STA 671 Kaggle Competition Report

Introduction

This is a Kaggle competition project about the prices of Airbnb listing places in Buenos Aires, the capital and largest city of Argentina. Airbnb is an online marketplace that provides a platform to connect hosts and tenants. It has become one of the largest hotel chains around the world. With its great popularity, two problems have attracted wide interest:

- What are the important features that influence the Airbnb price?
- How to make use of these features to build a predictive model?

The major objective of this project is to select the important features based on exploratory data analysis and construct a predictive model to predict the price of Airbnb using these features, in the hope that the produced mechanism will provide a price reference for both the Airbnb hosts and the tenants. The price in the Kaggle dataset has been divided into 4 categories, from the cheapest to the most expensive labelled as 1 to 4. Thus the price prediction problem in this project is a multiclass classification problem instead of a regression problem.

According to the official website of Airbnb, it describes its reservation price setting as a combination of the nightly rate set by the host and the additional fees set by either the host or Airbnb. The additional fees may include the Airbnb service fee, cleaning fee, local taxes etc. The additional fees are straightforward, yet the nightly rate is somewhat subjective which depends on the host's preference. Fortunately, the Kaggle dataset offers several features that may be highly correlated with this part of price. The overall information in the Kaggle dataset contains:

- Price per night: this is on the scale of 1 to 4 and serves as the outcome variable;
- Characteristics of the host: this includes basic information about the host neighborhood, room type, number of bedrooms and bathrooms etc.;
- Additional fees: this part is about cleaning fee and extra person fee set by the host.

Based on these features, a multiclass classification model can be built to predict the price category. The metric applied to evaluate the model performance is categorization accuracy. This metric will also be employed to implement model selection as well as hyperparameter tuning.

Exploratory Data Analysis

To start with the exploratory data analysis, the first step is to analyze the feature definitions, based on which the initial feature engineering can be conducted. The original training dataset contains 24 features and the response variable *price*. Among the 24 features, *id* is the primary key of the dataset, thus having no helpful information about price prediction. Besides, *neighbourhood* has 45 distinct values, so it's impractical to treat it as a pure categorical variable. The better option is to bring into an external dataset that contains detailed information such as area, population and number of communes about each neighborhood of Buenos Aires. This external dataset is from an open data source [1]. To more accurately capture the prosperity of a certain area, a new variable *density* is constructed using the division of population and area features rather than directly applying these features into the model. Also, since *is_business_travel_ready* only contains one value, this is an unneeded variable and can be dropped from the dataset. *last_review* and *host_since* are date variables that indicate the date of the last review of the corresponding host at Airbnb website and the date of the host's first listing. It seems to make more sense if the difference of them is used to express the host's experience. Therefore, another new variable *host_duration* is defined as *last_review* minus *host_since* in terms of the number of days. Then *cancellation_policy* needs to be paid attention to. This is a categorical variable that contains five categories: 'flexible', 'moderate', 'strict_14_with_grace_period', 'super_strict_30' and 'super_strict_60'. Based on the level of cancellation relaxation, it can be treated as an ordinal values with 0 representing the most flexible type 'flexible' and 4 representing the most strict type 'super_strict_60'.

According to the above analysis, the remaining categorical features are:

- *room_type*
- *host_is_superhost*
- *bed_type*
- *instant_bookable*
- *required_guest_profile_picture*
- *required_guest_phone_verification*

Figure 1 provides an overall visualization about the association between categorical features and outcome variable *price*. Each circle size in the plots illustrates the number of observations occurred at each combination of values. Interesting patterns are trying to be figured out from this figure. For example, if the host is superhost, which means that *host_is_superhost* is true, the circle size is larger for price 4 than that for price 1, indicating that superhosts tend to set their prices higher than the others'. However, the judgement of correlation based on this plot is subjective, thus it can only serve as a tool to help with variable importance intuition instead of a tool to directly select variables.

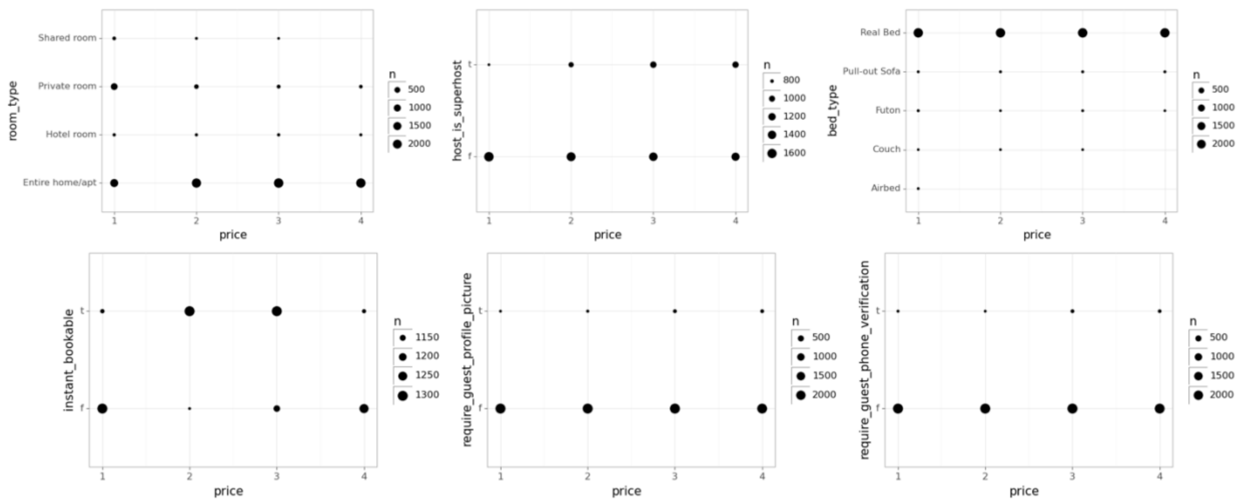


Figure 1: Plots of categorical predictors versus *price*

The remaining numeric features are:

- *minimum_nights*
- *number_of_reviews*
- *reviews_per_month*
- *calculated_host_listings_count*
- *availability_365*
- *bathrooms*
- *bedrooms*
- *beds*
- *cleaning_fee*
- *guests_included*
- *extra_people*
- *maximum_nights*
- *cancellation_policy*
- *density*
- *Commune*
- *host_duration*

Similarly, **Figure 2** is an overall visualization to explore on the correlation between numeric features and *price*. Some of the correlations are quite explicit. For instance, higher cleaning fee indicates higher price. This makes sense according to Airbnb's reservation price setting rule. Higher additional fees like cleaning fee will be directly added to the final price. Similar as before, the inference can only help with intuition construction and should not be directly used to select features due to its subjective property.

One potential problem in the train-test evaluation mechanism is the possible disparity of categories between training set and testing set. This problem can happen to the categorical features of the dataset. After inspection, feature *bed_type* does have disparity problem. In the training set, this variable contains values 'Airbed', 'Couch', 'Futon', 'Pull-out Sofa' and 'Real Bed'. But in the testing set, it only includes 'Airbed', 'Futon', 'Pull-out Sofa' and 'Real Bed'. One feasible solution is to conduct value combination. For example, the 'Couch' category in the training set can be directly combined with 'Airbed' to eliminate disparity. But before implementation, a detailed inspection should be done to *bed_type* feature.

