Kaggle InClass Competition Report: Airbnb Availability Data

Fangzhu Shen
Kaggle Username: `fangzhushen`
Duke University

1 Exploratory Analysis

As we can see, there are 20 features in the dataset and they can be divided to three types:continuous features, categorical features and binary features. And 4 features among them contains missing values.

- **Continuous features**: There are 10 continuous features in total in this dataset. I used the function `StandardScaler` to do standardization on them to eliminate effects of different scales. Continuous features are normalized to have zero mean and unit variance. The original distributions are shown in Figure 1:
- **Categorical features**: There are 6 categorical features. Histograms of them are shown in Figure 2. Since the feature `Property_type` contains too many categories, we can generalized and reduce the number of categories to three: `Entire type, Private type, Others`. The distribution after the process is illustrated in Figure 3.
  One-hot encoding is used for processing categorical features, because many machine learning algorithms can not deal with label data directly. I use the function `get_dummies()` to convert them into numerical values.
- **Binary features**: There are 4 binary features whose values are "t" or "f". I converted "f" to 0 and "t" to 1. Their distributions are shown in Figure 4.
  Since the label of the feature `Host_has_profile_pic` is all "t", it does not provide useful information for `Decision`. `Host_has_profile_pic` will be dropped during the data processing.
- **Feature Selections**: Since there are up to 20 features in the dataset, checking the multicollinearity is necessary. The correlation relationships of all features except `Host_has_profile_pic` are shown by heatmap in Figure 5.
  Because the categorical feature `Bedrooms_text` contains too many categories, I tentatively drop this feature to see correlation relationships of other features(Figure 6).
  Apparently, `Accommodates, Beds, Bedrooms` have high collinearity, so I drop `Beds` and `Bedrooms`. Moreover, collinearities between dummy varibles generated by `Month` is significantly high, so `Month` will also be dropped. I tried to drop feature `Property_type`, as it shows high collinearities with others. But the multicollinearity became worse after the deletion(Figure 7).
- **Missing values**: Finally 4 features are dropped(Host_has_profile_pic, Beds, Bedrooms, Month). There still are two features contains missing values: `Host_response_time` and `Review_scores_rating`. I created a label "missing" to fill NaN values in `Host_response_time` and picked the median value of `Review_scores_rating` to fill NaN values in it.

2 Methods

2.1 Models

I used three models: SVM, random forest and gradient boosting. SVM and random forests are implemented using `sklearn`. Gradient boosting trees are implemented with the `XGBoost` library.

