# University of Economics in Prague

# Faculty of Informations and Statistics

# 

## 

## Semester's thesis Data-X – applied data analytics models in real world tasks

### Summer 2025

Tereza Hradská, Josef Maaz, Michaela Nová, Michal Pražský, Mario Slavev, Petr Sychra

# Contents

[Contents 2](#_heading=)

[Data Understanding 3](#_heading=)

[Missing values 3](#_heading=)

[Outliers’ analysis 4](#_heading=)

[Unique values 5](#_heading=)

[**Data preparation 8**](#_heading=h.sbdqptl78gkm)

[**Data visualization 11**](#_heading=h.75lwjlyebpar)

[**Modelling 13**](#_heading=h.fxhiraxga778)

[Model Architecture 13](#_heading=h.802zk0sm4tyv)

[Validation Process 14](#_heading=h.jem95dagqmz5)

[Model Limitations and Considerations 14](#_heading=h.kqhxbl2iw7m9)

[Ideas to Improve the Model 14](#_heading=h.22pfgaslonsr)

[Explain How You Chose the Values for the Hyperparameters of Your Model 15](#_heading=h.2a02suo20gd6)

[**Model evaluation & interpretation 16**](#_heading=h.n06ducbutsmu)

[Basic Numeric Metrics 16](#_heading=h.h6cfrropii8o)

[Actual vs Predicted Comparison 17](#_heading=h.5gvljhv82qnm)

[Residual Analysis 18](#_heading=h.cbjdg7uolf3j)

[Feature Importance 19](#_heading=h.nssfat99dce2)

[Permutation importance 21](#_heading=h.qcdlf5596lk6)

[**Literature 23**](#_heading=h.dgmfmd607nuz)

# Data Understanding

The dataset used for predicting the nightly price of accommodation in Prague originates from the Airbnb platform and consists of 10,108 records and 79 attributes describing individual listings. Each record represents a unique accommodation offer, as confirmed by the analysis of the id column, which contains exactly 10,108 unique values. This indicates that there are no duplicate entries in the dataset, and each row corresponds to a distinct listing.

The dataset includes 24 columns of type integer, 22 columns of type float, and 33 columns of type object. These object-type columns typically contain textual or categorical data.

## Missing values

Several columns in the dataset contain missing values. Most notably, the columns neighbourhood\_group\_cleansed, license, and calendar\_updated are entirely empty, with 100% missing values, and will likely be excluded from further analysis. In addition, neighbourhood\_overview and neighbourhood each have missing values in 54.1% of the records, while host\_about and host\_location are missing in 39.3% and 22.9% of cases, respectively. The price column, although essential for modeling, is missing in 12.9% of the entries and requires careful treatment. Similarly, estimated\_revenue\_l365d, beds, and bathrooms are missing in around 12–13% of listings, all of which are important features for price prediction.

Several review-related columns, such as review\_scores\_rating, review\_scores\_accuracy, review\_scores\_location, review\_scores\_communication, review\_scores\_checkin, first\_review, last\_review, and reviews\_per\_month, are missing in approximately 9.6% of the records, mostly for listings without guest reviews. Other fields, including host\_response\_time, host\_response\_rate, host\_neighbourhood, host\_is\_superhost, and host\_acceptance\_rate, are partially missing, typically in 3–9% of cases. A few remaining attributes, such as description, has\_availability, or various host profile details, are missing only in a small fraction of the data—usually less than 3%.

| **Column** | **Missing values** | **Percentage %** |
| --- | --- | --- |
| Neighbourhood\_group\_cleansed | 10108 | 100 |
| License | 10108 | 100 |
| Calendar\_updated | 10108 | 100 |
| Neighborhood\_over\_view | 5470 | 54.12 |
| Neighbourhood | 5470 | 54.12 |
| Host\_about | 3976 | 39.34 |
| Host\_location | 2311 | 22.86 |
| Price | 1300 | 12.86 |
| Estimated\_revenue\_l365d | 1300 | 12.86 |
| Beds | 1254 | 12.41 |
| bathrooms | 1240 | 12.27 |
| Review\_scores\_checkin | 976 | 9.66 |
| Review\_scores\_location | 976 | 9.66 |
| Review\_scores\_value | 976 | 9.66 |
| Review\_scores\_accuracy | 976 | 9.66 |
| Review\_score\_rating | 975 | 9.65 |
| Last\_review | 975 | 9.65 |
| First\_review | 975 | 9.65 |
| Review\_score\_cleanliness | 975 | 9.65 |
| Review\_scores\_communication | 975 | 9.65 |
| Reviews\_per\_month | 975 | 9.65 |
| Host\_response\_time | 929 | 9.19 |
| Host\_response\_rate | 929 | 9.19 |
| Host\_neighbourhood | 680 | 6.73 |
| Host\_is\_superhost | 423 | 4.18 |
| Host\_acceptance\_rate | 391 | 3.87 |
| Bedrooms | 276 | 2.73 |
| Description | 241 | 2.38 |
| Has\_availibility | 83 | 0.82 |
| Bathrooms\_text | 20 | 0.2 |
| Host\_listings\_count | 1 | 0.01 |
| Host\_thumbnail\_url | 1 | 0.01 |
| Host\_total\_listings\_count | 1 | 0.01 |
| Host\_has\_profile\_pic | 1 | 0.01 |
| Host\_identity\_verified | 1 | 0.01 |
| Host\_since | 1 | 0.01 |
| Host\_name | 1 | 0.01 |
| Host\_picture\_url | 1 | 0.01 |

## Outliers’ analysis

Outliers were detected in several numerical columns. The most notable is the price column, where the majority of listings fall within a reasonable price range, but a small number of entries exceed 100 000 CZK per night, with the maximum reaching over 1.4 million CZK. These extreme values likely represent data entry errors or unusually high-end listings. The host\_listings\_count and host\_total\_listings\_count columns also contain outliers, with some hosts managing more than 3 000 and even 8 600 listings respectively, which may skew host-related patterns in the model.

