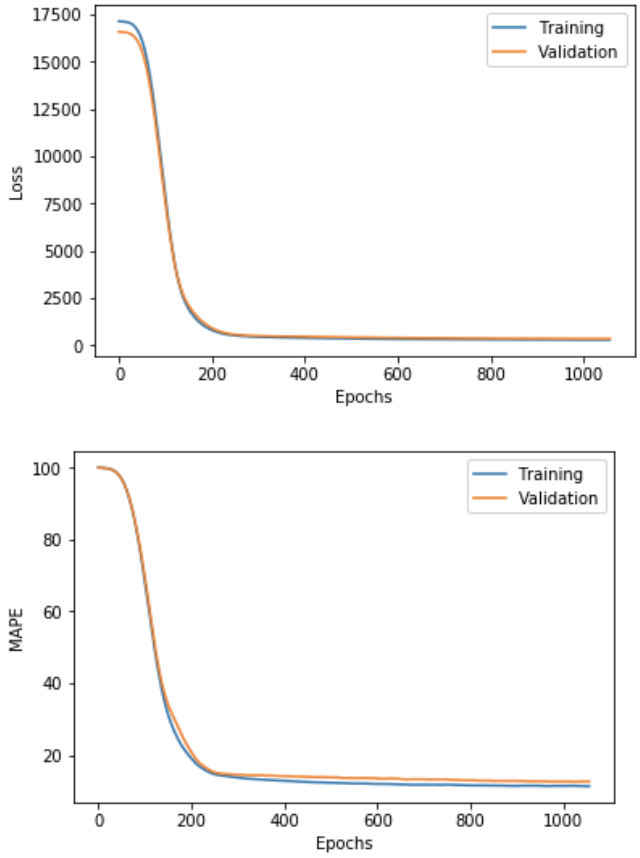


Figure 4.9.1 Base Model Standardized Data Training

We can see that now the model converges in 200 epochs. Accuracy on all dataset increased but out of sample accuracy remains same.

4.10. Tune The Neural Network Topology

There are many variables that can be optimized for a neural network model. Perhaps the most important one is the structure of the network itself, including the number of layers and the number of neurons in each layer. We evaluated two additional network topologies in an effort to further improve the performance of the model. We will look at both a deeper and a wider network topology.

4.10.1. Evaluate a Deeper Network Topology

One way to improve the performance of a neural network is to add more layers. This might allow the model to extract and recombine higher order features embedded in the data. We evaluated the effect of adding one more hidden layer to the model. This new layer will have about half the number of neurons, because there are heuristics that suggest that number of neurons in the next layer should not exceed half of the previous layer's number of neurons. Results show improvements.