Table 1: Detailed inspection of *bed_type* feature

Value	Count	Frequency (%)
Real Bed	9641	99.60%
Pull-out Sofa	23	0.20%
Futon	12	0.10%
Couch	3	< 0.1%
Airbed	2	< 0.1%

According to **Table 1**, more than 99.5% of the *bed_type* values are ‘Real Bed’. The information collected from other values is extremely limited. Therefore, it seems that compared with combining the values of *bed_type* in the training set, dropping this feature is a better choice.

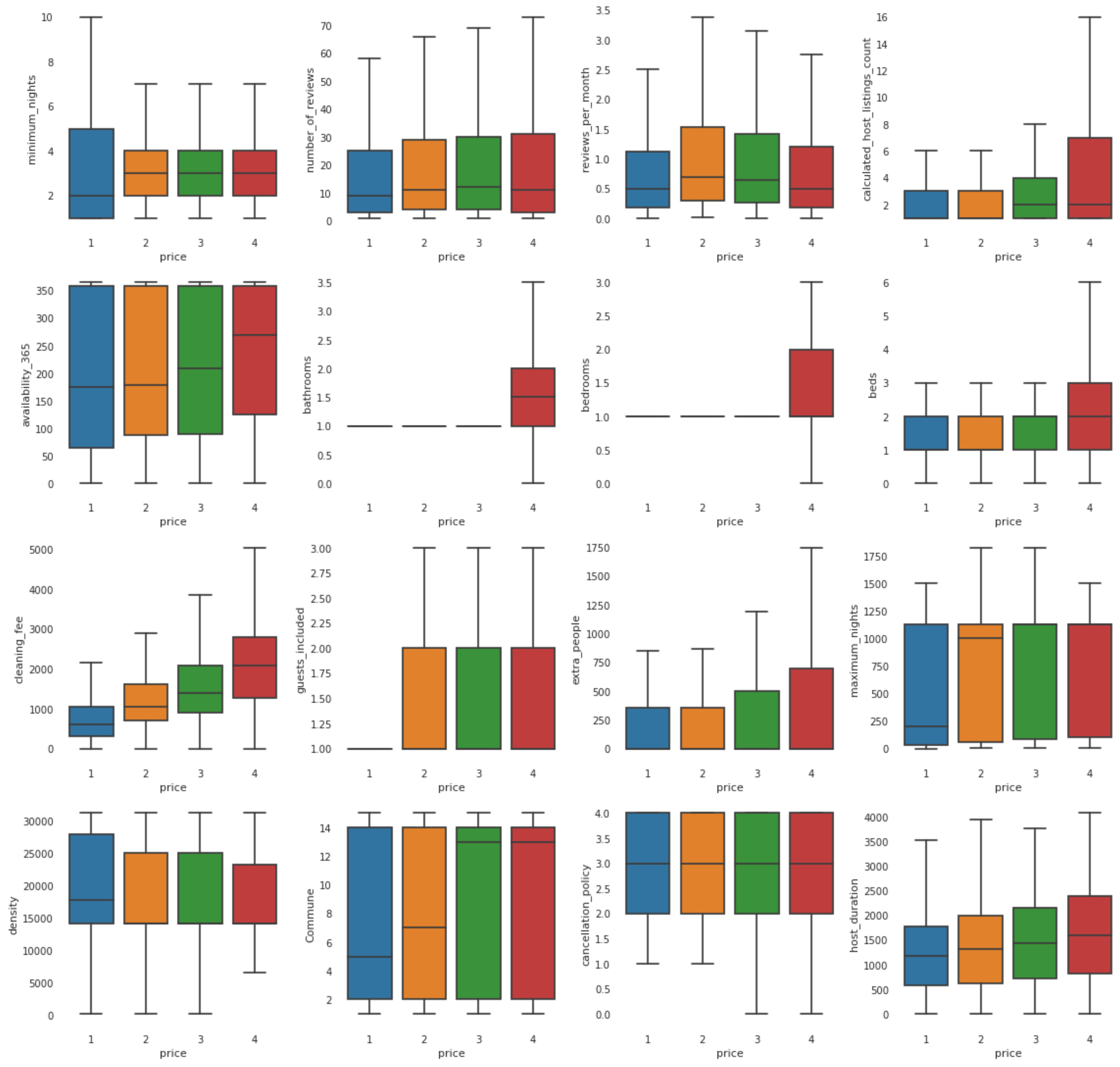


Figure 2: Plots of numeric predictors versus *price*

In the last step of exploratory data analysis, all of the remaining categorical features should be converted through hot encoding process. For instance, a categorical variable containing three categories A, B, C will have each of its category converted into vectors (1,0,0), (0,1,0) and (0,0,1). To avoid collinearity, the first element of each vector will be deleted. Through this procedure, the categorical variables are converted into a form that can be dealt with by the machine learning algorithms.

Models

This project involves a multiclass classification problem. Based on **Figure 1** and **Figure 2**, the correlation between predictors and the outcome variable *price* doesn't seem to be linear intuitively. Therefore, tree-based models are preferred in this circumstance. To verify the conjecture, an advanced Python library called *PyCaret* is applied. *PyCaret* is an open-source, low-code machine learning library in Python that automates machine learning workflows [2]. To use this package, the only thing that needs to be done is to input the dataset and evaluation metric. Everything else will be automatically processed by this package. In my opinion, this is an excellent initial step to conduct model selection.

Table 2: *PyCaret* model selection result

Model	Accuracy	TT (Sec)
Light Gradient Boosting Machine	0.5415	1.0289
CatBoost Classifier	0.5401	13.687
Extreme Gradient Boosting	0.5363	7.6988
Gradient Boosting Classifier	0.5242	5.2793
Ada Boost Classifier	0.5077	1.5569
Random Forest Classifier	0.5072	0.5797
Extra Trees Classifier	0.5049	1.2872
Ridge Classifier	0.4787	0.0544
Linear Discriminant Analysis	0.4752	0.2699
Naive Bayes	0.4414	0.0347
Decision Tree Classifier	0.412	0.2661
Logistic Regression	0.3995	0.4097
K Neighbors Classifier	0.3949	0.2009
SVM - Linear Kernel	0.3183	0.7584
Quadratic Discriminant Analysis	0.2798	0.1163

After entering the processed training dataset, partial result of *PyCaret* is shown in **Table 2**. The other part is dropped merely because accuracy is the evaluation metric in this project rather than AUC, recall, precision etc. The accuracy in the table is calculated through cross validation. Based on **Table 2**, among the top 6 classification models, Random Forest Classifier and Light Gradient Boosting Machine are the fastest using 0.5797 seconds and 1.0289 seconds respectively. Besides, Random Forest is a tree-based bagging algorithm, while Light Gradient Boosting is a tree-based boosting algorithm. From the perspective of model diversity as well as computational efficiency, Random Forest Classifier and Light Gradient Boosting Machine will be the algorithms used in this project.

For Random Forest Classifier and Light Gradient Boosting Machine, high-quality libraries are already available. Scikit-learn package [3] is a very widely used Python machine learning toolkit. Its in-built function for Random Forest (*RandomForestClassifier*) is easy and efficient to implement. For Light Gradient Boosting Machine, LightGBM package [4] is a great option. It provides faster gradient boosting framework with lower memory usage as well as better accuracy.

