

Predicting Paris Airbnb Prices with Machine Learning: What I Learned About Outliers and Overfitting

A practical regression project using real Airbnb data, from data cleaning and outlier analysis to model tuning and evaluation.

Published on Medium by Feyza Nur Demirbas

Airbnb prices can vary a lot, even between listings that seem quite similar. Location obviously matters, but it is not the only factor. The number of bedrooms, guest capacity, room type, minimum stay requirements, availability, and many other details can also affect the nightly price.

For this project, I wanted to explore a simple question: **Can we predict the nightly price of an Airbnb listing in Paris using the information available in the listing data?**

The Dataset

I used the Paris listings dataset from Inside Airbnb. The original dataset contained 77,679 listings and 90 columns, so the first step was deciding which variables were actually useful for the analysis.

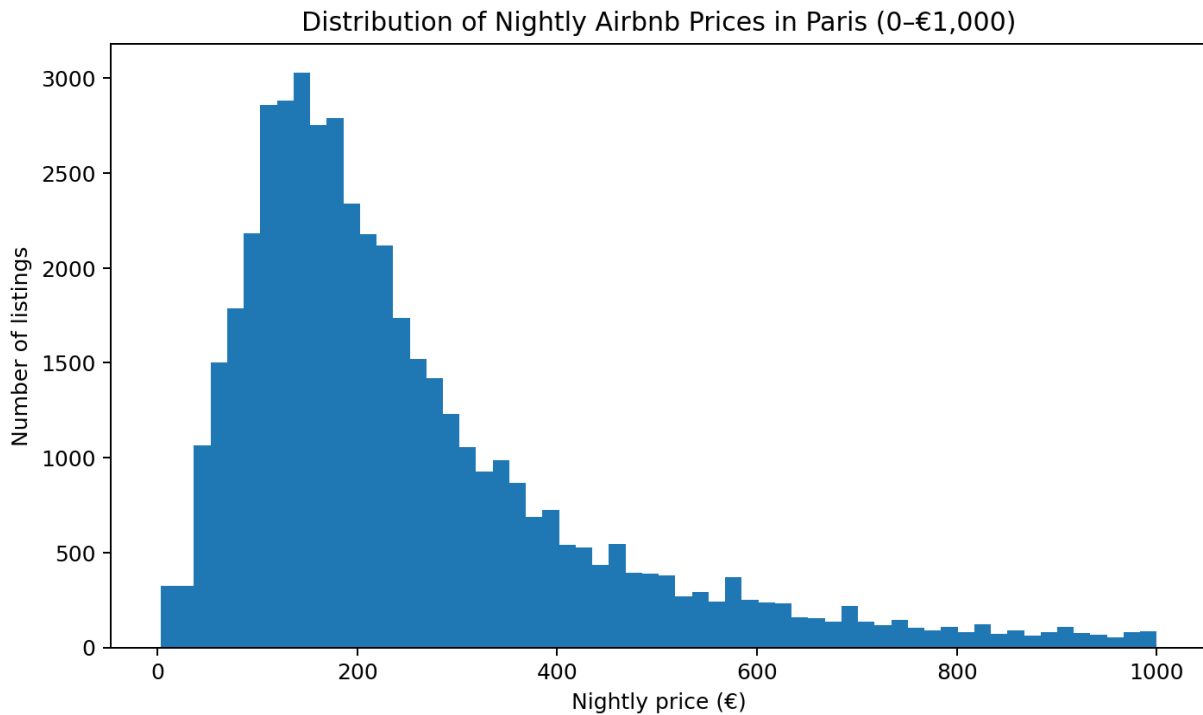
I focused on features such as location, room type, guest capacity, number of bedrooms and bathrooms, reviews, minimum stay requirements, and yearly availability. The target variable was the nightly price in euros.

After converting the price column into a numerical format and removing listings without a usable price, 48,402 listings remained for the analysis.

First Look at Airbnb Prices

One of the first things I noticed was how skewed the price distribution was. The median nightly price was around €205.50, while the mean was approximately €321.23. This difference suggested that a relatively small number of expensive listings were pulling the average upward.

This was also my first indication that outliers would need more attention.



