Prediction of PurchasePrice

Group: Firat Saritas, Simon Staehli

The goal of these notebooks is to find the model that has the best prediction for the PurchasePrice. Different models with different Pre-Processing are created for this. At the end of this notebook you will also find the trained model that is for the kagglechallange.

1. Index:

2. LinearRegression

2.1. Import data

At the beginning we start to import the dataset and see what shape we have:

```
Shape of Dataframe: (153627, 69)
```

2.2. Pre-Processing

- Dropping Cols and Rows
- Filling missing Values
- Tranformation of the Data
- Encoding of Categorical Features

First of all Drop all columns which are not necessary because of redundancy and rows with a lot of missing values.

We dropped especially the coulmns, who occur in three sizes (S M L). The sizes S and L were removed because we could not consider any large correlation deviations between the sizes (see notebook: EDA_master) At the beginning we dropped more columns, but then got a worse mape, which is why we keep the remaining attributes.

We also removed the rows that had a lot of missing information. In this case there were only 2 rows.

```
The new Shape of Dataframe after dropping: (153624, 56)
```

Next we filled in the missing values. The missing values occurred especially for specific columns. (see notebook: EDA_master) Thats why we treated them there specifically.

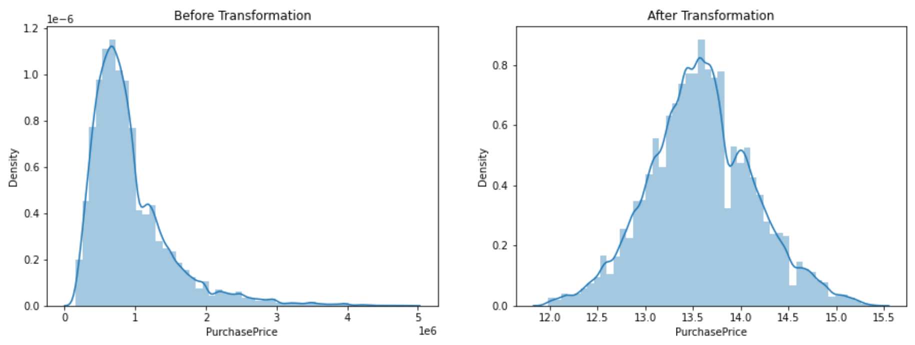
We also discovered incorrect information such as floornumber 1000. This was corrected in the same step

```
FloorNumber filled.
RenovationYear filled.
Politics filled.
AreaProperty filled.
Remaining Attributes filled.
```

A check to see if we have filled in all the missing values:

```
Any missing values: False
```

The natur of distribution often results in an extremely efficient calculation. Thts why we transform the distribution of Purchase Price into a Normal distribution



at the end machine learning require all input and output variables to be numeric. Since we have categorical data in our dataset, we have to encode them in numbers.

	Id	AreaLiving	AreaProperty	BuiltYear	FloorNumber	ForestDensityM	GroupNa
0	7135329	140	501	2016	2.0	0.555985	
1	7170979	143	277	2004	2.0	0.074061	

2 rows × 56 columns

2.3. Train-Test-Split

Before we start training the model, we have to split it up. So we can then test our trained model. We will train the model on 80% of the data and test it on the remaining 20%.