Training

In this section, the training mechanisms of algorithms chosen at the previous section, which are Light Gradient Boosting Machine and Random Forest Classifier, will be detailed introduced.

Light Gradient Boosting Machine is a tree-based Gradient Boosting algorithm with high performance. The general tree-based Gradient Boosting classifier requires several hyperparameters including the loss function L, the number of trees to fit M, the learning rate λ, and the stopping condition. With these hyperparameters, the pseudocode for the general gradient boosting algorithm is:

```
• Initialize model with $F_0 = argmin_F \sum_{i=1}^{n} L(y_i, F(x_i))$.
• From $m = 1$ to $M$:
    o Set $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \dots, n$.
    o Fit a tree $h_m$ to the residuals.
    o Update the tree $F_m(x) = F_{m-1}(x) + \lambda h_m(x)$.
• Output $F_M(x)$.
```

The stopping condition can be determined by the maximum tree depth or tree leaves, implying that each fitted tree's depth and number of leaves should not exceed the maximum tree depth and maximum tree leaves. Compared with general Gradient Boosting algorithm, Light Gradient Boosting Machine makes some improvement in the perspective of coping with continuous features and tree construction. This algorithm is histogram-based, indicating that it will bucket continuous feature values into discrete bins [4]. Besides, it grows trees leaf-wise rather than depth-wise, which will increase training speed and achieve lower loss.

Random Forest is a tree-based bagging algorithm. The hyperparameters involving in Random Forest are the number of trees in the forest B, the number of features to be considered in each split m, and stopping condition. The pseudocode for Random Forest is

```
• From $i = 1$ to $B$:
    o Take a bootstrap samples.
    o Fit a tree $h_i$ on the bootstrap samples. Each time a split is considered, a random
      sample of $m$ predictors is chosen.
• Output $F_B(x) = \frac{1}{B}\sum_{i=1}^{B} h_i(x)$.
```

The stopping condition of Random Forest is similar to that of Light Gradient Boosting Machine. However, unlike Light Gradient Boosting, the trees in Random Forest are independent. Since a random selection of predictors is conducted for each split, the trees are decorrelated, thus their combination will reduce the result variance.

The running times of Light Gradient Boosting Machine and Random Forest are shown in Table 3. The results are the training time of each algorithm using the default hyperparameters measured in wall time.

Table 3: Algorithm training time

Model	Wall Time
Light Gradient Boosting	1.51s
Random Forest	1.57s

Hyperparameter Tuning

In the training section, the hyperparameters of Light Gradient Boosting Machine and Random Forest are introduced. In this section, certain hyperparameters will be tuned based on the accuracy performance.

For Light Gradient Boosting Machine, I'm interested in the number of trees to fit (M) and the learning rate (λ). If M is too large, boosting can easily overfit. And λ controls the rate at which the boosting learns. These two parameters are closely dependent. Smaller learning rate might require larger number of trees to improve performance. Therefore, these parameters should be treated carefully. Other parameters will use the default values of LightGBM package. To evaluate the performance of parameters, 5-fold cross validation is implemented. Each combination of M and λ will be trained on four folds of data and the resulting model will be used to predict the accuracy of the remaining fold. The mean of 5 iterations will be applied as the final score of this combination of parameters. If the possible values of M are 50, 100, 150, 200, 250, 300, 350, 400, and the possible values of λ are 0.001, 0.01, 0.1, 1, 10, the final combination is $M = 400$ and $\lambda = 0.1$.

Similarly, for Random Forest algorithm, the number of trees in the forest B and the number of features to be considered in each split m arouse my interest. These features will greatly influence the performance of Random Forest algorithm. The selection mechanism is the same as the case for Light Gradient Boosting Machine. 5-fold cross validation is applied to select the best combination in terms of accuracy when B is chosen from 50, 100, 150, 200, 250, 300, 350, 400, and m is chosen from 0.1, 'log2', 'sqrt', 0.3, 0.5. The value of m represents the proportion of features to select from in each split. The final combination is $B = 300$ and $m =$'log2'.

The parameter tuning process is shown in **Figure 3**. The x axis is the parameter being tuned, and the y axis is the 5-fold cross validation accuracy. All the other parameters within our range of study are set to be the optimal values. And the parameters outside our range of study are set to be the default values.

Cross Validation Result

5-fold cross validation is applied to implement parameter tuning as well as model evaluation. The training dataset is split into 5 folds with roughly equal size. For every iteration, four folds are used to train the model until no more new combination is available. The trained model will then be applied to predict the price in the remaining fold, based on which the accuracy of the remaining fold can be obtained. After cross validation, five accuracy scores will be accessible for each algorithm and these scores are shown in **Table 4**. As can be seen, Random Forest has higher accuracy than Light Gradient Boosting Machine in every cross validation result, thus Random Forest seems to be more appropriate in this project.

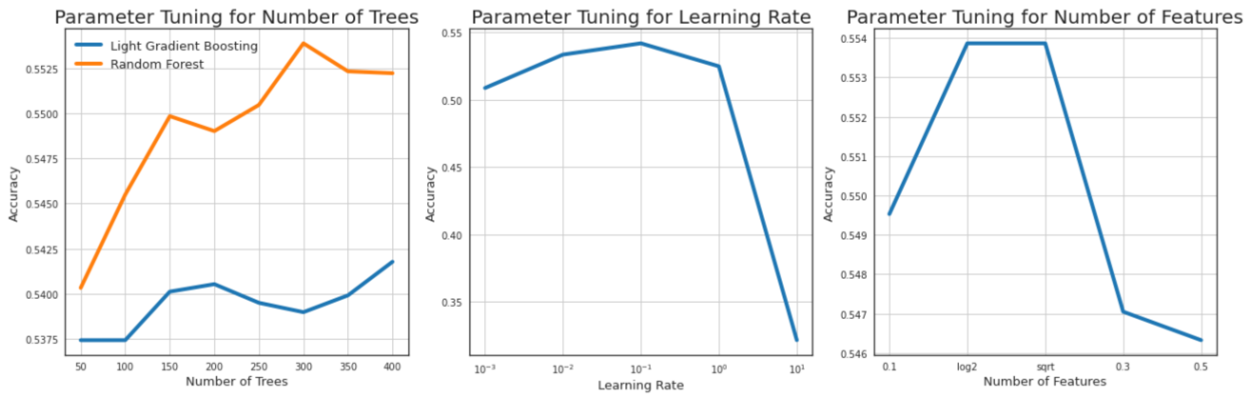


Figure 3: Parameter tuning plot measured in cross validation accuracy

Cross validation is also an efficient tool to detect overfitting. If the cross validation accuracy stops increasing and starts decreasing, the model is starting to overfit. **Figure 3**, the parameter tuning figure, can also be used to detect overfitting. Since the optimal parameter at the Hyperparameter Tuning section is decided when the cross validation reaches its highest point, our model is not overfitted.

Table 4: Cross validation result

Model	CV1	CV2	CV3	CV4	CV5
Light Gradient Boosting	0.559	0.533	0.528	0.535	0.554
Random Forest	0.569	0.555	0.55	0.536	0.56

Reflection (Errors and Mistakes)

During this competition, the hardest part for me is the data engineering part. The data manipulation requires many statistical intuition. And among all of the variable transformations that I have come up with, not all of them are helpful, and some of them even make the result worse than before. Careful selection needs to be conducted among those transformed variables, and it's really a huge amount of work.