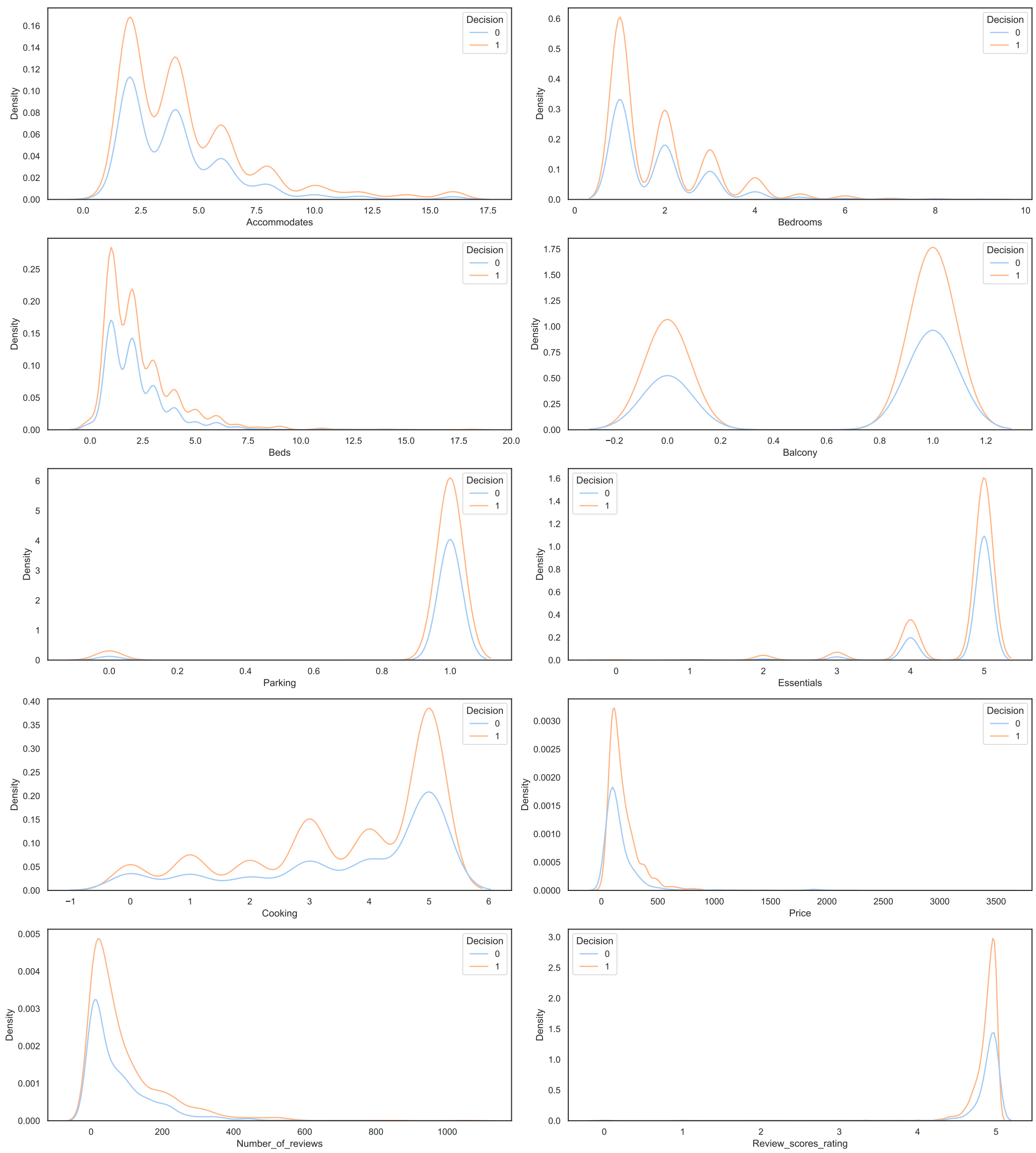


Figure 1: Distributions of Continuous Features

- **Support Vector Machine:** I firstly proposed SVM algorithm, since it is a good method to construct nonlinear classifier by applying the kernel trick in high dimensional spaces. Here I we convert categorical features to one-hot representations, features become very sparse, so SVM should performs well and efficiently. SVM might be more robust since the model only depends on support vectors, not whole training datasets.

- **Random Forest:** Random forest also performs well in high dimensional spaces. In addition, it can handle categorical features, binary features and numerical features, and it does not need the data to be re-scaled, so the pre-processing is easier. Because random forest averages all decision trees, so the bias and variances is low and it is unlikely to overfit.

- **Gradient Boosting:** I use XGBoost library, which is an implementation of gradient boosted decision trees created by Tianqi Chen. It performs well in both computational speed and model performance, so it is suitable for a large training dataset. Besides, XGBoost should perform well for a mixture of categorical and numeric features.