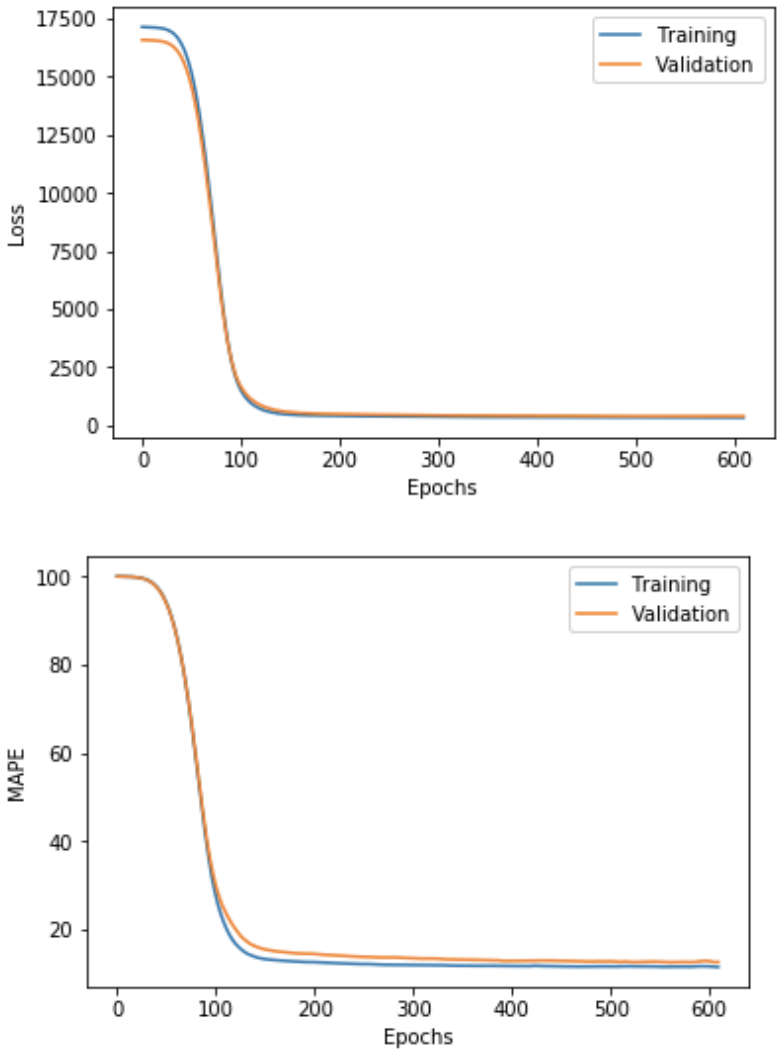


Figure 4.10.1 Deeper Model training

Model converges even faster and out sample accuracy increased to 84.3%.

4.10.2. Evaluate a Wider Network Topology

Another approach to increasing the representational capability of the model is to create a wider network. We evaluated the effect of keeping a shallow network architecture and nearly doubling the number of neurons in one hidden layer. Here, we have increased the number of neurons in the hidden layer compared to the baseline model from 9 to 15. Model convergences become worst and out of sample accuracy dropped to 78.4%.

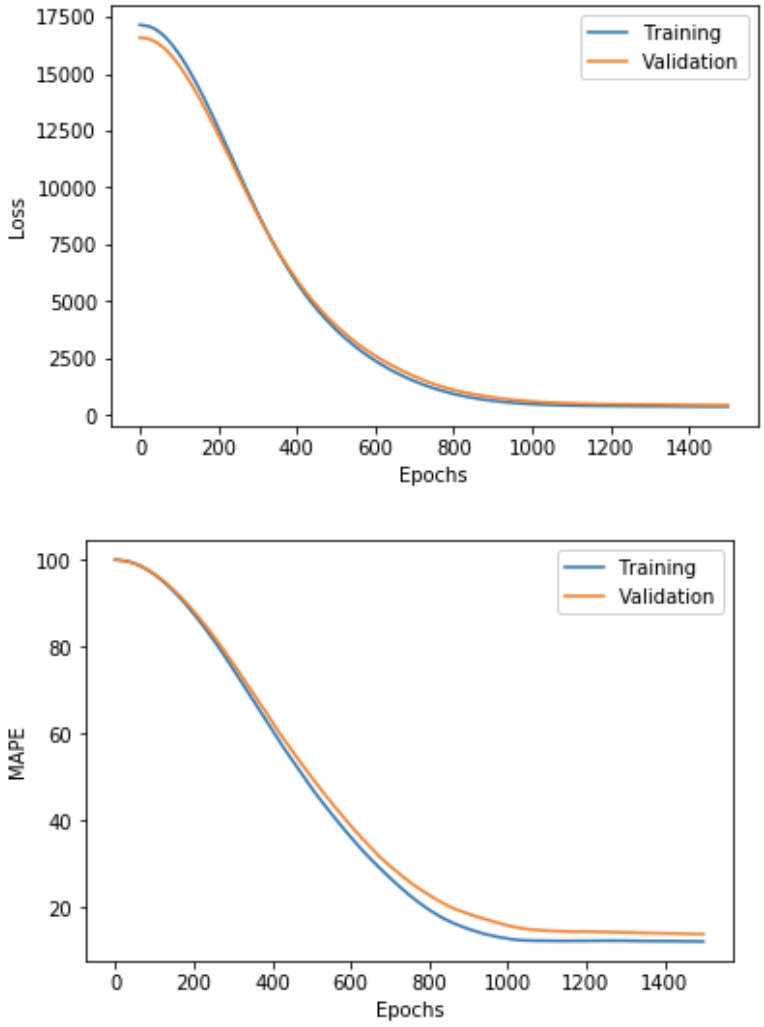


Figure 4.10.2 Wider Model training

4.10.3. Evaluate 3 Hidden Layer Network Topology

As we observed the best accuracy from the deeper model, we tried to make it even deeper by adding one more layer. Results show that model did not show improvement over previous best model.

