Assignment 2 - Airbnb prediction models

Course: ECBS6067 - Prediction with Machine Learning for Economists

Anne Valder

2023-12-02

The report below summarizes pricing strategies for small and mid-size apartments in Copenhagen. Using a data-driven approach, I determine that the average price for apartments hosting 2 to 6 guests is 1166 DKK (approx. 156 Euro) per night, as predicted by the best-performing model. The driving factor influencing apartment prices on average the most is the number of people an Airbnb can accommodate. The first part of the report delves into data preparation and pre-processing. Subsequently, the three distinct price prediction models, along with all modelling decisions, are explained. This is followed by a comparison of the results with predicted apartment prices in London. In addition, and to further assess the predictive power of the models out of sample, predictions are made not only on a holdout set but also on an additional data set covering a different reporting date, essentially mimicking live data. Finally, the report provides a detailed evaluation of the model performance of the best-performing machine learning model, utilizing variance-important measures such as Shapely values and partial dependence plots.

Data Preparation: The data for apartments in Copenhagen is taken from Inside Airbnb. The two selected data sets include the report date 24 September 2023 and the report date 29 December 2022. After joining the data sets, I proceed with the **sample design** as follows. First, I drop variables that are of no use to the prediction exercise, e.g. variables containing URLs, or specifics about the host etc. Next, I make sure that all variables are cleaned (i.e. remove characters or symbols) and of the correct type and class for the prediction exercise (e.g. conversion to binary, numeric, and factors). Moreover, I encode the 'price' variable to have the right format (no commas for thousands). Furthermore, I transform the variable 'amenity', which originally contains a listing of all kinds of amenities a certain Airbnb has, into dummy variables. In order to prevent to obtain over 2000 dummy variables, I group the most important amenities together and create approximately 15 dummies. To ensure that the sample relates to the policy question at hand, I filter for apartments that can house only 2 to 6 people and that correspond to "standard" property types for representative Airbnb prices. To circumvent that the analysis is distorted by errors, I drop extreme values by making sure that the minimum number of nights is at least equal to one. At last, I create the variable 'days since first review' which allows me to analyse approximately for how long an apartment has been used as an Airbnb, i.e. a proxy for age.
In terms of **label engineering** I assure that minimum and maximum values are reasonable and drop extreme values above 15,000 DKK (i.e. 2000 Euro per night) which are likely to be errors after carefully considering the characteristics of those high-priced Airbnbs. Moreover, I check whether a log-transformation is feasible, since it might be interesting to analyse relative changes rather than changes per monetary unit. However, the figure 3 in the appendix indicate that transforming the target variable to logs does not alter the shape of the distribution significantly. Therefore, I continue to use the target variable in levels.
Next, I turn to the **feature engineering:** choices. First, I inspect *missing values.* Here I drop variables if they contain too many missing values and are not of importance for further analysis (e.g.'calendar updated' and 'licence'). Other variables, like 'bathrooms' or 'bedrooms', I impute with the help of descriptive variables like 'bathroom text' or approximate with the variable 'number of beds' and 'accommodates'. Some variables that contain only a few missing variables get the missing values replaced with the median of the non-missing values (e.g.n_days_since, review). In the following, I consider the *functional form* of some predictors. However, these feature modifications are only relevant for the LASSO model. The other two (machine-learning) models

are non-parametric algorithms, which should be able to find interactions between variables and non-linear behaviour. After visual inspection (see for example figure 4 in the appendix) and some basic regression analysis, I add squared, cubic and log terms for the following variables: 'accommodates', 'beds', 'number_of_reviews', 'n_days_since', and 'review_scores_rating'. Last, I consider the *interactions of predictors*. Again, this is only relevant for the parametric OLS (LASSO) model. I assume interactions with 'f_property_type', 'f_room_type' and 'accommodates' and visualize those (see appendix figure 5). The interaction follows the belief, that the impact of room type on the response variable (e.g., price) varies depending on the specific property type, an interaction term can capture these differential effects. For example, the effect of upgrading to a higher room type might have different implications for prices in different property types (e.g., apartments vs. houses). Moreover, I include interactions between each 'f_property_type' or 'f_room_type' and all the amenity dummies, similar as to the London example. The final data set (including data for review date September 23 only) thus consists of **99 variables with 11880 observations**. Regarding the target variable, the mean price per night of an Airbnb in Copenhagen is 1184 DKK, the maximum is 15000 DKK and the minimum price is 75DKK. Further summary statistics of the final data set can be found in the markdown file in the chunk: Summary statistics.

