Kaggle Project: Airbnb Housing Price Prediction

Gongjinghao Cheng

1 Exploratory Analysis

In this project, we are provided with a dataset of the Airbnb listing places in Buenos Aires. We are going to predict the price of Airbnb with difference features of the places. We are given a training set for model development and a test set for model evaluation. The training set consists of 9681 samples and in total 25 features (including the target variable 'price'). For the 24 predictive variables, there are 9 categorical variables, 13 numerical variables and 2 date variables. Firstly, I observe that the frequencies of target classes are even in the training set, which means that our dataset is balanced. Therefore techniques such as oversampling or stratified sampling for data balance are not reserved.

With inspection of the categorical variables, each of them contains less than 5 categories except variable 'neighbourhood' which has 45 difference locations. However, most of them have low frequencies and I group locations with frequencies lower than 100 together into one class 'other', for purpose of better visualization.

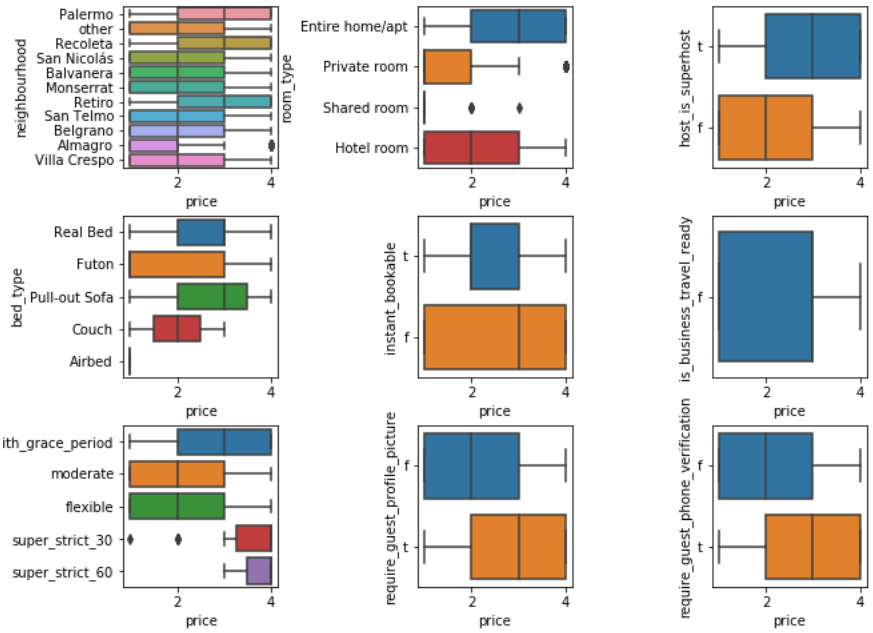


Figure 1: boxplots of categorical features

The boxplots in Figure 1 illustrate the distribution of target variable 'price' in classes of each categorical variable. It turns out that the target variables have reasonably distinguishable distributions

w.r.t. most categorical variables, except variable 'is business travel ready' which only contains False class. By any mean, this variable will not be useful for my modeling and I will delete it during data pre-processing.

As for the rest of the variables, I will first convert the date variables into numerical variables by imputing their day differences from 11/18/2020 (I also tried to convert the date into categorical variables of month and year, but the results are less preferred because my models are more adaptive to numerical variables).

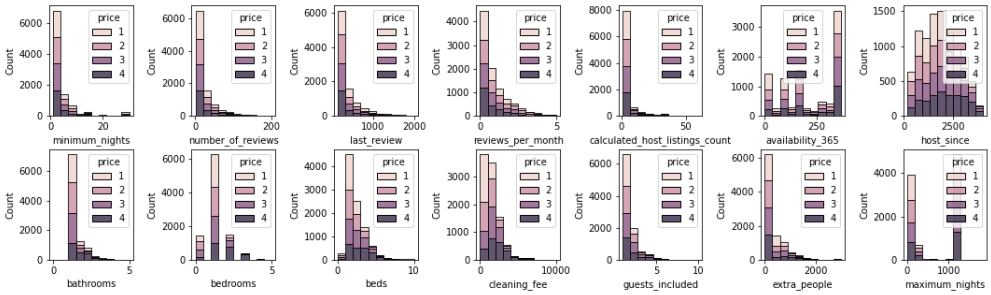


Figure 2: boxplots of numerical features

With the transformed date variables, there are in total 15 numerical variables; After inspection, the variable 'id' is just an index for the housing and it should not be helpful for modeling intuitively. Hence I will not include it in my training set. Figure 2 displays the distribution of 'price' classes w.r.t. each numerical variables (I have prune the displayed data by deleting small amount of outliers for better visualization). Based on the plots, most of the histograms from most of the plots have similar shares for each target class, which means that with the corresponding feature solely, the frequencies of each target class are very similar; Thus, it implies difficulty for the classification task in terms of distribution similarity of target classes.

In order to understand the importance of the variables, I also fit a default random forest model to the data and extract the feature importance. Table 1 illustrates the top 5 most frequently selected features in splits of random forest. Accordingly, 'cleaning fee' is the most influential feature; If we associate it with the corresponding plot in Figure 2, we can observe that frequencies of target classes are relatively uneven (different) in the 'cleaning fee' histogram.

Feature	Importance
cleaning fee	0.1334
reviews per month	0.0892
availability 365	0.0803
number of reviews	0.0747
neighbourhood	0.0660

Table 1: Top 5 most important features

2 Models

2.1 Baseline: Logistic Regression

As of a classification problem, a multi-class logistic regression model will be fitted as a benchmark for model performance. The motivations are various: Logistic regression is intuitively used for classification; It is simple and easy for training comparing to other ensemble or boosting methods; It is computationally efficient and converges in relatively fewer iterations; Logistic regression also has good interpretability since its coefficients quantify the effects from each feature upon the results.

Logistic regression model is available in the scikit-learn package for easy implementation. scikit-learn is a popular Python package for data science. It contains a comprehensive amount of techniques and

modules for various date related works. It has a coded logistic regression authored by Gael Varoquaux and others [3].

2.2 Random Forest

Tree-based methods are widely used for classification purpose, and intuitively random forest as an ensemble algorithm of trees will probably work in this case. There are several reasons for using random forest: Random forest works with both categorical and numerical variables; There is no need for feature scaling (data normalization); It is often robust with outliers; It handles with non-linear relationships. More importantly, random forest will reduce the variance of model prediction with its ensemble nature and it is resisting overfitting problems, which further improves model accuracy. In addition, random forest provides importance of features to make model interpretable.

Random forest is also available in scikit-learn package. The source codes are written by Gilles Louppe and others [3].

2.3 XGBoost

XGboost is one of the most popular boosting algorithms used in Kaggle competitions. It is an enhanced version of GBM (Gradient Boosting Machine). There are advantages of XGBoost. Firstly, XGBoost is a flexible algorithm that is able to improve training accuracy efficiently. It is a regularized boosting technique and the user can reduce overfitting by control the complexity of trees with regularization parameters. Unlike GBM, XGBoost is less time consuming because of parallel processing. We know that ordianry boosting technique is a sequential process, and current model is built upon results of previous one, and this leads to denial of parallel structure; However, XGBoost applies an approximate greedy algorithm for gradient search and yields weighted quantile sketch algorithm to establish parallel processing during each tree building period. As a result, XGBoost is highly efficient and less time consuming. Besides, XGBoost also offers feature importance for interpretability.

XGBClassifier has been implemented in xgboost package. It is compatible to many of the data mining, feature engineering and model evaluation modules in scikit-learn, which provides convenience for application. The source codes are created by Tianqi Chen and others [2].

2.4 AdaBoost

AdaBoost is another type of boosting algorithm which accumulates models built upon data with updated weights. It has advantages such as robustness to overfitting, efficient implementation. One thing I want to mention in particular is that it has minimum amount of parameters for tuning comparing to other algorithms. This provides convenience for model development.

AdaBoost has its module in scikit-learn package, authored by Noel Dawe and others [3].

2.5 CatBoost

CatBoost is a relatively new boosting technique published by Liudmila Prokhorenkova and others in 2017 [4]. It provides state-of-art results and outperforms other algorithms. Besides efficiency and implementation easiness, CatBoost is known for dealing with categorical variables automatically without explicit data pre-processing. More specifically, it transforms categorical variables into numerical features with target encoder [1] based on various statistics of all the variables. This not only shrinks time needed for pre-processing, but also potentially exceeds simple factorization of categorical variables as I did in the other models with LabelEncoder.

CatBoostClassifier is already implemented in CatBoost package by the team of Liudmila Prokhorenkova in [4] and training this model is convenient.

2.6 Vote Classifier

This last classifier is an ensemble of previously models. This is a commonly used technique to reduce individual model bias and variance, further achieving a small amount of improvement overall. The

vote classifier can predict based on both probabilities and exact predictions from consisted models. Here I will apply it by including best performed models.

3 Training

3.1 Logistic Regression

To train a logitstic regression model with L2 regularization, scikit-learn uses Broyden–Fletcher–Goldfarb–Shanno algorithm for $min_{w,c}\frac{1}{2}w^T w + C\sum_i^n log(exp(-y_i(X_i^T w + c)) + 1)$. This algorithm belongs to quasi-Newton methods. In general, the algorithm initializes with a guess x_0 and approximate Hessian B_0. It will then compute the derivative and solve for direction of new step; After line search of optimal stepsize, it will update x, y, and Hessian B based on the values from previous iterations. Optimally the algorithm will end once it converges to the minimum.

3.2 Random Forest

Random Forest is basically an ensemble model of decision trees. To fit the model, random forest will generate a large number of trees that grows and be pruned based on the boostrapped data. The user has various parameters to control the tree growth, including number_of_trees, split_criterion, max_depth and etc. A collection of trees are gathered into one ensemble model of random forest in terms of averaging probabilistic predictions.

3.3 XGBoost

XGBoost is an additive model that sums up weak classifiers to minimize the loss function. It uses a gradient tree boosting framework. After each iteration, XGBoost will search for the direction (classifier) that reduces loss the most; Eventually it will achieve an ensemble that makes small training loss. The key factor that makes XGBoost efficient is the parallel processing. According to Tianqi [2], a common tree boosting algorithm will need to use the whole dataset collectively for split finding; XGBoost uses an approximate greedy algorithm for split finding and take advantages of weighted quantile Sketch. In this way, data are divided into different column blocks and split finding can be parallelized for each column blocks.

3.4 AdaBoost

AdaBoost is another ensemble model that collects a sequence of weak learners on repeatedly modified versions of data. It initializes the data with equal weights and uses weak learner algorithm to fit a weak classifier. Then AdaBoost will decrease the weights of correctly classified sample and increase the weights of samples which are predicted incorrectly. The idea is to rise the importance of incorrect predictions. The above steps will repeat until convergence. Adding up all the weak classifier along iterations will lead to the AdaBoost model.

3.5 CatBoost

CatBoost is also based on gradient boosted decision trees. Likewise, a collection of decision trees will be built sequentially during training. For building each tree, CatBoost will first do a preliminary calculation of splits to count the possible ways to split data, it will then transform categorical features into numerical features; Next, the tree structure will be chosen with feature candidates from previous steps by a greedy algorithm. The new tree is selected according to the improving direction of certain metric function.

3.6 Runtime

Model	Runtime (seconds)
Logistic Regression	17.9923 $\pm$0.3496
Random Forest	7.896 $\pm$0.0556
XGBoost	10.7640 $\pm$0.5541
AdaBoost	29.2191 $\pm$0.1079
CatBoost	5.9157 $\pm$0.3559

Table 2: Runtimes for optimal models

Table 2 displays the runtimes for each of the models with tuned parameters. While this in some sense illustrates the efficiency of algorithms, we need to take account of the number of iterations and number of estimators.

4 Hyperparameter Selection

For all the algorithms I used, the hyperparameter selection are determined based on a 5-fold stratified cross validation. The tuning steps are as following:

1. Decide parameters for tuning.
2. Perform a gridsearch based on the mean of cross validation scores.
3. With the optimal parameters from gridsearch, perform greedy 1D search for all parameters sequentially within small range around the current optimal.

Below is the table of parameters tuned for each model.

Model	Paramters
Logistic Regression	solver,C
Random Forest	ccp_alpha,criterion,max_depth,max_features
XGBoost	max_depth,learning_rate,gamma,colsample_bytree
AdaBoost	base_estimator,learning_rate
CatBoost	one_hot_max_size,depth,l2_leaf_reg

Table 3: Parameter Tuned

Due to page limits, I will only present the sequential tuning for XGBoost as an example (Step 3). The tuning process for other models will be similar.

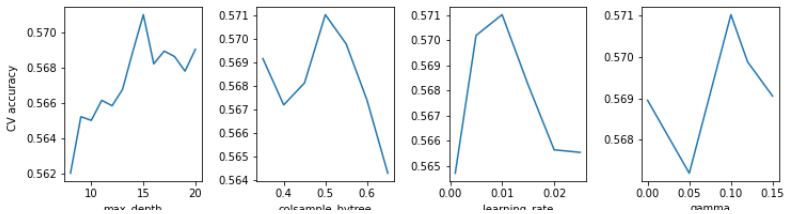


Figure 3: Greedy Tuning

As shown in Figure 3, the optimal parameters are selected to be at least close to local maximum with others fixed. After locating potential ranges of optimal parameters with gridsearch, the greedy search method within the ranges will lead to an approximation of best estimators locally.

5 Data Split

For data split scheme, I first use a stratified train test split to get balanced training set and validation set (size ratio 9:1). Then, for the training set, I apply a 5-fold cross validation for either hyperparameter

tuning or model evaluation. The tuned parameters will then be used for fitting the whole training set and be evaluated on validation set. In order to examine overfitting, for both model evaluation on validation set and along cross validation folds, training and testing scores will be recorded.

6 Errors and Mistakes

Admittedly, there are some unsolved puzzles in this project. For the technical part, CatBoost is not compatible to some evaluation functions of scikit-learn, which means that I am not able to utilize established modules easily. In addition, XGBoost will not do parallel process if I set up parallel structure for cross validation score functions in scikit-learn.

