Airbnb Price Prediction in New York City

Machine Learning Project

Gabriel Maccione Cyrille Malongo Julien Maronne

December 4, 2025

Abstract

This report presents a machine learning project aiming at predicting the nightly price of Airbnb listings in New York City using the public `AB_NYC_2019` dataset. After an exploratory analysis and a careful preprocessing of the data, several regression algorithms are compared, including XGBoost, Random Forest, Ridge Regression, Gradient Boosting and LightGBM. The target variable is modeled in the logarithmic scale to reduce skewness and stabilize the variance. The final model is selected based on standard regression metrics and its ability to generalize. We also discuss the business implications of the results and the main limitations of the study.

Contents

1 Business scope

Airbnb hosts face the recurring problem of choosing an appropriate nightly price for their listings. An overestimated price may lead to low occupancy, while an underestimated price reduces potential revenue and may distort competition. A data-driven pricing tool can therefore:

- help hosts set an *optimal* price given the attributes of their listing;
- improve transparency and consistency of prices across similar listings;
- support strategic decisions such as investing in specific neighbourhoods or amenities.

The scope of this project is restricted to New York City listings in 2019. The objective is to build a predictive model that, given a listing's characteristics (location, room type, reviews, availability, etc.), outputs an estimate of its nightly price. From a business perspective, such a model can be integrated into a recommendation system, a dashboard for hosts, or used as a benchmarking tool by Airbnb itself.

The work is implemented in Python and organized around a main Jupyter notebook `Projet_MachineLearnin` and a companion script `Projet_Machine_Learning.py`. The dataset `AB_NYC_2019.csv` contains real Airbnb listings, and several open-source libraries are used, such as `pandas`, `scikit-learn`, `xgboost`, `lightgbm`, `geopandas` and `contextily`.

2 Problem formulation and methods

We formulate the task as a supervised regression problem. For each listing i, we observe a feature vector $\mathbf{x}_i$ (describing the listing) and a scalar response y_i (its observed nightly price). Our goal is to learn a function f such that

$$\hat{y}_i = f(\mathbf{x}_i)$$

is as close as possible to the true price y_i for unseen listings.

In practice, we model the transformed target

$$z_i = \log(1 + y_i)$$

instead of y_i directly, and we learn a function g such that $\hat{z}_i = g(\mathbf{x}_i)$. The final price prediction is then obtained as $\hat{y}_i = \exp(\hat{z}_i) - 1$.

2.1 Algorithm description

The general pipeline implemented in the notebook and script is the following:

1. **Data loading and exploration.** The dataset `AB_NYC_2019.csv` is loaded using `pandas`. Basic inspections (`head`, `info`, `describe`, histogram of prices, scatter plots) are used to understand the structure and quality of the data.
2. **Feature engineering and cleaning.**
   - Categorical variables (e.g., `neighbourhood_group`, `room_type`, `host_name`) are first filled with a "Missing" category when necessary and then encoded using their empirical frequency in the dataset. This corresponds to a simple *frequency encoding*.
   - Listings with non-positive prices (`price` $= 0$) are removed since they are considered outliers or invalid records.
   - Remaining missing values are removed or later imputed depending on the step.
   - A new target variable `price_log` is created as $\log(1+$`price`$)$, and the original `price` column is dropped.

- Geographical coordinates (`latitude`, `longitude`) and identifier columns (`host_id`, `id`) are dropped from the final feature set, as they either pose difficulties for the models or are not directly interpretable features.
- High-cardinality text features such as `name` and `neighbourhood` are also dropped to reduce dimensionality.

3. **Modeling.** The main algorithms evaluated are:

   - **XGBoost** (`XGBRegressor`), a gradient boosting model based on decision trees.
   - **Random Forest**, an ensemble of decision trees trained with bootstrap aggregation.
   - **Ridge Regression**, a linear regression with L_2 regularization.
   - **Gradient Boosting Regressor**, the classic gradient boosting implementation from `scikit-learn`.
   - **LightGBM**, a highly efficient gradient boosting framework optimized for speed and performance.

   All models are wrapped into `scikit-learn` `Pipeline`s that include preprocessing steps and the regressor itself.

4. **Evaluation.** The dataset is split into training and validation subsets (80% / 20%). Models are evaluated mainly using the Root Mean Squared Error (RMSE) and the coefficient of determination R^2 on the log-price scale. Additional diagnostic plots such as residual histograms and predicted vs. true values are used to visually assess the fit.