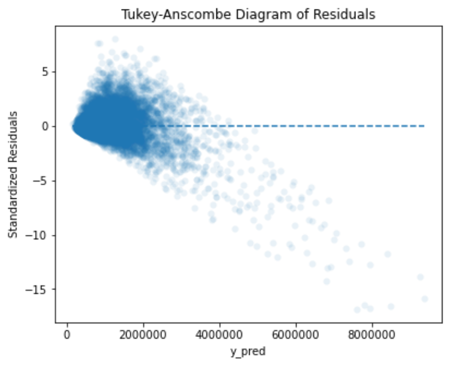
2.4. Train Model

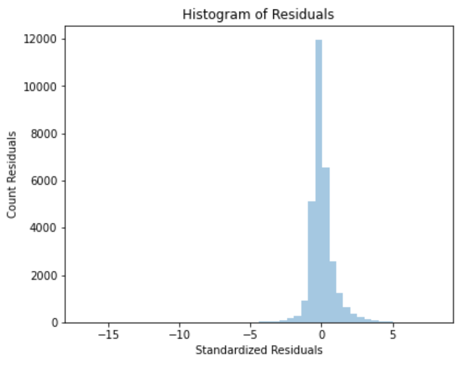
We will train the data using a linear regression. We also want to measure how long it takes to train and what an R2-Score it achieves. The R2-Score ranges from 0 to 1. The closer you are to the 1, the better your model!

```
Time:  0.3570408999999987
R2-Score:  0.665543757206443
```

The linear model runs very quickly! it only takes half a second and it's done. The R2-Score still has plenty of room for improvement. But for us, however, it is not the R2-Score that is important, but the mean absolute percentage error (mape). We would like to get as low a mape as possible.

```
Mean ABS Error:  232258.37
Mean ABS Percentage Error:  0.2516397870221667
Median ABS Error:  137399.12
```





We get a mape of 25% here. Unfortunately, this is still a rather bad result. It will be a sufficient one if we manage it under 15%.

Let's try other models, maybe they do better...

3. RandomForestRegressor

Welcome to the forest of decision trees!

Next we get into the RandomForestRegressor. This should help us to efficiently recognize the connection between classes and to come to a good prediction.

The individual steps will be very similar to the linear model, which is why it will contain fewer comments.

3.1. Import data

All steps are repeated as before in order to avoid errors and confusion. That's why we import the data again.

```
Shape of Dataframe: (153627, 69)
```

3.2. Pre-Processing

we will use the same steps here as well:

- Dropping Cols and Rows
- Filling missing Values
- Tranformation of the Data
- Encoding of Categorical Features

```
The new Shape of Dataframe after dropping: (153624, 56)
```

```
FloorNumber filled.
RenovationYear filled.
Politics filled.
AreaProperty filled.
```

3.3. Train-Test-Split

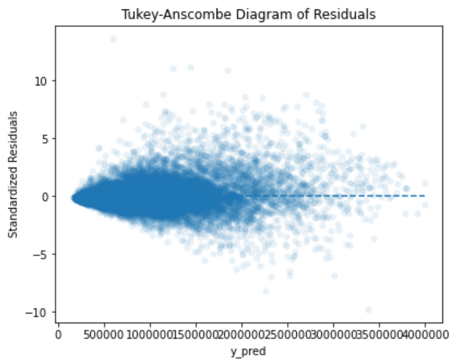
Now we come back to splitting We split the same as the linear model and therefore we set the same "random_state" to get the identical split in order to compare the models better.

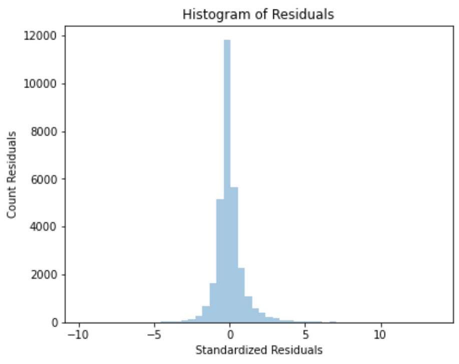
3.4. Train Model

```
Time:  297.085687
R2-Score:  0.9804236590252978
```

This model ran much longer now, approx. 300 seconds. But the r2-Score is very good. let's see what the mape says now.

```
Mean ABS Error:  144500.15
Mean ABS Percentage Error:  0.15379239726241428
Median ABS Error:  81674.96
```





The mape is already very close to the 15% that we would like to safely reach. In addition, the distribution of the Tukey-Anscombe Diagram of Residuals looks much better.