Distribution of nightly Airbnb prices in Paris

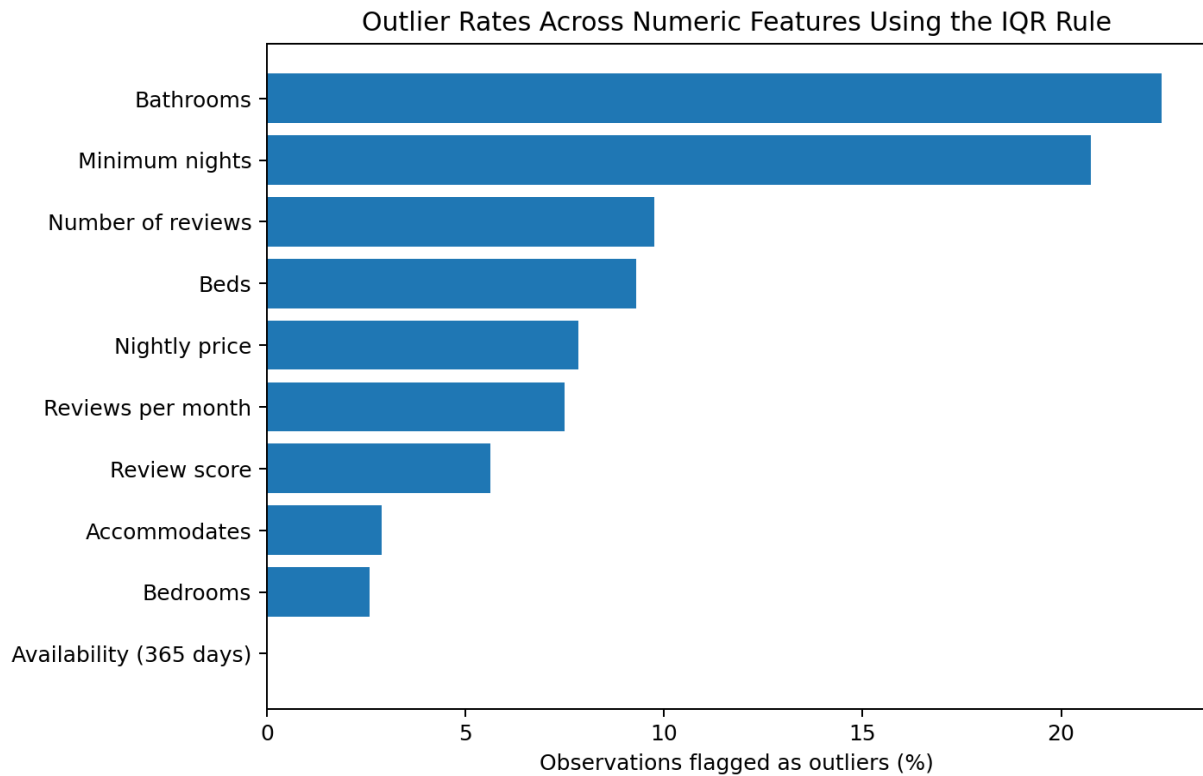
Outliers Were More Complicated Than I Expected

My first approach was to use the standard IQR method to identify potential outliers. It worked well as a starting point, but I quickly realized that applying the rule blindly could also remove perfectly valid listings.

For example, the `bathrooms` variable had both `Q1` and `Q3` equal to 1, which made the IQR equal to zero. As a result, many listings with more than one bathroom were classified as outliers, even though they were not necessarily incorrect observations.

This was an important reminder that an outlier is not automatically a data error. A large apartment with several bedrooms or bathrooms may be unusual statistically, but it can still be a completely valid Airbnb listing.

Because of this, I used IQR mainly as a diagnostic tool rather than deleting every observation it flagged. For the explanatory variables, I applied a more conservative approach and limited only the most extreme values based on thresholds learned from the training data.



Outlier rates across numeric Airbnb features using the IQR rule

What Seems to Affect Airbnb Prices?

Before building the models, I looked at the relationships between price and the numerical features. Since the price distribution was not normally distributed and contained extreme values, I used Spearman correlation to examine these relationships.

The strongest positive relationships with price were found for guest capacity, number of beds, bedrooms, and bathrooms. For example, accommodates had a correlation of about 0.60 with price, while bedrooms and beds were both above 0.50.

These results were not very surprising: larger listings that can accommodate more guests generally tend to cost more. Still, correlation only shows part of the picture, since variables can behave differently when they are considered together in a prediction model.