The major mistake I have made is that during my training on the Random Forest model, I happened to get my best result on the public leaderboard. However, since I just regarded this submission as a trial, I didn't set a random seed for the Random Forest model. Therefore, this result cannot be reproduced and it's really a pity.

Predictive Accuracy

Kaggle username: Random

The comparison of my models are shown in **Figure 4**. The accuracy scores on the test dataset as well as the cross validation scores for both algorithms are provided. The testing accuracy is a little bit higher than the cross validation score, possibly due to the fact that the model used to predict the testing accuracy is trained by more data than the cross validation set. And Random Forest does have better performance than Light Gradient Boosting Machine, which accords with the result in the Cross Validation Result section.

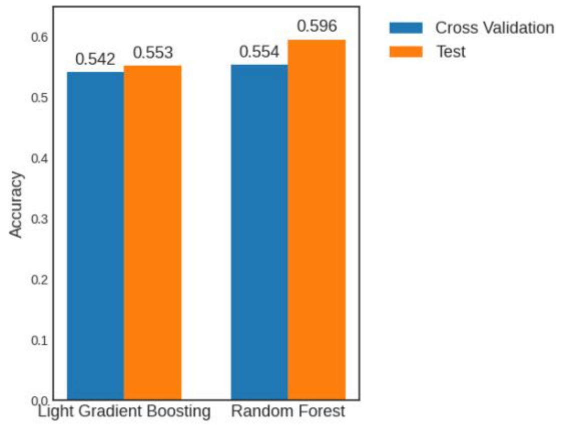


Figure 4: Model results

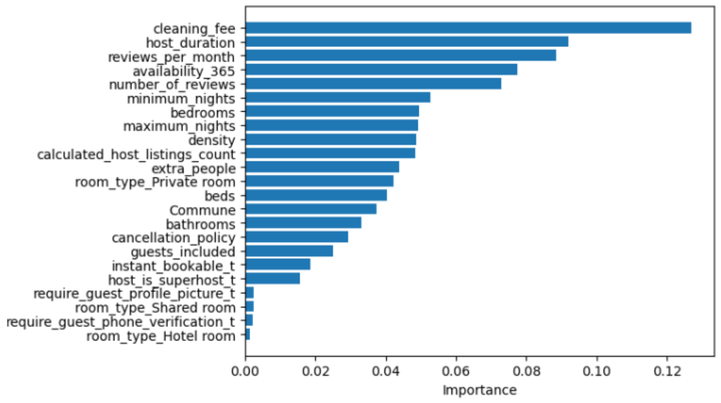


Figure 5: Feature importance plot

Figure 5 is the feature importance plot of Random Forest model. It indicates that cleaning fee is the most important feature to determine Airbnb price. It is reasonable since cleaning fee is part of the additional fee that will directly influence the price. The second most important feature is host duration. This feature represents the experience of the host at Airbnb platform. More sophisticated host should be more qualified to set high price. The number of reviews per month is the third most important variable, which makes sense as well since more reviews means additional visibility. This additional visibility might be highly correlated with more advanced facilities or more attractive homepage of the host. All of these factors can increase the final price.

In conclusion, the Random Forest model has great performance in both the accuracy prediction and plausibility. It suits with the condition of Airbnb price prediction quite well. I hope this model will provide a price reference for both the Airbnb hosts and the tenants.

Reference

[1] https://en.wikipedia.org/wiki/Neighbourhoods_of_Buenos_Aires
[2] https://github.com/pycaret/pycaret
[3] https://scikit-learn.org/stable/
[4] https://lightgbm.readthedocs.io/en/latest/

Code

In [1]:
```
import numpy as np
import pandas as pd
import pandas_profiling as pp
from pycaret.classification import setup, compare_models, plot_model, cr
eate_model, tune_model, predict_model, stack_models, save_model, load_mo
del
from sklearn.metrics import accuracy_score
from plotnine import *
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
from sklearn import model_selection
from tqdm import tqdm

import warnings
warnings.simplefilter('ignore', FutureWarning)
warnings.simplefilter('ignore', UserWarning)
```

Data Cleaning

In [2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
sample = pd.read_csv('sample_submission.csv')
neighbourhood = pd.read_excel('neighbourhood.xlsx')
```

In [3]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9681 entries, 0 to 9680
Data columns (total 25 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   id                                9681 non-null   int64  
 1   neighbourhood                     9681 non-null   object 
 2   room_type                         9681 non-null   object 
 3   minimum_nights                    9681 non-null   int64  
 4   number_of_reviews                 9681 non-null   int64  
 5   last_review                       9681 non-null   object 
 6   reviews_per_month                 9681 non-null   float64
 7   calculated_host_listings_count    9681 non-null   int64  
 8   availability_365                  9681 non-null   int64  
 9   host_since                        9681 non-null   object 
 10  host_is_superhost                 9681 non-null   object 
 11  bathrooms                         9681 non-null   float64
 12  bedrooms                          9681 non-null   int64  
 13  beds                              9681 non-null   int64  
 14  bed_type                          9681 non-null   object 
 15  cleaning_fee                      9681 non-null   int64  
 16  guests_included                   9681 non-null   int64  
 17  extra_people                      9681 non-null   int64  
 18  maximum_nights                    9681 non-null   int64  
 19  instant_bookable                  9681 non-null   object 
 20  is_business_travel_ready          9681 non-null   object 
 21  cancellation_policy               9681 non-null   object 
 22  require_guest_profile_picture     9681 non-null   object 
 23  require_guest_phone_verification  9681 non-null   object 
 24  price                             9681 non-null   int64  
dtypes: float64(2), int64(12), object(11)
memory usage: 1.8+ MB
```

```
In [4]: pp.ProfileReport(train)
```

Overview

Dataset statistics

Number of variables	25
Number of observations	9681
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.8 MiB
Average record size in memory	200.0 B