Outliers were also identified in number\_of\_reviews, where some listings received over 1,800 reviews, and in several review score columns. Although review scores are officially bounded between 1 and 5, scores below the calculated lower threshold (e.g., for review\_scores\_rating) are statistically considered outliers. These values might reflect negative guest experiences or incomplete feedback and should be evaluated accordingly.

## Unique values

The dataset contains columns with varying levels of uniqueness:

* **100% Unique**: Columns like id, listing\_url, and host\_id serve as identifiers and are useful for referencing, not modeling.
* **70–99% Unique**: Columns such as picture\_url (96.92%) and name (95.37%) often contain free-form or personalized data. Amenities (90.76%) is valuable for feature engineering.
* **20–70% Unique**: Columns like description, longitude, latitude, estimated\_revenue\_l365d, and price hold meaningful and analyzable information.
* **<20% Unique**: Columns such as room\_type, accommodates, beds, and bedrooms have low cardinality and are suitable for summarization and encoding.
* **Binary Columns (2 values)**: Flags like instant\_bookable, host\_is\_superhost, and host\_identity\_verified are ready for direct analysis.
* **0 Unique Values**: Columns like license, calendar\_updated, and neighbourhood\_group\_cleansed are empty and should be removed.
* **1 Unique Value**: Technical columns such as scrape\_id, last\_scraped, and calendar\_last\_scraped offer no analytical value.

Of course columns like picture\_url and others are not suitable for analysis and are going to be removed due to no information gain but categories defined above are more of a guideline on how to think about columns from the perspective of unique values.

| **Column** | **Unique Values** | **Percentage %** |
| --- | --- | --- |
| id | 10108 | 100 |
| listing\_url | 10108 | 100 |
| picture\_url | 9796 | 96.92 |
| name | 9639 | 95.37 |
| amenities | 9173 | 90.76 |
| description | 7895 | 78.09 |
| longitude | 6997 | 69.23 |
| latitude | 6257 | 61.91 |
| estimated\_revenue\_l365d | 5947 | 58.84 |
| price | 3292 | 32.57 |
| host\_id | 3284 | 32.48 |
| host\_url | 3284 | 32.48 |
| host\_thumbnail\_url | 3115 | 30.81 |
| host\_picture\_url | 3115 | 30.81 |
| neighborhood\_overview | 2919 | 28.88 |
| first\_review | 2812 | 27.83 |
| host\_since | 2285 | 22.61 |
| host\_about | 1475 | 14.60 |
| host\_name | 1397 | 13.82 |
| last\_review | 905 | 8.95 |
| reviews\_per\_month | 808 | 7.99 |
| number\_of\_reviews | 577 | 5.71 |
| maximum\_nights\_avg\_ntm | 413 | 4.09 |
| availability\_365 | 366 | 3.62 |
| availability\_eoy | 292 | 2.89 |
| neighbourhood | 216 | 2.14 |
| minimum\_nights\_avg\_ntm | 214 | 2.11 |
| host\_location | 186 | 1.84 |
| review\_scores\_cleanliness | 158 | 1.56 |
| review\_scores\_rating | 136 | 1.34 |
| number\_of\_reviews\_ltm | 136 | 1.34 |
| number\_of\_reviews\_ly | 132 | 1.31 |
| review\_scores\_value | 131 | 1.30 |
| host\_neighbourhood | 131 | 1.30 |
| maximum\_nights | 129 | 1.28 |
| review\_scores\_checkin | 128 | 1.27 |
| review\_scores\_accuracy | 125 | 1.24 |
| review\_scores\_communication | 124 | 1.23 |
| review\_scores\_location | 113 | 1.12 |
| maximum\_maximum\_nights | 107 | 1.06 |
| minimum\_maximum\_nights | 104 | 1.03 |
| host\_total\_listings\_count | 94 | 0.93 |
| host\_acceptance\_rate | 92 | 0.91 |
| availability\_90 | 91 | 0.89 |
| estimated\_occupancy\_l365d | 86 | 0.85 |
| host\_listings\_count | 76 | 0.75 |
| maximum\_minimum\_nights | 62 | 0.61 |
| availability\_60 | 61 | 0.60 |
| calculated\_host\_listings\_count | 59 | 0.58 |
| minimum\_nights | 58 | 0.57 |
| calculated\_host\_listings\_count\_entire\_homes | 55 | 0.54 |
| minimum\_minimum\_nights | 55 | 0.54 |
| property\_type | 52 | 0.51 |
| neighbourhood\_cleansed | 52 | 0.51 |
| host\_response\_rate | 46 | 0.45 |
| availability\_30 | 31 | 0.31 |
| bathrooms\_text | 31 | 0.31 |
| number\_of\_reviews\_l30d | 26 | 0.26 |
| beds | 23 | 0.23 |
| calculated\_host\_listings\_count\_private\_rooms | 23 | 0.23 |
| bathrooms | 21 | 0.21 |
| accommodates | 16 | 0.16 |
| bedrooms | 15 | 0.15 |
| calculated\_host\_listings\_count\_shared\_rooms | 6 | 0.06 |
| host\_verifications | 5 | 0.05 |
| room\_type | 4 | 0.04 |
| host\_response\_time | 4 | 0.04 |
| instant\_bookable | 2 | 0.02 |
| host\_is\_superhost | 2 | 0.02 |
| source | 2 | 0.02 |
| host\_has\_profile\_pic | 2 | 0.02 |
| host\_identity\_verified | 2 | 0.02 |
| has\_availability | 2 | 0.02 |
| scrape\_id | 1 | 0.01 |
| last\_scraped | 1 | 0.01 |
| calendar\_last\_scraped | 1 | 0.01 |
| neighbourhood\_group\_cleansed | 0 | 0 |
| calendar\_updated | 0 | 0 |
| license | 0 | 0 |

# Data preparation

As part of the data preparation, firstly we have dropped duplicates, because we do not want to train model on duplicate rows. Next step was to choose columns that should be dropped.