Spearman correlations between Airbnb prices and numeric listing features

Preparing the Data for Modeling

I split the data into 80% training and 20% testing sets. One thing I paid particular attention to was keeping the test set separate from preprocessing decisions. Missing-value imputation, outlier thresholds, encoding, and scaling parameters were learned from the training data and then applied to the test data.

Categorical variables such as neighbourhood and room type were converted using one-hot encoding. Numerical features were standardized for the linear models, while tree-based models such as Decision Tree and Random Forest were trained without scaling.

Comparing Different Regression Models

Instead of relying on a single algorithm, I compared several regression models: Linear Regression, Ridge, Lasso, ElasticNet, Decision Tree, Gradient Boosting, and Random Forest.

I evaluated the models using R^2 , Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). I also compared training and test performance, especially for the tree-based models, to check whether a model was learning useful patterns or simply fitting the training data too closely.

The Overfitting Problem

One of the most interesting problems I encountered was overfitting. My first unrestricted Decision Tree achieved a training R^2 of almost 1.00, which initially looked like an excellent result. However, its performance dropped significantly on the test set. The model was fitting the training data extremely well, but it was not generalizing to unseen listings.

To address this, I limited the complexity of the tree using parameters such as `max_depth`, `min_samples_split`, and `min_samples_leaf`. After these restrictions, the gap between training and test performance became much smaller.

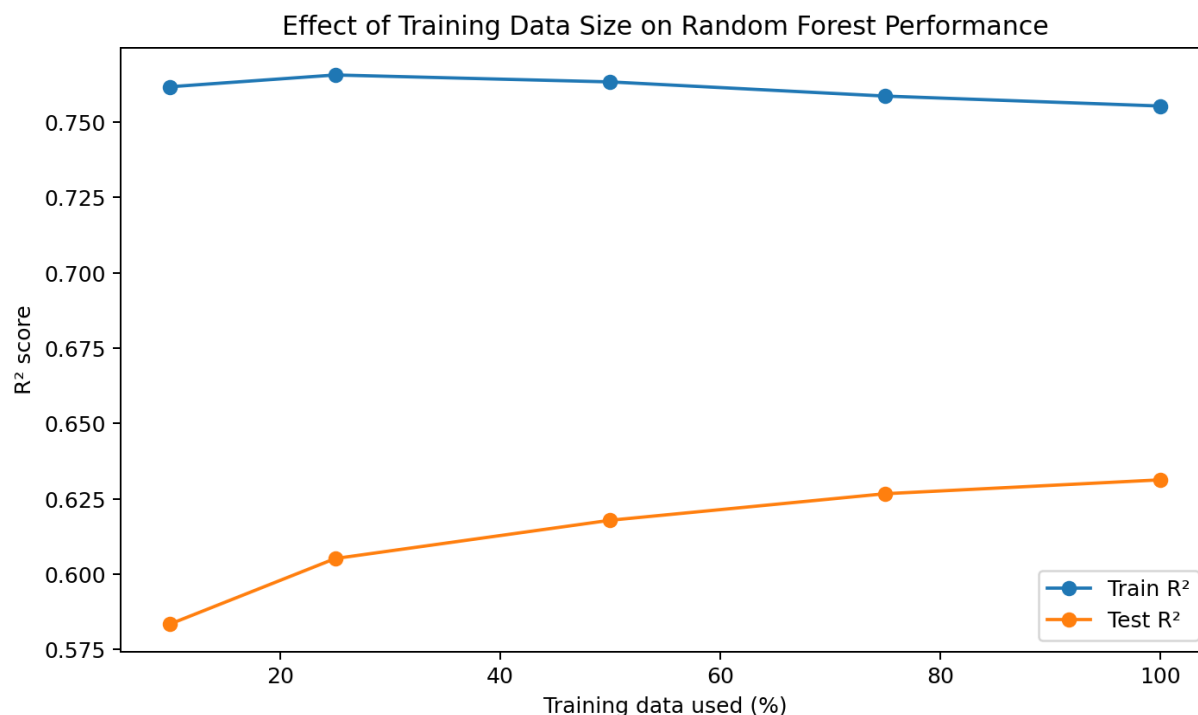
This was a useful reminder that a nearly perfect training score does not necessarily mean that a model is good.

Does Using Less Data Help?

I also wanted to see whether the amount of training data was contributing to the problem. To test this, I kept the test set fixed and trained the same Random Forest model using 10%, 25%, 50%, 75%, and 100% of the available training data.