2.2 Limitations

The chosen methodology is subject to several limitations:

- **Single train/validation split.** The evaluation relies on a single 80/20 split, which may not be fully representative of the true generalization performance. Cross-validation would provide more stable estimates.
- **Frequency encoding of categorical variables.** Mapping string categories to their global frequency is simple and compact, but it may discard useful information and introduce subtle target leakage if not carefully controlled. A more standard approach would be one-hot encoding or target encoding with proper regularization.
- **Limited hyperparameter tuning.** For all models, only basic hyperparameters are used (e.g., `n_estimators = 100`). Systematic tuning (grid search, random search or Bayesian optimization) could further improve performance.
- **Temporal and market dynamics.** The dataset is static (2019 snapshot) and restricted to New York City. Seasonal effects, regulatory changes and market dynamics are not modeled, which limits the temporal validity and geographical generality of the conclusions.
- **Unobserved factors.** Important determinants of price such as photos, exact quality of the apartment, host response time or demand shocks are not included. The model therefore captures only part of the true price formation process.

3 Methodology

3.1 Data description and exploration

The `AB_NYC_2019` dataset contains close to 49 000 Airbnb listings in New York City and 16 variables, including:

- unique identifiers for listing and host (`id`, `host_id`);
- textual information (`name`, `host_name`);
- location variables (`neighbourhood_group`, `neighbourhood`, `latitude`, `longitude`);
- characteristics of the listing (`room_type`, `minimum_nights`, `availability_365`);
- social proof variables (`number_of_reviews`, `last_review`, `reviews_per_month`);
- the target variable `price` (nightly price in USD).

Exploratory analysis in the notebook includes:

- descriptive statistics of all numerical features and counts of categorical levels;
- a histogram of prices showing a highly right-skewed distribution with many low-priced listings and a long tail of expensive ones;
- a scatter plot of index vs. price highlighting outliers;
- boxplots of price by `neighbourhood_group` and by `room_type`;
- a map of New York City where listings are colored by price or log-price using `geopandas` and `contextily`.

Geographical visualization confirms that:

- Manhattan and parts of Brooklyn concentrate the most expensive listings;
- outer boroughs tend to present more moderate prices;
- price patterns are spatially clustered, justifying the importance of location features.

3.1.1 Missing values

The exploratory phase reveals several missing values:

- `last_review` and `reviews_per_month` contain many missing entries because not all listings have received reviews;
- some textual fields such as `name` or `host_name` may also contain a small number of missing values.

The handling of missing values is twofold:

- At an early stage, categorical variables are filled with a "Missing" category before being frequency-encoded.
- Later, in the modeling pipelines, numerical variables are imputed with the median and (potential) categorical variables with the most frequent category using `SimpleImputer`.

For some experiments, rows with remaining missing values after basic preprocessing are simply dropped to simplify the pipeline.

3.1.2 Imbalanced data

The data presents several forms of imbalance:

- **Price distribution.** Prices are strongly right-skewed: most listings are relatively affordable, with a few extremely expensive ones. This is problematic for models that assume homoscedasticity and symmetric errors.
- **Neighbourhood groups.** The proportion of listings is not homogeneous across neighbourhood groups: Manhattan and Brooklyn dominate the dataset, while Staten Island has comparatively few listings.
- **Room types.** Entire homes/apartments and private rooms represent the majority of listings; shared rooms and other categories are rare.

The main mitigation strategy is the log transformation of the target, which reduces the impact of very expensive listings on the loss function. No specific rebalancing is applied to neighbourhood or room type because the task is regression and the models used (tree-based ensembles) are robust to such distributional imbalances.

3.1.3 Outliers

The analysis reveals several types of outliers:

- Listings with `price` equal to 0, which are considered invalid or placeholder entries.
- Extremely high prices, which may correspond to luxury properties, multi-night pricing mistakes, or data errors.

The main cleaning step is to remove listings with price ≤ 0. The remaining extreme prices are not explicitly removed, but their effect is mitigated through the log transformation of the target. Tree-based models such as Random Forest and gradient boosting are also relatively robust to a small number of extreme observations.

3.2 Data splitting for train/test

After preprocessing, the cleaned dataset contains fewer rows than the original (only listings with strictly positive prices and complete information are kept). We split the data into:

- a training set (80% of the listings), used to fit the models;
- a validation set (20%), used to estimate performance and to compare models.

The split is random but controlled by a fixed `random_state` to ensure reproducibility. Formally, if $\mathcal{D}$ denotes the full dataset, it is partitioned into:

$$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}} \subset \mathcal{D}, \quad \mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{valid}} = \emptyset, \quad |\mathcal{D}_{\text{train}}| \approx 0.8|\mathcal{D}|, \quad |\mathcal{D}_{\text{valid}}| \approx 0.2|\mathcal{D}|.$$

This simple hold-out strategy avoids information leakage between training and evaluation, at the cost of some variance in the performance estimates.