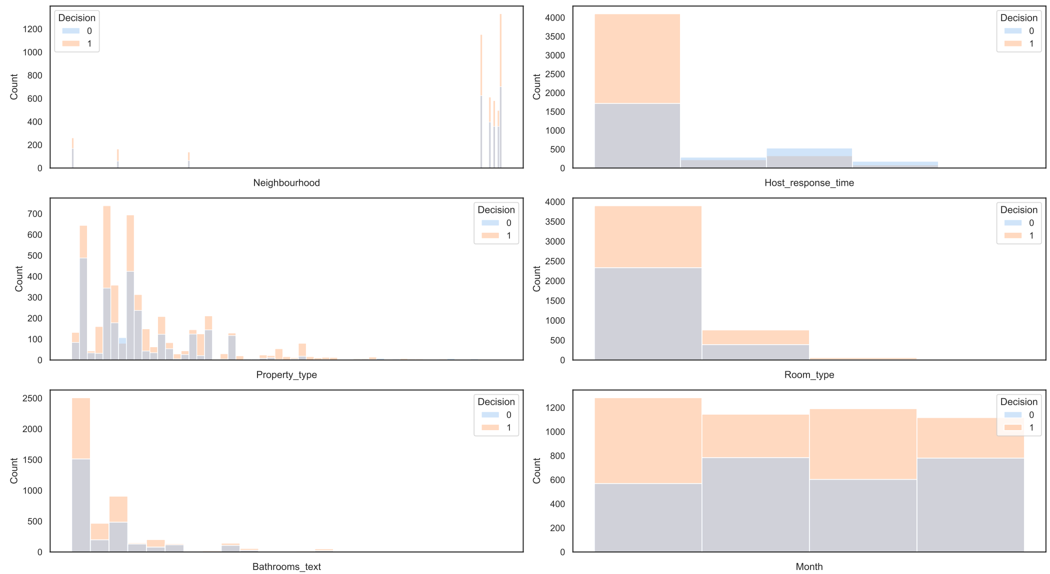


Figure 2: Distributions of Categorical Features

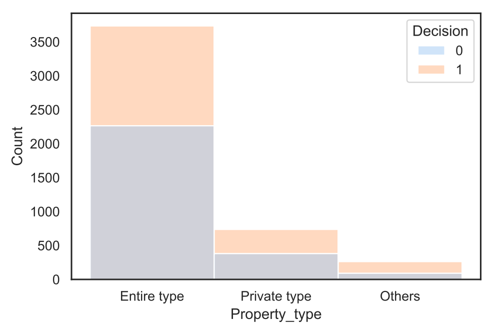


Figure 3: Distributions of Property_type after process

2.2 Training

We describe the training procedure and the profiled training time of each model in this section.

2.2.1 Random Forest

The random forest is often a strong baseline for the binary classification with these combination of categorical and numerical features. Random forest is an ensemble model that each tree is built from a sample drawn (with replacement) from the training set. Compared to a single decision tree, random can reduce the variance, since a single decision tree can easily exhibit high variance and overfit.

For training random forest with number of estimators set to 1000 and maximum depth set to 23, the training takes about 11.8sec on a laptop with Intel Core i7-7820HQ CPU on the full training set provided.

2.2.2 SVM

We have talked about SVM in class. SVM classifier for binary classification is quite intuitive and has been well studied in class. It simply finds the hyperplane that can maximally separate the two classes

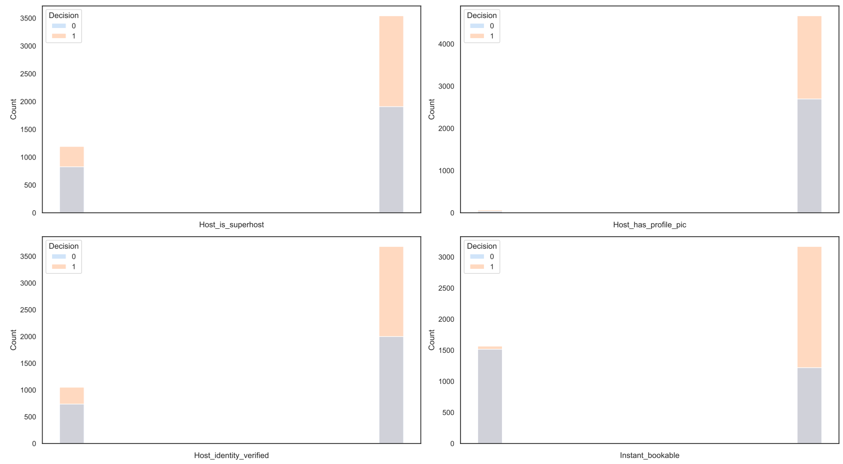


Figure 4: Distributions of Binary Features

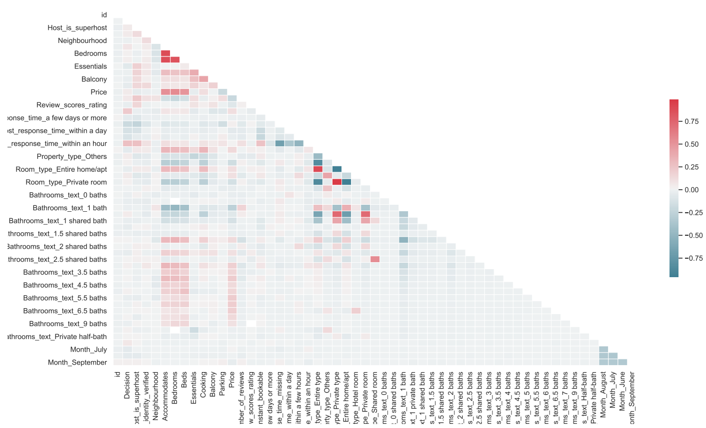


Figure 5: Correlation Heatmap between features1

of data points. SVM would seek the maximum margin. It solves the following problem[1]:

$$\begin{gathered} \min_{w,b,\zeta} \frac{1}{2}w^T w + C\sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i\left(w^T\phi\left(x_i\right)+b\right) \geq 1-\zeta_i \\ \zeta_i \geq 0, i=1,\ldots,n \end{gathered}$$

For training SVM with RBF kernel and $C = 0.1$, it takes about 8.5sec to train on a laptop with Intel Core i7-7820HQ CPU.