| Columns to drop | Reason |
| --- | --- |
| id | unique value, not correlated |
| name | no correlation between target |
| description | text |
| license | missing values |
| host\_id | unique value, not correlated |
| host\_name | no correlation between target |
| host\_about | text |
| neighborhood\_overview | text |
| neighbourhood\_group\_cleansed | other similar column is used, missing values |
| listing\_url | text |
| calendar\_updated | missing values |
| scrape\_id | unique value not correlated |
| last\_scraped | unique value not correlated |
| host\_thumbnail\_url | text, unique values |
| neighbourhood | other similar column is used |
| listing\_url | many unique values |
| picture\_url | many unique values |
| host\_url | many unique values |
| host\_picture\_url | many unique values |
| latitude | neigbourhood cleansed used |
| longitude | neigbourhood cleansed used |
| calendar\_last\_scraped | not correlated |
| estimated\_revenue\_l365 | highly correlated, estimation data leakage |
| estimated\_occupancy\_l365 | highly correlated, estimation data leakage |

One of the first steps we did was data exploration, we visualized our dataset to understand the distribution and relationships between variables. For numerical features, we used histograms to observe the overall shape and skewness of the data, and boxplots to detect potential outliers. For categorical features, we employed bar charts to assess the frequency of each category, which helped us evaluate cardinality and data quality.

Additionally, we created a correlation heatmap using Seaborn to visualize correlations between numerical features. This helped us to make decisions on which features to keep or drop, especially when identifying highly correlated columns that might lead to multicollinearity. These visualizations played a crucial role in shaping our preprocessing steps such as outlier removal, feature selection, and encoding strategies.

We also explicitly casted our numerical columns to “float64” and our categorical columns to “category”, which should help us with compatibility and also improve memory efficiency in further preprocessing and machine learning.

Simplification of date columns like host\_since, first review and last\_review from its default format took place. We extracted only the YEAR component which reduced granularity, while keeping the meaningful information, such as from which year is the host active on the platform or which year last review was added.

The column amenities in the original dataset was a string column that listed multiple amenities in a single text column. This column would not suit a machine learning model and because we decided that it contains some valuable information for better predicting our target variable we have separated each amenity into a separate binary column where 1 indicated the presence of that amenity in the listing and 0 indicated the absence of amenity in the listing.

We also cleaned the target variable price by removing currency symbols and filtering out values that were missing or close to zero. To ensure compatibility with machine learning pipelines, we standardized column names by removing special characters and spaces. The preprocessed dataset was saved as an interim CSV file for future use in modeling.

As part of the preprocessing, the goal is to do all preprocessing functions locally after splitting the train set into the train set and validation set so there is no data leakage between the train set and validation set. Firstly we detected outliers in the train set by estimating borders where value is considered as outlier. Then we use the same borders from the train data for validation dataset so we could detect outliers the same way, because we would not want to estimate borders based on the whole train data before splitting them into train and val. We detected outliers and decided to remove them from the training (and validation) set, as this led to improved model performance compared to keeping them. In contrast, replacing outliers with the median resulted in worse performance. When removing outliers, we ensured that rows were consistently removed from both x\_train and y\_train, maintaining alignment between inputs and targets. Since we wanted to detect only extreme outliers we have used iqr\_multiplier as 3 instead of common 1,5 for detection of mild outliers. While this process eliminated around 22% of the dataset, the model still performed better than when outliers were retained or replaced with the median but still had better results then leaving outliers as they were.

For numerical missing values we created class NumericalImputer that is using supervised machine learning techniques to fill the missing values. It leads to more accurate data than if we would have used simple mean or median imputations. In our case after testing a few techniques we decided to use HistGradientBoostingRegressor because of its good trade off between speed and accuracy. It is especially useful because our dataset is composed of a lot of unique columns. Firstly we tried RandomForestRegressor but HistGradientBoostingRegressor performed quicker and accuracy wasn't impacted that much so it saved us valuable runtime without compromising quality.

Missing values in categorical features needed to be filled as well. A common approach is imputing values with the most frequent category. The most frequent category was obtained from the training data and then used for all categorical columns with missing data in both the training and validation sets (the most frequent category was derived from the training data), as well as for the test data during the testing phase. This approach ensures that there is no data leakage.

The next step was to encode these features, with one-hot encoding chosen due to the lack of an ordinal order. One approach to ensure the encoder learns every single category would be to encode the features before the train-test split, but this would result in encoding NaN values as well. Therefore, the encoder is applied after filling the missing values in categorical features, ensuring that no column represents missing values.

