Prague University of Economics and Business

Faculty of Informatics and Statistics

****

Airbnb: Price prediction in Prague

4IT439 – Data-X – Applied data analytics models in real world tasks

Term paper

Team 3: Tomáš Mikulenka, Lukáš Kuthan, Adéla Smrčková, Jana Štolcová

Date: 6. 5. 2024

Content

[1 Data Understanding 3](#_Toc165913851)

[2 Data Preparation 7](#_Toc165913852)

[3 Data Visualization 11](#_Toc165913853)

[4 Modelling 13](#_Toc165913854)

[4.1 X a Y values 13](#_Toc165913855)

[4.2 Linear regression 14](#_Toc165913856)

[4.3 Lasso regression 15](#_Toc165913857)

[4.4 Random forest 15](#_Toc165913858)

[4.5 XG Boost 17](#_Toc165913859)

[5 Model interpretation 19](#_Toc165913860)

[5.1 Linear regression 19](#_Toc165913861)

[5.2 Lasso regression 20](#_Toc165913862)

[5.3 Random forest 21](#_Toc165913863)

[5.4 XG Boost 22](#_Toc165913864)

[5.5 Overall 24](#_Toc165913865)

# Data Understanding

We used python to explore the data. At first we downloaded all three versions of everything from the website. Then we went and looked at it one by one.

* Neighbourhoods – the same, kept one for possible binning (already included in listings)
* Reviews – all reviews in one file, kept the newest one from December
* Calendars – the data from December seem strange, so we decided to use only the one from June and September
* Listings – also a bit strange data from December, so we used only June and September

Here are some insights from our data exploration of the listings:

* price – wrong currency, needs to be a float, but won´t need to use it, we will use the price from calendar (the same case)
* bathroom\_text – text, needs to be transformed, gives us information about the number of bathrooms and also if it is shared or not
* bathroom – empty column, will be dropped
* neighbourhood – can be dropped, cleaned data are already in the column neighbourhood\_cleansed
* amenities – all equipment of the accommodation, could be helpful in the model if properly handled
* price, room type, accommodates, beds, bedrooms, bathrooms, neighbourhood will be probably the most important features for the model
* multiple other empty columns, will be dropped
* true false columns will be also transformed into Boolean 1/0

For the purpose of this report we have used python to give us a table (on the following page), which describes all the columns in listings.