2.2.3 Gradient Boosting

[2] Gradient boosted trees are just an ensemble of decision trees. In the ensemble, the prediction will be generated via the "collective wisdom of the crowd":

$$\hat{y}_i = \sum_{k=1}^{K} f_k\left(x_i\right), f_k \in \mathcal{F},$$

[1]The equations are borrowed from: `https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation`

[2]See XGBoost's document: `https://xgboost.readthedocs.io/en/latest/tutorials/model.html`

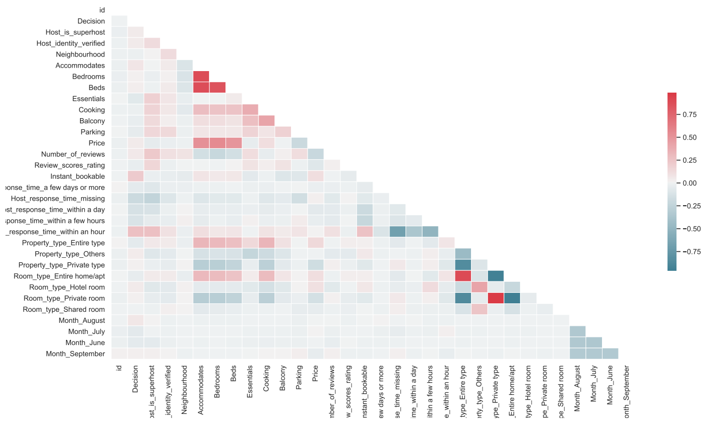


Figure 6: Correlation Heatmap between features2

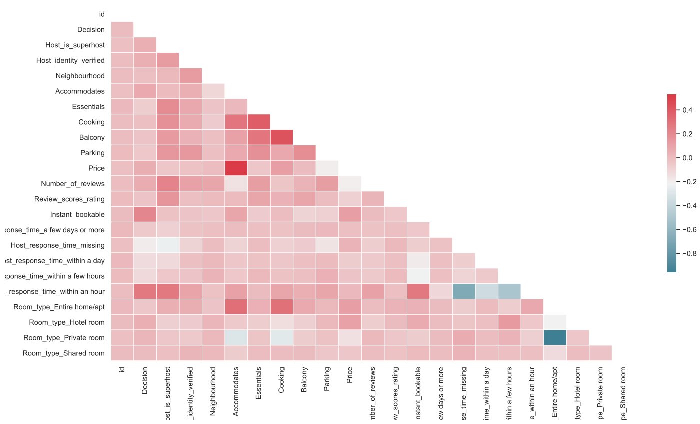


Figure 7: Correlation Heatmap between features3

where K is the number of trees. $\mathcal{F}$ is the set of all possible trees. The objective function is then given by:

$$\text{obj}(\theta) = \sum_{i}^{n} l\left(y_i, \hat{y}_i\right) + \sum_{k=1}^{K} \Omega\left(f_k\right)$$

However, it is intractable to learn all the trees at once, as learning the (non-parametric) tree struct is much harder than traditional optimization problem which we can rely on graident. Instead, we incrementally add one new tree at a time. We denote the prediction value at time step t as $\hat{y}_i^{(t)}$. The objective function becomes:

$$\begin{aligned}
\text{obj}^{(t)} &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega\left(f_i\right) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t\left(x_i\right)\right) + \Omega\left(f_t\right) + \text{ constant }.
\end{aligned}$$

We then take the Taylor expansion of the loss function to the second order. As seen in the objective function, We impose a regularization score on each of the tree, to control the model complexity. Since it is also infeasible to enumerate all possible trees and pick the best one, we result to optimize one level of the tree at a time. Each time, we split a leaf into the two leaves and compute the gain.