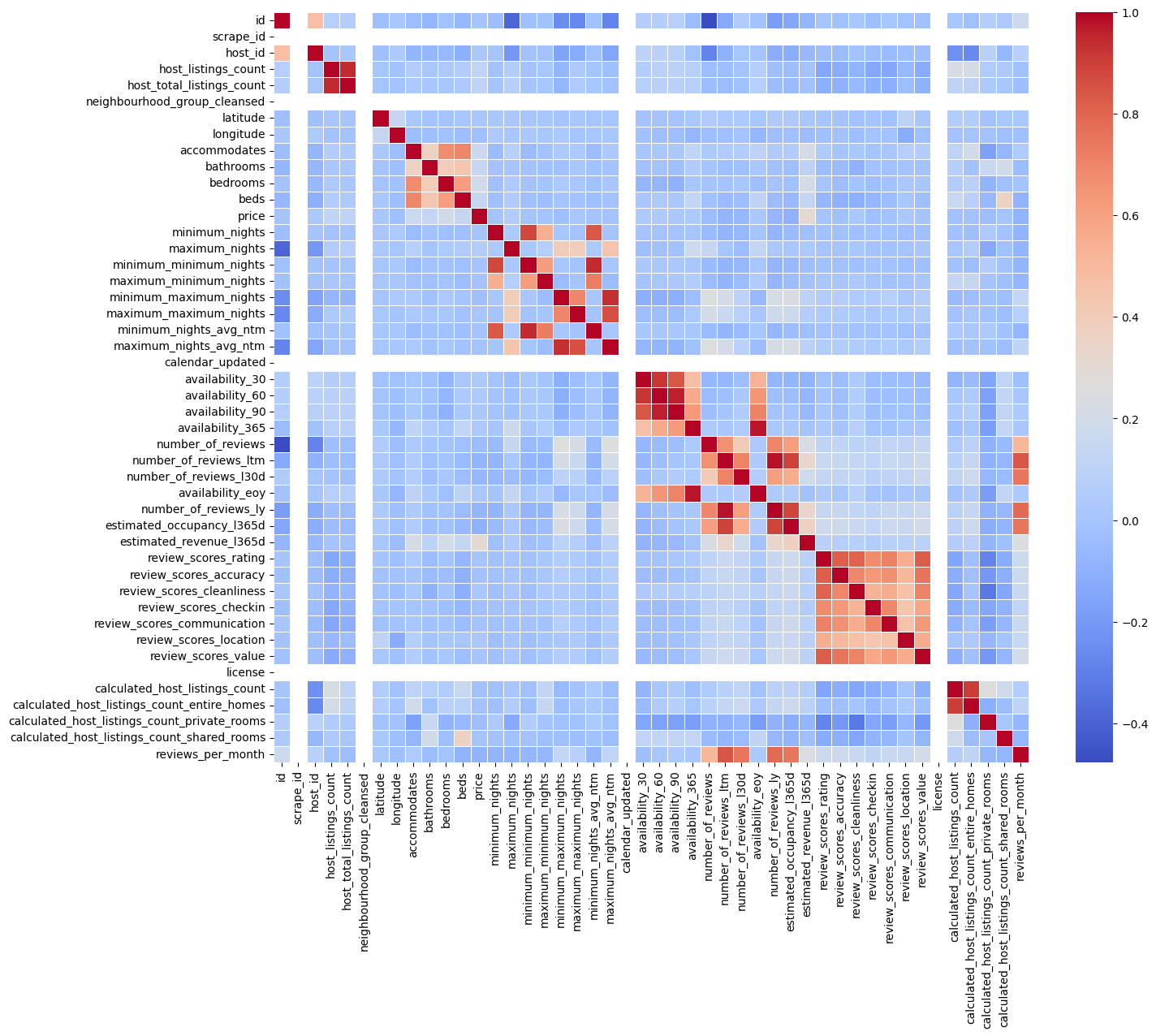
Since there could be categories that were not included in the training data, the encoder is set to ignore these new categories to prevent errors. This also helps prevent multicollinearity (problem with linear regression) as the encoder also drops the first category. After encoding, encoded features are then renamed with names of the categories so we could easily use these column names in feature importance. Finally, we applied feature scaling. Although XGBoost does not require scaled input features, scaling can be beneficial when strong L1 or L2 regularization is used, as it ensures that all features are on a similar scale—especially when using Min-Max scaling to compress feature ranges. MinMax scaler was fitted locally to train data in cross validation and then applied to the validation and test part.

# Data visualization

We looked at the correlation matrix which can give us a basic understanding of columns and if they are important or not (see Figure 1). Based on this we removed some columns with low correlation which we described in the data preparation section. We have also removed columns with high correlation as for example estimate of revenue which could lead to data leakage.

We also created an interpretation of missing data in the columns for better readability and visibility when deciding which columns are beneficial to keep in the dataset and which ones should be removed (see Figure 2).

When evaluating the model's performance, we plotted a learning curve for both training and validation data (see Figure 3). The learning curve helped us adjust the hyperparameters where the model was overfitting, as the error on the validation set did not decrease with more samples. After adjusting the hyperparameters, we observed a healthier learning curve, where the error on the training dataset remained relatively constant, while the error on the validation data decreased with more samples. Although there is still a difference between validation and train error; slight difference (slight overfit) between error on train and validation data is for real case scenario normal.