As for the modeling part, it is evident that the training accuracy is significantly larger than the test accuracy in random forest and XGBoost. This means that there exists considerable overfitting. However, after I adjust the regularization parameters and force the training accuracy to decrease, the test accuracy shows no sign of improvement but even gets worse. A potential answer to this is that the whole training set contains a set of samples that are not frequent enough to generate meaningful features. After data split, this set split evenly into the training and validation set. While the model with small regularization is forced to recognize such data in training set, those in validation set will be consistently unrecognizable. The solution for this will be a larger dataset.

7 Predictive Accuracy

My Kaggle username is Kim2.

7.1 Experimental results

This is the model evaluation part that shows the cross validation scores for each of the model with different data pre-processing.

Model	Data pre-processing	optimal model CV accuracy
Logistic Regression	Date to category	0.495
	Date to numeric	
Random Forest	Date to category	0.559
	Date to numeric	0.556
XGBoost	Date to category	0.563
	Date to numeric	**0.571**
AdaBoost	Date to category	0.540
	Date to numeric	0.538
CatBoost	Date to category	0.554
	Date to numeric	0.549
Ensemble	Date to numeric	**0.572**

Due to page limits, I restrain my table to a small amount of results. It is evident that XGBoost outperforms other algorithms; Ensemble several algorithms will also improve model performance.

8 Conclusion

Overall, for this specific classification task, XGBoost outperforms other 4 methods; Random Forest and CatBoost are slightly worse; The next best will be AdaBoost; Logistic Regression is the least accurate. Ensemble model consists of top models will perform better, even though it will be time consuming.

For direction of further investigation, a large dataset will be preferred since it will potentially provide solution for the modeling obstacle I mentioned in section 6. Improvements is also possible with neural networks, which I did not use for now.

References

[1] Patricio Cerda and Gaël Varoquaux. "Encoding high-cardinality string categorical variables". In: (2019). DOI: 10.1109/TKDE.2020.2992529. eprint: arXiv:1907.01860.

[2] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Liudmila Prokhorenkova et al. *CatBoost: unbiased boosting with categorical features*. 2017. eprint: arXiv:1706.09516.

In [1]:

```
# !pip install mord
# !pip install xgboost
# !pip install catboost

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, StratifiedKFold, KFold, cross_val
_score, GridSearchCV, ParameterGrid, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, mean_absolute_error, make_scorer, f1_score
from sklearn.ensemble import RandomForestClassifier, VotingClassifier, AdaBoostClassifi
er
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, Normalizer, MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier, OutputCodeClassifier
from sklearn.datasets import make_classification
from sklearn.tree import export_graphviz
from sklearn import metrics

from mord import LogisticAT

import xgboost as xgb
from xgboost import XGBClassifier

from catboost import CatBoostClassifier

from datetime import datetime

import time

%matplotlib inline
# pd.set_option('display.max_columns', None)
# pd.set_option('display.max_rows', None)
```

1. Import Data

In [2]:

```
# import data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

test_id = test['id']
```

2. EDA

In [3]:

```
train.head()
```

Out[3]:

	id	neighbourhood	room_type	minimum_nights	number_of_reviews	last_review	reviev
0	727	Palermo	Entire home/apt	1	170	5/12/20	
1	6274	Colegiales	Private room	1	11	7/1/19	
2	6025	Recoleta	Entire home/apt	3	2	11/23/19	
3	8931	Recoleta	Entire home/apt	2	1	11/1/19	
4	7524	San Nicolás	Entire home/apt	2	31	12/26/19	

5 rows × 25 columns

In [4]:

```
train.shape
```

Out[4]:

```
(9681, 25)
```

In [5]:

```
train.columns
```

Out[5]:

```
Index(['id', 'neighbourhood', 'room_type', 'minimum_nights',
       'number_of_reviews', 'last_review', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'host_since',
       'host_is_superhost', 'bathrooms', 'bedrooms', 'beds', 'bed_type',
       'cleaning_fee', 'guests_included', 'extra_people', 'maximum_night
s',
       'instant_bookable', 'is_business_travel_ready', 'cancellation_polic
y',
       'require_guest_profile_picture', 'require_guest_phone_verificatio
n',
       'price'],
      dtype='object')
```

The training data contains 9681 samples and in total 25 variables. There are 24 predictors and 1 response variable, 'price'. Note that there are features such as 'id', which intuitively is not associate with the price, therefore we will discard this feature.

In [6]:

```
factor_feature = train.columns[train.dtypes==object]
factor_feature
```

Out[6]:

```
Index(['neighbourhood', 'room_type', 'last_review', 'host_since',
       'host_is_superhost', 'bed_type', 'instant_bookable',
       'is_business_travel_ready', 'cancellation_policy',
       'require_guest_profile_picture', 'require_guest_phone_verificatio
n'],
      dtype='object')
```

In [7]:

```
for factor in factor_feature:
    print('{} variables:{}'.format(factor, len(train[factor].unique())))
```

```
neighbourhood variables:45
room_type variables:4
last_review variables:1190
host_since variables:2762
host_is_superhost variables:2
bed_type variables:5
instant_bookable variables:2
is_business_travel_ready variables:1
cancellation_policy variables:5
require_guest_profile_picture variables:2
require_guest_phone_verification variables:2
```

Here are the categorical features, note that features 'neighbourhood', 'last_review', 'host_since' have huge amount of categories, we might need to fix them by clustering or turning to numeric variables.

In [8]:

```
quant_feature = train.columns[train.dtypes!=object]
quant_feature
```

Out[8]:

```
Index(['id', 'minimum_nights', 'number_of_reviews', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'bathrooms',
       'bedrooms', 'beds', 'cleaning_fee', 'guests_included', 'extra_peopl
e',
       'maximum_nights', 'price'],
      dtype='object')
```

In [9]:

```
train[quant_feature].describe()
```

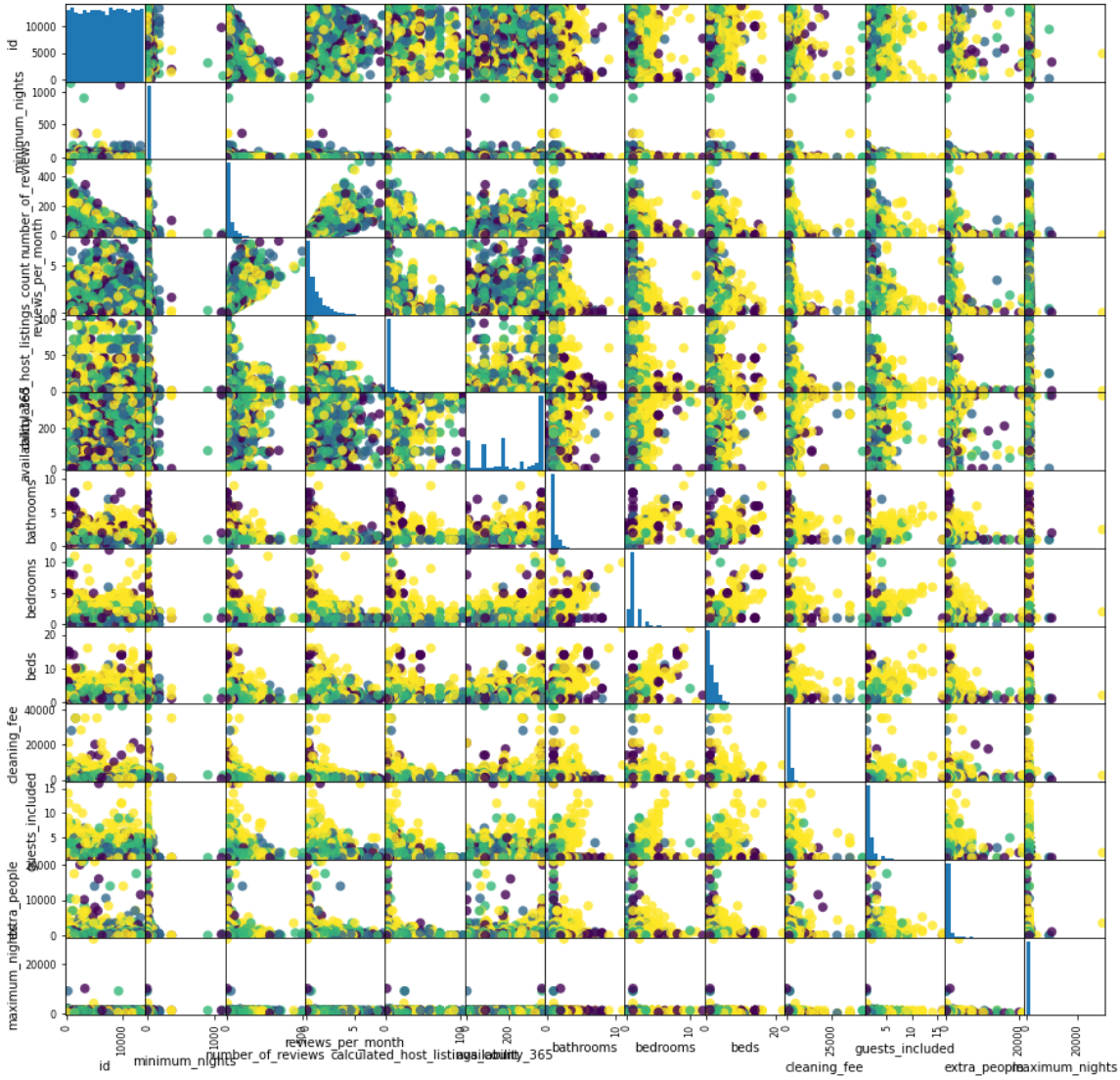
Out[9]:

	id	minimum_nights	number_of_reviews	reviews_per_month	calculated_hos
count	9681.000000	9681.000000	9681.000000	9681.000000	
mean	6935.858796	5.283235	23.959715	0.940867	
std	3997.261605	21.897973	36.662277	1.003831	
min	0.000000	1.000000	1.000000	0.010000	
25%	3487.000000	2.000000	3.000000	0.230000	
50%	6950.000000	3.000000	11.000000	0.580000	
75%	10386.000000	4.000000	28.000000	1.310000	
max	13829.000000	1124.000000	500.000000	7.870000	

The above are numeric variables, and there are bounded differently, thus we might need to normalize the data.

In [22]:

```
pd.plotting.scatter_matrix(train.iloc[:,:-1], c=train.iloc[:,-1], figsize=(15, 15), mar
ker='o',
                           hist_kwds={'bins': 20}, s=60, alpha=.8)
plt.show()
```


id
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

Take a look at the distribution of price in each features. It seems that in most classes, the different price classes are miexd and hard to be seperated. Some of the features are better than others for classification.

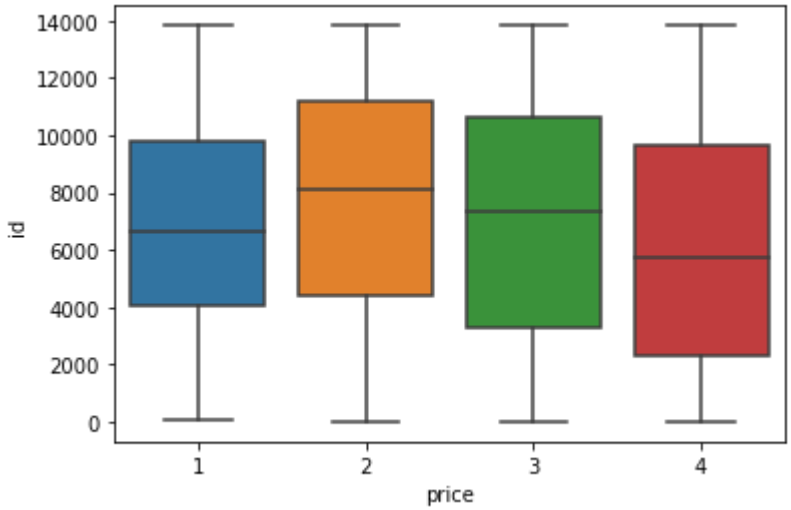
In [11]:

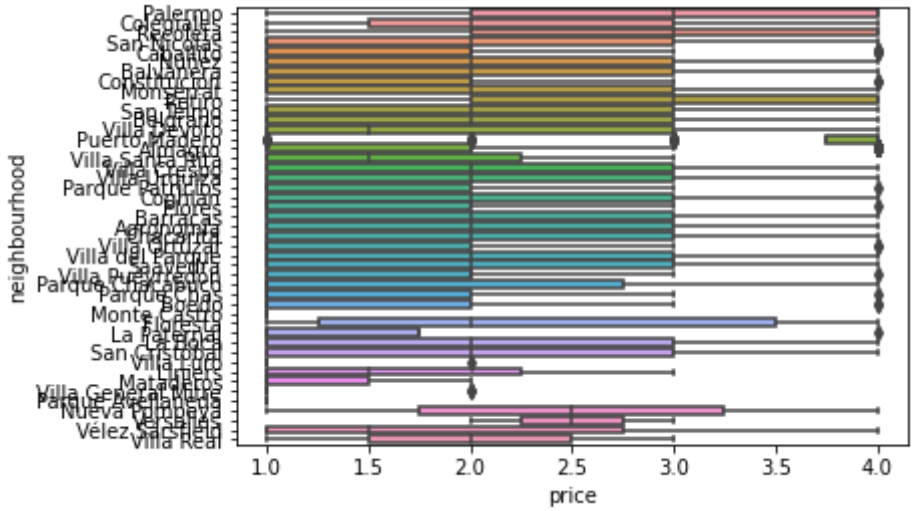
```
plt.hist(train.price)
plt.title('freq of 4 price classes')
plt.show()
```

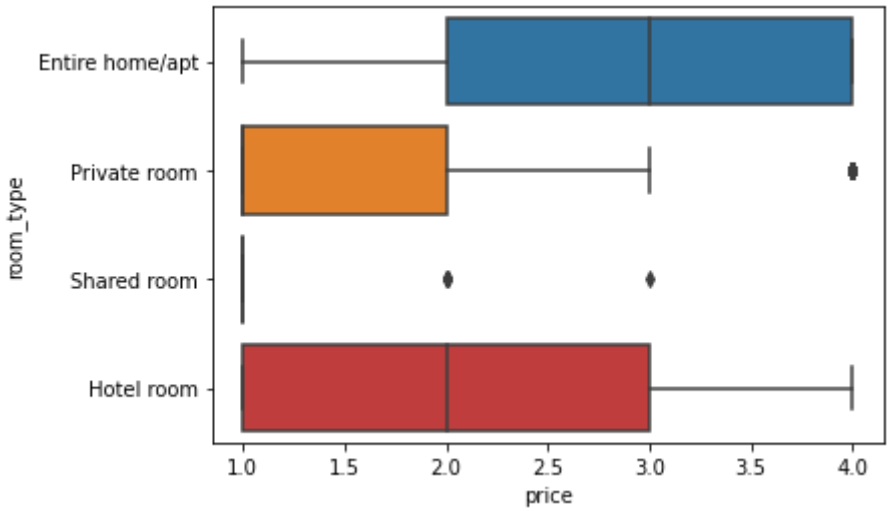
Roughly there are same amount of price classes in training set; It might be fine to use a normal K-fold CV instead of a stratified one.