The tree-based method is much better than the linear one, so let's stick with the regression trees and go to the method that is said to be the winner of many kaggle challenges:

4. XGBRegressor

With the help of eXtreme Gradient Boosting (XGBoost), target variables can be determined more precisely by combining several simpler and weaker models and making better predictions. So lets try!

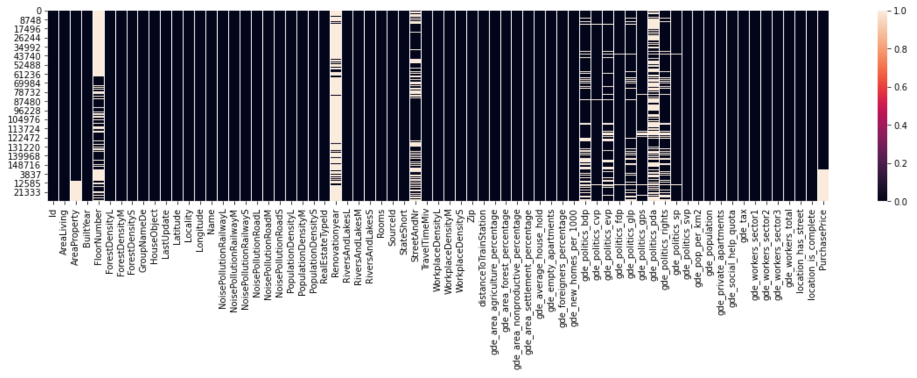
We will repeat some of the steps as we did before.

4.1. Import data

This time we will also include the test data for the Kaggle-Challenge. Thus, this data can go through the same pre-processing and be used for the Kaggle-prediction. We're going to concatenate the two datasets together for further processing.

```
Shape of Dataframe: (153627, 69)
Shape of Dataframe: (30074, 68)
```

```
New Shape of concatenated Dataframe:  (183701, 69)
```



The two dataframe have been successfully imported as we can see in matrix above. The PurchasePrice clearly shows misssing values of the test dataframe in the lower bound.

Next we will add new features which we extracted from the internet on swissopendata, which shows the amount of GA and Halbtax owners per Zip.

Add Features Amount of People how own GA and Halbtax per Zip

	PLZ	Amount_GA	Amount_HT
12725	1000	78.0	1078.0
12726	1003	845.0	3347.0

```
(183701, 71)
```

	Id	AreaLiving	AreaProperty	BuiltYear	FloorNumber	ForestDensityL	ForestDe
0	7135329	140	501.0	2016	NaN	0.418964	0.
1	7170979	143	277.0	2004	NaN	0.033259	0.

2 rows × 71 columns

We see that we have now successfully added two new columns with the information about GA and Halbtax.

Additionally, we add new information to dataframe with has_bahnhof. The data is from the Wikipedia Webpage which we scraped by using pandas read_html command. The data consist of all train stations in Switzerland.

	Locality	has_bhf
75	Roggwil BE	1
77	Puidoux	1

```
(183701, 72)
```

Now, this column was also added to our dataset

4.2. Pre-Processing

Wow we come back to the pre-processing part. The same processes are carried out here as with the other models. But this time we want to add a few extra columns with the information that we can read from our dataset.

- Dropping Cols and Rows
- Filling missing Values
- Tranformation of the Data
- Generate New Features
- Encoding of Categorical Features

```
The new Shape of Dataframe after dropping: (183698, 59)
```

```
FloorNumber filled.
RenovationYear filled.
Politics filled.
AreaProperty filled.
```

Now we would like to add the new additional features/columns. The new columns are information like:

- count AreaLiving and AreaProperty
- if AreaProperty has value
- years since Renovation
- if its rennovated
- count inserts per zip

	Id	AreaLiving	AreaProperty	BuiltYear	FloorNumber	ForestDensityM	GroupNa
0	7135329	140	501.0	2016	2.0	0.555985	
1	7170979	143	277.0	2004	2.0	0.074061	