Model selection: I chose to conduct the analysis using first an **LASSO model**, second a **Random forest model** and last a **Boosting algorithm**. In the following I will go through each model, argue for its choice and modelling decisions and then compare their performance. To begin the prediction exercise, I start by separating my data set several times. First, I filter so that it only contains the data from the first report date (29 December 2022). Second, I separate the data into a holdout set for evaluation and a working data set to estimate the three different models. The working data set gets further divided into a training and a test set for cross validation when tuning the right model parameters. This gets done automatically via the 'train' function in the caret package, when used with cross-validation (method = "cv").
The first model is the **LASSO model**. I chose it in order to not fully depend on domain knowledge and to not be too concerned with variable selection, but instead let the model itself choose the most important predictors automatically. Moreover, this way potential overfitting is avoided. I set a tuning grid to do cross validation and choose the value for lambda (between 0.5 and 1), the parameter that sets the strength of the variable selection. The larger the lambda, the more aggressive is the selection and thus fewer variables are left in the regression. The best performing LASSO model has a lambda of 0.75. From all possible predictors (including interactions, polynomials etc.) the model shrinks the weakest coefficients to zero to reduce variance and ends up picking 61 out of 74 predictors and (see markdown file chunk LASSO) and has a RMSE of $725,64$.
The second model is the **Random forest model**. It seems promising since there are many possible predictors, which gives the Random forest many opportunities for randomly selecting m variables to split on in the first step (i.e. decorrelation). This will lead to more differently looking bagged trees. Aggregating those leads then to more reduction in variance and thus a more robust outcome, increased stability and less overfitting. Moreover, using this model there is no need to make decisions regarding functional form, interaction, variable selection. In addition, the Random forest needs relatively little tuning. I again set the tuning parameters (number of trees, number of variables checked for a spit, and depth of trees (size)) by trying out several combinations via five-fold cross validation. In the cross-validation process I try out 3 different values (5, 7, 9) for the parameter 'mtry' (number of variables randomly sampled as candidates at each split). Usually, the parameter roughly equals the square root of the number of variables. In this case, the number of predictors is 26, so I set the starting value to be equal to its square root, which is approximately 5.09. The next parameter, '.min.node.size' specifies the minimum size of terminal nodes (leaves) in the trees. Here I also chose 3 different values to try out, since a smaller value allows the model to create more complex trees, potentially capturing more details in the data, but at the same time it may also lead to overfitting. A larger value results in simpler trees, which may generalize better to new data but might miss some patterns in the training data. The 'ntree' parameter, number of trees to grow in the forest, is left at its default value of 500. More trees can increase model performance, but it also requires more computational resources. Last, I set the splitting rule for building the trees to be variance reduction, since this splitting rule is less sensitive towards outliers and because I have a regression with a continuous target variable. The best performing model (according to the RMSE) uses: mtry = 9, splitrule = variance and min.node.size = 5 and has a RMSE of 674.63.
The last model I consider to predict Airbnb pricing in Copenhagen is a **Boosting algorithm**, or more

specific gradient boosting using the gbm library. Similarly to the Random forest, it is an ensemble method based on trees. However, it differs by building sequential trees that depend on each other and in the end combines results from many weak and imperfect trees to make a final strong prediction. Ensemble approaches often lead to improved predictive performance compared to individual models, since they capture diverse patterns and reduce overfitting. In the case of predicting Airbnb prices, there may be intricate interactions and dependencies between various features (e.g., neighbourhood, amenities, property type) that boosting models can model effectively. In addition, Boosting models are less sensitive to outliers (i.e.here unusual or extreme listings) compared to other algorithms. Similar to before, I begin by fine-tuning the model to the specific characteristics of the Airbnb data set using cross-validation. Compared to Random forest, there is a greater emphasis on tuning with Boosting. The first parameter 'interaction.depth' guides the complexity of the tree. It represents the maximum depth of an individual tree. In this case, I try three different values (1, 5, and 10) since there is a trade-off between model complexity and interpretability. Lower values (e.g. 1) are chosen when the relationships in the data are suspected to be relatively simple. This leads to shallow trees, which can be computationally efficient and less prone to overfitting. Slightly larger values (e.g. 5) are appropriate for data sets with moderately complex relationships and balance between model complexity and computational efficiency. Higher values indicate (e.g. 10) that relationships in the data are highly intricate and nonlinear but are prone to over fit. The next parameter, 'n.trees' specifies the number of boosting iterations or trees, here I cross validate over a sequence from 200 to 700 with a step size of 50. The parameter depends on the trade-off between model performance and training time. The third parameter of interest is 'shrinkage' or the learning rate, this parameter controls the contribution of each tree to the final prediction. A lower value results in a more robust model, but requires more trees. It is set to 0.1, which is a common starting value. Last, the parameter 'n.minobsinnode', the minimum number of training set samples required in a node to initiate a split during the construction of a tree, is set to 20. As it helps control the size of the tree nodes, there is again a trade-off between model complexity and interpretability. Small values lead to complex trees and potential overfitting vice versa for large values. A value of 20 is commonly used in practice. The RMSE is used to select the optimal model using the smallest value. The final values used for the best model are n.trees = 200, interaction.depth = 10, shrinkage = 0.1 and n.minobsinnode = 20. It has a RMSE of 702.20.
After being split into train and test data set during cross-validation, the best model of LASSO, Random forest and Boosting gets reevaluated on the working data set. The results are shown in table 1. We can see in column 1 that out of the three models according to the RMSE (for the work data set) **the best performing one is model 2 — the Random forest**. It has a RMSE of: 674.63 which is substantially lower than those of the other two models. Also in terms of other evaluation methods like the mean absolute error or the R-squared model 2 performs best (see appendix table 2. The second best is the Boosting algorithm and the last the LASSO. Moreover, from a practical perspective Random forest is the most convenient model as it is fully automatic, needs little tuning and works very fast compared to more complex Machine Learning algorithms like Boosting. Column 2 of table 1 shows the results for reevaluating the three models on the hold-out data set. In terms of the performance ranking, nothing changes and the Random forest model remains the winner. It is unexpected though that the RMSEs in the hold out set are lower for all three models, usually the opposite is the case since the model has to generalize to new, unseen data. The lower holdout RMSE might be due to the randomness involved in splitting the data into work and holdout set, which can lead to variability in performance metrics. Other explanations are model simplicity or outliers. Compared to the results from the London case study (slide 51) where the RMSE for the different models are all values between 45 and 50, the RMSEs are substantially larger. Also, the differences between the RMSE of the three models are quite big, compared to the results from the London case study. However, the absolute value of the RMSE does not have any meaning and the two data sets for London and Copenhagen differ in many regards. Furthermore, I believe the large size of the RMSE comes from the fact that in this data set the price is given in DKK instead of Euro, which is 7,46 times as much. It would be a good robustness check to transform the prices into EUR and recalculate the models.