We decided not to include the grey columns in the model, because they don´t seem to us as relevant. The others we want in the model, the green ones will stay as they are and the orange ones need to be transformed. The transformations are described in the next chapter.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column** | **Data Type** | **Non-null val** | **Missing val** | **Unique Count** | **Min Value** | **Max Value** | **Mean Value** | **SD** | **Outlier Count** |
| id | int64 | 8949 | 0 | 8949 |  |  |  |  |  |
| listing\_url | object | 8949 | 0 | 8949 |  |  |  |  |  |
| scrape\_id | int64 | 8949 | 0 | 1 |  |  |  |  |  |
| last\_scraped | object | 8949 | 0 | 1 |  |  |  |  |  |
| source | object | 8949 | 0 | 2 |  |  |  |  |  |
| name | object | 8949 | 0 | 6728 |  |  |  |  |  |
| description | object | 8785 | 164 | 7599 |  |  |  |  |  |
| neighborhood\_overview | object | 4904 | 4045 | 3126 |  |  |  |  |  |
| picture\_url | object | 8949 | 0 | 8627 |  |  |  |  |  |
| host\_id | int64 | 8949 | 0 | 3180 | 3128 | 537229454 | 167351187,56 | 176419788 | 0 |
| host\_url | object | 8949 | 0 | 3180 |  |  |  |  |  |
| host\_name | object | 8949 | 0 | 1383 |  |  |  |  |  |
| host\_since | object | 8949 | 0 | 2144 |  |  |  |  |  |
| host\_location | object | 7025 | 1924 | 191 |  |  |  |  |  |
| host\_about | object | 5259 | 3690 | 1479 |  |  |  |  |  |
| host\_response\_time | object | 7860 | 1089 | 4 |  |  |  |  |  |
| host\_response\_rate | object | 7860 | 1089 | 47 |  |  |  |  |  |
| host\_acceptance\_rate | object | 8284 | 665 | 92 |  |  |  |  |  |
| host\_is\_superhost | object | 8637 | 312 | 2 |  |  |  |  |  |
| host\_thumbnail\_url | object | 8949 | 0 | 3048 |  |  |  |  |  |
| host\_picture\_url | object | 8949 | 0 | 3048 |  |  |  |  |  |
| host\_neighbourhood | object | 8197 | 752 | 132 |  |  |  |  |  |
| host\_listings\_count | int64 | 8949 | 0 | 68 | 0 | 2562 | 21,96 | 72 | 27 |
| host\_total\_listings\_count | int64 | 8949 | 0 | 83 | 0 | 5324 | 36,50 | 219 | 15 |
| host\_verifications | object | 8949 | 0 | 6 |  |  |  |  |  |
| host\_has\_profile\_pic | object | 8949 | 0 | 2 |  |  |  |  |  |
| host\_identity\_verified | object | 8949 | 0 | 2 |  |  |  |  |  |
| neighbourhood | object | 4904 | 4045 | 236 |  |  |  |  |  |
| neighbourhood\_cleansed | object | 8949 | 0 | 52 |  |  |  |  |  |
| neighbourhood\_group\_cleansed | float64 | 0 | 8949 | 0 |  |  |  |  | 0 |
| latitude | float64 | 8949 | 0 | 5334 | 50 | 50 | 50,08 | 0 | 19 |
| longitude | float64 | 8949 | 0 | 6193 | 14 | 15 | 14,43 | 0 | 19 |
| property\_type | object | 8949 | 0 | 60 |  |  |  |  |  |
| room\_type | object | 8949 | 0 | 4 |  |  |  |  |  |
| accommodates | int64 | 8949 | 0 | 16 | 1 | 16 | 3,93 | 2 | 0 |
| bathrooms | float64 | 0 | 8949 | 0 |  |  |  |  | 0 |
| bathrooms\_text | object | 8933 | 16 | 34 |  |  |  |  |  |