3.3 Algorithm implementation and hyperparameters

For each model, a dedicated `scikit-learn` pipeline is defined. A typical example is:

```
numeric_features = [...]
categorical_features = [...]

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median'))
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', XGBRegressor(n_estimators=100, random_state=42))
])
```

In practice, because categorical variables are already frequency-encoded at an earlier stage, the list `categorical_features` is often empty and only numerical preprocessing is effectively applied.

The main hyperparameters chosen for the models are:

- **XGBoost:** number of estimators $n_{\text{estimators}} = 100$, default learning rate and tree depth, `random_state = 42`.
- **Random Forest:** $n_{\text{estimators}} = 100$, `random_state = 42`.
- **Ridge Regression:** regularization parameter $\alpha = 1.0$.
- **Gradient Boosting Regressor:** $n_{\text{estimators}} = 100$, `random_state = 42`.
- **LightGBM:** $n_{\text{estimators}} = 100$, `random_state = 42`.

No extensive hyperparameter search is performed; the focus is on comparing the algorithms under a reasonable, but not fully optimized, configuration.

4 Results

4.1 Metrics

To evaluate model performance, we use mainly the Root Mean Squared Error (RMSE) and the coefficient of determination R^2 on the log-price scale.

Given a set of n observations with true targets z_i and predictions $\hat{z}_i$, the RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_i - \hat{z}_i)^2}.$$

RMSE is expressed in the same units as the target (here, in terms of $\log(1 + \text{price})$) and penalizes large errors more strongly.

The R^2 score is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(z_i - \hat{z}_i)^2}{\sum_{i=1}^{n}(z_i - \bar{z})^2},$$

where $\bar{z}$ is the mean of the true targets. An R^2 close to 1 indicates that the model explains most of the variance in the data, while an R^2 close to 0 means that the model does not perform better than predicting the mean. Negative values indicate that the model is worse than the naive mean predictor.

In addition to these scalar metrics, the notebook reports:

- histograms of prediction errors $z_i - \hat{z}_i$ (residuals);
- scatter plots of $\hat{z}_i$ vs. z_i with a 45-degree reference line.

The residual distribution is centered around zero, with approximately 99% of errors between -1 and 1 in the log-price scale, meaning that most predictions are within a multiplicative factor of roughly e of the true price.

4.2 Overfitting / underfitting / imbalance

The models exhibit different degrees of bias and variance:

- The initial XGBoost model fitted on the preprocessed data achieves a moderate R^2 on the validation set (around half of the variance explained). This suggests that the model underfits the complex relationship between features and prices, possibly due to non-optimized hyperparameters.
- Random Forest and LightGBM typically achieve higher R^2 and lower RMSE on the training set. Their training performance is significantly better than that of simpler models such as Ridge Regression, indicating a lower bias but a higher risk of overfitting.
- Residual plots show that most predictions align with the reference line, but some points are scattered far away, corresponding to outliers or rare configurations (for example, extremely high prices or atypical locations). These observations are more difficult to model and contribute to heavier tails in the residual distribution.
- Because the distribution of neighbourhood groups and room types is imbalanced, models may fit better the majority patterns (e.g., Manhattan entire apartments) than the minority ones (e.g., shared rooms in Staten Island). However, given the regression setting and the relatively large dataset, this imbalance is not critical.

Overall, the best-performing models strike a compromise between underfitting and overfitting: they capture substantial structure in the data without perfectly memorizing the training set.

4.3 Evaluation, comparison with the baseline

As a reference, a naive baseline model can be defined as predicting the mean of `price_log` for all listings. This baseline has $R^2 = 0$ by construction and an RMSE equal to the standard deviation of the log-price.

The machine learning models clearly outperform this baseline:

- XGBoost already improves R^2 substantially compared to the baseline, but its validation performance remains limited, indicating that there is room for improvement.
- Random Forest achieves the best overall scores among the tested models, with the highest R^2 and the lowest RMSE on the training data. Validation metrics also show a clear gain compared to simpler models and the baseline.
- Gradient Boosting and LightGBM perform competitively, often close to Random Forest. Ridge Regression is systematically less accurate, which is expected given the non-linear nature of the problem.