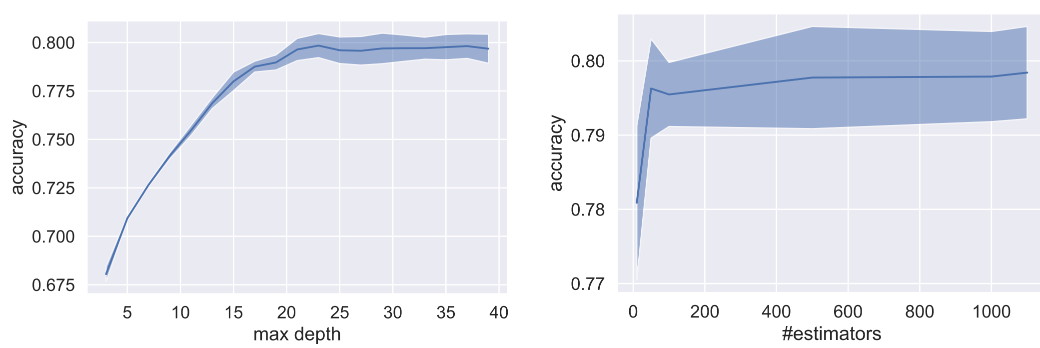
For training of gradient boosting (XGBoost) with 1100 estimators and maximum depth is set to 11, it takes about 23.9sec to train a laptop with Intel Core i7-7820HQ CPU.

2.3 Hyper-parameter Selection

Cross-validation is used to tune hyper-parameters. We use 3-fold cross validation on the provided training set.

2.3.1 Random Forest

Random forest seems to be a very performant model on this dataset. We mainly tune the number of estimators and the maximum depth for random forest to improve its performance. The maximum depth is searched between 3 and 40 with a step of 2. For the number of estimators, we consider it in $[10, 50, 100, 500, 1000, 1100]$. We tune one of the parameters at a time. We first tune the maximum depth with number of estimators set to 1100, and then we tune the number of estimators by setting the maximum depth to the optimal one obtained during the previous step tuning (with max depth set to 23). The results is shown in Figure 8.



(a) Maximum depth

(b) Number of estimators

Figure 8: Hyper-parameter tuning of random forests.

2.3.2 SVM

For SVM, we tune the parameter C. RBF kernel is used. The results are shown in Figure 9.

2.3.3 Gradient Boosting

For gradient boosting (XGBoost), we tune the maximum depth. The maximum depth is tuned within the range of 3 to 15 (excluding 15) with a step of 2. The results are illustrated in Figure 10. For other parameters, we set learning rate to 0.1, minimal child weight is set to 1, gamma is set to 1, number of estimators is set to 1100.

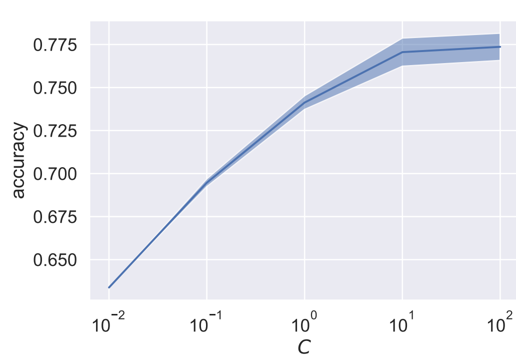


Figure 9: Hyper-parameter tuning of SVM.

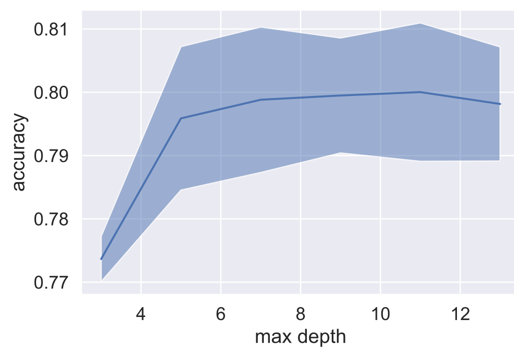


Figure 10: Hyper-parameter tuning of Gradient boosting.

3 Results

3.1 Prediction

3.1.1 Leaderboard Score

The best learderboard accuracy we obtain is 0.28005 (MAE), with random forests.

