Methodology

In this section, five models were built to predict the prices based on the available features. The three best models (Lasso, XGBoost and Model stacking), selected based on the scores given by Kaggle, will be thoroughly discussed, and the other two models (Random Forest and LightGBM) will be briefly explained. Other models, including Ridge regression and Catboost, were also explored but will not be discussed in this report.

Model 1: Lasso

The Lasso model was selected as the best linear model based on Kaggle scores. Lasso refers to the Least Absolute Shrinkage and Selection Operator, which performs ℓ_1 regularisation to enhance model performance with two main properties: variable selection and shrinkage. Through shrinking the coefficients of features, the lasso method is able to significantly reduce variance without substantially increasing bias. This particular application of shrinkage also allows it to perform variable selection, as the coefficients for features are reduced to 0 individually, compared to the ridge method, which simultaneously sets all feature coefficients to 0.

Therefore, an important advantage of Lasso over Ridge regression is its interpretability. Following data processing and feature engineering, the training dataset contained over 10,000 observations and 130 features. Therefore, following the goal of identifying important features to enhance the stakeholders' decision-making, the lasso method was deemed more appropriate than the ridge regression method due to its feature selection capabilities.

When compared with subset selection, which chooses the best subset of variables based on certain criteria, Lasso is much more computationally efficient. The subset selection method is much more time-consuming because of the combinatorial explosion of the parameters. It also leads to a nonconvex problem that is more difficult to fit.

Hyperparameter tuning

$$\widehat{\beta_{lasso}} = \underset{\beta}{argmin}\left\{\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{i=1}^{p}\left|\beta_j\right|\right\}$$

The most critical part of the formula is the hyperparameter λ. The hyperparameter value is essential in enhancing prediction accuracy and interpretability as it controls the strength of variable selection and shrinkage.

When the hyperparameter λ equals 0, the Lasso will simply reduce to an OLS problem.

1. When $\lambda = \infty$, all of the coefficients in the model will be shrined to zero.

2. Increasing λ will increase the bias and decrease the variance, leading to an underfitting problem and vice versa.

Commonly, the value of λ is chosen by cross-validation. The lowest CV score is achieved when the hyperparameter for Lasso in this project is set to be 0.0001 (4 d.p.)

Since the lasso method constrains the size of the coefficients based on the magnitude of each variable, we first standardised the predictors in both the training and test dataset through the robust scaler to ensure that all predictors are on the same scale.

Model results & limitation

First, we use the k-fold cross-validation method to find how the Lasso model we built fits our training data. The method involves dividing the dataset into k equal groups, each of which was utilised once as a test group (to be predicted) and the rest of the time as a training group (to predict the others). Here we choose k=5, which divides the dataset into 5 equal groups. The cross-validation score of the lasso method was evaluated using the 'cross_val_score' function. When λ =0.0001, the lowest root mean squared error (RMSE) of this LASSO model on the training dataset is 0.3909. This model will be used to predict the daily Price of Airbnb rentals contained in the validation dataset, and an exponential transformation has been applied to the predictions, as the response variable used to train the model was log-transformed.

The main disadvantage of the lasso method is that it is not robust to highly correlated variables in the dataset. The steps taken in the data processing and feature engineering sections aimed to overcome this limitation by removing features exhibiting high correlation.

Model 2: XGBoost

XGBoost refers to Extreme Gradient Boosting. The XGBoost model was the best nonparametric model based on Kaggle scores. XGBoost was used in this project as there are more than 10,000 observations with more than 100 features after feature engineering, and the dataset contains both categorical and numeric variables. The XGBoost model is a widely used application of gradient boosting with significant advantages compared to other models. In particular, XGBoost tends to have greater execution speed and predictive performance (Jason, 2016). Furthermore, as the XGBoost model is well documented, it can be easily accessed and tuned to the requirements of the relevant stakeholders for future predictions, with a relatively lower level of required training.

In boosting, we use an additive model with decision trees as basis functions that are essentially weak learners to add up a strong learner. The model's capacity increases with the number of basis functions, eventually adapting to the size of the training data. However, XGBoost uses a regularised form of gradient boosting to penalise the complexity of each tree and thus prevent overfitting (Neetika, 2020). It solves the regularised risk minimisation problem with a penalty factor on complexity $\Omega(\theta_m)$.