2 rows × 66 columns

4.3. Train-Test-Split

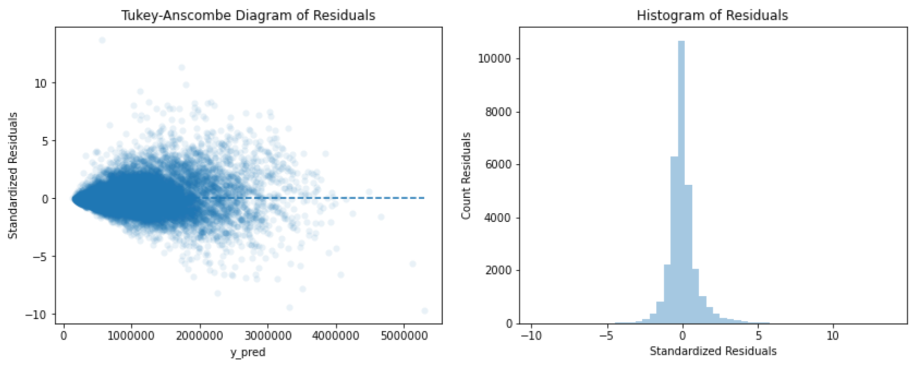
Here we are again at splitting the data with the same randim_state.

4.4. Train Model

First we perform a model with no adjustments on the hyperparameters and we have to look on the important features as well.

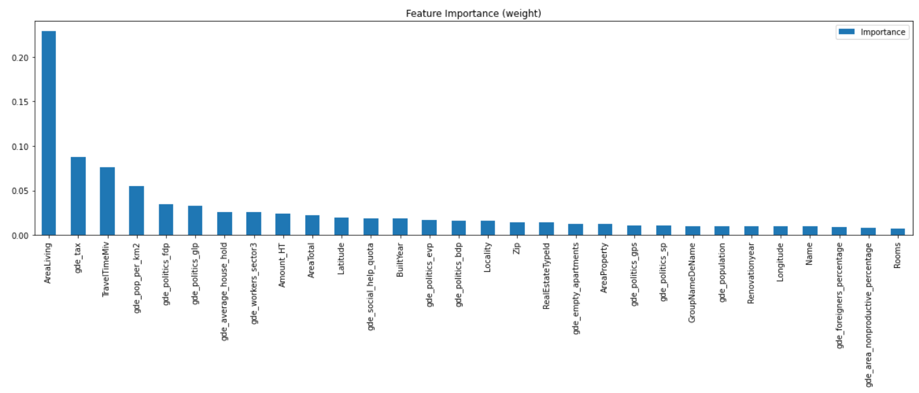
```
Time:  4.668166800000108
R2-Score:  0.8875905279349239


Mean ABS Error:  150065.16
Mean ABS Percentage Error:  0.15839418885190523
Median ABS Error:  89839.25
```



Compared to the RandomForestRegressor, the xgboost is a lot faster. The mape is very similar for both.

Let's just take a quick look which feature play an important role in the prediction:



You can clearly see that AreaLiving plays an important role here. But we detected that our model gets worser with less data (information) and the goal of the challenge is to get a score as good as possible, we decided to keep all features for the final prediction of the PurchasePrice.

4.5. Train Model with modified hyperparameters

To search for optimal parameters we performed a RandomizedSearchCV first to get a clue in which directions the paremeters should be adapted. Afterwards we performed a GridSearchCV to get through all possible parameter ranges found in Randomized Search.