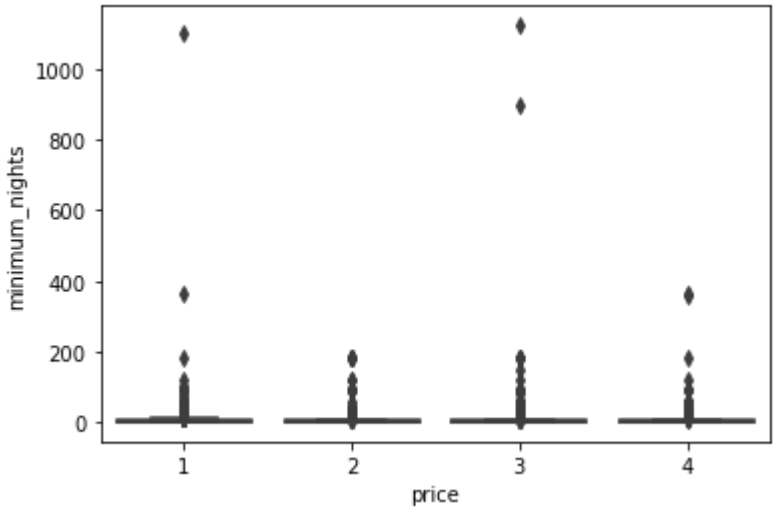
In [11]:

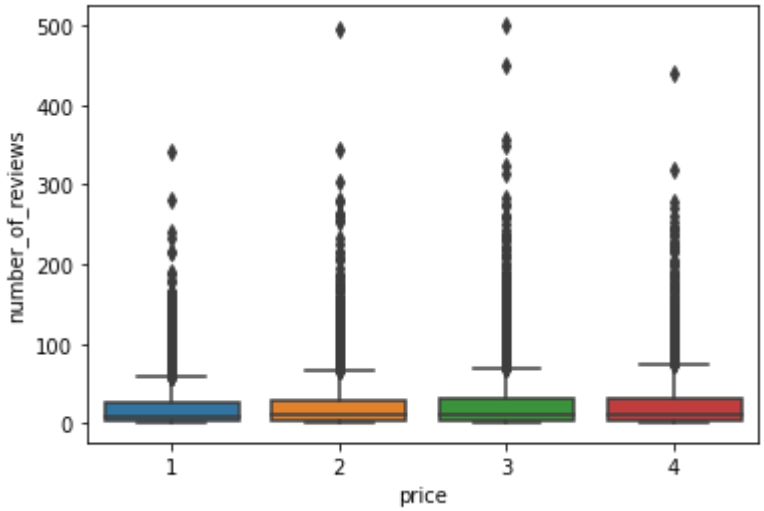
```
drop = ['last_review', 'host_since', 'price']
for col in [c for c in train.columns if c not in drop]:
    sns.boxplot(x=train['price'], y=train[col])
    plt.show()
```

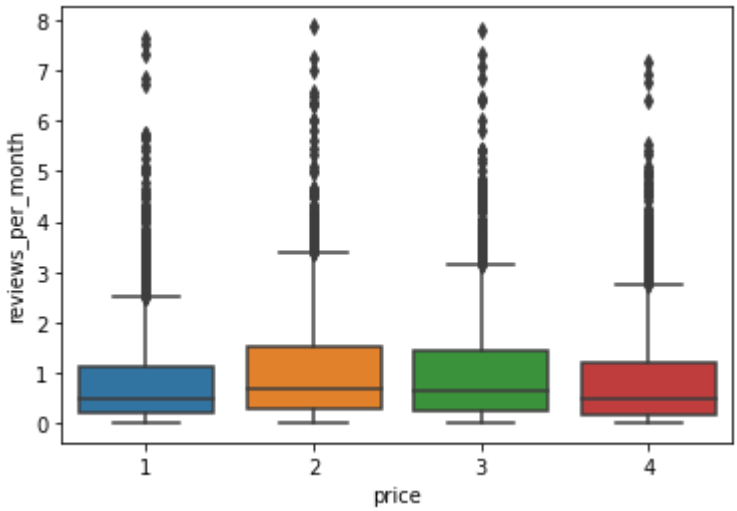

14000
12000
10000
8000
6000
4000
2000
0
id
1
2
3
4
price

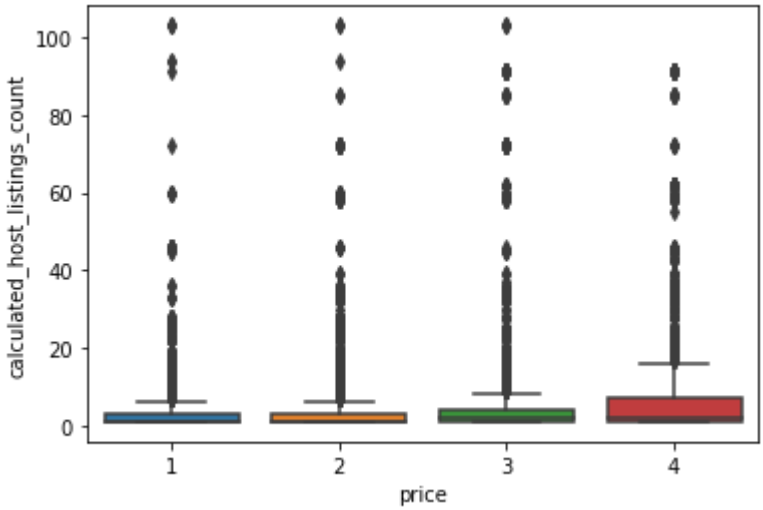

neighbourhood
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price

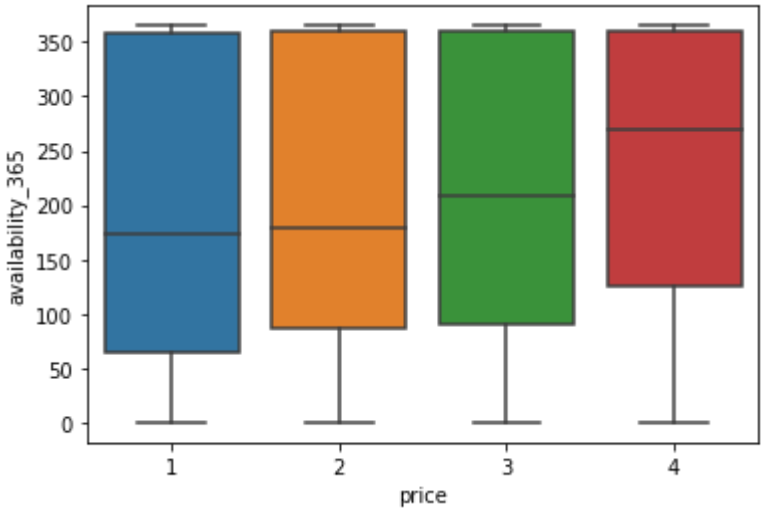

Entire home/apt
Private room
Shared room
Hotel room
room_type
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price