We specified the penalty term by:

$$\Omega(\theta_m) = \gamma|T_m| + \frac{\lambda}{2}\sum_{j=1}^{|T_m|} w_{j_m}^2$$

The penalty term here can be considered as a ℓ_1 regularisation and the hyperparameters in $\Omega(\theta_m)$ is γ and λ.

Hyperparameter tuning

In python, we use the XGBRegressor package to perform the XGBoost model. In our implementation, we optimised the following hyperparameters through the random search method:

1. n_estimators: The number of gradient boosting trees (Python API Reference, n.d.). which is similar to the n_estimator in the RandomForestRegressor. Here we set the n_estimator to follow a discrete uniform distribution between 100 and 2500, and the best n_estimator found by the randomised search is 1623.

2. learning_rate: Boosting learning rate (Python API Reference, n.d.). This is the

shrinkage factor in gradient boosting, adjusted to slow down the learning in the model and prevent overfitting; the model's performance and computational costs increase as the learning rate decreases. Here we set the learning_rate to follow a uniform distribution between 0.005 and 0.1, and the best learning_rate found by the randomised search is 0.033.

3. max_depth: Maximum tree depth for base learners (Python API Reference, n.d.). The higher the value of max_depth, the better the results. However, a higher maximum depth can also lead to overfitting as the model will capture too much information, picking up 'noise' from the training set. In this case, the optimal parameter is 4.

4. sub_sample: Subsample ratio of the training instance. A sub_sample of 0.5 means the XGBoost will randomly sample only half of the training data before growing trees to prevent overfitting (Python API Reference, n.d.). We set the sub_sample to follow a uniform distribution between 0.5 and 1, and the best sub_sample found by the randomised search is 0.695.

5. reg_alpha & reg_lamda : ℓ_1& ℓ_2 regularisation term (XGBoost Python Package, n.d.). The higher the value, the more conservative the model will be. The tuned parameter is 0.744.

Model results & limitation

The cross-validation RMSE of this XGBoost model is 0.326, which is significantly less than that of the Lasso model. This model will be used to predict the daily Price of Airbnb rentals contained in the validation dataset. Although this method makes no assumptions on the normality of the features and response variable, it was found that a better generalisation performance was achieved when a log-transformation was applied to the response variable. This is most likely due to the way in which the underlying decision trees split the data.

Although XGBoost is designed to handle large datasets, it can be memory and time-consuming, especially when compared to LightGBM. If Airbnb wants to use this model to fit a larger dataset, it will require much more computing power and resources.

Model 3: Random Forest

Random Forest is based on the idea of bagging. Bagging is a form of bootstrap aggregation where bootstrap samples are generated by randomly sampling from the training data with replacement, after which a decision tree is fitted to each sample, and the final prediction is taken as the average prediction across all trees (Marcel, 2021). The random forest method further improves model performance by only selecting a subset of the features for each decision node. Additionally, Random Forests can decorrelate trees to largely reduce the variance and improve prediction stability (James et al. 2013). The details of this algorithm are shown in Appendix M.

In this algorithm, multiple hyperparameters require tuning. The number of trees in the forest was set to 1000. The optimised value of max_features, the number of features to consider when looking for the best split, was 34. The minimum number of samples required to be at a leaf node (RandomForestRegressor, n.d.) was optimised at 1. This value will lead to better test set performance, but also raises concerns regarding overfitting.

The cross-validation RMSE of this Random Forest model is 0.355. The Random Forest algorithm has various limitations. Firstly, it requires a greater amount of computing power, time and resources than other models. Due to limited computational power, most hyperparameters were set to default values in this project, which may have affected the performance of the model. Additionally, the Random Forest model is more challenging to interpret due to the complexity of random sampling, which is not ideal for companies like Airbnb that want to assess the detailed influence of individual factors.

Model 4: LightGBM

Similar to XGBoost, LightGBM is another efficient implementation of Gradient Boosting Decision Trees (GBDT). Compared to XGBoost, LightGBM further optimises memory usage, efficiency and accuracy.

Several key features of LightGBM are described below (Ke et al., 2017):

1. LightGBM employs a histogram-based algorithm, which consumes less memory and simplifies data separation;

2. LightGBM abandons the level-wise decision tree growth strategy used by most GBDT tools and uses a leaf-wise algorithm with depth restrictions to obtain better accuracy for the same number of splits;

3. LightGBM has optimised support for categorical features, forgoing the need for categorical processing through dummy variables (forgoing the need for dummy variables).