| bedrooms | float64 | 7030 | 1919 | 15 | 1 | 34 | 1,57 | 1 | 33 |
| beds | float64 | 8833 | 116 | 26 | 1 | 50 | 2,61 | 2 | 83 |
| amenities | object | 8949 | 0 | 8040 |  |  |  |  |  |
| price | object | 8949 | 0 | 3121 |  |  |  |  |  |
| minimum\_nights | int64 | 8949 | 0 | 63 | 1 | 1115 | 4,97 | 32 | 295 |
| maximum\_nights | int64 | 8949 | 0 | 128 | 1 | 9000 | 590,46 | 464 | 2 |
| minimum\_minimum\_nights | int64 | 8949 | 0 | 61 | 1 | 1115 | 4,43 | 29 | 269 |
| maximum\_minimum\_nights | int64 | 8949 | 0 | 66 | 1 | 1115 | 9,64 | 41 | 285 |
| minimum\_maximum\_nights | int64 | 8949 | 0 | 97 | 1 | 3333 | 693,11 | 472 | 1 |
| maximum\_maximum\_nights | int64 | 8949 | 0 | 97 | 1 | 3333 | 751,94 | 451 | 1 |
| minimum\_nights\_avg\_ntm | float64 | 8949 | 0 | 231 | 1 | 1115 | 6,16 | 32 | 218 |
| maximum\_nights\_avg\_ntm | float64 | 8949 | 0 | 298 | 1 | 3333 | 734,00 | 451 | 1 |
| calendar\_updated | float64 | 0 | 8949 | 0 |  |  |  |  | 0 |
| has\_availability | object | 8949 | 0 | 2 |  |  |  |  |  |
| availability\_30 | int64 | 8949 | 0 | 31 | 0 | 30 | 9,90 | 9 | 0 |
| availability\_60 | int64 | 8949 | 0 | 61 | 0 | 60 | 25,93 | 20 | 0 |
| availability\_90 | int64 | 8949 | 0 | 91 | 0 | 90 | 43,06 | 31 | 0 |
| availability\_365 | int64 | 8949 | 0 | 366 | 0 | 365 | 159,07 | 133 | 0 |
| calendar\_last\_scraped | object | 8949 | 0 | 1 |  |  |  |  |  |
| number\_of\_reviews | int64 | 8949 | 0 | 499 | 0 | 1693 | 63,26 | 100 | 6 |
| number\_of\_reviews\_ltm | int64 | 8949 | 0 | 121 | 0 | 385 | 17,72 | 23 | 6 |
| number\_of\_reviews\_l30d | int64 | 8949 | 0 | 19 | 0 | 23 | 1,50 | 2 | 4 |
| first\_review | object | 8066 | 883 | 2585 |  |  |  |  |  |
| last\_review | object | 8066 | 883 | 932 |  |  |  |  |  |
| review\_scores\_rating | float64 | 8066 | 883 | 146 | 0 | 5 | 4,69 | 0 | 49 |
| review\_scores\_accuracy | float64 | 8050 | 899 | 132 | 1 | 5 | 4,74 | 0 | 42 |
| review\_scores\_cleanliness | float64 | 8050 | 899 | 164 | 1 | 5 | 4,68 | 0 | 36 |
| review\_scores\_checkin | float64 | 8050 | 899 | 133 | 1 | 5 | 4,79 | 0 | 73 |
| review\_scores\_communication | float64 | 8051 | 898 | 129 | 1 | 5 | 4,79 | 0 | 79 |
| review\_scores\_location | float64 | 8050 | 899 | 124 | 1 | 5 | 4,77 | 0 | 55 |
| review\_scores\_value | float64 | 8050 | 899 | 136 | 1 | 5 | 4,66 | 0 | 35 |
| license | float64 | 0 | 8949 | 0 |  |  |  |  | 0 |
| instant\_bookable | object | 8949 | 0 | 2 |  |  |  |  |  |
| calculated\_host\_listings\_count | int64 | 8949 | 0 | 53 | 1 | 96 | 17,06 | 24 | 0 |
| calculated\_host\_listings\_count\_entire\_homes | int64 | 8949 | 0 | 50 | 0 | 96 | 14,30 | 23 | 0 |
| calculated\_host\_listings\_count\_private\_rooms | int64 | 8949 | 0 | 22 | 0 | 59 | 1,82 | 7 | 112 |
| calculated\_host\_listings\_count\_shared\_rooms | int64 | 8949 | 0 | 7 | 0 | 77 | 0,78 | 7 | 186 |
| reviews\_per\_month | float64 | 8066 | 883 | 782 | 0 | 27 | 2,00 | 2 | 5 |