Model evaluation with second report date: Next, I evaluate the three models again, but this time on the 'new' data set which has September 2023 as a reference date. I do this to further investigate the out of sample prediction performance of the three models. Using the September data set is a way to mimic live data which we cannot access in another way. The results in column 3 of table 1 show that again in terms of the performance ranking nothing changes and the Random forest model remains the model with the lowest RMSE of 726.10. The LASSO model remains the worst performing model with a RMSE of 773.99. As expected from theory, the RMSEs of the 'unseen' data set are larger than the RMSE of the holdout data set and the working data set. Which leads to the conclusion that the predictive power of the models decreases when using 'live' data. The problem of the lower holdout RMSE seems not to prevail here if a different 'unseen' data set is used.

Table 1: RMSE Task 1 & 2

	Work set RMSE	Holdout set RMSE	September RMSE
LASSO	725.64	705.01	773.99
Random forest	674.63	669.31	726.10
GBM	702.20	695.72	768.17

Model evaluation and visualization Random forest: Since Machine Learning methods are often seen as "black-box" models, and we often do not really know how an actual prediction is created, I want to further investigate why the Random forest model was the best performing model in all scenarios discussed above. One way to do this is by looking at the features' contribution to predicted values on average, i.e. variable importance, using variance importance plots and Shapley values. Understanding what factors influence Airbnb prices the most can help users and hosts to make informed decisions. First, looking at figure 1 which displays the ten most important features (in percentage terms) of the Random forest, we can see that the variables 'accommodates', 'beds'& 'n_days_since' contribute the most. Together, these three variables explain about 30% of the prediction performance. The variables 'review_score_rating' and 'neigborhood_cleansedIndre By' also appear to be influential. For all other variables, the importance lays below 6% and diminishes rather quickly. The results are similar to the London case study, where also 'accommodates' was the most important variable. Grouping the factor variables together and recalculating the variable importance indicates no change in the importance ordering (see appendix figure 6). In addition, figure 7 in the appendix displays the importance of all features in the model above a cut-off (for readability). Here again we see the typical shape that three to four variables are quite important in explaining the prediction outcome and the others contribute relatively little. Turning to the Shapley values, a local model agnostic approach, which calculates marginal importance of the variables. In detail, the Shapley plot (figure 2) puts the baseline value (= sample mean of price_n: 1494.67) on the horizontal axis and shows how each feature adds (or subtracts) for a given observation. It illustrates that 'neigborhood_cleansedIndre By' has the highest relative importance by adding a value of +277.42 to the sample mean. This outcome seems reasonable since the neighbourhood, Indre By in Copenhagen, is very central, close to the harbour and close to most of the touristic hot-spots. Further, the graph highlights that the features 'accommodates', 'beds'& 'n_days_since are again important. However, now we can see in much more detail how each of their contributions distributes to the prediction. For example, if an Airbnb can accommodate 3 people, the sample mean of price_n drops by -70.64. If the Airbnb has two beds, the first review is older than 10 year or the review score rating is equal to 5 there is a positive effect on the sample mean of the Airbnb price (see also figure 8 in the appendix for the Shapley variable importance). An even deeper analysis between each value of a predictor and the predicted values gives us the Partial dependence plot, see for example figure 9 in the appendix for the most important feature 'accommodates'). Even more insights could be obtained from an analysis across subsamples. This is however neglected here due to space & time constraints.