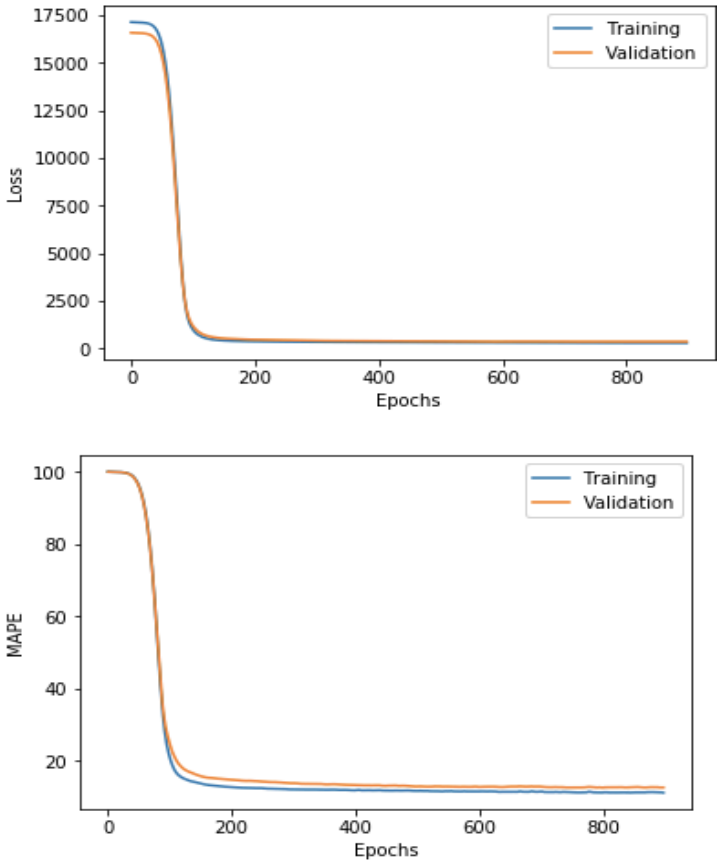


Figure 4.10.3 3 Hidden Layer Model training

5. Results

Random Forest models make better predictions as compared to the Neural Network models. We believe that this it is due to lack of training samples, 79th region contains 537 samples and after removal of the outliers the data contained 507 samples.

Random Forest	**Neural Network***
Without removing Outliers: 87.15	Baseline Model: 81.37
With removing Outliers: 89.95	Deeper Model: 84.31
	Wider Model: 78.43
	Deepest Model: 83.33

Table 5 Accuracy of models on out of sample data for 79. Region

*All NN models used outlier removed dataset.

6. Creating User Interface

After fitting several different models, we decided to continue using Random Forest model as it has the best performance on our data as can be seen from *Table-5.* We believe that the biggest reason that Neural Network was outperformed by Random Forest is the limited training dataset. In support of this claim, the following figure, taken from Andrew Ng's course, shows that with small amount of labeled data, the order of performance of different models is not certain.

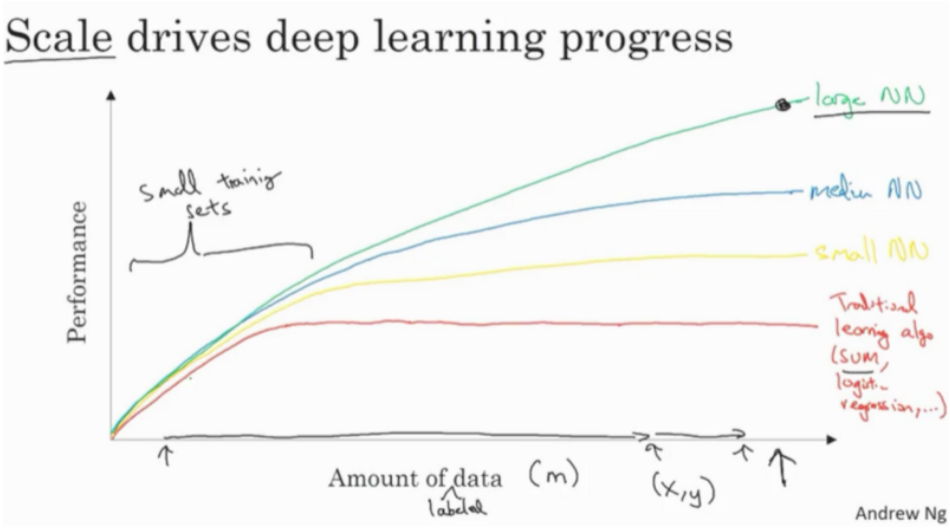


Figure 6.1 How scale drives performance in DNN's (Ng)