A summary table of results, as computed in the notebook, compares the models based on their train R^2 and RMSE. Although exact numbers depend on the specific random split, a typical ranking is:

Model	Train R^2 (approx.)	Train RMSE (approx.)
Random Forest	highest	lowest
LightGBM	high	low
Gradient Boosting	high	low–medium
XGBoost	medium	medium
Ridge Regression	lowest	highest

This table is computed on the training set and used mainly to rank the models before validating the best candidate on the held-out data. These results justify the selection of Random Forest as the final model for this project.

5 Encountered issues

During the development of the notebook, we faced a few important issues that influenced the final design of the pipeline and our understanding of the results.

5.1 Evaluating on the training set instead of the test set

In an intermediate version of the notebook, when comparing several models in a loop, we mistakenly evaluated the performance on the *training* data instead of on the held-out validation set. Concretely, after fitting each pipeline on `X_train`, we computed predictions on `X_train` again and then calculated R^2 and RMSE using ($y_train, \hat{y}_$train). This produced very optimistic scores that did not reflect the true generalization ability of the models.

This error is a classic pitfall in machine learning: a model can always perform very well on the data it has seen during training, especially for flexible models such as Random Forest or gradient boosting, but this says little about its performance on new, unseen listings.

The issue was detected when we started analysing residuals and scatter plots on the validation set and noticed a clear gap between the apparent performance (based on the training metrics) and the behaviour on unseen data. We then corrected the code so that, for the model comparison, metrics are computed on `X_valid` and `y_valid`. After this correction, the scores became lower but more realistic, and we could fairly compare the models and select Random Forest as the final one.

5.2 Need for richer feature engineering

At the beginning of the project, we trained models on a relatively raw version of the dataset, keeping many columns that were noisy or weakly informative (such as identifiers or high-cardinality text fields) and without transforming the target variable. The resulting performance was disappointing, with high errors and unstable behaviour across splits.

To address this, we had to significantly enrich and refine the feature engineering:

- removing listings with zero price and dropping the raw `price` in favour of its logarithm `price_log`;
- dropping `id`, `host_id`, `latitude`, `longitude`, `name` and `neighbourhood`, which either carried little predictive signal for our models or introduced unnecessary complexity;
- frequency-encoding the main categorical variables and carefully handling missing values;
- analysing correlations with `price_log` to keep only the most relevant variables.

Only after this more thorough feature engineering did the models start to achieve acceptable R^2 and RMSE on the validation set. This experience highlighted the importance of data preprocessing: improving the quality and representation of the features was at least as impactful as changing the machine learning algorithm itself.

6 Discussion and conclusion

This project demonstrates how classical machine learning techniques can be used to predict Airbnb prices from a real-world dataset. By combining data cleaning, exploratory analysis, and several regression algorithms, we obtain a model that explains a substantial proportion of the variance in log-prices and produces reasonably accurate predictions for unseen listings.

From a business perspective, the analysis confirms that:

- **Location** is a key driver of price: listings in Manhattan and central areas of Brooklyn command higher prices.
- **Room type** matters: entire homes and apartments are significantly more expensive than private or shared rooms.
- **Activity indicators** such as the number of reviews and availability also provide useful signals about price.

The final Random Forest model can serve as a decision-support tool for hosts and platform managers. Given the characteristics of a listing, it returns a recommended price that is consistent with market patterns observed in 2019 New York data.

However, several limitations should be stressed:

- The model is trained on a single city and year; its predictions may not generalize to other cities or to later periods, especially after regulatory changes or market shocks.
- Some important determinants of price (quality of pictures, description text, host behavior, seasonality) are not included in the dataset.
- The frequency encoding of categorical variables, while simple, might not be the most expressive representation, and it may cause a loss of interpretability.
- Hyperparameter tuning and more robust validation strategies (e.g., k-fold cross-validation) could further improve the performance and reliability of the results.

Future work. Several extensions could be considered:

- perform systematic hyperparameter optimization for tree-based models;
- explicitly model spatial effects, e.g., using spatial cross-validation or distance-based features;
- incorporate temporal information (booking date, season) when available;
- explore interpretable models and feature importance methods (e.g., SHAP values) to better understand the contribution of each variable;
- deploy the model in a simple web interface to assist hosts in pricing their listings.

In conclusion, the project successfully builds a working machine learning pipeline for Airbnb price prediction in New York City, highlights the key steps of data preparation and model evaluation, and opens avenues for more advanced and robust approaches.

Code and resources

All the experiments described in this report are implemented in the files `Projet_MachineLearning_Vfinal(2).i` and `Projet_Machine_Learning.py`, using the dependencies listed in `requirements.txt`.