Legend

|  |
| --- |
| not for model |
| adjust |
| keep |

# Data Preparation

Here is the next version of the previous table, where each parameter is reasoned with why keep it or not and what changes we did.

Specifics can be found in the code. We have merged the June and September data.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Keep** | **Change** | **Reason** |
| id | No |  | Not relevant, could be misleading |
| listing\_url | No |  | Not relevant, could be misleading |
| scrape\_id | No |  | Not relevant, could be misleading |
| last\_scraped | No |  | Not relevant, could be misleading |
| source | No |  | Not relevant, could be misleading |
| name | No |  | Not relevant, could be misleading |
| description | Yes, but | Count number of signs | Could be relevant |
| neighborhood\_overview | Yes, but | Is or not - boolean | Could be relevant |
| picture\_url | No |  | Not relevant, could be misleading |
| host\_id | No |  | Not relevant, could be misleading |
| host\_url | No |  | Not relevant, could be misleading |
| host\_name | No |  | Not relevant, could be misleading |
| host\_since | Yes, but | Difference from today and in bins (less than year, more than year) | Could be relevant |
| host\_location | No |  | Not relevant, could be misleading |
| host\_about | Yes, but | Is or not - boolean | Could be relevant |
| host\_response\_time | No |  | Not relevant, could be misleading |
| host\_response\_rate | No |  | Too many null values |
| host\_acceptance\_rate | No |  | Not relevant, could be misleading |
| host\_is\_superhost | Yes, but | 1/0 boolean | Could be relevant |
| host\_thumbnail\_url | No |  | Not relevant, could be misleading |
| host\_picture\_url | No |  | Not relevant, could be misleading |
| host\_neighbourhood | No |  | Not relevant, could be misleading |
| host\_listings\_count | No |  | Not relevant, could be misleading |
| host\_total\_listings\_count | No |  | Not relevant, could be misleading |
| host\_verifications | No |  | Not relevant, could be misleading |
| host\_has\_profile\_pic | Yes, but | 1/0 boolean | Could be relevant |
| host\_identity\_verified | Yes, but | 1/0 boolean | Could be relevant |
| neighbourhood | No |  | Not relevant, could be misleading |
| neighbourhood\_cleansed | Yes, but | Make bins | Relevant |
| neighbourhood\_group\_cleansed | No |  | Empty |
| latitude | No |  | Not relevant, could be misleading |
| longitude | No |  | Not relevant, could be misleading |
| property\_type | No |  | Not relevant, could be misleading |
| room\_type | Yes |  | Relevant |
| accommodates | Yes |  | Relevant |
| bathrooms | No |  | Empty |
| bathrooms\_text | Yes, but | Convert to number and create a new column boolean shared\_bathroom | Relevant |
| bedrooms | Yes, but | Replace blanks with 1 | Relevant |
| beds | Yes, but | Replace blanks with 1 | Relevant |
| amenities | No |  | Too many individual values |
| price | No |  | Not relevant, could be misleading |
| minimum\_nights | No |  | Not relevant, could be misleading |
| maximum\_nights | No |  | Not relevant, could be misleading |
| minimum\_minimum\_nights | No |  | Not relevant, could be misleading |
| maximum\_minimum\_nights | No |  | Not relevant, could be misleading |
| minimum\_maximum\_nights | No |  | Not relevant, could be misleading |
| maximum\_maximum\_nights | No |  | Not relevant, could be misleading |
| minimum\_nights\_avg\_ntm | No |  | Not relevant, could be misleading |
| maximum\_nights\_avg\_ntm | No |  | Not relevant, could be misleading |
| calendar\_updated | No |  | Empty |
| has\_availability | No |  | Not relevant, could be misleading |
| availability\_30 | No |  | Not relevant, could be misleading |
| availability\_60 | No |  | Not relevant, could be misleading |
| availability\_90 | No |  | Not relevant, could be misleading |
| availability\_365 | No |  | Not relevant, could be misleading |
| calendar\_last\_scraped | No |  | Not relevant, could be misleading |
| number\_of\_reviews | Yes |  | Could be relevant |
| number\_of\_reviews\_ltm | No |  | Not relevant, could be misleading |
| number\_of\_reviews\_l30d | No |  | Not relevant, could be misleading |
| first\_review | No |  | Not relevant, could be misleading |
| last\_review | No |  | Not relevant, could be misleading |
| review\_scores\_rating | Yes |  | Could be relevant |
| review\_scores\_accuracy | No |  | Not relevant, could be misleading |
| review\_scores\_cleanliness | No |  | Not relevant, could be misleading |
| review\_scores\_checkin | No |  | Not relevant, could be misleading |
| review\_scores\_communication | No |  | Not relevant, could be misleading |
| review\_scores\_location | No |  | Not relevant, could be misleading |
| review\_scores\_value | No |  | Not relevant, could be misleading |
| license | No |  | Empty |
| instant\_bookable | Yes, but | 1/0 boolean | Could be relevant |
| calculated\_host\_listings\_count | No |  | Not relevant, could be misleading |
| calculated\_host\_listings\_count\_entire\_homes | No |  | Not relevant, could be misleading |
| calculated\_host\_listings\_count\_private\_rooms | No |  | Not relevant, could be misleading |
| calculated\_host\_listings\_count\_shared\_rooms | No |  | Not relevant, could be misleading |
| reviews\_per\_month | No |  | Not relevant, could be misleading |

From the calendar we have decided to keep only the “listing\_id”, “date” and “price” features.

First, we had to work with the price a lot, where we converted it to number and then we removed outliers (using quantile 0.01 and 0.96). Then we grouped the date for every listing id by year and month and averaged the price.

Afterwards, we joined the listings with calendar to get the final dataset for the modelling.

# Data Visualization

This is a graph which helped us to determine whether we need to focus on removing outliers and we decided that it is necessary, so our models are more accurate. It is not a very nice-looking graph, but it shows us what we needed.

A diagram of a box plot

Description automatically generated

This following graph was done on the already transformed and prepared data just to see if our expectation that the price should be lowest in outskirts and highest in center is correct and that we have done the binning properly. When we look at the graph, we are satisfied with what this graph is showing us.A graph with blue rectangular bars

Description automatically generated with medium confidence

# Modelling

In the following chapter, we present the models we selected for training on our data. Each model was first trained on the training data, then we compared their success on the validation data, and then we tested the best models on the test data.