After choosing the Random Forest model, we wanted to create a user interface for the model. Doing so turns the model into an interactive tool that can be used by people with no programming or data analysis background. Initially the model was written in Python so we searched for libraries for Python to enable creation of a user-interface. Currently, there is no library for Python that is flexible and powerful enough for our needs. On the other hand, R has well-known package called Shiny that lets users interact with and analyze data (Shiny). To be able to use Shiny, the model needed to be coded in R. In order to avoid rewriting the model in R, we decided to do some research on how to use Python in R, and found the R library *reticulate*. The library is described as "Interface to 'Python' modules, classes, and functions. When calling into 'Python', R data types are automatically converted to their equivalent 'Python' types. When

values are returned from 'Python' to R they are converted back to R types" (reticulate). Using this package we were able to upload our model written in Python into R as a R object.

Firstly, we worked on subset of the data containing only the 79th region with 2 districts. Resulting UI is shown below:

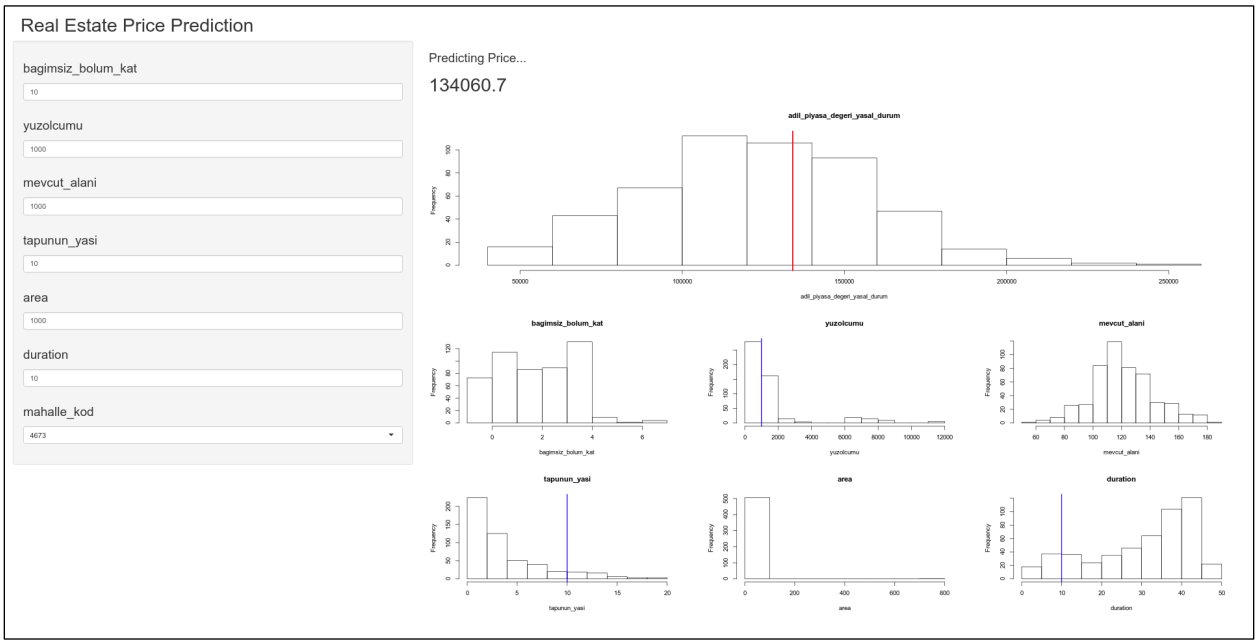


Figure 6.2 Shiny Application for 79. region Random Forest Model

The major benefit of this UI is that a user can select parameters of the model which are the features of homes using left panel, and can select the district from the combo-box. Whenever the user changes a parameter, the parameters will be updated and a new price prediction will be calculated. Instead of just printing out the result price, we decided to show the parameters and resulting price in the respective histograms of the data. For example, in the figure above, the user has selected 10 for the number floors and we can actually see where the selected number drops into range of the values from dataset. The floor number 10 is actually outside the range, which means that the model has never seen a training entry with 10 number of floors. We expect that model's prediction accuracy will be higher on the cases that are similar to ones that are seen in training phase. This information can be useful for someone who uses this model on new cases for prediction. Also, we highlighted the resulting price prediction in the distribution of prices of the training dataset. Now, we have an idea about how this predicted price compares to all other

prices in dataset and whether we expect such a price prediction. We can also have an idea about the distributions of the parameters in the dataset. From the figure above, we see that field *area* has an exponential distribution whereas *mevcut_alani* has normal distribution.