To achieve the highest level of predictive accuracy, several hyperparameters must be optimised for the LightGBM model. The optimised hyperparameters and their values were found through a cross-validated random search and shown in Table 14.

Table 14: Hyperparameters of LightGBM

Hyperparameter	Value
Learning rate	0.067
Number of estimators	2039
Number of leaves	5
Regression lambda	0.157
Subsample	0.519

The resulting LightGBM model had a validation set RMSE of 0.327, showing a significant performance improvement over the linear models.

A possible disadvantage of the LightGBM model is that the leaf-wise strategy may result in greater tree depth, potentially leading to over-fitting. Another limitation is that as a new method, LightGBM not only has less information from the literature but also lacks the community support that might facilitate the optimised utilisation of this model (Kasturi, 2019).

Model 5: Model Stacking (Final Model)

Of the models assessed, the model stack containing the LightGBM, XGBoost, random forest, and ordinary least squares models demonstrated the highest generalisation ability. This model stacking method is considered an ensemble modelling method, combining predictions from the selected base models to overcome weaknesses in the individual models and improve generalisation performance. The weights of the base models within the ensemble are typically set to be equal or optimised using meta regressors, with models demonstrating greater generalisation ability often possessing greater weights. Many meta regressors were tested when optimising the weights of the base models, including linear regression, regression tree, and boosting models. Of these, the meta regressor based on the ordinary least squares method had the best validation set performance and was thus selected to be the final model. The addition of a third layer was considered. However, this was not realistic given our resource constraints.

The ensemble was first generated based on all other base models and reduced to contain only those with positive weights. These were the random forest, XGBoost, LightGBM, and ordinary least squares models, with weights of 0.065, 0.517, 0.424, and ≈0.000 respectively. This outcome is consistent with our findings in the previous sections, as the XGBoost model showed the highest predictive accuracy on the validation set, followed by the LightGBM and random forest models.

To assess the performance of the model, it was analysed based on five-fold cross-validation. The model displayed an RMSE of 0.320. Compared with previous models, this showed a slight improvement In predictive accuracy, with RMSE being 0.002 higher than the XGBoost model. Indeed, the model stack also showed higher performance on the final validation set, returning an RMSE score of 0.325

However, due to the ensembled nature of the model stack, it can be challenging to interpret. Indeed, we could extract the most influential variables and their respective effect for the overall model. However, the importance of individual features could potentially be extrapolated as a function of the weight of the base model and its importance in the base model. One main limitation of ensemble methods in practice is that it can be computationally expensive and time-consuming to train, as it requires the assembly of many base models to be effectively used.

Model Validation

All predictions generated by the models in this report were based on the data provided from the Kaggle competition. The validation score of each model was taken from their respective public root mean squared logarithmic errors from Kaggle. These scores have been summarised in Table 15. The scores in table 14 have been rounded to 4 d.p. to better represent differences in predictive accuracy.

Table 15: Kaggle Scores of Five Models

Rank	Model	Validation set Kaggle results
1st	Model Stacking (Final Model)	0.3245
2nd	XGBoost (Bench Mark Model)	0.3259
3rd	LightGBM	0.3275
4th	Random Forest	0.3580
5th	Lasso	0.3939

The XGBoost model was chosen as the benchmark model due to its widespread use in the industry and its strong predictive capabilities. While the XGBoost model performed better than all other non-ensemble models, the LightGBM model achieved a similar level of predictive accuracy with a fraction of the computational cost. Overall, the XGBost model and the LightGBM model demonstrated significantly greater generalisation ability over other assessed non-ensemble methods, with the random forest method performing better than linear models. The differences in performance can likely be attributed to two characteristics: the ability of tree-based methods to model nonlinear relationships, and the generalisation ability of the XGBoost and LightGBM models.

The model stack demonstrated the best generalisation performance, showing an improvement over the benchmark XGBoost model of 0.0014. This result suggests that the ensemble was able to improve the likelihood of achieving the best performance by combining the learning algorithms. One caveat, however, is that a complex meta-model may lead to overfitting concerns, thus affecting its performance on the test dataset. Therefore, this should be taken into account if the company wishes to continue using this model in the future.