The result was quite clear: test performance improved as more training data was used. The Test R^2 increased from about 0.58 with 10% of the training data to about 0.63 with the full training set.

In this case, using less data did not reduce the problem. More training data actually helped the model generalize better, which suggested that controlling model complexity was more useful than simply reducing the dataset.



Train and test R^2 scores as the amount of training data increases

Tuning the Random Forest

Since Random Forest showed the strongest performance among the models I tested, I focused on tuning it more carefully. I used GridSearchCV with 5-fold cross-validation to compare different combinations of tree depth, minimum leaf size, feature sampling, and other parameters.

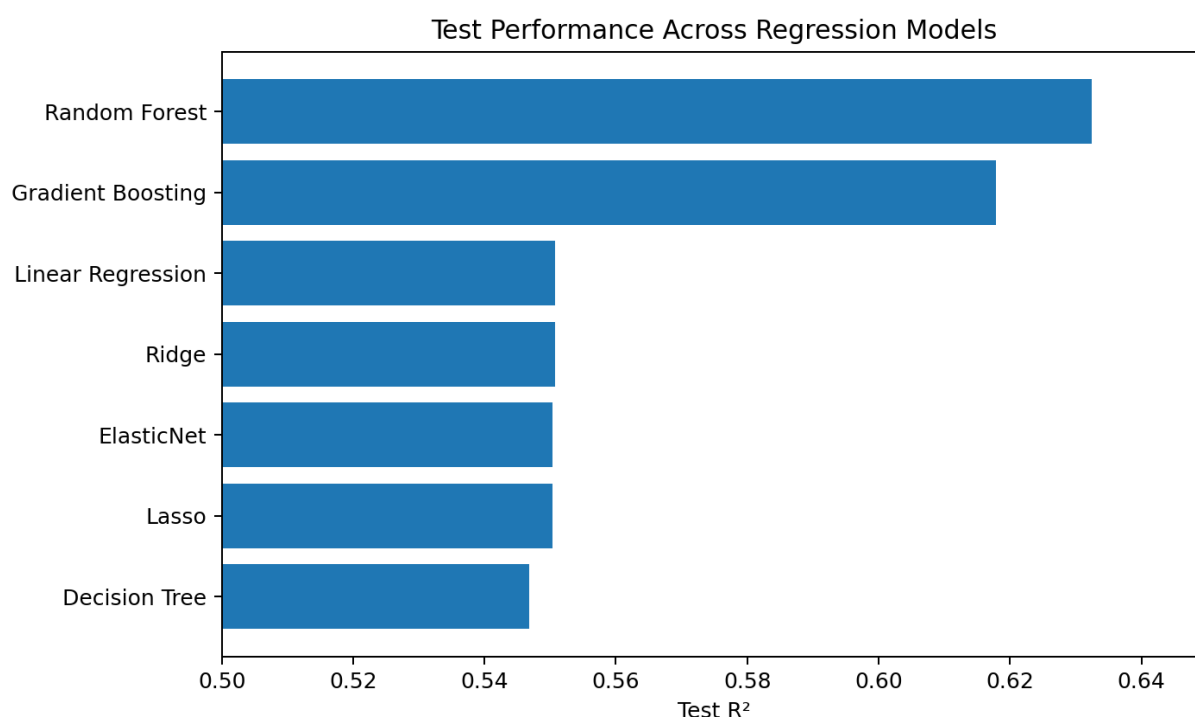
I did not want to choose a model only because it had the highest training score. Instead, I looked for a combination that maintained good validation and test performance while keeping the gap between training and unseen data under control.

Final Results

After comparing all the models, Random Forest gave the best overall performance. The final model achieved a Train R^2 of 0.756 and a Test R^2 of 0.632, with an RMSE of €81.71 and an MAE of €58.06.

The difference between the training and test scores shows that some overfitting still remains, so I would not say that the problem disappeared completely. However, compared with the initial models, the gap was considerably reduced while maintaining the best test performance.

In practical terms, the MAE of €58.06 means that the model's predictions differed from the actual nightly prices by about €58 on average.