Figure 1. Correlation Matrix

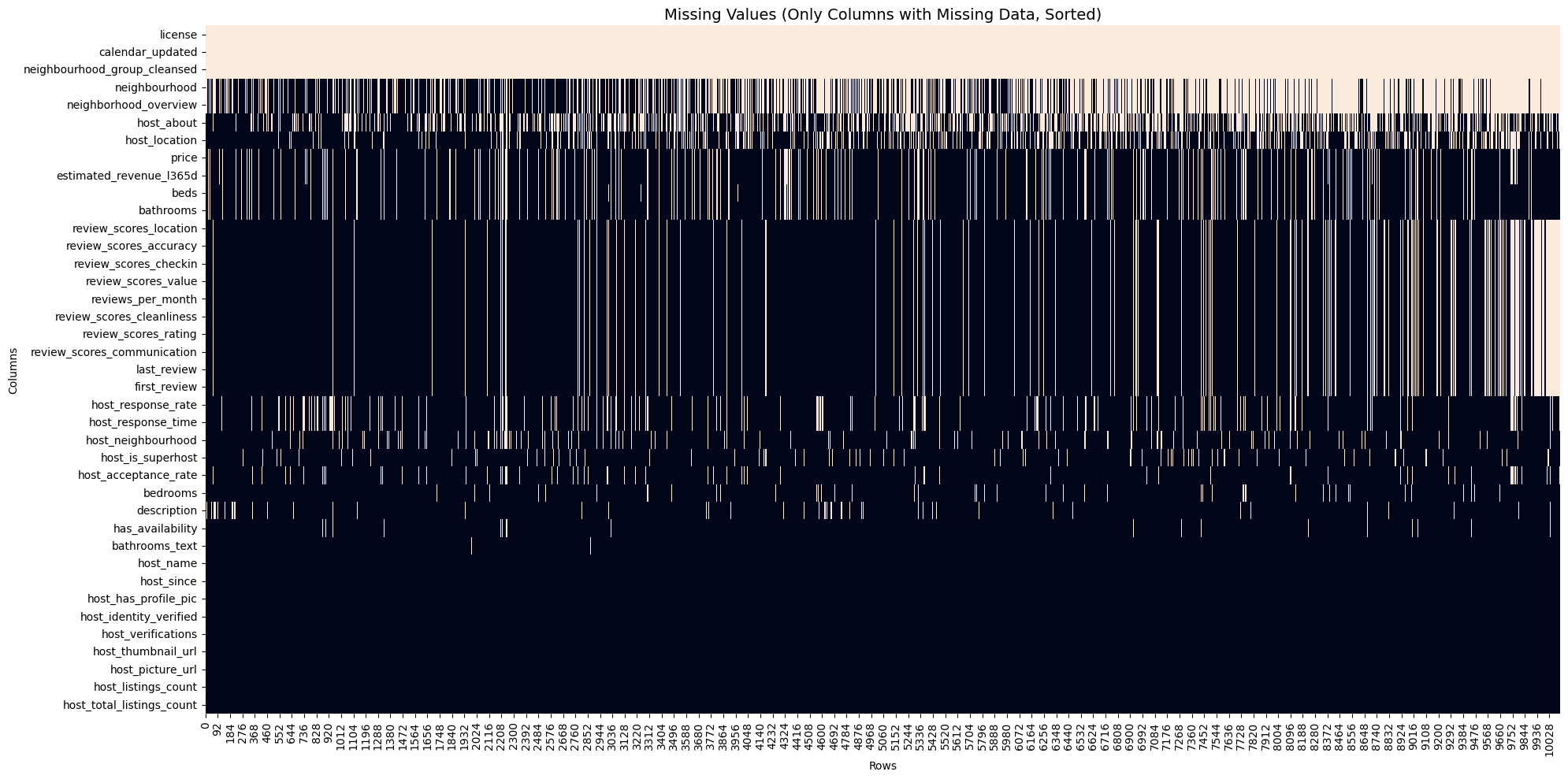


Figure 2. Missing Data by Columns

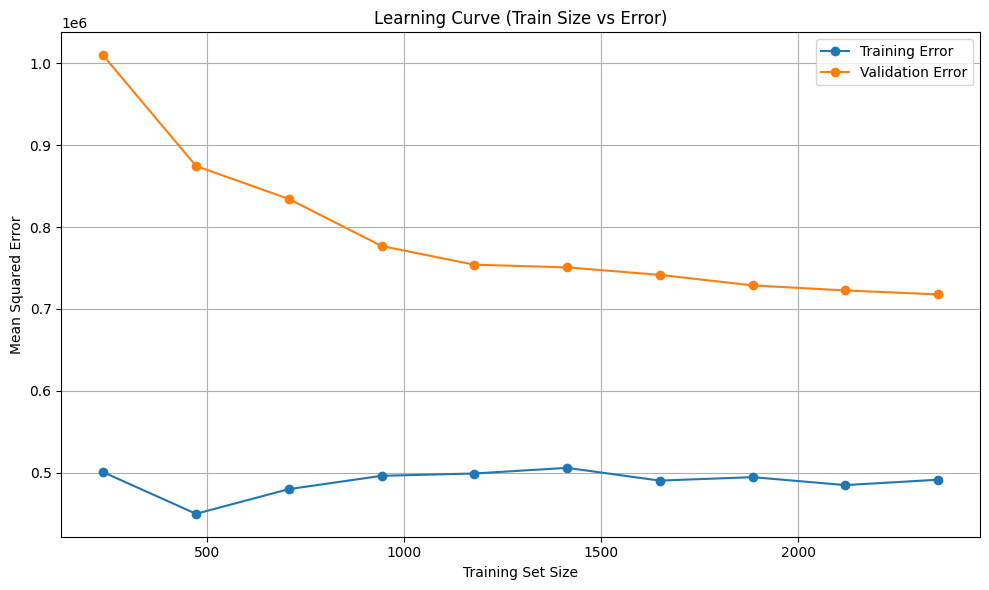


Figure 3. Learning Curve

# Modelling

In this project, several machine learning models could be tested for predicting the target variable (price). Common choices for regression tasks include Linear Regression, Decision Trees, Random Forests, and Gradient Boosting Machines (GBM). Among these, XGBoost was selected for the following reasons:

Performance: XGBoost has proven to be highly effective in a variety of regression tasks, often outperforming other models due to its ability to handle complex relationships and non-linear patterns in data. (Chen and Guestrin, 2016)

Regularization: XGBoost includes regularization techniques (L1 and L2), which help prevent overfitting, making it ideal for datasets with a large number of features. (Chen and Guestrin, 2016)

Handling Missing Data: XGBoost handles missing values internally, reducing the need for complex imputation strategies and simplifying the preprocessing steps.

Flexibility: The model is flexible and can be adjusted with multiple hyperparameters (e.g., n\_estimators, max\_depth, learning\_rate) to improve performance. (Brownlee, 2017)

## Model Architecture

The architecture of the model used in this case is based on an ensemble of decision trees (boosted trees), specifically using the XGBoost model. The architecture consists of multiple weak learners (small decision trees) that are trained iteratively, with each tree attempting to correct the errors of the previous tree. The model uses a gradient-boosting framework to minimize the residuals (errors), allowing it to build a strong predictive model through boosting. (Chen and Guestrin, 2016)

## Validation Process

To ensure the robustness and generalizability of the model, the following validation process was used:

Train-Test Split: The dataset was split into training and testing sets (e.g., 80% training and 20% testing). This allows the model to be trained on one subset of data and evaluated on another, unseen subset. (Géron, 2019)

Cross-validation: To further validate the model, k-fold cross-validation was used. This process divides the dataset into 'k' subsets, and the model is trained on 'k-1' subsets while the remaining subset is used for validation. This ensures that the model's performance is evaluated across different data splits, leading to a more reliable estimate of its performance. (Géron, 2019)

Performance Metrics: The R² Score was chosen as metric for choosing the best model on the validation dataset. R², MAPE, RMSE and Mean Absolute Error (MAE) were used to evaluate the model’s performance on test data. The R² score measures how well the model fits the data, and the MAE provides an interpretable measure of the average prediction error. MAPE is relative percentage error which says how relatively is the model precise and RMSE is root mean squared error and It measures the average magnitude of the error between the predicted values and the actual values. (James et al., 2013)