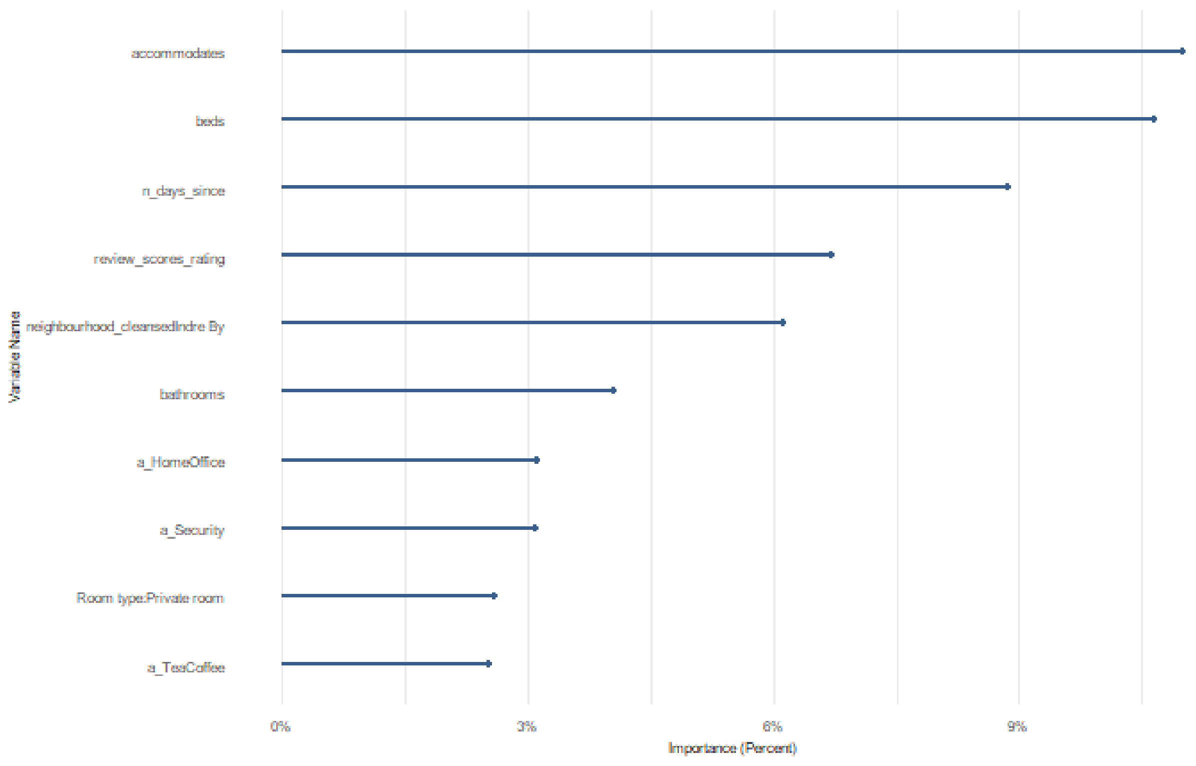

Figure 1: Variable importance plot of Top 10 features

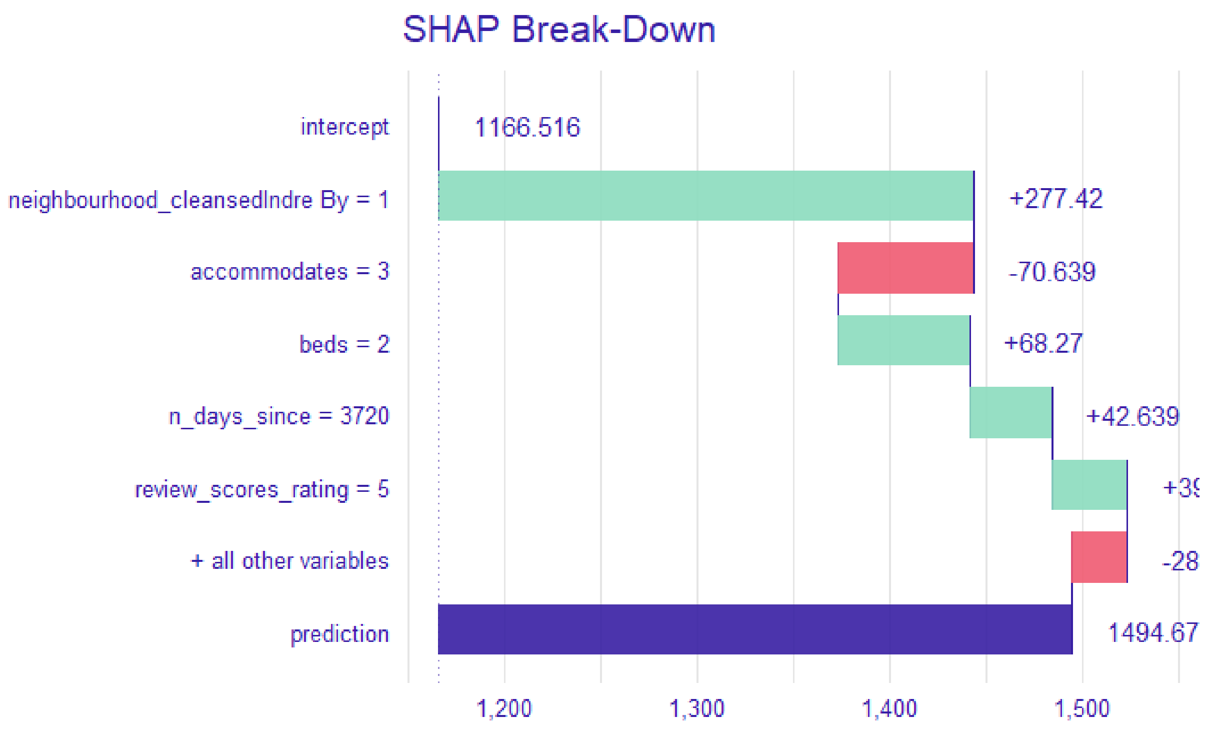

Figure 2: Shapley plot, features by contribution

Appendix

Table 2: Summary evaluation metrics

	LASSO	Random forest	GBM
MAE.Min.	354.82	331.29	356.98
MAE.1st.Qu.	363.58	351.31	371.67
MAE.Median	377.40	354.51	374.93
MAE.Mean	371.44	357.11	372.03
MAE.3rd.Qu.	380.32	369.37	377.78
MAE.Max.	381.07	379.08	378.78
RMSE.Min.	609.56	538.38	627.05
RMSE.1st.Qu.	703.95	600.36	691.17
RMSE.Median	738.75	604.25	708.04
RMSE.Mean	725.64	674.63	702.20
RMSE.3rd.Qu.	742.04	779.61	719.08
RMSE.Max.	833.90	850.54	765.68
Rsquared.Min.	0.17	0.26	0.22
Rsquared.1st.Qu.	0.21	0.26	0.24
Rsquared.Median	0.21	0.28	0.29
Rsquared.Mean	0.22	0.32	0.28
Rsquared.3rd.Qu.	0.23	0.40	0.30
Rsquared.Max.	0.28	0.42	0.34