After successfully implementing the UI for a subset of the data, as a final step, we wanted to train the model using all the data and to expand the UI accordingly. There were some technical difficulties to convert the user inputs (categorical variables) into a form that model can accept but at the end, we were successful.

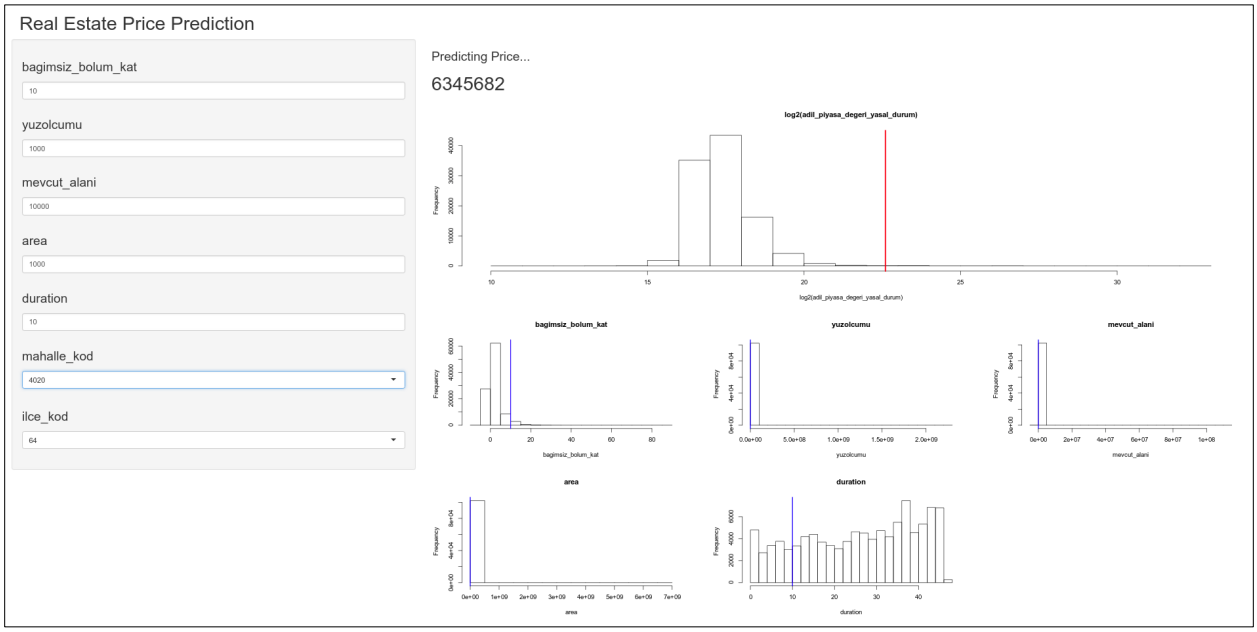


Figure 6.3 Shiny Application for all dataset Random Forest Model

It is noticeable that distributions of the features and the prices are different when we look at the whole dataset. Also, as we change regions and districts prices change, which give us an idea about the relative prices of locations.

One nice thing about Shiny is that now we can upload this UI into server and anyone can use it simultaneously from the Web without any need to install R, Python or run the code that we wrote to build and run the model.

Conclusion:

In conclusion, we found that random forest outperformed neural network models, most probably because of the small size of the training dataset. We noticed that different network topologies changed the performance of neural networks and the time of convergence. The data was messy and had many errors. Therefore, cleaning and formatting the data took substantial time and effort. Furthermore, large number of one hot encoded categorical variables made the training computationally expensive. We were able to use Python and R together in order to get the best features of both languages. For instance, we used the Python's famous scikit-learn library together with R's Shiny. The UI made the model interactive and easy to analyze for users.

References:

Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M., eds. *A density-based algorithm for discovering clusters in large spatial databases with noise*. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. CiteSeerX 10.1.1.121.9220. ISBN 1-57735-004-9.

Geisser, Seymour (1993). *Predictive Inference*. New York, NY: Chapman and Hall. ISBN 0-412-03471-9.

Ho, Tin Kam (1995). *Random Decision Forests* (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

"Interface to 'Python' [R Package Reticulate Version 1.7]." README, Comprehensive R Archive Network (CRAN), cran.r-project.org/web/packages/reticulate/index.html.

Ng, Andrew. "Neural Networks and Deep Learning." Coursera, Michigan State University, www.coursera.org/learn/neural-networks-deep-learning.

Shiny, shiny.rstudio.com/.