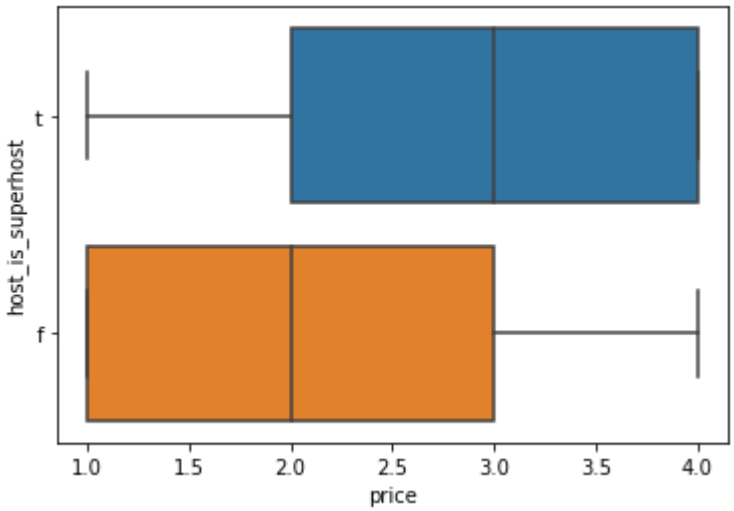

minimum_nights
1000
800
600
400
200
0
1
2
3
4
price

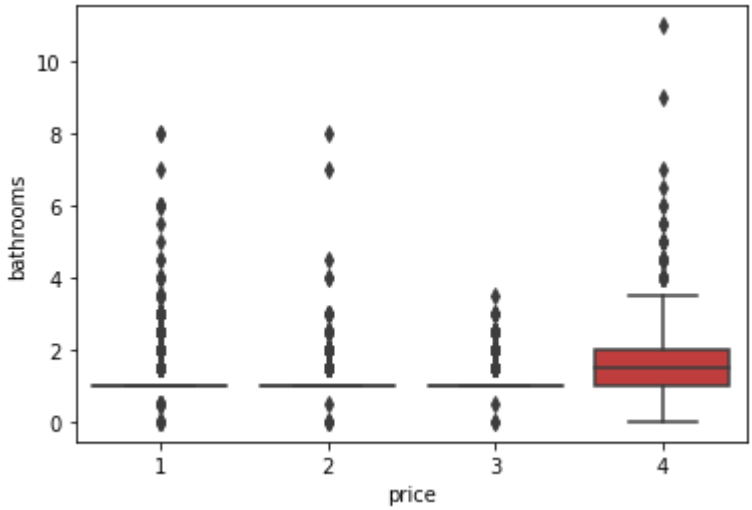

number_of_reviews
500
400
300
200
100
0
1
2
3
4
price

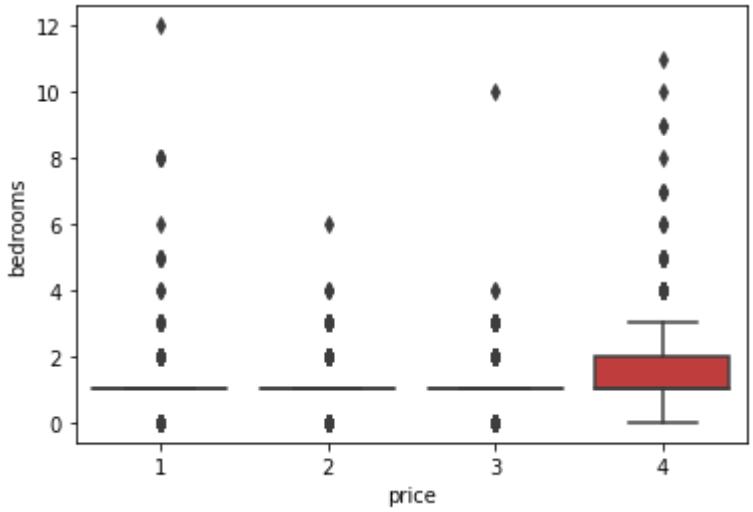

reviews_per_month
8
7
6
5
4
3
2
1
0
1
2
3
4
price

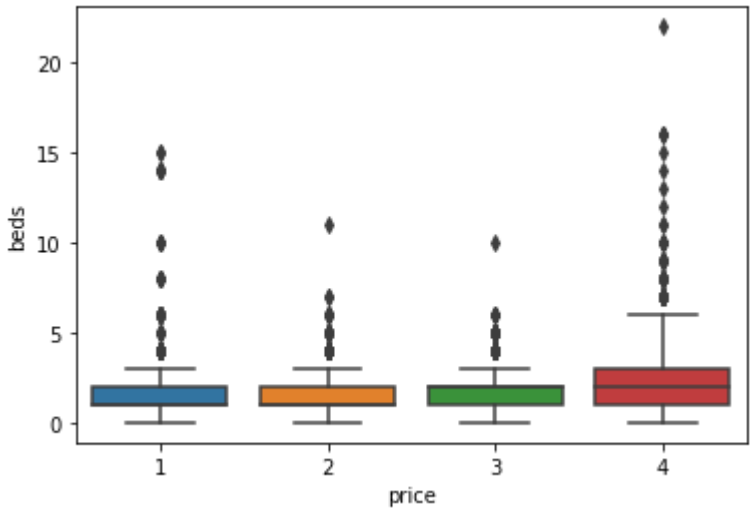

calculated_host_listings_count
100
80
60
40
20
0
1
2
3
4
price

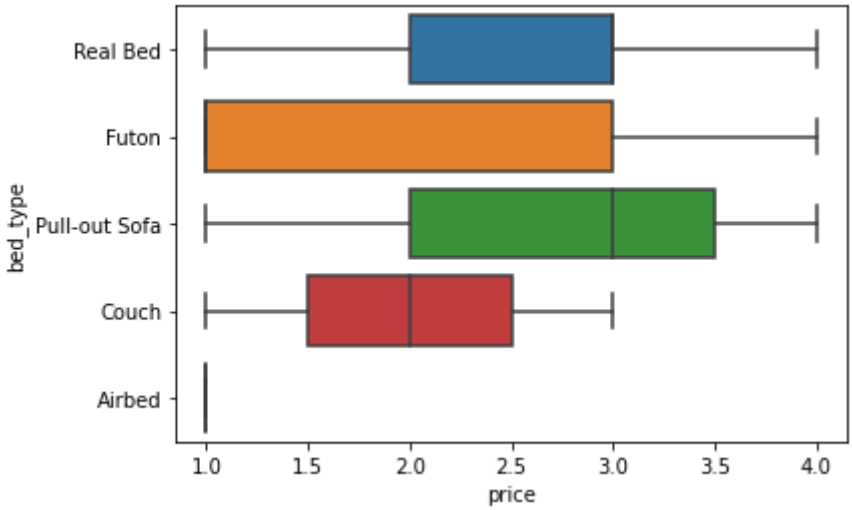

availability_365
350
300
250
200
150
100
50
0
1
2
3
4
price

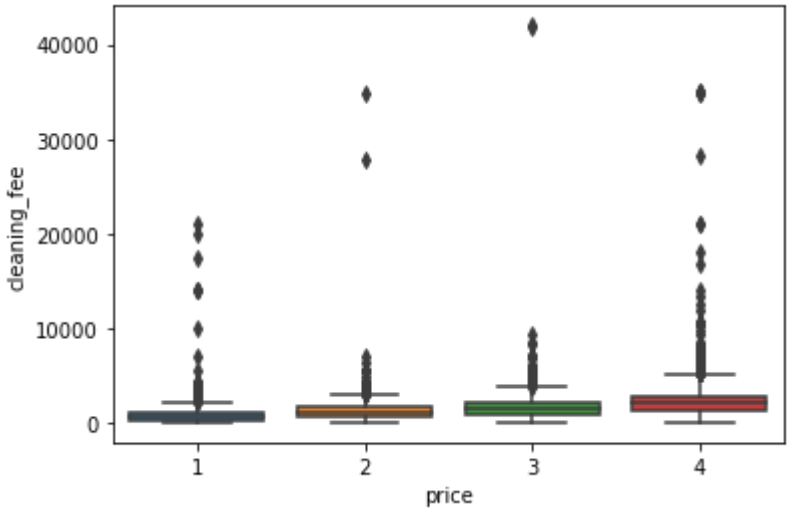

host_is_superhost
t
f
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price

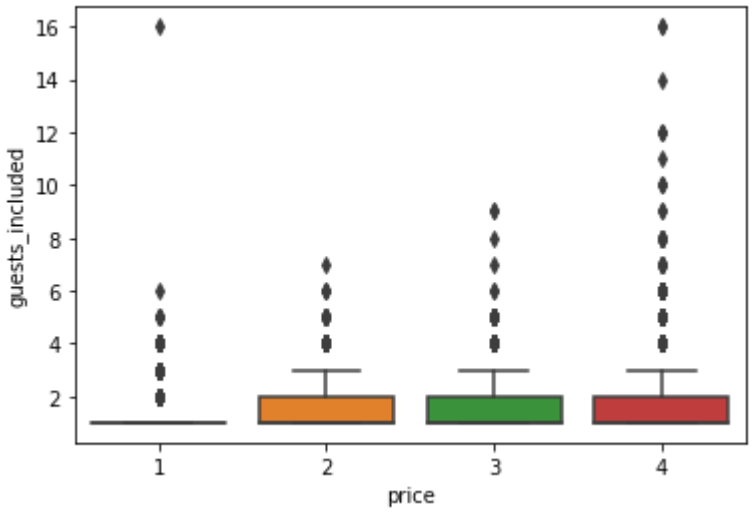

bathrooms
price
0
2
4
6
8
10
1
2
3
4

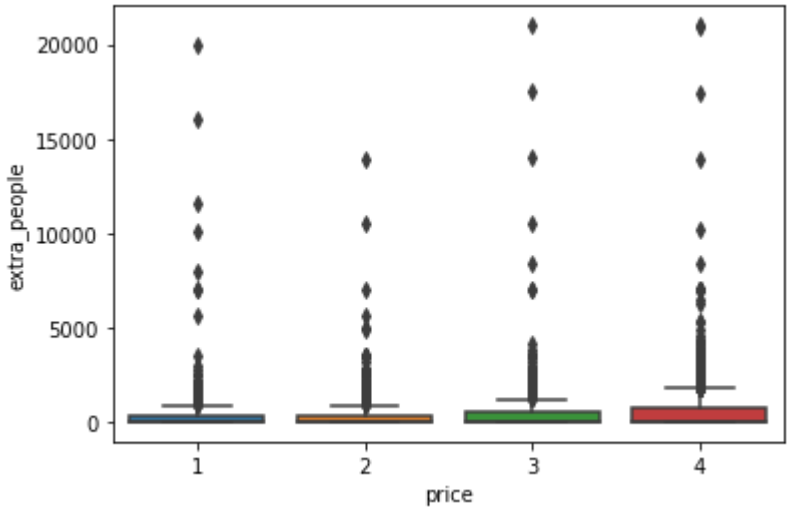

bedrooms
price
0
2
4
6
8
10
12
1
2
3
4

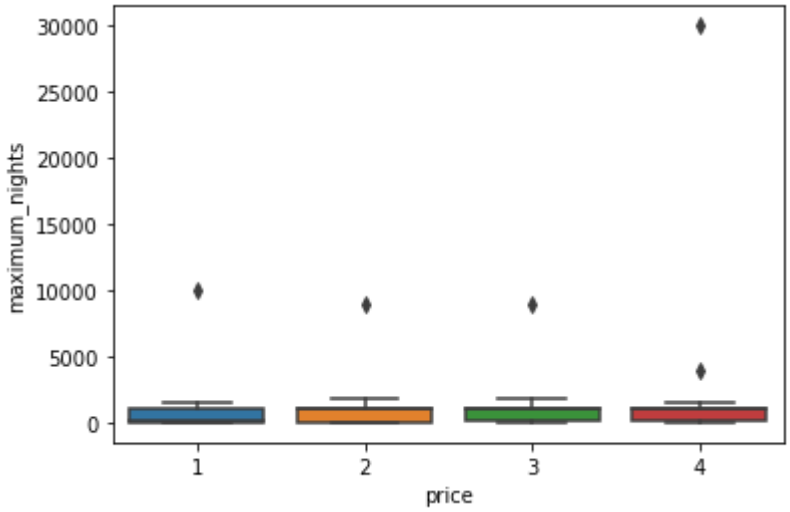

beds
price
0
5
10
15
20
1
2
3
4

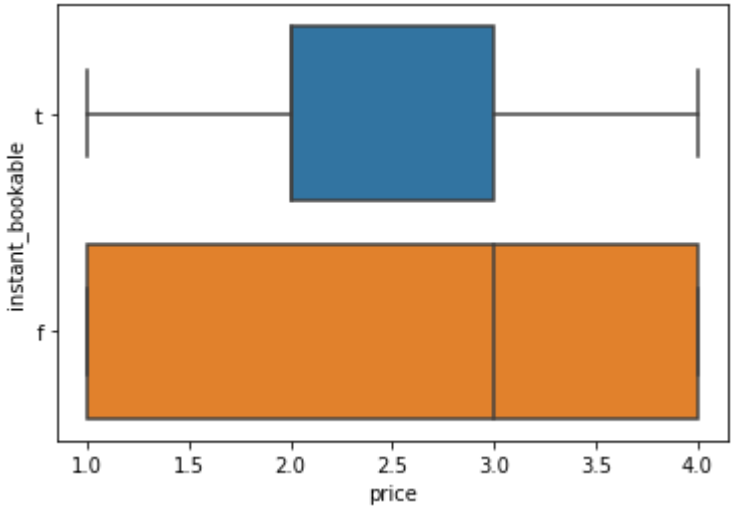

Real Bed
Futon
Pull-out Sofa
Couch
Airbed
bed_type
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price

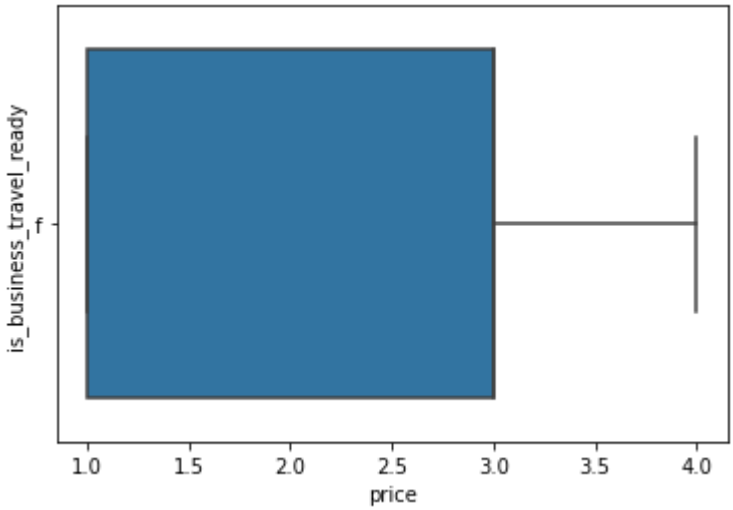

40000
30000
20000
10000
0
deaning_fee
1
2
3
4
price

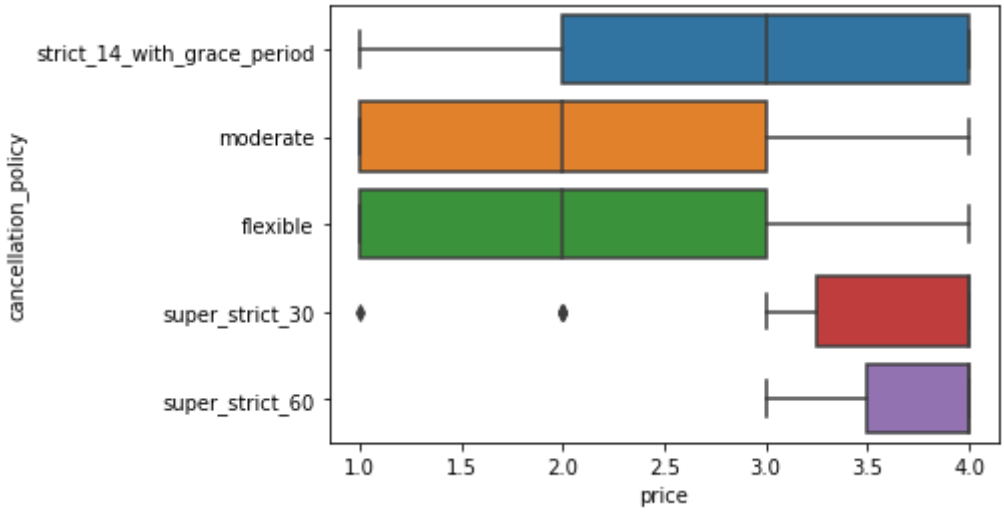

16
14
12
10
8
6
4
2
guests_included
1
2
3
4
price

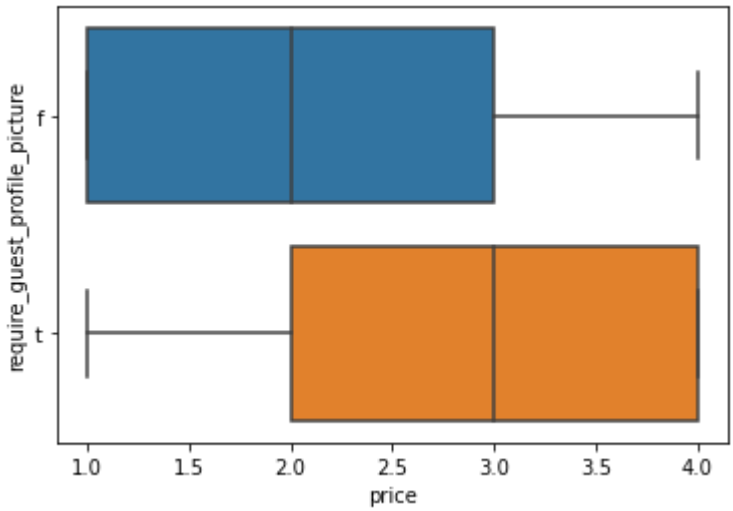

extra_people
20000
15000
10000
5000
0
1
2
3
4
price

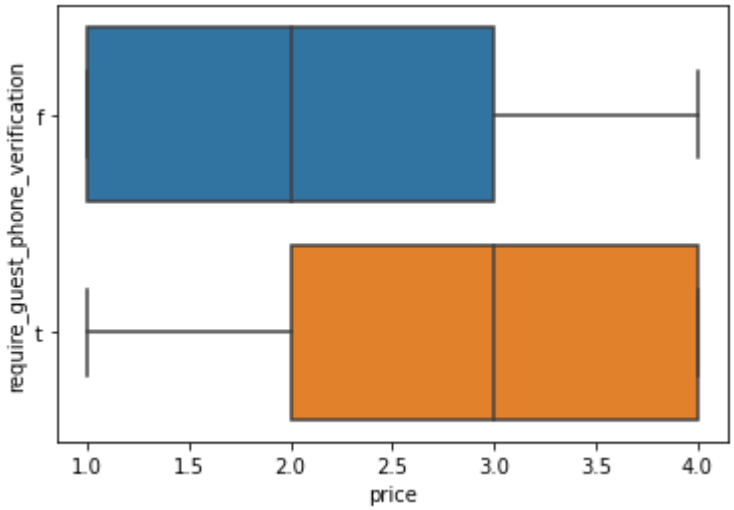

maximum_nights
30000
25000
20000
15000
10000
5000
0
1
2
3
4
price


instant_bookable
t
f
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price


is_business_travel_ready
f
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price


cancellation_policy
strict_14_with_grace_period
moderate
flexible
super_strict_30
super_strict_60
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price


require_guest_profile_picture
f
t
1.0
1.5
2.0
2.5
3.0
3.5
4.0
price



These are the boxplots represent the distribution of price classes in each features. Some of the features have outliers and we might need to deal with it later. Note that 'is_business_travel_ready' only has one category and we might drop it.