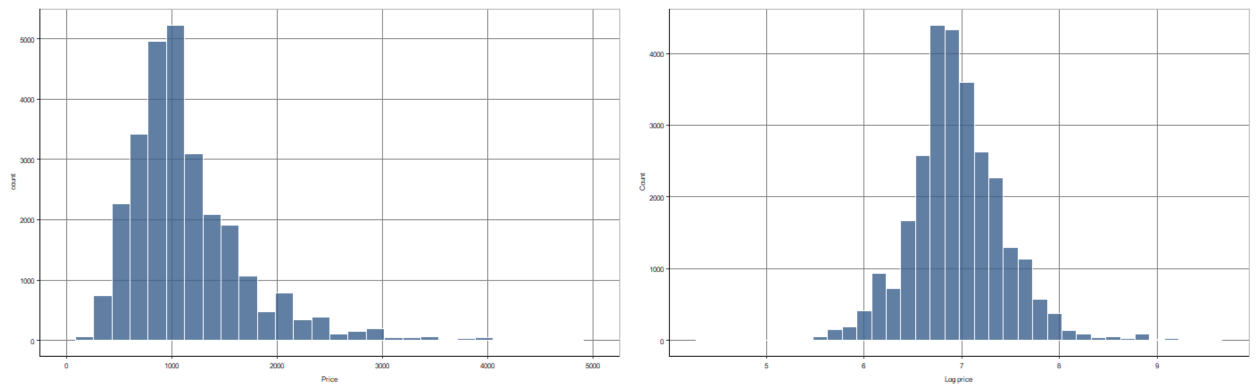

Figure 3: Label engineering: Log transformation

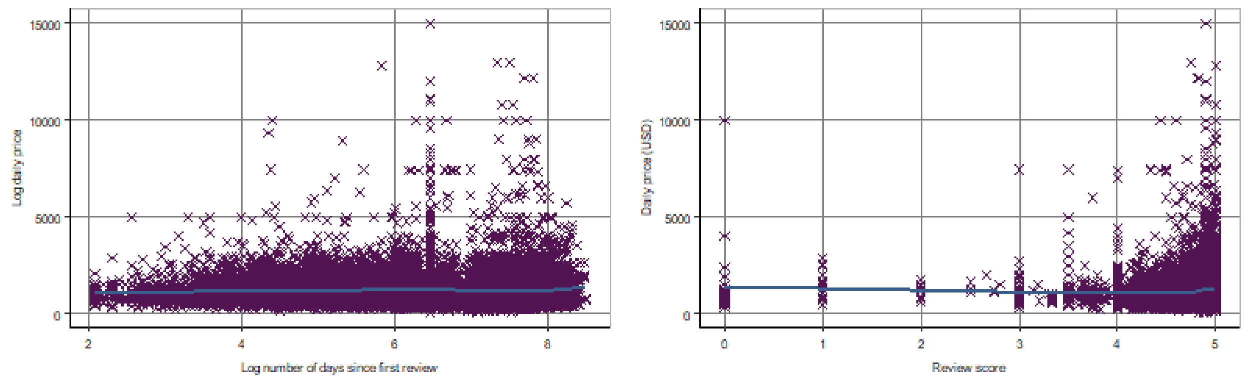

Figure 4: Feature engineering