Variable types

NUM	13
CAT	8
BOOL	4

Reproduction

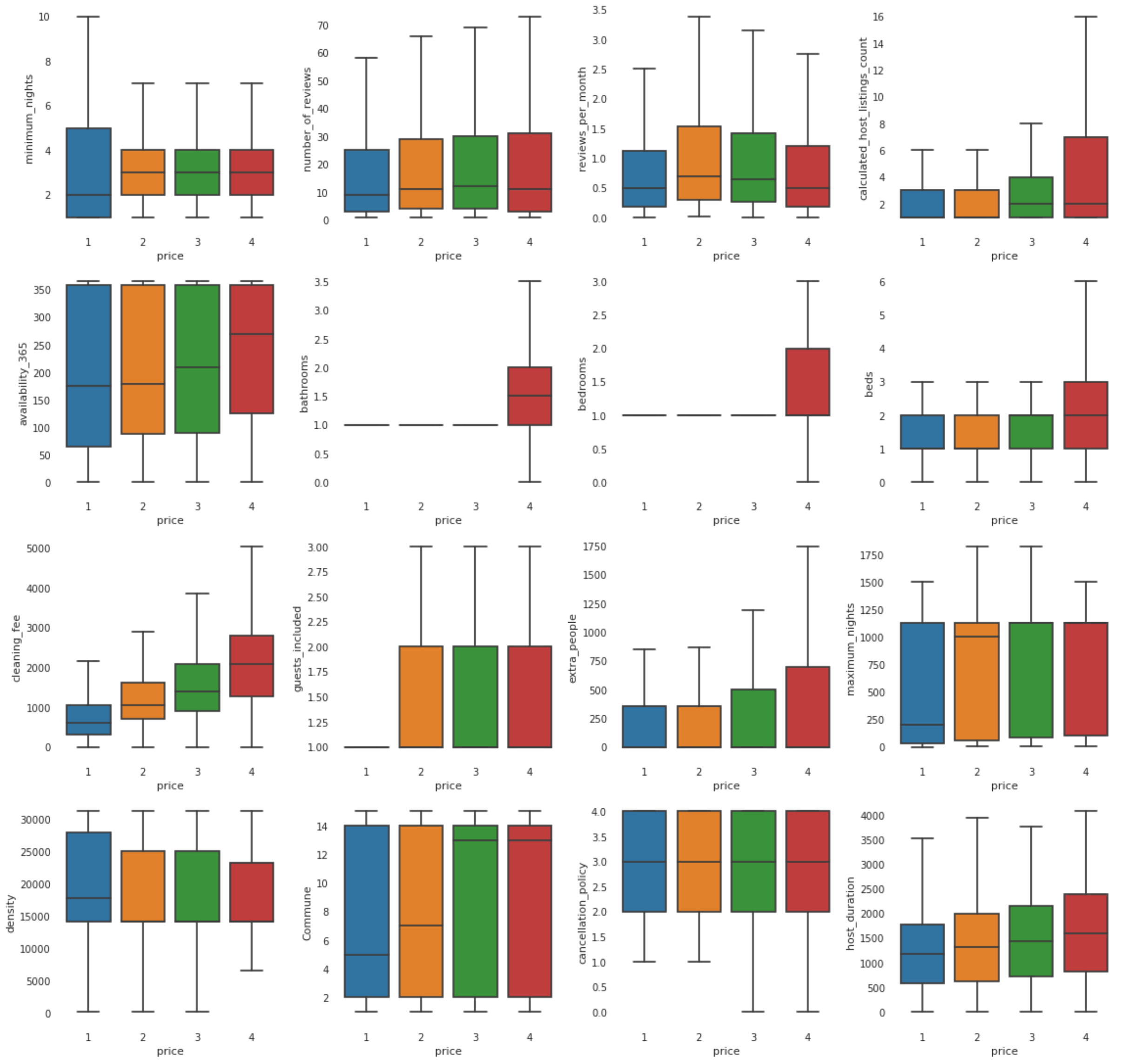
Analysis started	2020-11-19 08:27:42.395695
Analysis finished	2020-11-19 08:28:19.339904

```
Out[4]:
```

In [5]:
```
# id is primary key, cannot use
# neighbourhood has 45 distinct values, can use external data to extract
information
# last_review and host_since are date variable, can combine them to get
 a variable to indicate experience
# is_business_travel_ready contains the same value and is invalid variab
le
# density makes more sense than Population and Area to indicate populari
ty of certain areas
# cancellation_policy can be changed to continuous variable to indicate
 the relaxation level of policy

# train
train['host_duration'] = (pd.to_datetime(train['last_review']) - pd.to_d
atetime(train['host_since'])).dt.days
train = pd.merge(train, neighbourhood, left_on = 'neighbourhood', right_
on = 'Name').drop(['neighbourhood', 'Name'], axis = 1)
train['density'] = train['Population']/train['Area in km2']
train = train.drop(['id', 'is_business_travel_ready', 'last_review', 'ho
st_since', 'Population', 'Area in km2'], axis = 1)
train.replace({'flexible': 4, 'moderate': 3, 'strict_14_with_grace_perio
d': 2, 'super_strict_30': 1,
               'super_strict_60': 0}, inplace = True)

# test
test['host_duration'] = (pd.to_datetime(test['last_review']) - pd.to_dat
etime(test['host_since'])).dt.days
test = pd.merge(test, neighbourhood, left_on = 'neighbourhood', right_on
= 'Name').drop(['neighbourhood', 'Name'], axis = 1)
test['density'] = test['Population']/test['Area in km2']
test = test.drop(['is_business_travel_ready', 'last_review', 'host_sinc
e', 'Population', 'Area in km2'], axis = 1)
test.replace({'flexible': 4, 'moderate': 3, 'strict_14_with_grace_perio
d': 2, 'super_strict_30': 1,
              'super_strict_60': 0}, inplace = True)
```

In [6]:
```
cat = ['room_type', 'host_is_superhost', 'bed_type', 'instant_bookable',
'require_guest_profile_picture',
       'require_guest_phone_verification']
num = ['minimum_nights', 'number_of_reviews', 'reviews_per_month', 'calc
ulated_host_listings_count', 'availability_365',
       'bathrooms', 'bedrooms', 'beds', 'cleaning_fee', 'guests_include
d', 'extra_people', 'maximum_nights', 'density',
       'Commune', 'cancellation_policy', 'host_duration']
```

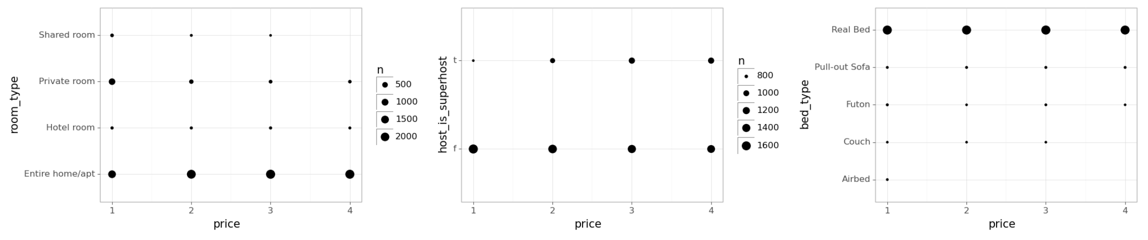
In [7]:
```
fig = plt.figure(figsize = (16,16))
for i in range(len(num)):
    sns.boxplot(data = train, x = 'price', y = num[i], ax = fig.add_subp
lot(4,4,i+1), showfliers = False)
# fig.suptitle('Plots of Continuous Predictors versus Price', fontsize =
16)
plt.tight_layout(rect=[0, 0.03, 1, 0.98])
```



In [8]:
```
for i in cat:
    p = (ggplot(data = train) +
         geom_count(mapping = aes(x = 'price', y = i)) +
         theme_light(base_size = 15))
    ggsave(p, filename = 'test_' + i + '.png')
```