## Model Limitations and Considerations

Overfitting: While XGBoost is less prone to overfitting compared to simpler models (L1 a L2 regularization), it can still overfit if the hyperparameters are not tuned properly or if the model is too complex (e.g., too many trees or a high maximum depth). (Brownlee, 2017)

Interpretability: XGBoost, being an ensemble method, is harder to interpret compared to simpler models like linear regression. It can be challenging to directly interpret the relationships between features and the target variable. (Lundberg and Lee, 2017)

Feature Importance Bias: Some features may be overrepresented in terms of their importance, especially if they have many unique values or high cardinality. This can lead to biased feature importance rankings. (Lundberg and Lee, 2017)

## Ideas to Improve the Model

Feature Engineering: Improving feature engineering, such as creating additional interaction features or transforming existing ones, may capture more relevant patterns in the data. (Zheng and Casari, 2018)

Hyperparameter Tuning: A more exhaustive hyperparameter tuning process using Optuna with more complex hyperparameters and increased number of trials. (Akiba et al., 2019)

Ensemble Methods: Combining XGBoost with other models (e.g., Random Forests, or even stacking multiple models) could lead to improved performance by leveraging the strengths of different algorithms. (Wolpert, 1992)

Cross-validation Strategy: Exploring different cross-validation strategies, such as stratified k-fold or time-series cross-validation (if the data has temporal dependencies), could provide more robust performance estimates.(Géron, 2019)

Before production performance of the model could also increase (based on error in learning curve) by fitting model on the whole data.

## Explain How You Chose the Values for the Hyperparameters of Your Model

The hyperparameters for XGBoost were chosen based on a combination of prior knowledge and Optuna-based hyperparameter optimization. Hyperparameters were iteratively adjusted based on results from learning curve.

The following key hyperparameters were considered:

n\_estimators: The number of trees (boosting rounds) in the model. Initially set to a high value (e.g., 500), and then optimized to prevent overfitting by tuning learning\_rate and max\_depth. (Brownlee, 2017)

max\_depth: The maximum depth of the decision trees. A higher value can make the model more complex, while a lower value reduces overfitting. The value was chosen by evaluating the performance of different depths (e.g., 3-10). (Brownlee, 2017)

learning\_rate: This determines the step size at each iteration while moving toward a minimum of the loss function. A lower learning rate can improve accuracy but requires more boosting rounds. The rate 0.01 to 0.05 was selected by using cross-validation. (Chen and Guestrin, 2016)

subsample: This parameter controls the fraction of samples used to build each tree. A value of 0.4 to 0.7 was tested to balance performance and model variance.

colsample\_bytree: The fraction of features to be randomly sampled for each tree. The optimal value was found through cross-validation. (Brownlee, 2017)

Since the model was overfit based on the learning curve regarding validation and train data, there was reduction in complexity of hyperparametrs and strong regularization L1, L2 and gamma was applied.

# Model evaluation & interpretation

To interpret the performance of the model, the following methods were applied:

## Basic Numeric Metrics

For the test and validation set, the basic numeric metrics were as follows:

| **Metric** | **Training sample (last from crossval)** | **Validation sample**  **(last from crossval)** |
| --- | --- | --- |
| Mean Absolute Error (MAE) | 481.35 | 527.87 |
| Mean Absolute Percentage Error (MAPE) | 26.39% | 29.24% |
| Mean Squared Error (MSE) | 572,483.62 | 688,361.44 |
| Root Mean Squared Error (RMSE) | 756.63 | 829.68 |
| R² (coefficient of determination) | 0.6421 | 0.5534 |

For these samples, the MAE was quite low, around 500 Kc. R² also remained above 0.55 in both cases. The testing sample, however, did not display equally good results:

| **Metric** | **Testing sample** |
| --- | --- |
| Mean Absolute Error (MAE) | 1,374.31 |
| Mean Absolute Percentage Error (MAPE) | 35.81% |
| Mean Squared Error (MSE) | 102,886,029.62 |
| Root Mean Squared Error (RMSE) | 10,143.28 |
| R² (coefficient of determination) | 0.0237 |

Based on these numbers, the model’s predictions deviate from the actual values by 1 374 Kc on average, or by 35,81 %. While the MAE is not too high, the MAPE shows quite a large error.

Because RMSE is more sensitive to large individual errors, the wrong predictions drove the RMSE up to 10 143 Kc. The very low R² suggests that using the model is not much better than using the mean. This is also influenced by R²’s sensitivity to big individual mistakes. A

This was likely caused by deleting the outliers in both the training and validation sample, which meant that the model was unequipped to predict for more luxurious listings that were present in the test set.

## Actual vs Predicted Comparison

The full‑scale scatter plot (see Figure 4) shows almost all points close to the x axis, on the right side far below the 45 ° line of agreement. In other words, the model systematically under‑prices expensive listings—it never predicts above roughly 6 000 Kč even when the true nightly rate exceeds 250 000 Kč. This corresponds with the big penalisation presented in RMSE and R².

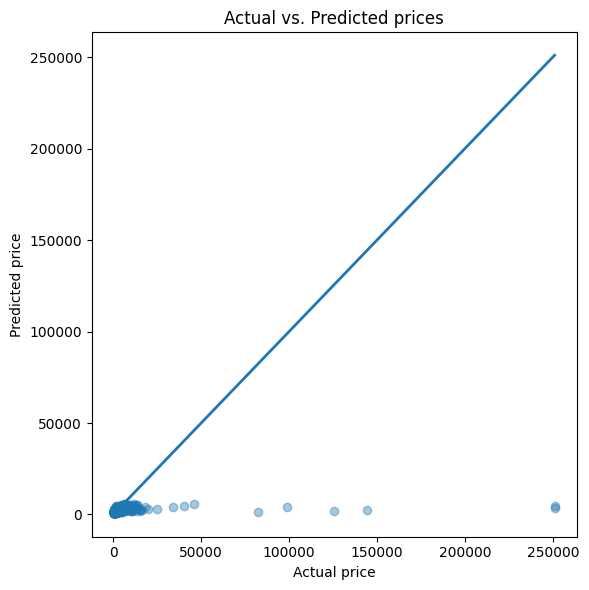
****

Figure 4. Actual vs Predicted prices

Focusing on the sub‑50 000 Kc segment (see Figure 5) reveals a denser cloud with a faint upward trend, but many points still lie beneath the diagonal. The model therefore exhibits reasonable performance for listings priced under 5000 Kc, but then fails regarding the more expensive accommodation offers, as predicted in the paragraphs above.