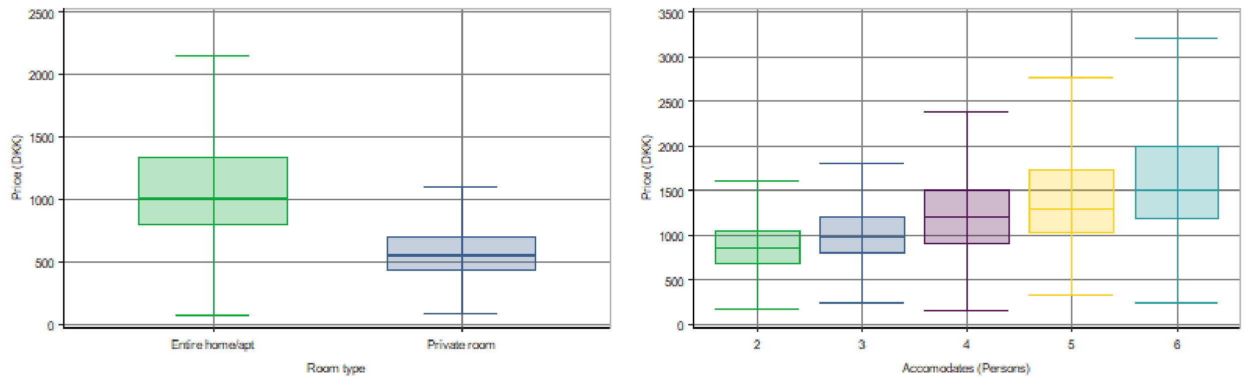

Figure 5: Interactions

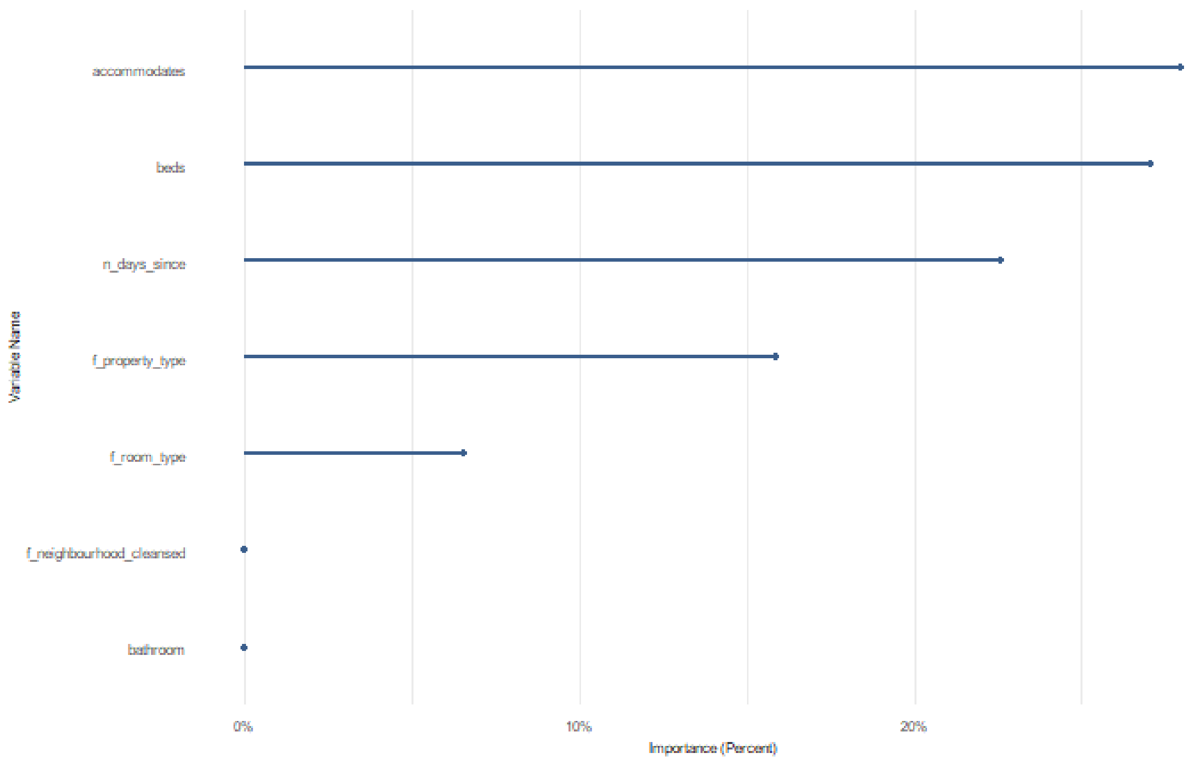

Figure 6: Variable importance plot of Top 10 features (grouped)