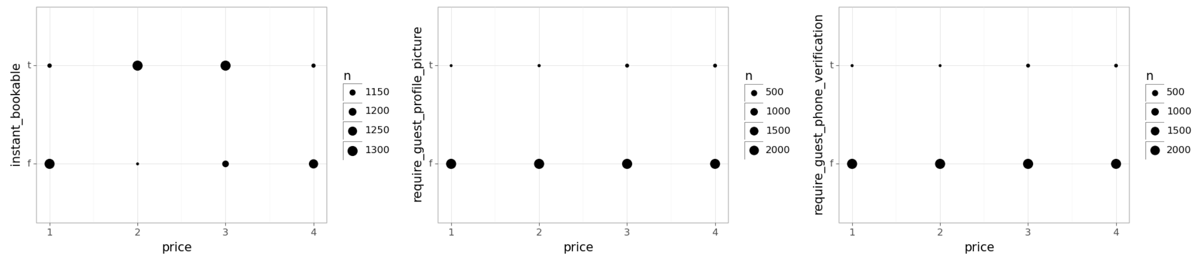
```
In [9]: images = [Image.open(x) for x in ['test_' + i + '.png' for i in cat]]

        widths, heights = zip(*(i.size for i in images))

        sum_width = sum(widths[:3])
        max_height = max(heights[:3])

        new_im = Image.new('RGB', (sum_width, max_height))

        x_offset = 0
        for im in images[:3]:
            new_im.paste(im, (x_offset,0))
            x_offset += im.size[0]
        new_im
```

Out[9]:



```
In [10]: sum_width = sum(widths[3:])
         max_height = max(heights[3:])

         new_im = Image.new('RGB', (sum_width, max_height))

         x_offset = 0
         for im in images[3:]:
             new_im.paste(im, (x_offset,0))
             x_offset += im.size[0]
         new_im
```

Out[10]:



```
In [11]: # check if there is any disparity between training set and test set
         for i in cat:
             print(i, ':', list(set(test[i])) == list(set(train[i])))

room_type : True
host_is_superhost : True
bed_type : False
instant_bookable : True
require_guest_profile_picture : True
require_guest_phone_verification : True
```

```
In [12]: # together with pp figure, should drop bed_type
         # train
         train.drop('bed_type', axis = 1, inplace = True)
         train = pd.get_dummies(train,
                                columns = ['room_type', 'host_is_superhost', 'ins
         tant_bookable', 'require_guest_profile_picture', 'require_guest_phone_ve
         rification'],
                                drop_first = True)

         # test
         test.drop('bed_type', axis = 1, inplace = True)
         test = pd.get_dummies(test,
                               columns = ['room_type', 'host_is_superhost', 'inst
         ant_bookable', 'require_guest_profile_picture', 'require_guest_phone_ver
         ification'],
                               drop_first = True)
```

Model Selection

In [13]:
```
clfs = setup(
    data = train,
    target = 'price',
    silent = True,
    session_id = 1
)
```

```
Setup Succesfully Completed!
```

	Description	Value
0	session_id	1
1	Target Type	Multiclass
2	Label Encoded	None
3	Original Data	(9681, 24)
4	Missing Values	False
5	Numeric Features	18
6	Categorical Features	5
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(9681, 24)
11	Transformed Train Set	(6776, 82)
12	Transformed Test Set	(2905, 82)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	False
16	Normalize Method	None
17	Transformation	False
18	Transformation Method	None
19	PCA	False
20	PCA Method	None
21	PCA Components	None
22	Ignore Low Variance	False
23	Combine Rare Levels	False
24	Rare Level Threshold	None
25	Numeric Binning	False
26	Remove Outliers	False
27	Outliers Threshold	None
28	Remove Multicollinearity	False
29	Multicollinearity Threshold	None
30	Clustering	False
31	Clustering Iteration	None
32	Polynomial Features	False
33	Polynomial Degree	None

	Description	Value
34	Trignometry Features	False
35	Polynomial Threshold	None
36	Group Features	False
37	Feature Selection	False
38	Features Selection Threshold	None
39	Feature Interaction	False
40	Feature Ratio	False
41	Interaction Threshold	None
42	Fix Imbalance	False
43	Fix Imbalance Method	SMOTE

In [14]:
```
best_model = compare_models(sort = 'Accuracy')
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
0	Light Gradient Boosting Machine	0.5415	0.0000	0.5425	0.5397	0.5400	0.3886	0.3888	1.0356
1	CatBoost Classifier	0.5401	0.0000	0.5410	0.5408	0.5399	0.3868	0.3870	13.7450
2	Extreme Gradient Boosting	0.5363	0.0000	0.5373	0.5353	0.5353	0.3817	0.3820	7.6378
3	Gradient Boosting Classifier	0.5242	0.0000	0.5250	0.5246	0.5237	0.3655	0.3658	5.3062
4	Ada Boost Classifier	0.5077	0.0000	0.5088	0.4999	0.5024	0.3436	0.3442	1.6393
5	Random Forest Classifier	0.5072	0.0000	0.5083	0.5048	0.5042	0.3430	0.3439	0.5682
6	Extra Trees Classifier	0.5049	0.0000	0.5059	0.5048	0.5044	0.3398	0.3400	1.3684
7	Ridge Classifier	0.4787	0.0000	0.4797	0.4855	0.4808	0.3049	0.3054	0.0579
8	Linear Discriminant Analysis	0.4752	0.0000	0.4758	0.5011	0.4827	0.3000	0.3019	0.2740
9	Naive Bayes	0.4414	0.0000	0.4431	0.4954	0.4279	0.2557	0.2807	0.0354
10	Decision Tree Classifier	0.4120	0.0000	0.4141	0.4400	0.3960	0.2174	0.2310	0.2692
11	Logistic Regression	0.3995	0.0000	0.4013	0.3678	0.3606	0.2002	0.2086	0.4114
12	K Neighbors Classifier	0.3949	0.0000	0.3960	0.3969	0.3927	0.1934	0.1944	0.2139
13	SVM - Linear Kernel	0.3183	0.0000	0.3182	0.2919	0.2178	0.0903	0.1428	0.7712
14	Quadratic Discriminant Analysis	0.2798	0.0000	0.2789	0.4781	0.1741	0.0384	0.0821	0.2206

Running Time

In [15]:
```
%%time

clf = LGBMClassifier()
pred = clf.fit(train.drop('price', axis = 1), train['price']).predict(te
st.drop('id', axis = 1))
```

```
CPU times: user 1.48 s, sys: 32 ms, total: 1.52 s
Wall time: 1.51 s
```

In [16]:
```
%%time

clf = RandomForestClassifier()
pred = clf.fit(train.drop('price', axis = 1), train['price']).predict(te
st.drop('id', axis = 1))
```

```
CPU times: user 1.55 s, sys: 16 ms, total: 1.57 s
Wall time: 1.57 s
```

Hyperparameter Tuning

Light Gradient Boosting Machine

In [17]:
```
kfold = model_selection.KFold(
    n_splits = 5,
    shuffle = True,
    random_state = 10
)
store1 = np.zeros((5, 8))
p1s = [0.001, 0.01, 0.1, 1, 10]
p2s = [50, 100, 150, 200, 250, 300, 350, 400]
with tqdm(total=40) as pbar:
    for i in range(len(p1s)):
        for j in range(len(p2s)):
            clf = LGBMClassifier(n_jobs = -1, random_state = 10, learnin
g_rate = p1s[i], n_estimators = p2s[j])
            s = model_selection.cross_val_score(
                clf, train.drop('price', axis = 1), train['price'], scor
ing='accuracy', cv=kfold,
            )
            store1[i,j] = np.mean(s)
            pbar.update(1)
```