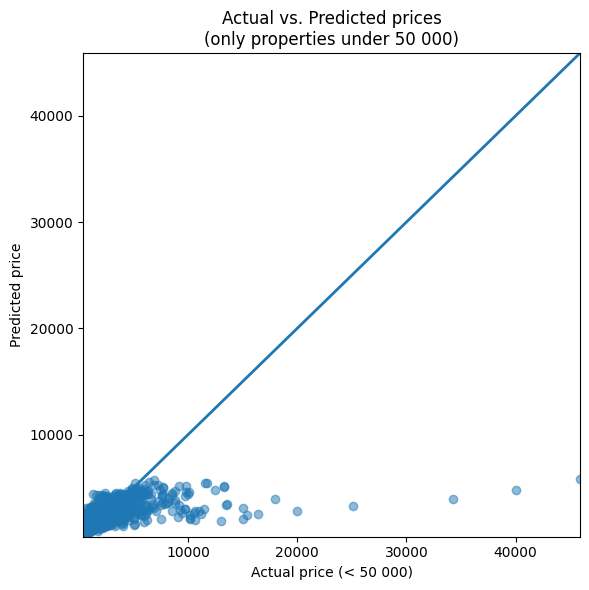
****

Figure 5. Actual vs Predicted prices <50 000 Kc

## Residual Analysis

An analysis of the residuals, obtained by plotting the differences between actual and predicted values against the predicted prices, reveals a visible pattern (see Figure 6). Specifically, residuals tend to remain relatively small and symmetrically distributed around zero for lower predicted values (up to approximately 2000 Kc). However, as predicted prices increase, the residuals stray from the x axis more often and can be found more towards the top of the plot.

This pattern again suggests that the variance of the prediction error increases with the magnitude of the predicted price. It reflects the model’s difficulty in accurately estimating prices for high-end listings.

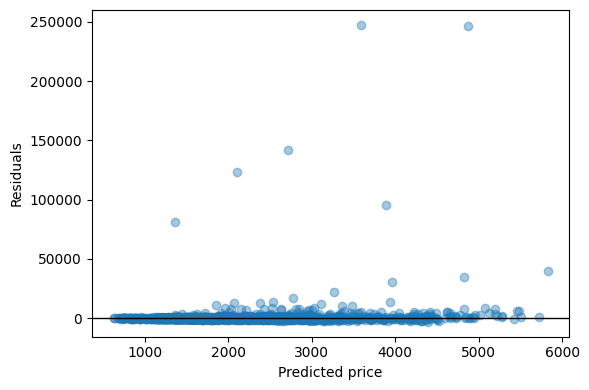


Figure 6. Residuals against predicted prices

## Feature Importance

XGBoost provides built-in methods for evaluating feature importance, explaining which variables contribute most significantly to the model’s performance. Two common metrics are used: gain, which reflects the average improvement in the loss function from splits using a particular feature, and frequency, which counts how often a feature is used in the decision trees (Lundberg, 2017). For clarity, the output was restricted to the top 15 most important features according to each metric.

According to the gain-based importance (see Figure 7), the most influential features are “bedrooms”, “beds”, “accommodates” and “bathrooms”, all of which are related to the physical capacity of the listing. Among these, “bedrooms” shows the highest contribution, likely due to its relationship with both guest capacity and space.

Interesting is the presence of “Dishwasher\_1”, which is the only representative of amenities in the top 15.

Certain location indicators, particularly “neighbourhood\_cleansed\_Praha 1” followed by Praha 2, also demonstrate one of the higher gain values. This observation may be linked to these areas being relatively more expensive and centrally located.

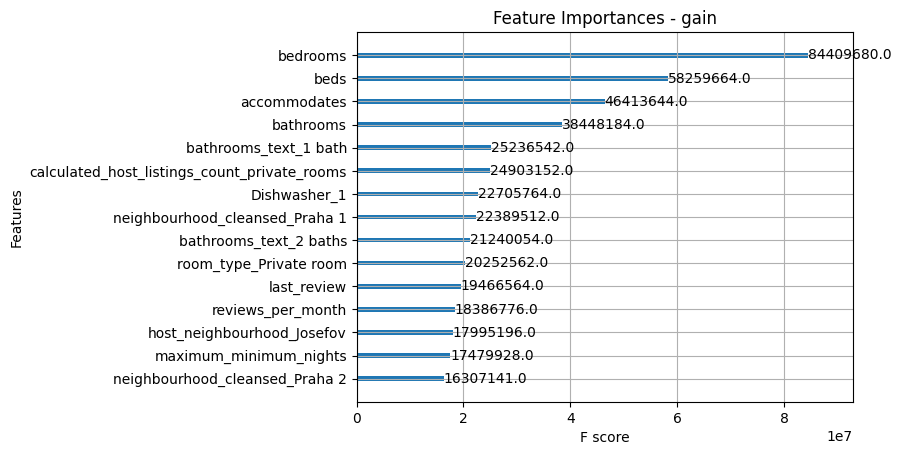


Figure 7. Feature importances by gain

The frequency-based importance highlights features that are used more often across the ensemble of trees. The feature “accommodates” appears near the top in both gain and frequency rankings (see Figure 8), suggesting that it is both commonly selected and effective. Conversely, while “beds” is associated with high gain, it is not used as often, implying that it is selected in fewer but more impactful splits.