3. Preprocess

Conclude the preprocessing steps needed in EDA:

- Factorize categorical varaibels
- Modify 'neighbourhood', 'last_review', 'host_since' if applicable
- Drop 'is_business_travel_ready', 'id'
- Normalize numeric variables
- Outliers

In [10]:

```
class MultiColumnLabelEncoder:
    def __init__(self,columns = None):
        self.columns = columns # array of column names to encode

    def fit(self,X,y=None):
        return self

    def transform(self,X):

        output = X.copy()
        if self.columns is not None:
            for col in self.columns:
                if col in X.columns:
                    output[col] = LabelEncoder().fit_transform(output[col])
                else:
                    continue
        else:
            for colname,col in output.iteritems():
                output[colname] = LabelEncoder().fit_transform(col)
        return output

    def fit_transform(self,X,y=None):
        return self.fit(X,y).transform(X)
```

In [11]:

```
def get_my(date):
    n = len(date)
    mm, dd, yy = np.zeros(n), np.zeros(n), np.zeros(n)
    for i in range(n):
        sep = date.iloc[i].split('/')
        mm[i]=sep[0]
        yy[i]=sep[2]

    return mm, yy
```

In [12]:

```
def date_diff(df, col):
    now = datetime.now()
    for c in col:
        date = pd.to_datetime(df[c])
        df[c] = date.apply(lambda x: (now-x).days)
```

In [13]:

```
def create_onehot(train, test, factor_feature):
    enc = OneHotEncoder()
    enc.fit(train[factor_feature])
    train_transformed = enc.transform(train[factor_feature])
    test_transformed = enc.transform(test[factor_feature])

    train_ohe_df = pd.DataFrame(train_transformed.toarray(),columns=enc.get_feature_nam
es())
    test_ohe_df = pd.DataFrame(test_transformed.toarray(),columns=enc.get_feature_names
())

    train = pd.concat([train, train_ohe_df], axis=1).drop(factor_feature, axis=1)
    test = pd.concat([test, test_ohe_df], axis=1).drop(factor_feature, axis=1)

    return train, test
```

In [14]:

```
def preprocess(train, test, factor_feature, quant_feature, cate='less', scalar='MinMaxS
caler'):
    X_train = train.copy()
    X_test = test.copy()
    y_train = train['price'].copy()

    # only record years
    if cate=='less':
        X_train['last_review_mm'], X_train['last_review_yy'] = get_my(X_train.last_revi
ew)
        X_train['host_since_mm'], X_train['host_since_yy'] = get_my(X_train.host_since)
        X_test['last_review_mm'], X_test['last_review_yy'] = get_my(X_test.last_review)
        X_test['host_since_mm'], X_test['host_since_yy'] = get_my(X_test.host_since)
        X_train = X_train.drop(['last_review','host_since'],axis=1)
        X_test = X_test.drop(['last_review','host_since'],axis=1)
        factor_feature = list(factor_feature) + ['last_review_mm','last_review_yy','hos
t_since_mm','host_since_yy']
    elif cate=='num':
        col=['last_review', 'host_since']
        date_diff(X_train, col)
        date_diff(X_test, col)
        factor_feature = [f for f in list(factor_feature) if f not in col]
        quant_feature = list(quant_feature) + col

    # drop variables
    X_train = X_train.drop(['id','is_business_travel_ready', 'price'],axis=1)
    X_test = X_test.drop(['id','is_business_travel_ready'],axis=1)

    # factorize
    enc = MultiColumnLabelEncoder(factor_feature)
    X_train = enc.fit_transform(X_train)
    X_test = enc.transform(X_test)

    # normalize
    if scalar == 'MinMaxScaler':
        sca = MinMaxScaler()
        feature = [qf for qf in quant_feature if qf in X_train.columns and qf in X_test
.columns]
        X_train[feature] = sca.fit_transform(X_train[feature])
        X_test[feature] = sca.transform(X_test[feature])

    return X_train, y_train, X_test
```

In [15]:

```
quant_feature
```

Out[15]:

```
Index(['id', 'minimum_nights', 'number_of_reviews', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'bathrooms',
       'bedrooms', 'beds', 'cleaning_fee', 'guests_included', 'extra_peopl
e',
       'maximum_nights', 'price'],
      dtype='object')
```

In [16]:

```
X_train_all, y_train_all, X_test = preprocess(train, test, factor_feature, quant_featur
e, cate='less')
```

In [17]:

```
X_train_all
```

Out[17]:

	neighbourhood	room_type	minimum_nights	number_of_reviews	reviews_per_month	c
0	21	0	0.000000	0.338677	0.314249	
1	9	2	0.000000	0.020040	0.071247	
2	27	0	0.001781	0.002004	0.008906	
3	27	0	0.000890	0.000000	0.015267	
4	31	0	0.000890	0.060120	0.194656	
...	...	...	...	...	...	
9676	26	0	0.005343	0.012024	0.111959	
9677	17	0	0.000000	0.008016	0.053435	
9678	21	0	0.001781	0.000000	0.036896	
9679	27	0	0.000890	0.054108	0.072519	
9680	9	2	0.000890	0.000000	0.003817	

9681 rows × 24 columns

In [18]:

```
y_train_all
```

Out[18]:

```
0       2
1       1
2       3
3       2
4       3
       ..
9676    4
9677    4
9678    3
9679    3
9680    1
Name: price, Length: 9681, dtype: int64
```

In [19]:

```
X_test
```

Out[19]:

	neighbourhood	room_type	minimum_nights	number_of_reviews	reviews_per_month	c
0	30	0	0.000000	0.056112	0.188295	
1	26	0	0.000890	0.010020	0.143766	
2	20	0	0.016919	0.004008	0.089059	
3	33	0	0.016919	0.136273	0.256997	
4	40	0	0.000890	0.012024	0.082697	
...	...	...	...	...	...	
4144	20	0	0.001781	0.000000	0.020356	
4145	33	0	0.000890	0.118236	0.362595	
4146	20	0	0.000890	0.004008	0.022901	
4147	16	2	0.000000	0.006012	0.061069	
4148	4	2	0.000890	0.008016	0.017812	

4149 rows × 24 columns

In [20]:

```
# def normal(train, test, quant_feature):
#     feature =  [qf for qf in quant_feature if qf in train.columns and qf in test.colu
mns]

#     d1 = train[feature].copy()
#     d2 = test[feature].copy()
#     mean = d1.mean()
#     std = d1.std

#     d1 = (d1-mean)/std
#     d2 = (d2-mean)/std

#     return d1, d2
```

Split training set into training and validation sets with stratified split.

In [21]:

```
X_train, X_val, y_train, y_val = train_test_split(X_train_all, y_train_all, test_size=
0.1, stratify=y_train_all)
```

4. Model

Temporarily, we have:

- Logistic regression
- Random Forest
- Xgboost
- SVM

In [22]:

```
# cross validation
def CV(model, X_train, y_train, cv, scoring, return_train_score=False):
    cv_summary = cross_validate(estimator = model, X=X_train, y = y_train, scoring=scor
ing, cv=folds, return_train_score=True)
    if return_train_score:
        train_score = cv_summary['train_score']
        train_mean = np.mean(train_score)
        train_std = np.std(train_score)
        print(f'{model} \n train acc mean: %.4f, std: %.4f '%(train_mean, train_std))
        print('-'*80)
    test_score = cv_summary['test_score']

    test_mean = np.mean(test_score)
    test_std = np.std(test_score)
    print(f'{model} \n test acc mean: %.4f, std: %.4f '%(test_mean, test_std))
    print('-'*80)
```

In [64]:

```
def tune(model, parameters, X_train, y_train, return_train_score = False, test_scores =
False, n_jobs=-1):
    clf = GridSearchCV(model, parameters, n_jobs=n_jobs, scoring='accuracy', return_tra
in_score=return_train_score)
    clf.fit(X_train, y_train)

    if test_scores:
        return clf.cv_results_['mean_test_score']
#     if return_train_score:
#         print(cv_results_['mean_train_score'])

    print('Best parameters: {}, best cv score: {}'.format(clf.best_params_, clf.best_sc
ore_))
    return clf.best_params_
```

In [24]:

```
def train_val_acc(model, X_train, y_train, X_val, y_val):
    model.fit(X_train, y_train)
    pred_train = model.predict(X_train)
    pred_val = model.predict(X_val)
    train_acc = np.mean(pred_train==y_train)
    val_acc = np.mean(pred_val==y_val)

    return train_acc, val_acc
```

In [25]:

```
def create_submission(model, X_train_all, y_train_all, X_test, file_name, is_cat=False
):
    model.fit(X_train_all, y_train_all)
    if is_cat:
        y_pred = model.predict(X_test).reshape(-1)
    else:
        y_pred = model.predict(X_test)

    sub = pd.DataFrame({'id':test_id, 'price':y_pred})

    sub.to_csv(file_name+'.csv',index=False)

    y_ratio = sub.price.value_counts()
    y_ratio = y_ratio/sum(y_ratio)

    return y_ratio
```

In [62]:

```
# timeit
def time_measure(model, X = X_train_all, y=y_train_all, n=10, cat=False):
    if not cat:
        T =[]
        for i in range(n):
            t0 = time.time()
            model.fit(X,y)
            t1 = time.time()
            T.append(t1-t0)
        return np.mean(T), np.std(T)
    elif cat:
        T =[]
        for i in range(n):
            t0 = time.time()
            model.fit(X,y,cat_features)
            t1 = time.time()
            T.append(t1-t0)
        return np.mean(T), np.std(T)
```

i. Logistic regression

This is the baseline model.

In [26]:

```
acc = make_scorer(accuracy_score)
folds = 5
```

In [41]:

```
model_mlr = LogisticRegression(solver='lbfgs',
                               class_weight='balanced',
                               max_iter=10000)
```

In [48]:

```
CV(model_mlr,X_train,y_train,folds,acc,True)
```

```
LogisticRegression(class_weight='balanced', max_iter=10000)
 train acc mean: 0.4832, std: 0.0010
------------------------------------------------------------------------
------
LogisticRegression(class_weight='balanced', max_iter=10000)
 test acc mean: 0.4752, std: 0.0081
------------------------------------------------------------------------
------
```

Tune for hyperparameters for better results.

In [65]:

```
# tune
model_mlr = LogisticRegression()
parameters = {'solver':['newton-cg', 'lbfgs', 'sag', 'saga'],
              'max_iter':[10000],
              'C':[0.1, 1, 5],
              'class_weight': ['balanced'],
              'n_jobs': [-1]}
```

In [66]:

```
mlr_opt_params = tune(model_mlr, parameters, X_train, y_train)
```

```
Best parameters: {'C': 5, 'class_weight': 'balanced', 'max_iter': 10000,
'n_jobs': -1, 'solver': 'sag'}, best cv score: 0.48737551485258734
```

Predict on val set.

In [60]:

```
mlr_opt_params = {'C': 5, 'class_weight': 'balanced', 'max_iter': 10000, 'n_jobs': -1,
'solver': 'sag'}
model_mlr_tuned = LogisticRegression(**mlr_opt_params)
```

In [46]:

```
train_val_acc(model_mlr_tuned, X_train, y_train, X_val, y_val)
```

Out[46]:

```
(0.49449035812672176, 0.4953560371517028)
```

In [61]:

```
# time
time_measure(model_mlr_tuned)
```

Out[61]:

```
(17.99238693714142, 0.34968830064305245)
```

There exists no overfitting.

ii. Random Forest

In [49]:

```
model_rf = RandomForestClassifier()
```

In [52]:

```
CV(model_rf,X_train,y_train,folds,acc,True)
```

```
RandomForestClassifier()
 train acc mean: 1.0000, std: 0.0000
------------------------------------------------------------------------
------
RandomForestClassifier()
 test acc mean: 0.5503, std: 0.0062
------------------------------------------------------------------------
------
```

Random Forest seems to be a good candidate, let's tune the tree.

In [141]:

```
parameters = {'n_estimators': [500],
              'criterion': ['gini', 'entropy'],
              'max_depth':[10, 15, 20, 25],
              'max_features':[3, 5, 10, 15, 'sqrt', 'log2'],
              'class_weight': ['balanced'],
              'ccp_alpha':[0,0.1,0.5,1]}
```

In [159]:

```
# tune will all data
tune(model_rf, parameters, X_train_all, y_train_all)
```

```
Best parameters: {'ccp_alpha': 0, 'class_weight': 'balanced', 'criterion':
'gini', 'max_depth': 25, 'max_features': 'log2', 'n_estimators': 500}, bes
t cv score: 0.5599618883252196
```

Out[159]:

```
{'ccp_alpha': 0,
 'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 25,
 'max_features': 'log2',
 'n_estimators': 500}
```

Results on val set

In [37]:

```
rf_opt_params = {'ccp_alpha': 0, 'class_weight': 'balanced', 'criterion': 'gini', 'max_
depth': 25, 'max_features': 'log2', 'n_estimators': 500}
model_rf_tuned = RandomForestClassifier(**rf_opt_params)
```

In [70]:

```
train_val_acc(model_rf_tuned, X_train, y_train, X_val, y_val)
```

Out[70]:

```
(0.9997704315886135, 0.5479876160990712)
```

There exists considerable overfitting, but the highest cv score is associated with the high train score.

We could also check the importance of features.

In [76]:

```
model_rf.fit(X_train, y_train)

importances = model_rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in model_rf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
print("Feature ranking:")

important_feature = []
threshold = 0.01

for f in range(X_train.shape[1]):
    print('{} {}: {}'.format(f + 1, X_train.columns[indices[f]], importances[indices[f
]]))
    if importances[indices[f]]>= threshold:
        important_feature.append(X_train.columns[indices[f]])
```

```
Feature ranking:
1 cleaning_fee: 0.13341905685938
2 reviews_per_month: 0.08925114567805666
3 availability_365: 0.0803251191899467
4 number_of_reviews: 0.07470977153151076
5 neighbourhood: 0.06607734655851721
6 host_since: 0.05856143720816513
7 minimum_nights: 0.05589668655230387
8 maximum_nights: 0.050437146468239734
9 calculated_host_listings_count: 0.04975527146362607
10 room_type: 0.04837711778633282
11 extra_people: 0.04653389740187774
12 bedrooms: 0.044603534709408865
13 beds: 0.0428600342882366
14 bathrooms: 0.036261506756484235
15 cancellation_policy: 0.030353156968246208
16 last_review: 0.02646573195443984
17 guests_included: 0.02630266440869404
18 instant_bookable: 0.018890309259746947
19 host_is_superhost: 0.015145973747917478
20 require_guest_profile_picture: 0.002647157395565314
21 require_guest_phone_verification: 0.002249966905885046
22 bed_type: 0.0008759669074186895
```

iii. Xgboost

In [129]:

```
model_xgb = XGBClassifier()
```

In [130]:

```
CV(model_xgb,X_train,y_train,folds,acc,True)
```

```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints
=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
 train acc mean: 0.9360, std: 0.0011
------------------------------------------------------------------------
------
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints
=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
 test acc mean: 0.5414, std: 0.0136
------------------------------------------------------------------------
------
```

We could tune xgb.