```
100%|██████████| 40/40 [07:40<00:00, 11.51s/it]
```

In [18]:
```
print('Best learning rate:', float(np.array(p1s)[np.where(store1 == stor
e1.max())[0]]))
print('Best number of trees:', int(np.array(p2s)[np.where(store1 == stor
e1.max())[1]]))
```

```
Best learning rate: 0.1
Best number of trees: 400
```

Random Forest

In [19]:
```
store2 = np.zeros((5, 8))
p1s2 = [0.1, 'log2', 'sqrt', 0.3, 0.5]
p2s2 = [50, 100, 150, 200, 250, 300, 350, 400]
with tqdm(total=40) as pbar:
    for i in range(len(p1s2)):
        for j in range(len(p2s2)):
            clf = RandomForestClassifier(n_jobs = -1, random_state = 10,
max_features = p1s2[i], n_estimators = p2s2[j])
            s = model_selection.cross_val_score(
                clf, train.drop('price', axis = 1), train['price'], scor
ing='accuracy', cv=kfold,
            )
            store2[i,j] = np.mean(s)
            pbar.update(1)
```

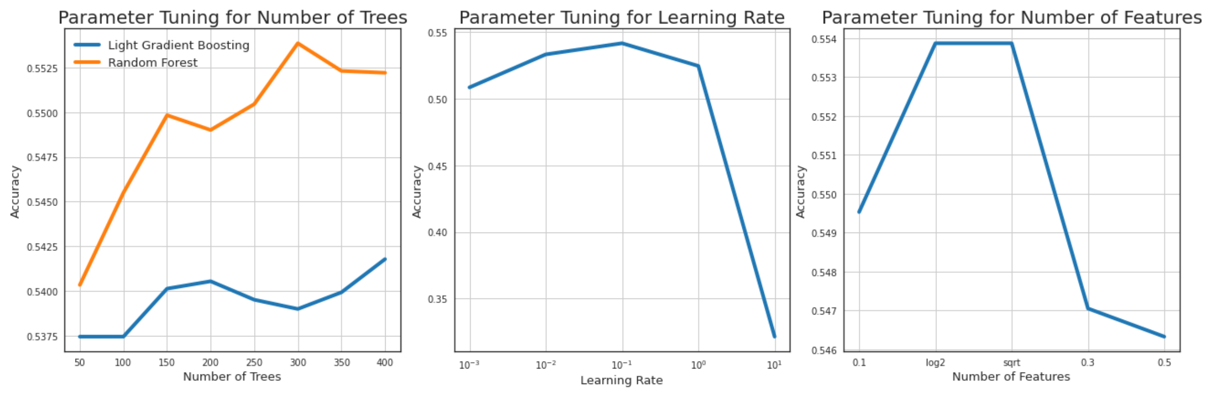
```
100%|██████████| 40/40 [03:01<00:00,  4.54s/it]
```

In [20]:
```
index = np.array(np.where(store2 == store2.max()))[:,0]
print('Best number of features:', np.array(p1s2)[index[0]])
print('Best number of trees:', int(np.array(p2s2)[index[1]]))
```

```
Best number of features: log2
Best number of trees: 300
```

Figure

```
In [21]: with plt.style.context("seaborn-whitegrid"):
             plt.rcParams["axes.edgecolor"] = "0.15"
             plt.rcParams["axes.linewidth"]  = 1.25

             fig, axs = plt.subplots(1, 3, figsize=(18, 6))
             axs = axs.ravel()

             accuracy11 = store1[np.where(store1 == store1.max())[0]]
             accuracy12 = store2[index[0]]
             accuracy2 = store1[:,np.where(store1 == store1.max())[1]]
             accuracy3 = store2[:,index[1]]

             axs[0].plot(p2s, accuracy11.reshape(-1), '-', linewidth=4.0, label=
         'Light Gradient Boosting')
             axs[0].plot(p2s2, accuracy12, '-', linewidth=4.0, label='Random Fore
         st')
             axs[0].set_title('Parameter Tuning for Number of Trees', size = 20)
             axs[0].set_xlabel('Number of Trees', size = 13)
             axs[0].set_ylabel('Accuracy', size = 13)
             axs[0].legend(loc='best', prop={'size':13})

             axs[1].set_xscale('log')
             axs[1].plot(p1s, accuracy2.reshape(-1), '-', linewidth=4.0, )
             axs[1].set_title('Parameter Tuning for Learning Rate', size = 20)
             axs[1].set_xlabel('Learning Rate', size = 13)
             axs[1].set_ylabel('Accuracy', size = 13)

             axs[2].plot(p1s2, accuracy3.reshape(-1), '-', linewidth=4.0, )
             axs[2].set_title('Parameter Tuning for Number of Features', size = 2
         0)
             axs[2].set_xlabel('Number of Features', size = 13)
             axs[2].set_ylabel('Accuracy', size = 13)