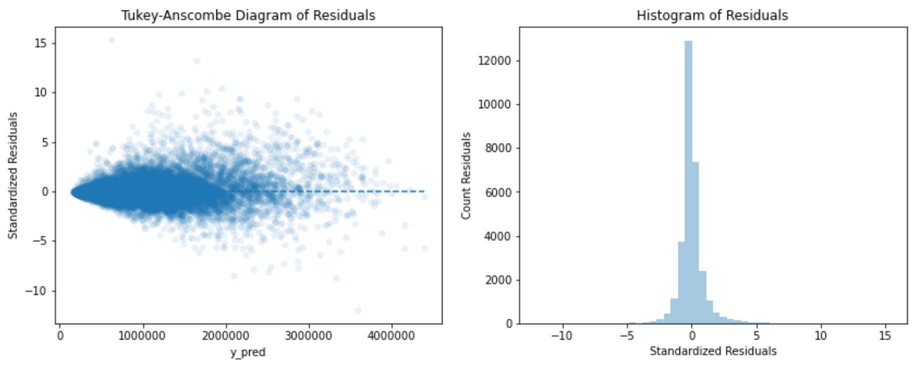
The optimal parameters we found based on our data was the following model which we submitted to the Kaggle-Challenge.

- n_estimators = 1000,
- eta = .02,
- max_depth = 15,
- alpha = .2,
- colsample_bytree = .6,
- subsample = .7

so lets train it and see the results:

```
Time:  314.0501340999999
R2-Score:  0.9982658104916367

Mean ABS Error:  125234.79
Mean ABS Percentage Error:  0.13199822769329156
Median ABS Error:  69211.0
```



By tuning with the parameters, we increase the running time extremely, but this allows us to achieve a mape of up to 13%!

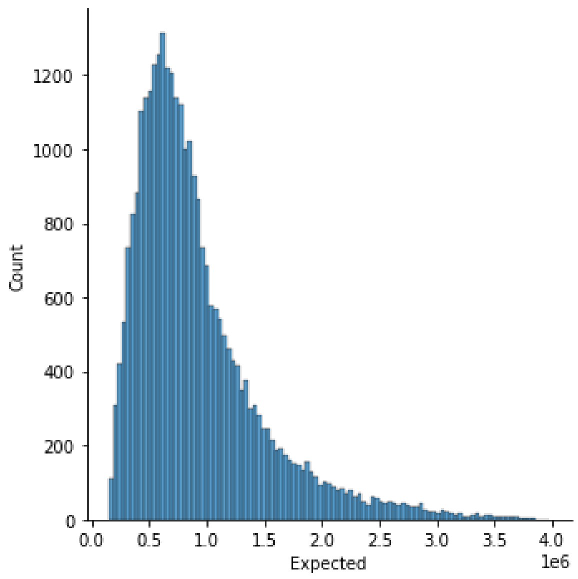
5. Kaggle-Challenge

5.1. train Model with the whole dataset

lets train the model with the same parameters on the whole dataset and save it.

```
XGBRegressor(alpha=0.2, base_score=0.5, booster='gbtree', colsample_byl
evel=1,
             colsample_bynode=1, colsample_bytree=0.6, eta=0.02, gamma=
0,
             gpu_id=-1, importance_type='gain', interaction_constraints
='',
             learning_rate=0.0199999996, max_delta_step=0, max_depth=1
5,
             min_child_weight=1, missing=nan, monotone_constraints
='()',
             n_estimators=2000, n_jobs=24, num_parallel_tree=1, random_
state=0,
             reg_alpha=0.200000003, reg_lambda=1, scale_pos_weight=1,
             subsample=0.7, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

5.2. Predict Test Data



This diagram serves primarily as a control. With the help of this diagram we can roughly check whether the prediction makes sense.

5.2. Save Prediction

And after saving the prediction as csv, we are ready to submit this file to the Kaggle-Challenge. With this model we achieve a MAPE of 12.97703

6. Sources

A Beginners Guide to Random Forest Regression https://medium.com/datadriveninvestor/random-forest-regression-9871bc9a25eb (https://medium.com/datadriveninvestor/random-forest-regression-9871bc9a25eb)

XGBoost Parameters: https://xgboost.readthedocs.io/en/latest/parameter.html (https://xgboost.readthedocs.io/en/latest/parameter.html)

Notes on Parameter Tuning: https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html (https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html)

Notebooks:

Firat Saritas, F. S. & Simon Staehli, S. S. (2020). EDA_Master.ipynb [Jupyter Notebook].