In [204]:

```
parameters = {'max_depth': [5, 7, 9],
              'learning_rate': [0.01, 0.1],
              'n_estimators': [500],
              'gamma': [0.1, 1, 2, 5, 8],
              'colsample_bytree':[0.1, 0.2, 0.3, 0.5, 0.7]}
```

In [205]:

```
tune(model_xgb, parameters, X_train_all, y_train_all, n_jobs=None)
```

```
Best parameters: {'colsample_bytree': 0.5, 'gamma': 2, 'learning_rate': 0.
1, 'max_depth': 7, 'n_estimators': 500}, best cv score: 0.563474871681095
```

Out[205]:

```
{'colsample_bytree': 0.5,
 'gamma': 2,
 'learning_rate': 0.1,
 'max_depth': 7,
 'n_estimators': 500}
```

Results on val set

In [41]:

```
xgb_opt_params = {'colsample_bytree': 0.5, 'gamma': 2, 'learning_rate': 0.1, 'max_dept
h': 7, 'n_estimators': 300}
model_xgb_tuned = XGBClassifier(**xgb_opt_params)
```

In [27]:

```
# cv on train
CV(model_xgb_tuned,X_train_all,y_train_all,folds,acc,True)
```

```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=0.5, gamma=2, gpu_id
=None,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.1, max_delta_step=None, max_depth=7,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              n_estimators=300, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
 train acc mean: 0.7822, std: 0.0044
------------------------------------------------------------------------
------
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=0.5, gamma=2, gpu_id
=None,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.1, max_delta_step=None, max_depth=7,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              n_estimators=300, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
 test acc mean: 0.5631, std: 0.0053
------------------------------------------------------------------------
------
```

In [177]:

```
train_val_acc(model_xgb_tuned, X_train, y_train, X_val, y_val)
```

Out[177]:

```
(0.7801882460973371, 0.5624355005159959)
```

Looks like there are considerable overfitting

sols:

- Increase regularization
- Early stop for Xgboost

In [38]:

```
# increase regulaization
xgb_params = {'colsample_bytree': 0.5, 'gamma': 2, 'learning_rate': 0.1, 'max_depth': 7
, 'n_estimators': 300}
model_xgb_ga = XGBClassifier(**xgb_params)
```

In [ ]:

```
train_val_acc(model_xgb_ga, X_train, y_train, X_val, y_val)
```

In [213]:

```
CV(model_xgb_ga,X_train_all,y_train_all,folds,acc,True)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=8, gpu_id=-
1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=300, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
 train acc mean: 0.5741, std: 0.0007
------------------------------------------------------------------------
------
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=8, gpu_id=-
1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=300, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
 test acc mean: 0.5433, std: 0.0086
------------------------------------------------------------------------
------
```

I tried to fit a model on residual...

In [73]:

```
model_xgb_ga.fit(X_train,y_train)
```

Out[73]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=8, gpu_id=-
1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=9,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=300, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [74]:

```
y_pred = model_xgb_ga.predict(X_train)
```

In [76]:

```
y_train - y_pred
```

Out[76]:

```
4191    0
6573    0
1512   -1
8035    0
7416    1
       ..
7318    2
5166    0
3640    0
4721    0
6770    0
Name: price, Length: 8712, dtype: int64
```

iv. Adaboost

In [208]:

```
model_ada = AdaBoostClassifier()
```

In [210]:

```
CV(model_ada,X_train_all,y_train_all,folds,acc,True)
```

```
AdaBoostClassifier()
 train acc mean: 0.5256, std: 0.0041
------------------------------------------------------------------------
------
AdaBoostClassifier()
 test acc mean: 0.5163, std: 0.0067
------------------------------------------------------------------------
------
```

Tune parameters

In [219]:

```
parameters ={'base_estimator': [DecisionTreeClassifier(max_depth=1),DecisionTreeClassif
ier(max_depth=2),DecisionTreeClassifier(max_depth=3),DecisionTreeClassifier(max_depth=4
),
                                DecisionTreeClassifier(max_depth=5)],
             'n_estimators':[50, 100, 200, 500,1000],
             'learning_rate':[0.01, 0.1, 0.5, 1]
            }
```

In [220]:

```
tune(model_ada, parameters, X_train_all, y_train_all, n_jobs=-1)
```

```
Best parameters: {'base_estimator': DecisionTreeClassifier(max_depth=4),
'learning_rate': 0.01, 'n_estimators': 1000}, best cv score: 0.54054306736
58251
```

Out[220]:

```
{'base_estimator': DecisionTreeClassifier(max_depth=4),
 'learning_rate': 0.01,
 'n_estimators': 1000}
```

In [39]:

```
ada_params = {'base_estimator': DecisionTreeClassifier(max_depth=4), 'learning_rate':
0.01, 'n_estimators': 1000}
model_ada_tuned =  AdaBoostClassifier(**ada_params)
```

v. CatBoost

In [27]:

```
factor_feature
```

Out[27]:

```
Index(['neighbourhood', 'room_type', 'last_review', 'host_since',
       'host_is_superhost', 'bed_type', 'instant_bookable',
       'is_business_travel_ready', 'cancellation_policy',
       'require_guest_profile_picture', 'require_guest_phone_verificatio
n'],
      dtype='object')
```

In [28]:

```
cat_features = ['neighbourhood', 'room_type', 'last_review_mm','last_review_yy', 'host_
since_mm',
       'host_since_yy','host_is_superhost', 'bed_type', 'instant_bookable','cancellatio
n_policy',
       'require_guest_profile_picture', 'require_guest_phone_verification']

model_cat = CatBoostClassifier(verbose=False)
```

In [120]:

```
parameters = {'depth': [4, 7, 10],
              'learning_rate' : [0.01, 0.1, 0.2],
              'l2_leaf_reg': [1, 3, 5, 9],
              'iterations': [100, 200, 300]}
```

In [121]:

```
tune(model_cat, parameters, X_train_all, y_train_all, n_jobs=None)
```

```
Best parameters: {'depth': 4, 'iterations': 300, 'l2_leaf_reg': 9, 'learni
ng_rate': 0.2}, best cv score: 0.5506659676504093
```

Out[121]:

```
{'depth': 4, 'iterations': 300, 'l2_leaf_reg': 9, 'learning_rate': 0.2}
```

In [35]:

```
cat_params = {'eval_metric':'Accuracy', 'one_hot_max_size':50, 'depth': 4, 'iterations'
: 300, 'l2_leaf_reg': 9, 'learning_rate': 0.2, 'verbose':False}
model_cat_tuned = CatBoostClassifier(**cat_params)
```

In [30]:

```
def CV_cat(model, X, y, folds, cat_features=None):
    skf = StratifiedKFold(n_splits=folds)
    cv_score=[]
    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
        y_train, y_test = y[train_index], y[test_index]

        model.fit(X_train, y_train, cat_features = cat_features, verbose=False)
        y_test_pred = model.predict(X_test)
        cv_score.append(np.mean(y_test_pred.reshape(-1)==y_test))

        return np.mean(cv_score)
```

In [144]:

```
CV_cat(model_cat_tuned, X_train_all, y_train_all,folds,cat_features)
```

Out[144]:

```
0.554981930820857
```

In [31]:

```
# tune parameters

def tune_cat(model, parameters, X, y, cat_features=cat_features):
    skf = StratifiedKFold(n_splits=folds)
    best_score=0
    best_estimator=[]
    skf = StratifiedKFold(n_splits=folds)

    param_grid = list(ParameterGrid(parameters))
    n = len(param_grid)

    for i in range(n):
        if i%5==0:
            print(f'epoch %d/%d done.'%(i,n))

        clf = model(**param_grid[i])
        cv_score=[]

        for train_index, test_index in skf.split(X, y):
            X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
            y_train, y_test = y[train_index], y[test_index]

            clf.fit(X_train, y_train, cat_features = cat_features, verbose=False)
            y_test_pred = clf.predict(X_test)
            cv_score.append(np.mean(y_test_pred.reshape(-1)==y_test))
        cv_score=np.mean(cv_score)

        if cv_score>best_score:
            best_score=cv_score
            best_estimator=param_grid[i]
    return best_estimator, best_score
```

In [32]:

```
parameters ={'eval_metric':['Accuracy'],
             'one_hot_max_size':[20,30,50],
             'depth': [4, 7, 10],
             'learning_rate' : [0.01, 0.1, 0.2],
             'l2_leaf_reg': [1, 4, 9],
             'iterations': [500],
             'verbose':[False]}
```

In [33]:

```
tune_cat(CatBoostClassifier, parameters, X_train_all, y_train_all, cat_features=cat_fea
tures)
```

```
epoch 0/81 done.
epoch 5/81 done.
epoch 10/81 done.
epoch 15/81 done.
epoch 20/81 done.
epoch 25/81 done.
epoch 30/81 done.
epoch 35/81 done.
epoch 40/81 done.
epoch 45/81 done.
epoch 50/81 done.
epoch 55/81 done.
epoch 60/81 done.
epoch 65/81 done.
epoch 70/81 done.
epoch 75/81 done.
epoch 80/81 done.
```

Out[33]:

```
({'depth': 10,
  'eval_metric': 'Accuracy',
  'iterations': 500,
  'l2_leaf_reg': 4,
  'learning_rate': 0.1,
  'one_hot_max_size': 20,
  'verbose': False},
 0.5498401080310782)
```

5. Try oneVSrest setting for candidate classifiers

i. Random Forest

In [50]:

```
model_rf_1vn = OneVsRestClassifier(RandomForestClassifier())
```

In [51]:

```
CV(model_rf_1vn,X_train,y_train,folds,acc)
```

```
OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True,
                                                     ccp_alpha=0.0,
                                                     class_weight=None,
                                                     criterion='gini',
                                                     max_depth=None,
                                                     max_features='auto',
                                                     max_leaf_nodes=None,
                                                     max_samples=None,
                                                     min_impurity_decrease
=0.0,
                                                     min_impurity_split=No
ne,
                                                     min_samples_leaf=1,
                                                     min_samples_split=2,
                                                     min_weight_fraction_l
eaf=0.0,
                                                     n_estimators=100,
                                                     n_jobs=None,
                                                     oob_score=False,
                                                     random_state=None,
                                                     verbose=0,
                                                     warm_start=False),
                    n_jobs=None)
 acc mean: 0.5544, std: 0.0128
------------------------------------------------------------------------
------
```

In [87]:

```
#### tune parameters

parameters = {'n_estimators': [200],
              'max_depth':[10, 20, 30],
              'max_features':[3, 5, 10, 15, 20],
              'min_samples_leaf':[1,2,3],
              'n_jobs':[-1]}
```

In [88]:

```
param_grid = list(ParameterGrid(parameters))
```

In [96]:

```
def tune_1vn(model, param_grid, X_train, y_train,folds=5,scoring=make_scorer(accuracy_s
core)):
    best_score = 0
    best_params = []
    n = len(param_grid)
    for i in range(n):
        if i%5==0:
            print(f'epoch %d/%d done.'%(i,n))
        single = model(**param_grid[i])
        clf = OneVsRestClassifier(single)
        cv_score = np.mean(cross_val_score(clf, X_train, y_train, cv=folds, scoring=sco
ring))
        if cv_score > best_score:
            best_score = cv_score
            best_params = param_grid[i]
    return best_params, best_score
```

In [90]:

```
tune_1vn(RandomForestClassifier, param_grid, X_train, y_train)
```

```
epoch 0/45 done.
epoch 5/45 done.
epoch 10/45 done.
epoch 15/45 done.
epoch 20/45 done.
epoch 25/45 done.
epoch 30/45 done.
epoch 35/45 done.
epoch 40/45 done.
```

Out[90]:

```
({'max_depth': 20,
  'max_features': 5,
  'min_samples_leaf': 1,
  'n_estimators': 200,
  'n_jobs': -1},
 0.5598590625359996)
```

In [88]:

```
opt_1vn_rf={'max_depth': 20,
  'max_features': 5,
  'min_samples_leaf': 1,
  'n_estimators': 200,
  'n_jobs': -1}
```

In [87]:

```
model_rf_1vn_tuned = OneVsRestClassifier(RandomForestClassifier(**opt_1vn_rf))
```

ii. xgboost

In [84]:

```
model_xgb_1vn = OneVsRestClassifier(XGBClassifier())
```

In [52]:

```
CV(model_xgb_1vn,X_train,y_train,folds,acc)
```

```
OneVsRestClassifier(estimator=XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=None, gamma=N
one,
                                            gpu_id=None, importance_type
='gain',
                                            interaction_constraints=None,
                                            learning_rate=None,
                                            max_delta_step=None, max_depth
=None,
                                            min_child_weight=None, missing
=nan,
                                            monotone_constraints=None,
                                            n_estimators=100, n_jobs=None,
                                            num_parallel_tree=None,
                                            objective='binary:logistic',
                                            random_state=None, reg_alpha=N
one,
                                            reg_lambda=None,
                                            scale_pos_weight=None,
                                            subsample=None, tree_method=No
ne,
                                            validate_parameters=None,
                                            verbosity=None),
                    n_jobs=None)
 acc mean: 0.5468, std: 0.0127
------------------------------------------------------------------------
------
```

In [92]:

```
# tune
parameters = {'max_depth': [3, 4, 5, 6, 7],
              'learning_rate': [0.05, 0.1, 0.2],
              'n_estimators': [200],
              'gamma': [0.1, 0.2, 0.4],
              'reg_alpha': [0, 1e-2, 1, 1e1]
             }
param_grid = list(ParameterGrid(parameters))
```

In [93]:

```
tune_1vn(XGBClassifier, param_grid, X_train, y_train)
```

```
epoch 0/180 done.
epoch 5/180 done.
epoch 10/180 done.
epoch 15/180 done.
epoch 20/180 done.
epoch 25/180 done.
epoch 30/180 done.
epoch 35/180 done.
epoch 40/180 done.
epoch 45/180 done.
epoch 50/180 done.
epoch 55/180 done.
epoch 60/180 done.
epoch 65/180 done.
epoch 70/180 done.
epoch 75/180 done.
epoch 80/180 done.
epoch 85/180 done.
epoch 90/180 done.
epoch 95/180 done.
epoch 100/180 done.
epoch 105/180 done.
epoch 110/180 done.
epoch 115/180 done.
epoch 120/180 done.
epoch 125/180 done.
epoch 130/180 done.
epoch 135/180 done.
epoch 140/180 done.
epoch 145/180 done.
epoch 150/180 done.
epoch 155/180 done.
epoch 160/180 done.
epoch 165/180 done.
epoch 170/180 done.
epoch 175/180 done.
```