Variables such as “reviews\_per\_month” and “review\_scores\_location” appear near the top of this ranking, indicating their frequent usage, despite each split contributing a bit less to loss reduction.

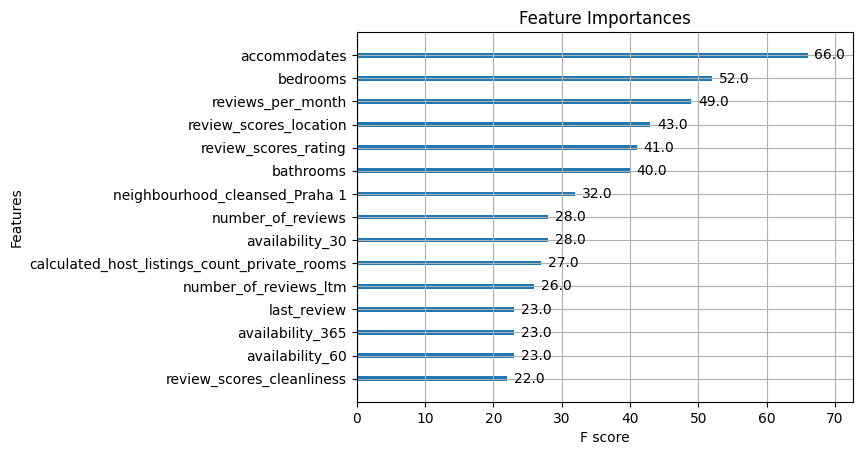


Figure 8. Feature importances by frequency

## Permutation importance

To complement the gain- and frequency-based feature importance provided by XGBoost, permutation importance was calculated (see Figure 9). This method assesses the impact of each feature on model performance by randomly shuffling its values and observing the decrease in the model’s R² score. A greater drop in R² indicates a more critical feature (Orlenko, 2019).

Unsurprisingly, the most impactful variable is “bedrooms”, followed closely by “bathrooms” and “accommodates”. These three, all indicative of a listing’s capacity and physical characteristics, consistently appear as top predictors across all feature importance analyses. Their dominant role reflects the strong link between size and pricing in Airbnb listings.

The next set of variables “reviews\_per\_month”, “neighbourhood\_cleansed\_Praha 1”, and “last\_review” all appeared among the top 15 feature importances by frequency, which suggests that reviews and locational factors also play a meaningful, albeit secondary, role. The appearance of “neighbourhood\_cleansed\_Praha 1” additionally supports earlier observations about location premium.

Last but not least, “Dishwasher\_1” made an appearance again, even though this time it is outperformed by “Coffeemaker\_1”.

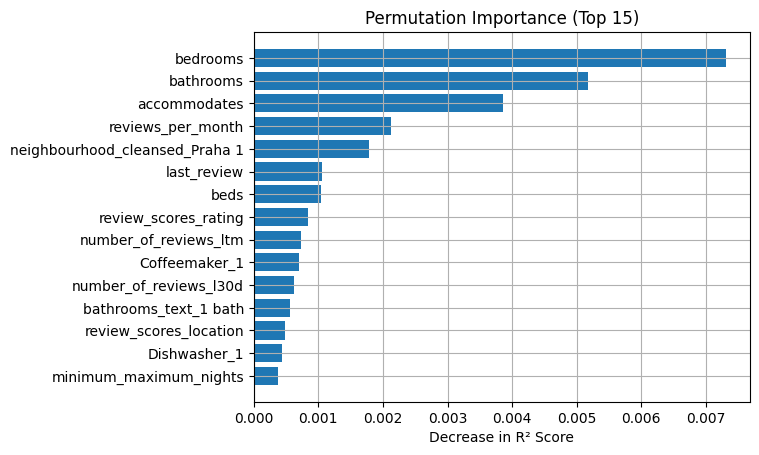


Figure 9. Permutation Importance

In summary, permutation importance reinforces the previous findings that the most important features seem to be those that determine the physical capacity of the listing, followed by location and reviews, and finally accompanied by a few amenities.

# Literature

AKIBA, Takuya et al. *Optuna: A Next-generation Hyperparameter Optimization Framework.* In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* New York: ACM, 2019, pp. 2623–2631.

BROWNLEE, Jason. *XGBoost With Python: Gradient Boosted Trees with XGBoost and Scikit-Learn.* Machine Learning Mastery, 2017.

CHEN, Tianqi and CARLOS, Guestrin. *XGBoost: A scalable tree boosting system.* In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York: ACM, 2016, pp. 785–794. ISBN 978-1-4503-4232-2.

GERON, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* 2nd ed. Sebastopol: O'Reilly Media, 2019. ISBN 978-1-492-03264-9.

JAMES, Gareth et al. *An Introduction to Statistical Learning: with Applications in R.* New York: Springer, 2013. ISBN 978-1-4614-7137-0.

LUNDBERG, Scott M. and LEE, Su-In. Consistent feature attribution for tree ensembles. *arXiv preprint*, 2017. Available from: https://doi.org/10.48550/ARXIV.1706.06060.

LUNDBERG, Scott M. and LEE, Su-In. *A unified approach to interpreting model predictions.* In: *Advances in Neural Information Processing Systems 30 (NIPS 2017).* 2017.

LUNDBERG, Scott M. et al. *From local explanations to global understanding with explainable AI for trees.* Nature Machine Intelligence, 2020, 2(1), pp. 56-67.

ORLENKO, Alexander and MOORE, Jason H. A comparison of methods for interpreting random forest models of genetic association in the presence of non-additive interactions. *BioData Mining*, 2021, 14(1). Available from: https://doi.org/10.1186/s13040-021-00243-0.

WOLPERT, David H. *Stacked generalization.* Neural Networks, 1992, 5(2), pp. 241-259. DOI: 10.1016/S0893-6080(05)80023-1.

ZHENG, Alice and CASARI, Amanda. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists.* Sebastopol: O'Reilly Media, 2018. ISBN 978-1-491-97322-3.