3.1.2 Validation Accuracy

Here we list the mean validation accuracy (over 3 validation sets, since we use 3-fold CV) of the different models under the best parameters:

- Random Forest: 0.79842
- SVM: 0.77366
- XGBoost: 0.80003

3.2 Fixing Mistakes

- We originally include `Month` (and other things) as a feature. After we exclude `Bedrooms, Beds, Month, Host_has_profile_pic`, the score on Kaggle become better.
- How to properly deal with categorical features is hard.

Code

Data Analysis

In [114…
```
###for the explotary analysis section
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

In [117…
```
columns = ['id', 'Decision', 'Host_response_time',
           'Host_is_superhost', 'Host_has_profile_pic', 'Host_identity_verified'
           'Property_type', 'Room_type', 'Accommodates','Bathrooms_text',
           'Bedrooms', 'Beds',
           'Essentials', 'Cooking',
           'Balcony', 'Parking',
           'Price',
           'Number_of_reviews',
           'Review_scores_rating',
           'Instant_bookable',
           'Month']
#we can split features like below:
categorical = ['Neighbourhood', 'Host_response_time','Property_type', 'Room_type
continuous = ['Accommodates','Bedrooms', 'Beds','Balcony', 'Parking','Essentials
              'Price','Review_scores_rating','Number_of_reviews']
binary = ['Host_is_superhost', 'Host_has_profile_pic', 'Host_identity_verified',
```

In [118…
```
#import data and convert bool features to numerical
bool_converter = lambda x: 1 if x == 't' else 0
train_df = pd.read_csv(
    "duke-cs671-fall21-airbnb-availability-data/train.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    },
)

test_df = pd.read_csv(
    "duke-cs671-fall21-airbnb-availability-data/test.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    }
)
```

In [119…
```
train_df
```

Out[119…

	id	Decision	Host_response_time	Host_is_superhost	Host_has_profile_pic	Host_identi
0	1	1	within an hour	1	1	

	id	Decision	Host_response_time	Host_is_superhost	Host_has_profile_pic	Host_identi
1	2	1	within an hour	1	1	
2	3	0	within a few hours	1	1	
3	4	1	within an hour	1	1	
4	5	0	within an hour	1	1	
...	...	...	...	...	...	
7466	7467	1	within an hour	0	1	
7467	7468	1	within an hour	1	1	
7468	7469	1	within an hour	1	1	
7469	7470	1	within an hour	1	1	
7470	7471	1	within an hour	1	1	

7471 rows × 22 columns

In [120…

```
test_df
```

Out[120…

	id	Host_response_time	Host_is_superhost	Host_has_profile_pic	Host_identity_verified
0	1	NaN	1	1	1
1	2	within an hour	1	1	1
2	3	within an hour	1	1	1
3	4	within an hour	1	1	1
4	5	within an hour	1	1	1
...	...	...	...	...	...
2435	2436	within an hour	0	1	0
2436	2437	within an hour	1	1	1
2437	2438	within a few hours	1	1	1

	id	Host_response_time	Host_is_superhost	Host_has_profile_pic	Host_identity_verified
2438	2439	within an hour	1	1	1
2439	2440	within an hour	1	1	1

2440 rows × 21 columns

In [121…

```
#check the nan values
print(train_df.isna().sum())
print(test_df.isna().sum())
#I find there are nan values in "Host_response_time","Bedrooms","Review_scores_r
```