Test R^2 scores across the regression models used in the project

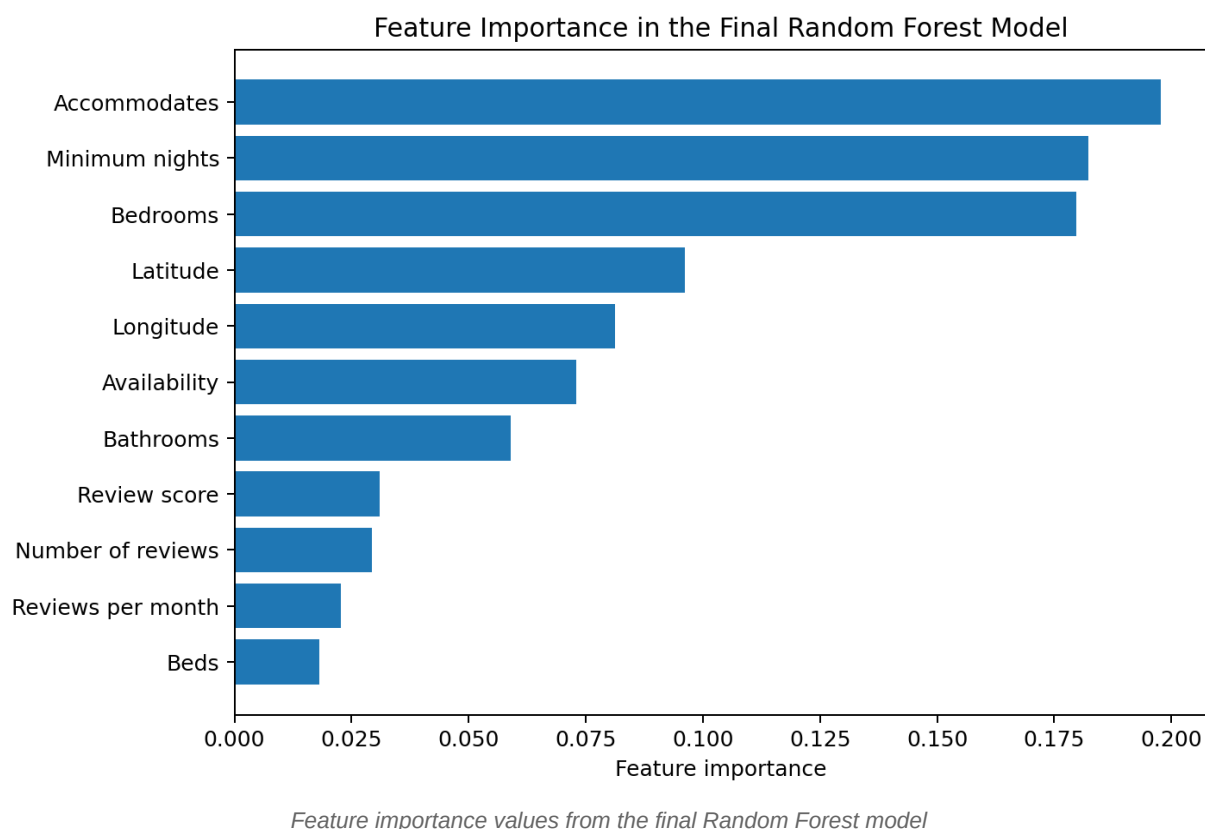
Which Features Matter Most?

After selecting the final Random Forest model, I also looked at feature importance to understand which variables contributed most to its predictions.

The most important features were guest capacity (accommodates), minimum stay requirements, and the number of bedrooms. Location variables such as latitude and longitude were also among

the more influential features.

I found `minimum_nights` particularly interesting. Its simple correlation with price was negative, but it still became one of the most important features in the Random Forest model. This shows why correlation alone does not tell the whole story: tree-based models can capture nonlinear relationships and interactions between variables.



What I Learned From This Project

The biggest takeaway from this project was that building a model was actually the easy part. Understanding the data, deciding how to handle unusual observations, and recognizing when a model was overfitting required much more attention.

I also learned that a better training score does not always mean a better model. The first Decision Tree was a good example of this: it performed almost perfectly on the training data but struggled with unseen observations. Using less data did not solve the problem either. What helped more was controlling model complexity and evaluating training and test performance together.

In the end, Random Forest gave the best balance for this dataset, with a Test R^2 of about 0.63 and an average absolute error of around €58. There is still room for improvement, especially by adding information such as seasonality, distance to major attractions, local events, or more detailed listing characteristics.

Overall, this project showed me that a reliable model is not necessarily the one with the highest score, but the one whose results make sense both on the training data and on data it has never seen before.