The results shown below are the results of validation data.

## X a Y values

We used the same set of explained and explanatory variables for all the models we trained. Which we have modified to be applicable to our models. For example, we made the neighborhood\_cleansed and room\_type variables into dummy variables that are understandable for the models.

# Making of dummy values for model procesing

# One-Hot Encoding for neighbourhood\_cleansed and room\_type

neighbourhood\_dummies = pd.get\_dummies(data['neighbourhood\_cleansed'], prefix='neighb-')

room\_dummies = pd.get\_dummies(data['room\_type'], prefix='room-')

# Adding new dummy columns to dataframe data

data = pd.concat([data, neighbourhood\_dummies], axis=1)

data = pd.concat([data, room\_dummies], axis=1)

# sorting data by date

data\_sorted = data.sort\_values(by='date', ascending=True)

# validation of succesful sorting logic

print(data\_sorted['date'])

# setting borders

total\_rows = len(data\_sorted)

train\_idx = int(total\_rows \* 0.6)

valid\_idx = int(total\_rows \* 0.8)

# train 0 - 60 %

X\_train = data\_sorted.iloc[:train\_idx].drop(['avg\_price','neighbourhood\_cleansed','room\_type','date','neighb\_num'], axis=1)

y\_train = data\_sorted.iloc[:train\_idx]['avg\_price']

# valid 60 - 80 %

X\_valid = data\_sorted.iloc[train\_idx:valid\_idx].drop(['avg\_price','neighbourhood\_cleansed','room\_type','date', 'neighb\_num'], axis=1)

y\_valid = data\_sorted.iloc[train\_idx:valid\_idx]['avg\_price']

# test 80 - 100 %

X\_test = data\_sorted.iloc[valid\_idx:].drop(['avg\_price','neighbourhood\_cleansed','room\_type','date', 'neighb\_num'], axis=1)

y\_test = data\_sorted.iloc[valid\_idx:]['avg\_price']

To avoid overfitting the model, we split the data into training, validation and test data. Therefore, the X and Y variables were also divided into train, valid and test. Because the data is collected over time and our prediction will take place in the future, it was necessary to sort the data by date before splitting. To train our models on the oldest data.

## Linear regression

We decided to train a linear regression model first. To train the model we used the sklearn library. The model was fit on the training data. The Python code used for the linear regression is shown below.

from sklearn.metrics import mean\_squared\_error

from sklearn.linear\_model import LinearRegression

import pickle

# Fit model to train dataset

model = LinearRegression()

model.fit(X\_train, y\_train)

.

# Predict the test set labels 'y\_pred'

y\_pred = model.predict(X\_valid)

# Evaluate model on validation dataset

mse = mean\_squared\_error(y\_valid, y\_pred)

rmse\_LR = mse\*\*(1/2)

# Print the results

print("Intercept:", model.intercept\_ )

print("Coefficients:", model.coef\_)

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:",rmse\_LR)

The RMSE of our linear regression model is 2403 Kč.

## Lasso regression

Second, we trained a lasso regression model, where we hoped that the regularization parameter alpha would reduce the number of coefficients and lead to better results. We tried many different alpha parameters, but in the end, parameter 1 worked best, which led to a lasso regression result that was slightly worse than the ordinary linear regression result. In the following lines you can see the code that was used to fit the model.

from sklearn.linear\_model import Lasso

import pickle

# Set alpha parameter and initialize lasso regression model

alpha = 1

lasso = Lasso(alpha=alpha)

# Fit model to train dataset

lasso.fit(X\_train, y\_train)

# Predict the test set labels 'y\_pred'

y\_pred\_LAR = lasso.predict(X\_valid)

# Evaluating the model on validation dataset

mse = mean\_squared\_error(y\_valid, y\_pred\_LAR)

rmse\_LAR = mse\*\*(1/2)

# Print the results

print("Intercept:", lasso.intercept\_)

print("Coefficients:", lasso.coef\_)

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:", rmse\_LAR)

The RMSE of our lasso regression model is 2406 Kč.