             plt.tight_layout()
```



Cross Validation

In [22]:
```
clf_lgb = LGBMClassifier(n_jobs = -1, random_state = 10, learning_rate = 0.1, n_estimators = 400)
clf_rf = RandomForestClassifier(n_jobs = -1, random_state = 10, max_features = 'log2', n_estimators = 300)
kfold = model_selection.KFold(
    n_splits = 5,
    shuffle = True,
    random_state = 10
)
s_lgb = model_selection.cross_val_score(
    clf_lgb, train.drop('price', axis = 1), train['price'], scoring='accuracy', cv=kfold)
s_rf = model_selection.cross_val_score(
    clf_rf, train.drop('price', axis = 1), train['price'], scoring='accuracy', cv=kfold)
```

In [23]:
```
np.round(s_lgb, 3)
```

Out[23]:
```
array([0.559, 0.533, 0.528, 0.535, 0.554])
```

In [24]:
```
np.round(s_rf, 3)
```

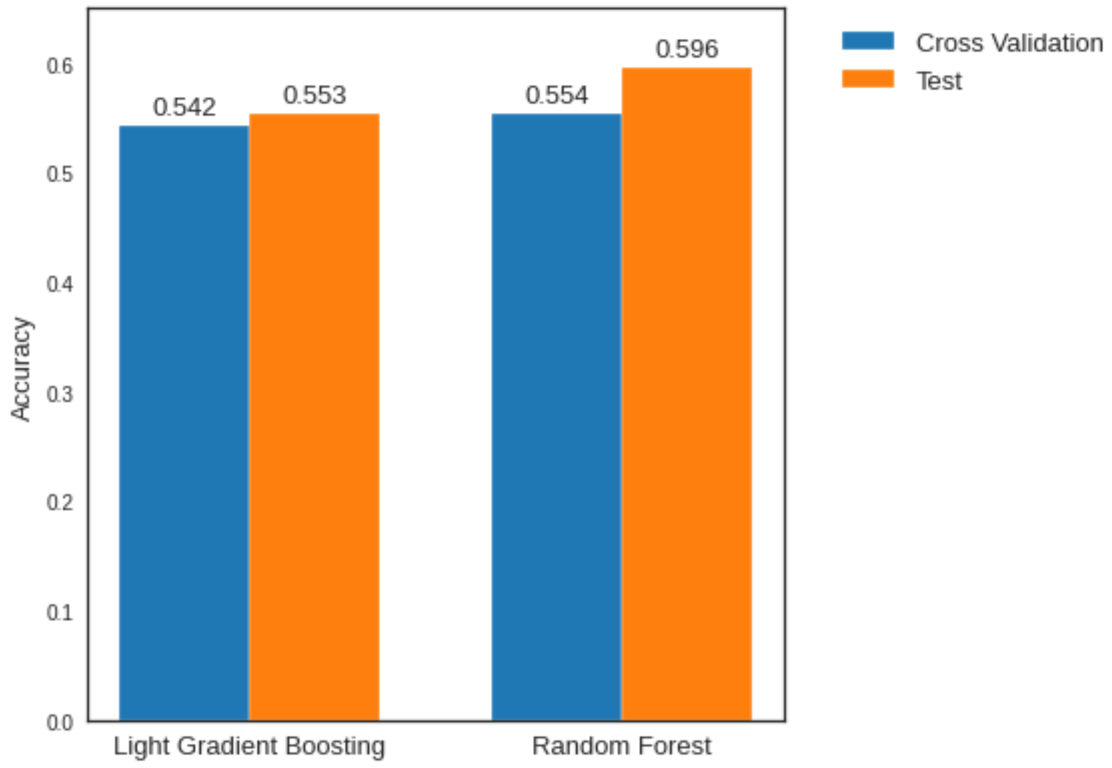
Out[24]:
```
array([0.569, 0.555, 0.55 , 0.536, 0.56 ])
```

Submission

In [25]:
```
clf_lgb = LGBMClassifier(n_jobs = -1, learning_rate = 0.1, n_estimators = 400)
clf_rf = RandomForestClassifier(n_jobs = -1, max_features = 'log2', n_estimators = 300)
pred_lgb = clf_lgb.fit(train.drop('price', axis = 1), train['price']).predict(test.drop('id', axis = 1))
pred_rf = clf_rf.fit(train.drop('price', axis = 1), train['price']).predict(test.drop('id', axis = 1))
```

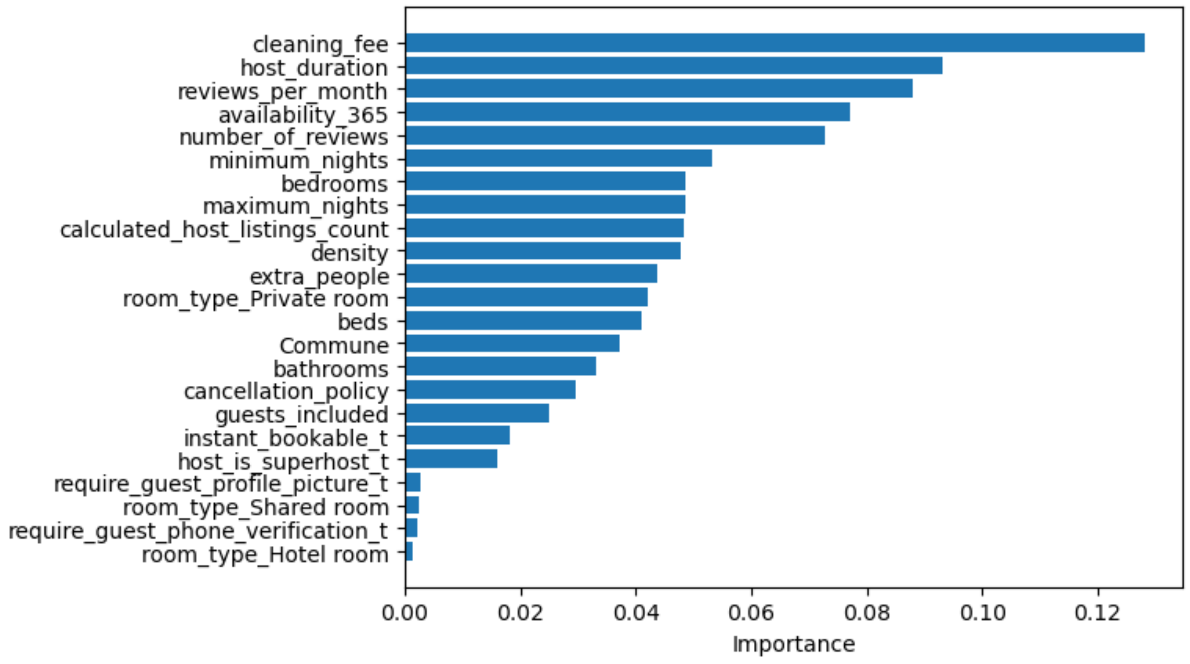
In [26]:
```
index = [test['id'].to_list().index(i) for i in sample['id']]
sample['price'] = pred_lgb[index]
sample.to_csv('lgb_submission.csv', index = False)

sample['price'] = pred_rf[index]
sample.to_csv('rf_submission.csv', index = False)
```

In [27]:
```
# reference: https://matplotlib.org/3.1.1/gallery/lines_bars_and_marker
s/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py
labels = ['Light Gradient Boosting', 'Random Forest']
cv_means = np.round([np.mean(s_lgb), np.mean(s_rf)], 3)
test_means = [0.553, 0.596]

x = np.arange(len(labels))
width = 0.35

with plt.style.context("seaborn-white"):
    plt.rcParams["axes.edgecolor"] = "0.15"
    plt.rcParams["axes.linewidth"]  = 1.25

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, cv_means, width, label='Cross Validatio
n')
    rects2 = ax.bar(x + width/2, test_means, width, label='Test')

    ax.set_ylabel('Accuracy', size = 13)
    ax.set_xticks(x)
    ax.set_ylim([0,0.65])
    ax.set_xticklabels(labels, size = 13)
    ax.legend(bbox_to_anchor=(1.05, 1), prop={'size':13})


    def autolabel(rects):
        """Attach a text label above each bar in *rects*, displaying its
height."""
        for rect in rects:
            height = rect.get_height()
            ax.annotate('{}'.format(height),
                        xy=(rect.get_x() + rect.get_width() / 2, height
),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom',
                        size = 13)

    autolabel(rects1)
    autolabel(rects2)
    fig.tight_layout()

    plt.show()
```



Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.542
0.553
0.554
0.596
Light Gradient Boosting
Random Forest
Cross Validation
Test

In [28]:
```
# reference: https://matplotlib.org/3.1.1/gallery/lines_bars_and_marker
s/barh.html
df_importance = (
    pd.DataFrame(dict(feature = train.drop('price', axis = 1).columns,
                      importance = clf_rf.feature_importances_)).
    sort_values('importance', ascending = False).
    reset_index(drop = True)
)

plt.rcdefaults()
fig, ax = plt.subplots()

# Example data
people = df_importance['feature']
y_pos = np.arange(len(people))
performance = df_importance['importance']

ax.barh(y_pos, performance)
ax.set_yticks(y_pos)
ax.set_yticklabels(people)
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('Importance')
ax.set_title('')

plt.show()
```


cleaning_fee
host_duration
reviews_per_month
availability_365
number_of_reviews
minimum_nights
bedrooms
maximum_nights
calculated_host_listings_count
density
extra_people
room_type_Private room
beds
Commune
bathrooms
cancellation_policy
guests_included
instant_bookable_t
host_is_superhost_t
require_guest_profile_picture_t
room_type_Shared room
require_guest_phone_verification_t
room_type_Hotel room
Importance