Out[93]:

```
({'gamma': 0.2,
  'learning_rate': 0.1,
  'max_depth': 7,
  'n_estimators': 200,
  'reg_alpha': 1},
 0.5598591692017562)
```

In [90]:

```
opt_1vn_xgb = {'gamma': 0.2,
  'learning_rate': 0.1,
  'max_depth': 7,
  'n_estimators': 200,
  'reg_alpha': 1}

model_xgb_1vn_tuned = OneVsRestClassifier(XGBClassifier(**opt_1vn_xgb))
```

iii. Adaboost

In [92]:

```
model_ada_1vn = OneVsRestClassifier(AdaBoostClassifier())
```

In [93]:

```
CV(model_ada_1vn,X_train_all,y_train_all,folds,acc)
```

```
OneVsRestClassifier(estimator=AdaBoostClassifier())
 test acc mean: 0.5236, std: 0.0153
------------------------------------------------------------------------
------
```

In [94]:

```
# tune
parameters = {'base_estimator': [DecisionTreeClassifier(max_depth=1),DecisionTreeClassi
fier(max_depth=2),DecisionTreeClassifier(max_depth=3),DecisionTreeClassifier(max_depth=
4),
                                 DecisionTreeClassifier(max_depth=5)],
              'n_estimators':[50, 100, 200, 500,1000],
              'learning_rate':[0.01, 0.1, 0.5, 1]
             }
param_grid = list(ParameterGrid(parameters))
```

In [98]:

```
tune_1vn(AdaBoostClassifier, param_grid, X_train_all, y_train_all)
```

```
epoch 0/100 done.
epoch 5/100 done.
epoch 10/100 done.
epoch 15/100 done.
epoch 20/100 done.
epoch 25/100 done.
epoch 30/100 done.
epoch 35/100 done.
epoch 40/100 done.
epoch 45/100 done.
epoch 50/100 done.
epoch 55/100 done.
epoch 60/100 done.
epoch 65/100 done.
epoch 70/100 done.
epoch 75/100 done.
epoch 80/100 done.
epoch 85/100 done.
epoch 90/100 done.
epoch 95/100 done.
```

Out[98]:

```
({'base_estimator': DecisionTreeClassifier(max_depth=3),
  'learning_rate': 0.01,
  'n_estimators': 1000},
 0.543745760036181)
```

iv. Losgistic regression

In [54]:

```
model_mlr_1vn = OneVsRestClassifier(LogisticRegression(max_iter=10000))
CV(model_mlr_1vn,X_train,y_train,folds,acc)
```

```
OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
                                                 dual=False, fit_intercept
=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=1
0000,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l
2',
                                                 random_state=None,
                                                 solver='lbfgs', tol=0.000
1,
                                                 verbose=0, warm_start=Fal
se),
                    n_jobs=None)
 acc mean: 0.4754, std: 0.0076
------------------------------------------------------------------------
------
```

6. Ensemble Method

Ensemble best performed single models (random forest and xgboost) and check performance.

In [42]:

```
model_rf_tuned, model_xgb_tuned, model_ada_tuned, model_cat_tuned
```

Out[42]:

```
(RandomForestClassifier(ccp_alpha=0, class_weight='balanced', max_depth=2
5,
                        max_features='log2', n_estimators=500),
 XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
               colsample_bynode=None, colsample_bytree=0.5, gamma=2, gpu_i
d=None,
               importance_type='gain', interaction_constraints=None,
               learning_rate=0.1, max_delta_step=None, max_depth=7,
               min_child_weight=None, missing=nan, monotone_constraints=No
ne,
               n_estimators=300, n_jobs=None, num_parallel_tree=None,
               random_state=None, reg_alpha=None, reg_lambda=None,
               scale_pos_weight=None, subsample=None, tree_method=None,
               validate_parameters=None, verbosity=None),
 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=4),
                    learning_rate=0.01, n_estimators=1000),
 <catboost.core.CatBoostClassifier at 0x7fdf6433c0d0>)
```

In [61]:

```
model_ens = VotingClassifier(estimators=[('xgb', model_xgb_tuned), ('cat', model_cat_tu
ned)], voting='soft')
```

In [62]:

```
CV(model_ens, X_train, y_train, folds, acc, True)
```

```
VotingClassifier(estimators=[('xgb',
                              XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=0.5, gamma=2,
                                            gpu_id=None, importance_type
='gain',
                                            interaction_constraints=None,
                                            learning_rate=0.1,
                                            max_delta_step=None, max_depth
=7,
                                            min_child_weight=None, missing
=nan,
                                            monotone_constraints=None,
                                            n_estimators=300, n_jobs=None,
                                            num_parallel_tree=None,
                                            random_state=None, reg_alpha=N
one,
                                            reg_lambda=None,
                                            scale_pos_weight=None,
                                            subsample=None, tree_method=No
ne,
                                            validate_parameters=None,
                                            verbosity=None)),
                             ('cat',
                              <catboost.core.CatBoostClassifier object at
0x7fdf6433c0d0>)],
                  voting='soft')
 train acc mean: 0.7476, std: 0.0062
------------------------------------------------------------------------
------
VotingClassifier(estimators=[('xgb',
                              XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=0.5, gamma=2,
                                            gpu_id=None, importance_type
='gain',
                                            interaction_constraints=None,
                                            learning_rate=0.1,
                                            max_delta_step=None, max_depth
=7,
                                            min_child_weight=None, missing
=nan,
                                            monotone_constraints=None,
                                            n_estimators=300, n_jobs=None,
                                            num_parallel_tree=None,
                                            random_state=None, reg_alpha=N
one,
                                            reg_lambda=None,
                                            scale_pos_weight=None,
                                            subsample=None, tree_method=No
ne,
                                            validate_parameters=None,
                                            verbosity=None)),
                             ('cat',
                              <catboost.core.CatBoostClassifier object at
0x7fdf6433c0d0>)],
                  voting='soft')
 test acc mean: 0.5523, std: 0.0094
```

```
------------------------------------------------------------------------------
------
```

In [52]:

```
model_ens_hard = VotingClassifier(estimators=[('rf', model_rf_tuned), ('xgb', model_xgb
_tuned), ('ada', model_ada_tuned)], voting='hard')
CV(model_ens_hard, X_train, y_train, folds, acc, True)
```

```
VotingClassifier(estimators=[('rf',
                              RandomForestClassifier(ccp_alpha=0,
                                                     class_weight='balance
d',
                                                     max_depth=25,
                                                     max_features='log2',
                                                     n_estimators=500)),
                             ('xgb',
                              XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=0.5, gamma=2,
                                            gpu_id=None, importance_type
='gain',
                                            interaction_constraints=None,
                                            learning_rate=0....
                                            min_child_weight=None, missing
=nan,
                                            monotone_constraints=None,
                                            n_estimators=300, n_jobs=None,
                                            num_parallel_tree=None,
                                            random_state=None, reg_alpha=N
one,
                                            reg_lambda=None,
                                            scale_pos_weight=None,
                                            subsample=None, tree_method=No
ne,
                                            validate_parameters=None,
                                            verbosity=None)),
                             ('ada',
                              AdaBoostClassifier(base_estimator=DecisionTr
eeClassifier(max_depth=4),
                                                 learning_rate=0.01,
                                                 n_estimators=1000))])
 train acc mean: 0.8396, std: 0.0040
------------------------------------------------------------------------
------
VotingClassifier(estimators=[('rf',
                              RandomForestClassifier(ccp_alpha=0,
                                                     class_weight='balance
d',
                                                     max_depth=25,
                                                     max_features='log2',
                                                     n_estimators=500)),
                             ('xgb',
                              XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=0.5, gamma=2,
                                            gpu_id=None, importance_type
='gain',
                                            interaction_constraints=None,
                                            learning_rate=0....
                                            min_child_weight=None, missing
=nan,
                                            monotone_constraints=None,
                                            n_estimators=300, n_jobs=None,
                                            num_parallel_tree=None,
                                            random_state=None, reg_alpha=N
one,
                                            reg_lambda=None,
```

```
                                          scale_pos_weight=None,
                                          subsample=None, tree_method=No
ne,
                                          validate_parameters=None,
                                          verbosity=None)),
                             ('ada',
                              AdaBoostClassifier(base_estimator=DecisionTr
eeClassifier(max_depth=4),
                                                 learning_rate=0.01,
                                                 n_estimators=1000))])
 test acc mean: 0.5569, std: 0.0064
------------------------------------------------------------------------
------
```

Create submission

In [58]:

```
create_submission(model_ens, X_train_all, y_train_all, X_test, '4ens')
```

Out[58]:

```
3    0.303206
4    0.254278
1    0.234996
2    0.207520
Name: price, dtype: float64
```

In [51]:

```
create_submission(model_cat_tuned, X_train_all, y_train_all, X_test, 'cat', True)
```

Out[51]:

```
3    0.288744
4    0.254519
1    0.244396
2    0.212340
Name: price, dtype: float64
```

In [ ]:

```

```

7. Explore by encoding datetime into numeric

Here, variable 'last_review' and 'host_since' are converted into number of days from now.

In [46]:

```
X_train_all, y_train_all, X_test = preprocess(train, test, factor_feature, quant_featur
e, cate='num')
```

i. Random forest

In [25]:

```
model_rf = RandomForestClassifier()
```

In [86]:

```
CV(model_rf,X_train_all,y_train_all,folds,acc,True)
```

```
RandomForestClassifier()
 train acc mean: 1.0000, std: 0.0000
--------------------------------------------------------------------------
------
RandomForestClassifier()
 test acc mean: 0.5487, std: 0.0112
--------------------------------------------------------------------------
------
```

In [88]:

```
# tune
parameters = {'n_estimators': [500],
              'criterion': ['gini', 'entropy'],
              'max_depth':[10, 15, 20, 25],
              'max_features':[3, 5, 10, 15, 'sqrt', 'log2'],
              'class_weight': ['balanced'],
              'ccp_alpha':[0,0.1,0.5,1]}

tune(model_rf, parameters, X_train_all, y_train_all)
```

```
Best parameters: {'ccp_alpha': 0, 'class_weight': 'balanced', 'criterion':
'gini', 'max_depth': 25, 'max_features': 'sqrt', 'n_estimators': 500}, bes
t cv score: 0.5562443733813472
```

Out[88]:

```
{'ccp_alpha': 0,
 'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 25,
 'max_features': 'sqrt',
 'n_estimators': 500}
```

In [28]:

```
rf_num_params = {'ccp_alpha': 0, 'class_weight': 'balanced', 'criterion': 'gini', 'max_
depth': 25, 'max_features': 'sqrt', 'n_estimators': 500}
```

In [29]:

```
model_rf_num = RandomForestClassifier(**rf_num_params)
```

Time

In [31]:

```
time_measure(model_rf_num)
```

Out[31]:

```
(7.896006798744201, 0.05563087149342594)
```

ii. xgboost

In [67]:

```
model_xgb = XGBClassifier()
```

In [90]:

```
CV(model_xgb,X_train_all,y_train_all,folds,acc,True)
```

```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints
=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
 train acc mean: 0.9564, std: 0.0062
------------------------------------------------------------------------
------
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints
=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
 test acc mean: 0.5451, std: 0.0092
------------------------------------------------------------------------
------
```

In [43]:

```
# tune
parameters = {'objective':['multi:softprob'],
              'max_depth': [5, 7, 9],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [500],
             'gamma': [0.1, 1, 2, 5, 8],
             'colsample_bytree':[0.1, 0.2, 0.3, 0.5, 0.7]}

tune(model_xgb, parameters, X_train_all, y_train_all, n_jobs=None)
```

```
Best parameters: {'colsample_bytree': 0.5, 'gamma': 0.1, 'learning_rate':
0.01, 'max_depth': 9, 'n_estimators': 500, 'objective': 'multi:softprob'},
best cv score: 0.5652303767007855
```

Out[43]:

```
{'colsample_bytree': 0.5,
 'gamma': 0.1,
 'learning_rate': 0.01,
 'max_depth': 9,
 'n_estimators': 500,
 'objective': 'multi:softprob'}
```

In [32]:

```
xgb_num_params={'colsample_bytree': 0.5, 'gamma': 0.1, 'learning_rate': 0.01, 'max_dept
h': 9, 'n_estimators': 500, 'objective': 'multi:softprob'}
model_xgb_num = XGBClassifier(**xgb_num_params)
```

In [33]:

```
# time
time_measure(model_xgb_num)
```

Out[33]:

```
(10.764060926437377, 0.5541887874397537)
```

Further tuning

In [73]:

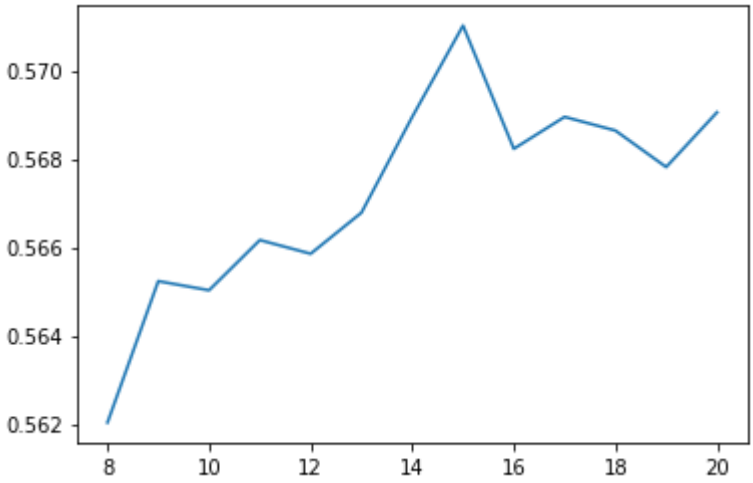
```
# tune max depth
parameters = {'objective':['multi:softprob'],
              'max_depth': [8,9,10,11,12,13,14,15,16,17,18,19,20],
             'learning_rate': [0.01],
             'n_estimators': [500],
             'gamma': [0.1],
             'colsample_bytree':[0.5]}
test_score_md = tune(model_xgb, parameters, X_train_all, y_train_all, test_scores=True,
n_jobs=None)
```