## Random forest

The next model we tried was random forest, where we expected better results. At first we have tried to strictly specify parameters as can be seen below:

# Initial model with strict parameters

forest\_model\_0 = RandomForestRegressor(n\_estimators=3000,

min\_samples\_split=10,

max\_features = 5,

random\_state=SEED)

# Fit RF to the training set

forest\_model\_0.fit(X\_train, y\_train.values.ravel()) #values.ravel() flattened array expected by RandomForestRegressor

# Predict the validation set labels 'y\_pred'

y\_pred= forest\_model\_0.predict(X\_valid)

# Evaluate the validation set RMSE

rmse\_RF\_0 = mean\_squared\_error(y\_valid, y\_pred, squared=False)

print(rmse\_RF\_0)

The RMSE of this random forest is 1601,52 Kč.

After that we have tried our most time-consuming model where we tried to create a selection of different parameters. Fitting 3 folds for each of 54 candidates, totalling 162 fits.

# Create the grid with multiple set of parameters

hyper\_grid = {'n\_estimators': [4000,5000,5500],

'max\_features': [5,9,12],

'min\_samples\_split': [10, 20],

'max\_depth': [None, 5, 10]

}

# Reinstantiate RandomForestRegressor model

forest\_model\_cv = RandomForestRegressor()

# Instantiate the GridSearchCV with forest\_model\_cv as estimator

grid\_search = GridSearchCV(estimator = forest\_model\_cv, param\_grid = hyper\_grid,

cv = 3, n\_jobs = -1, verbose = 2)

# Fit the grid search to the data

grid\_search.fit(X\_train, y\_train.values.ravel())

#best parameters

print(grid\_search.best\_params\_)

#best estimator

forest\_model\_opt= grid\_search.best\_estimator\_

The best option we got was this: {'max\_depth': None, 'max\_features': 12, 'min\_samples\_split': 10, 'n\_estimators': 4000}.

# Predict the validation set labels 'y\_pred'

y\_pred\_cv = forest\_model\_opt.predict(X\_valid)

rmse\_RF\_opt = mean\_squared\_error(y\_valid, y\_pred\_cv, squared=False)

print("RMSE for optimized RF by GridSearchCV is: " ,rmse\_RF\_opt)

The RMSE of this random forest is 1534 Kč, which makes it a litter better than our random forest before.

## XG Boost

Our final model was XG Boost, where we decided to do it similarly to random forest. First choose parameters, then selection. This model was faster than random forest and it gave us better results, the best of all models.

# Initial model with strict parameters

gbm0 = xgb.XGBRegressor(n\_estimators = 50, learning\_rate = 0.1, objective='reg:squarederror',seed = SEED)

gbm0.fit(X\_train, y\_train)

# Predict the validation set labels 'y\_pred0'

y\_pred0 = gbm0.predict(X\_valid)

# Evaluate the validation set RMSE

rmse\_test0 = mean\_squared\_error(y\_valid, y\_pred0, squared=False)

print(rmse\_test0)

This XG Boost model gave us the RMSE of 2208,7 Kč, which was worse than the random forest. Then we tried to optimize it. Fitting 3 folds for each of 162 candidates, totalling 486 fits.

# setup params grid

param\_grid = {'learning\_rate': [0.1,0.5,0.9], #alias eta, Step size shrinkage used in update to prevents overfitting.

'n\_estimators': [20, 50, 100],

'subsample': [0.5, 0.8, 1], #Subsample ratio of the training instances

'max\_depth': [5, 8, 11],

'colsample\_bytree': [0.5, 1] #colsample\_bytree is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.

}

# initiate XGBRegressor

gbm = xgb.XGBRegressor(seed=SEED, objective='reg:squarederror')

grid\_mse = GridSearchCV(estimator=gbm,

param\_grid=param\_grid,

scoring='neg\_mean\_squared\_error',

cv=3,

verbose=1,

n\_jobs=-1)

#fit GridSearchCV

grid\_mse.fit(X\_train, y\_train)

print("Best parameters found: ",grid\_mse.best\_params\_)

print("Lowest RMSE found: ", np.sqrt(np.abs(grid\_mse.best\_score\_)))

#extract the estimator best\_estimator\_

xgbm\_opt = grid\_mse.best\_estimator\_

The best parameters found were: {'colsample\_bytree': 0.5, 'learning\_rate': 0.5, 'max\_depth': 11, 'n\_estimators': 100, 'subsample': 1}.