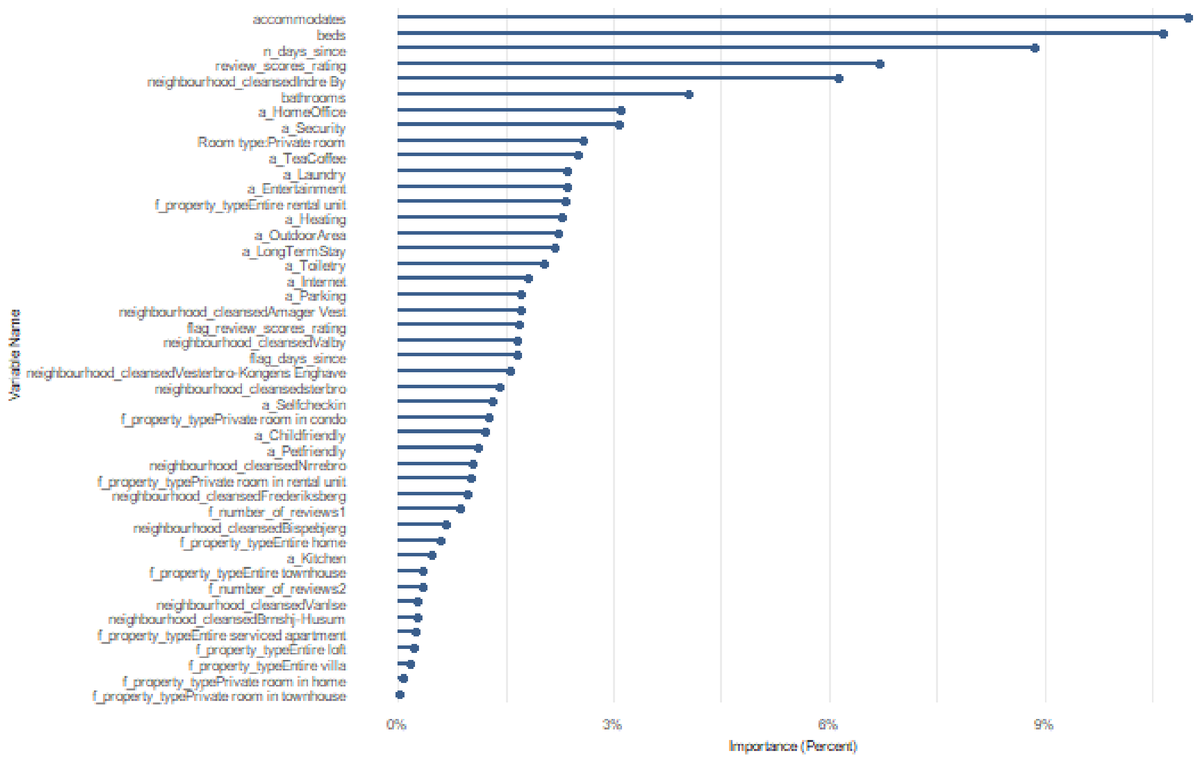

Figure 7: Variable importance plot of all features (above cutoff)