In [78]:

```
md = [8,9,10,11,12,13,14,15,16,17,18,19,20]
plt.plot(md, test_score_md)
# md = 15
```

Out[78]:

```
[<matplotlib.lines.Line2D at 0x7fa27308ef50>]
```



In [79]:

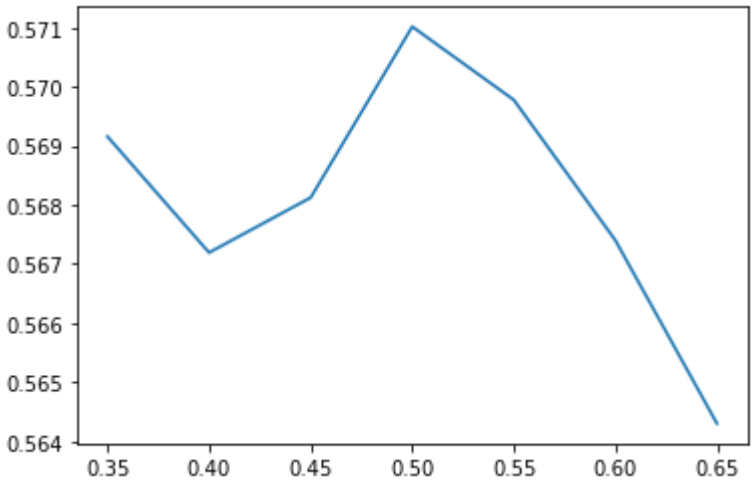
```
# tune colsample_bytree
parameters = {'objective':['multi:softprob'],
              'max_depth': [15],
            'learning_rate': [0.01],
            'n_estimators': [500],
            'gamma': [0.1],
            'colsample_bytree':[0.35, 0.4,0.45, 0.5, 0.55, 0.6, 0.65]}
test_score_cb = tune(model_xgb, parameters, X_train_all, y_train_all, test_scores=True,
n_jobs=None)
```

In [80]:

```
cb = [0.35, 0.4,0.45, 0.5, 0.55, 0.6, 0.65]
plt.plot(cb, test_score_cb)
```

Out[80]:

```
[<matplotlib.lines.Line2D at 0x7fa2729778d0>]
```



In [81]:

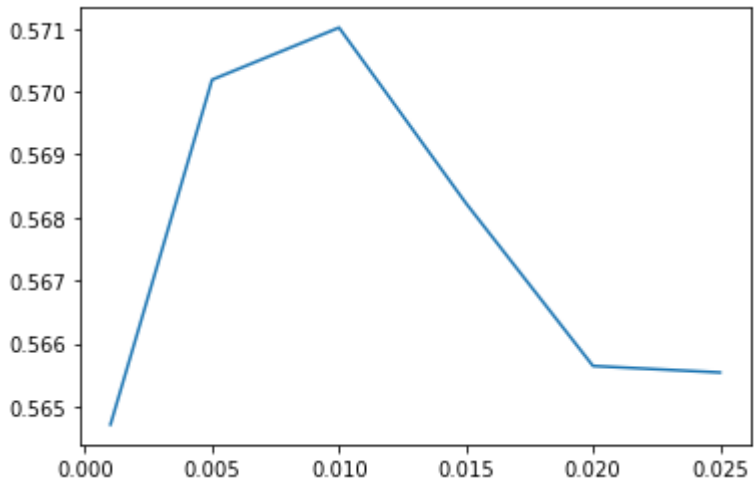
```
# tune lr
parameters = {'objective':['multi:softprob'],
              'max_depth': [15],
             'learning_rate': [0.001, 0.005, 0.01, 0.015, 0.02, 0.025],
             'n_estimators': [500],
             'gamma': [0.1],
             'colsample_bytree':[0.5]}
test_score_lr = tune(model_xgb, parameters, X_train_all, y_train_all, test_scores=True,
n_jobs=None)
```

In [82]:

```
lr = [0.001, 0.005, 0.01, 0.015, 0.02, 0.025]
plt.plot(lr, test_score_lr)
```

Out[82]:

```
[<matplotlib.lines.Line2D at 0x7fa2728e9a50>]
```



In [83]:

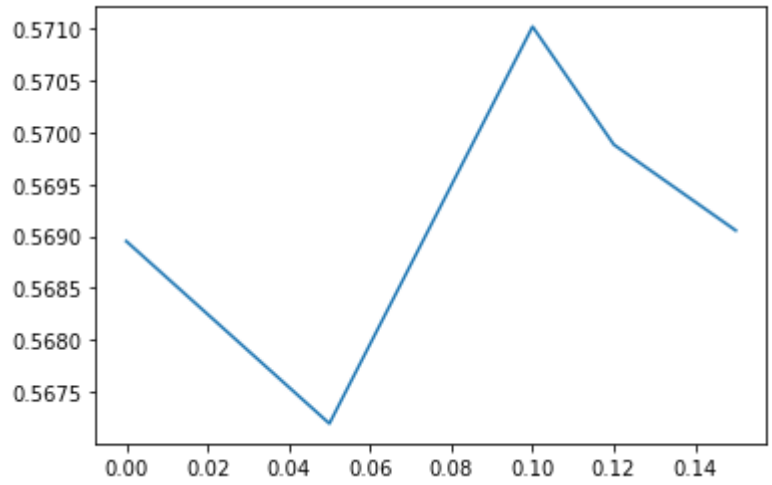
```
# tune gamma
parameters = {'objective':['multi:softprob'],
              'max_depth': [15],
             'learning_rate': [0.01],
             'n_estimators': [500],
             'gamma': [0, 0.05, 0.1, 0.12, 0.15],
             'colsample_bytree':[0.5]}
test_score_ga = tune(model_xgb, parameters, X_train_all, y_train_all, test_scores=True,
n_jobs=None)
```

In [84]:

```
ga = [0, 0.05, 0.1, 0.12, 0.15]
plt.plot(ga, test_score_ga)
```

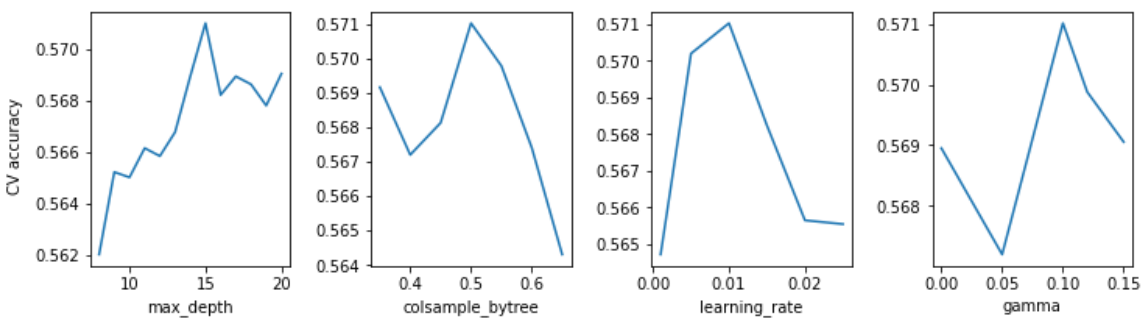
Out[84]:

```
[<matplotlib.lines.Line2D at 0x7fa2728d2e90>]
```



In [121]:

```
fig, axs = plt.subplots(1, 4, figsize=(12, 3))
fig.subplots_adjust(wspace=0.4)
axs = axs.ravel()

fig.text(0.07, 0.35, 'CV accuracy', ha='center', rotation='vertical')
axs[0].plot(md, test_score_md)
axs[0].set_xlabel('max_depth')

axs[1].plot(cb, test_score_cb)
axs[1].set_xlabel('colsample_bytree')

axs[2].plot(lr, test_score_lr)
axs[2].set_xlabel('learning_rate')

axs[3].plot(ga, test_score_ga)
axs[3].set_xlabel('gamma')

plt.savefig('tune.png')
```



In [138]:

```
xgb_params = {'objective':'multi:softprob',
              'max_depth': 15,
             'learning_rate':  0.01,
             'n_estimators': 500,
             'gamma': 0.1,
             'colsample_bytree':0.5}
model_xgb_v2 = XGBClassifier(**xgb_params)
```

In [139]:

```
create_submission(model_xgb_v2, X_train_all, y_train_all, X_test, 'ens_xgbv2_1', True)
```

Out[139]:

```
2    0.286334
1    0.244878
3    0.239576
4    0.229212
Name: price, dtype: float64
```

iii. Catboost

In [38]:

```
cat_features = ['neighbourhood', 'room_type','host_is_superhost', 'bed_type', 'instant_
bookable','cancellation_policy',
                'require_guest_profile_picture', 'require_guest_phone_verification']

model_cat = CatBoostClassifier(verbose=False)
```

In [97]:

```
parameters ={'eval_metric':['Accuracy'],
             'one_hot_max_size':[50],
             'depth': [4, 7, 10],
             'learning_rate' : [0.01, 0.1, 0.2],
             'l2_leaf_reg': [1, 4, 9],
             'iterations': [500],
             'verbose':[False]}
```

In [98]:

```
tune_cat(CatBoostClassifier, parameters, X_train_all, y_train_all, cat_features=cat_fea
tures)
```

```
epoch 0/27 done.
epoch 5/27 done.
epoch 10/27 done.
epoch 15/27 done.
epoch 20/27 done.
epoch 25/27 done.
```

Out[98]:

```
({'depth': 7,
  'eval_metric': 'Accuracy',
  'iterations': 500,
  'l2_leaf_reg': 1,
  'learning_rate': 0.1,
  'one_hot_max_size': 50,
  'verbose': False},
 0.5498406413598604)
```

In [41]:

```
cat_num_params={'depth': 7,
  'eval_metric': 'Accuracy',
  'iterations': 500,
  'l2_leaf_reg': 1,
  'learning_rate': 0.1,
  'one_hot_max_size': 50,
  'verbose': False}
model_cat_num = CatBoostClassifier(**cat_num_params)
```

In [42]:

```
# time
time_measure(model=model_cat_num, cat = True)
```

Out[42]:

```
(5.915794086456299, 0.35598828238307695)
```

iv. Adaboost

In [100]:

```
model_ada = AdaBoostClassifier()
```

In [102]:

```
CV(model_ada,X_train_all,y_train_all,folds,acc,True)
```

```
AdaBoostClassifier()
 train acc mean: 0.5276, std: 0.0045
--------------------------------------------------------------------------
------
AdaBoostClassifier()
 test acc mean: 0.5161, std: 0.0083
--------------------------------------------------------------------------
------
```

In [103]:

```
parameters ={'base_estimator': [DecisionTreeClassifier(max_depth=1),DecisionTreeClassif
ier(max_depth=2),DecisionTreeClassifier(max_depth=3),DecisionTreeClassifier(max_depth=4
),
                                DecisionTreeClassifier(max_depth=5)],
             'n_estimators':[50, 100, 200, 500,1000],
             'learning_rate':[0.01, 0.1, 0.5, 1]
            }
```

In [104]:

```
tune(model_ada, parameters, X_train_all, y_train_all, n_jobs=-1)
```

```
Best parameters: {'base_estimator': DecisionTreeClassifier(max_depth=4),
'learning_rate': 0.01, 'n_estimators': 1000}, best cv score: 0.53847791165
51538
```

Out[104]:

```
{'base_estimator': DecisionTreeClassifier(max_depth=4),
 'learning_rate': 0.01,
 'n_estimators': 1000}
```

In [50]:

```
ada_num_params= {'base_estimator': DecisionTreeClassifier(max_depth=4), 'learning_rate'
: 0.01, 'n_estimators': 1000}
model_ada_num = AdaBoostClassifier(**ada_num_params)
```

In [51]:

```
model_ada_num.fit(X_train_all, y_train_all)
```

Out[51]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=4),
                   learning_rate=0.01, n_estimators=1000)
```

In [54]:

```
# time
time_measure(model_ada_num)
```

Out[54]:

```
(29.21919255256653, 0.10798444327870389)
```

v. ensemble and create submission

In [55]:

```
model_ens_numv1 = VotingClassifier(estimators=[('xgb', model_xgb_num), ('rf', model_rf_
num),('ada', model_ada_num)], voting='soft')
```

In [56]:

```
create_submission(model_ens_numv1, X_train_all, y_train_all, X_test, 'ens_numv1_1', Tru
e)
```

Out[56]:

```
2    0.283201
3    0.244396
1    0.243673
4    0.228730
Name: price, dtype: float64
```

In [57]:

```
model_ens_numv2 = VotingClassifier(estimators=[('xgb', model_xgb_num), ('rf', model_rf_
num),('cat', model_cat_num)], voting='soft')
create_submission(model_ens_numv2, X_train_all, y_train_all, X_test, 'ens_numv2', True)
```

Out[57]:

```
2    0.270427
1    0.249940
3    0.243432
4    0.236201
Name: price, dtype: float64
```

In [58]:

```
model_ens_numv3 = VotingClassifier(estimators=[('xgb', model_xgb_num),('cat', model_cat
_num)], voting='soft')
create_submission(model_ens_numv3, X_train_all, y_train_all, X_test, 'ens_numv3', True)
```

Out[58]:

```
2    0.265606
1    0.252109
4    0.241745
3    0.240540
Name: price, dtype: float64
```

In [59]:

```
model_ens_numv4 = VotingClassifier(estimators=[('xgb', model_xgb_num),('rf', model_rf_n
um)], voting='soft')
create_submission(model_ens_numv4, X_train_all, y_train_all, X_test, 'ens_numv4', True)
```

Out[59]:

```
2    0.279103
1    0.246806
3    0.241745
4    0.232345
Name: price, dtype: float64
```

In [145]:

```
model_ens_numv5 = VotingClassifier(estimators=[('xgb', model_xgb_v2), ('rf', model_rf_n
um)], voting='soft')
create_submission(model_ens_numv5, X_train_all, y_train_all, X_test, 'ens_numv5', True)
```

Out[145]:

```
2    0.281514
1    0.247771
3    0.245601
4    0.225114
Name: price, dtype: float64
```

In [ ]:

```

```

In [ ]:

```

```