The RMSE of this XS Boost is 1287 Kč, which makes it a winner.

# Model interpretation

In the following section, the interpretations of the individual trained models and the presentation of their results are presented. We chose RMSE as the main comparison metric for all models, which we calculate for each model used.

## Linear regression

The following table shows the results of the linear regression trained on the training data. Mean squared error was calculated after control prediction on the validation data. The results show that the model is not very reliable there is mean squared error of 2404 Kč. The array of coefficients represents the values by which the values of the individual attributes entering the model are multiplied. For example, the very first is the coefficient value for the attribute description. The coefficients are for each attribute in turn as they entered the prediction model.

|  |  |
| --- | --- |
| Variable | Values |
| Intercept | -601152.3217238288 |
| Coeficients | [ 5.70560978 22.73911238 -130.36032637 -55.00277619 199.31731746  429.44269482 -918.90714623 403.68461469 209.79935586 -79.76090382  -2.39249414 203.46521552 245.94407167 -130.36032637 62.2399841  -503.21439886 297.31264558 38.99819051 -195.82174176 676.85574994  -481.03400817 56.06359818 639.56760892 108.49498473 -804.12619182] |
| Mean Squared Error | 5777962.777503724 |
| Root Mean Squared Error | 2403.739332270395 |

For a better idea, we decided to visualize the results of the linear regression. As can be seen from the following graph, there is no large linear dependence in the data. For better understanding, we added a red line to the graph showing the ideal regression.

Obsah obrázku text, Vykreslený graf, řada/pruh, diagram

Popis byl vytvořen automaticky

## Lasso regression

We trained the lasso regression model on the training data to obtain coefficients for each explanatory attribute and intercept. When predicting on the validation data, we then also obtained the mean squared error and root mean squared error. These results show that the prediction model is not very accurate. The root mean squared error is 2406 Kč. This is slightly worse than the result obtained using linear regression. We were surprised by this comparison of results, as we expected the lasso regression to perform better. However, there is very weak linear dependence between the data, so neither of the models using linear dependence performs very well.

|  |  |
| --- | --- |
| Variable | Values |
| Intercept | -554864.9217283456 |
| Coeficients | [ 6.51754046 18.64938677 -58.20277747 -50.45990542 193.25813266 371.24951444 -889.94211536 406.56873007 209.73652472 -82.97424289 -2.39099193 199.64321291 240.51406609 -148.89703873 56.10230389 -486.96076918 274.38286681 36.46507818 -0. 869.51694696 -259.65078537 -12.52204476 514.6849589 23.81181889 -777.3042681 ] |
| Mean Squared Error | 5789605.883727598 |
| Root Mean Squared Error | 2406.159987142916 |

To illustrate, we decided to visualize the results from the lasso regression prediction in the following graph. The attentive reader will notice that the results are almost the same as the linear regression plot. To give a better idea, we have added a red line to the plot showing the ideal linearity.

Obsah obrázku text, Vykreslený graf, řada/pruh, diagram

Popis byl vytvořen automaticky

## Random forest

The RMSE of our best random forest is 1601,52 Kč, which is way better than the linear or lasso regression as expected.

We have decided to use feature importance to explain the features in the random forest model. We also wanted to try the SHAP explainer, but due to very complex RF we have decided to skip it for random forest, since it is not the best model.

The feature importance is what we expected it to be, making the number of accommodates the highest ranked feature. All features can be seen in the plot below:

A graph with green and black text

Description automatically generated

## XG Boost

The RMSE of our best XG Boost model is 1287 Kč, which makes it a winner of all models.

For XG Boost we have used the SHAP values technique to explain the features in the model.

A screenshot of a graph

Description automatically generated

The features in the XG Boost model are also as we have expected. For better interpretation we have used also other plots as can be seen below. To our surprise, the number of reviews effects the model in other way than was expected – the more reviews, the lower the price (generally speaking).

A graph of different colored lines

Description automatically generated

In the following plot we have chosen the 100th row to see how the features effected the price.

A graph with blue rectangles and black text

Description automatically generated

We have tried even more plots which can be found in the jupyter notebook.

## Overall

In the end, we have tested the four winning models in its category on the testing data. The result we got was following:

A graph of a comparison of a test

Description automatically generated

This makes the XG Boost still the best option for the price prediction in Prague.