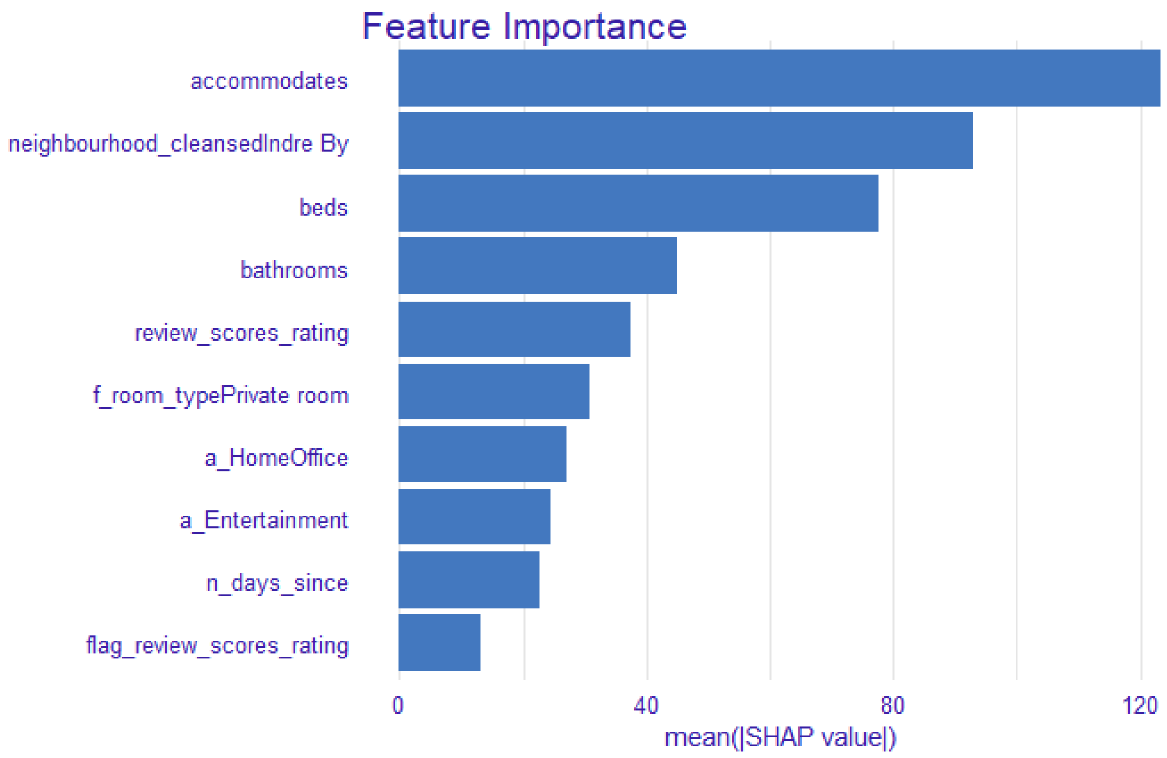

Figure 8: Shapley variable importance plot

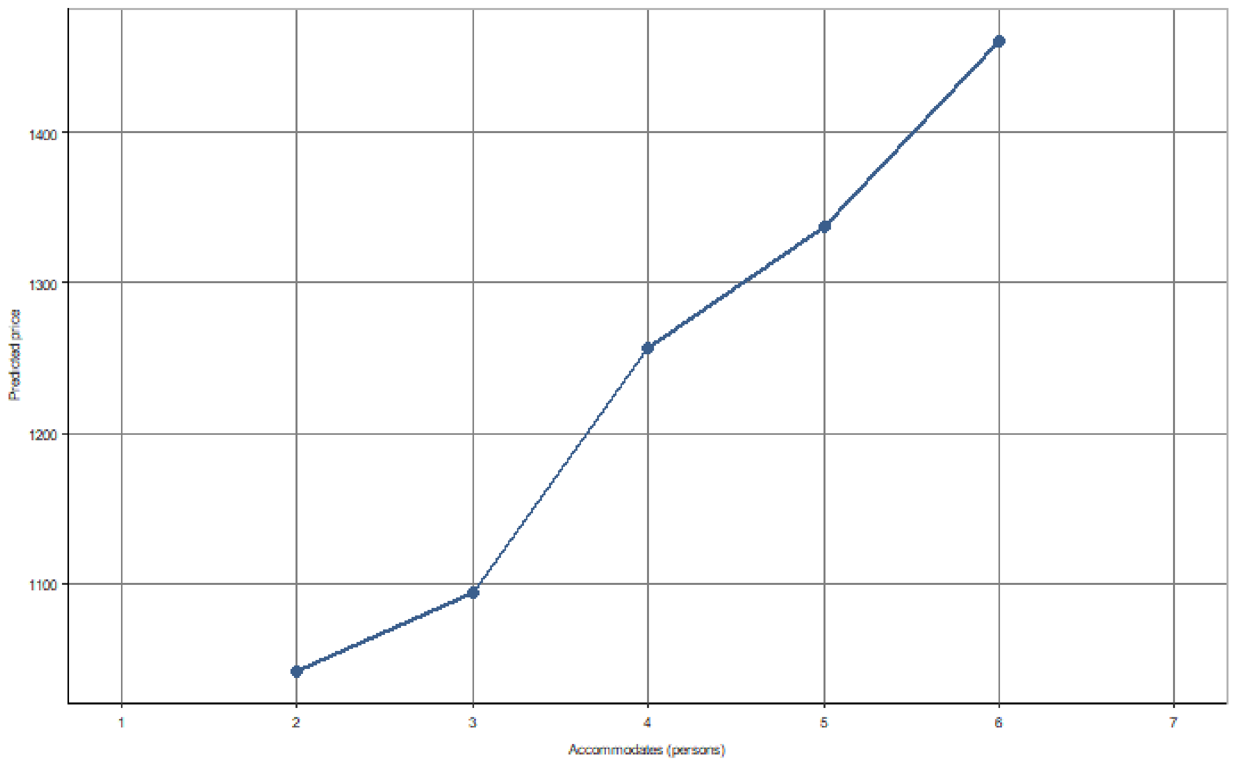

Figure 9: Partial dependence plot for: 'accomodates'