```
id                          0
Decision                    0
Host_response_time        858
Host_is_superhost           0
Host_has_profile_pic        0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Bedrooms                  585
Beds                       13
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      395
Instant_bookable            0
Month                       0
dtype: int64
id                          0
Host_response_time        293
Host_is_superhost           0
Host_has_profile_pic        0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Bedrooms                  149
Beds                        9
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      274
Instant_bookable            0
Month                       0
dtype: int64
```

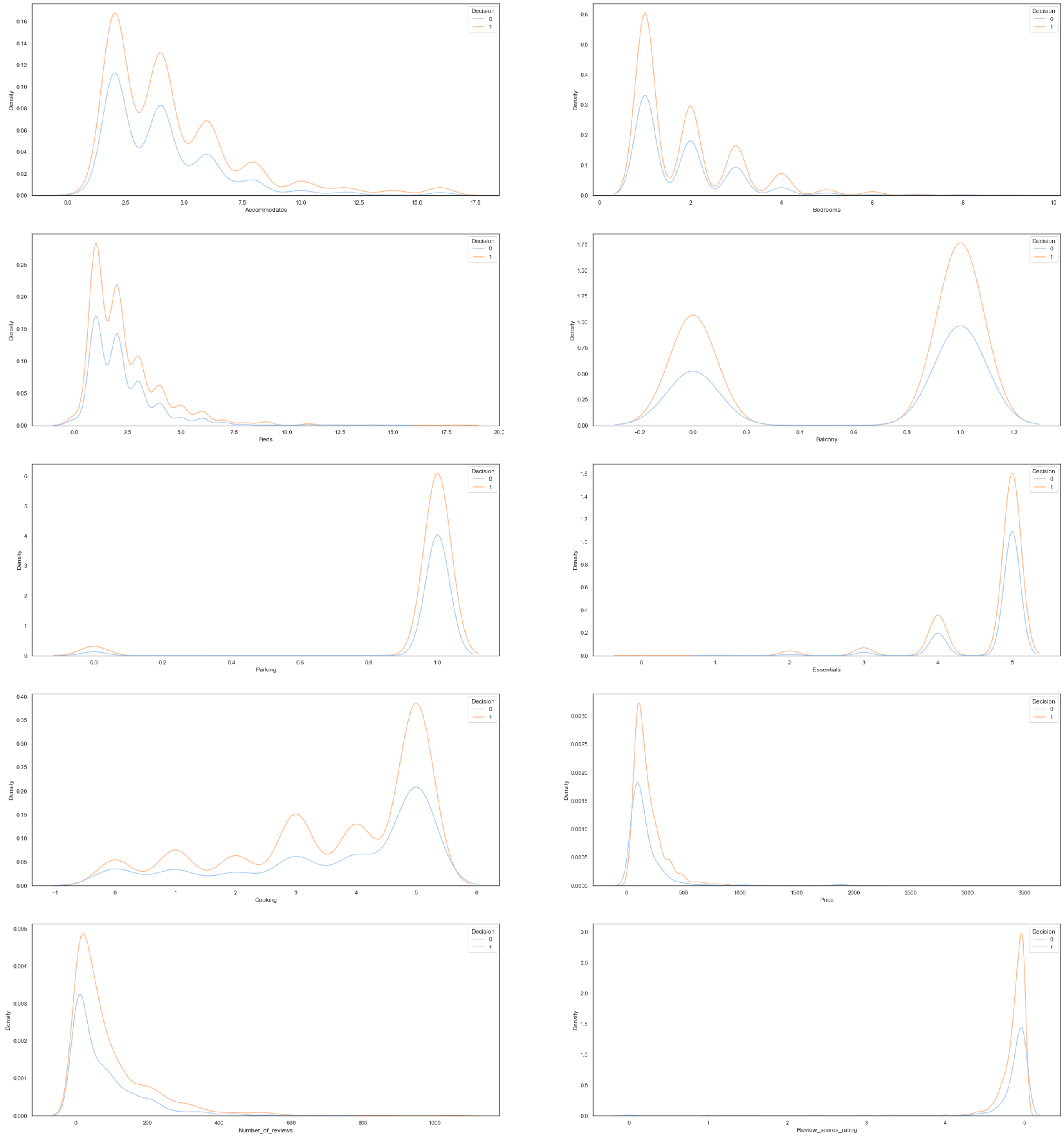
In [122…

```
#process price features: str to numerical
train_df['Price'] = train_df['Price'].replace({'\$':'',',':''},regex = True)
train_df['Price'] = train_df['Price'].astype('float')

test_df['Price'] = test_df['Price'].replace({'\$':'',',':''},regex = True)
test_df['Price'] = test_df['Price'].astype('float')
```

In [130…

```
#continuous features: draw kde plots
# draw kde plots for some of the continuous features
fig, ax = plt.subplots(5, 2,
                       figsize=(18,20))
sns.kdeplot(data=train_df, x='Accommodates', hue='Decision', palette='pastel', a
sns.kdeplot(data=train_df, x='Bedrooms', hue='Decision', palette='pastel', ax=ax
sns.kdeplot(data=train_df, x='Beds', hue='Decision', palette='pastel', ax=ax[1,0
sns.kdeplot(data=train_df, x='Balcony', hue='Decision', palette='pastel', ax=ax[
sns.kdeplot(data=train_df, x='Parking', hue='Decision', palette='pastel', ax=ax[
sns.kdeplot(data=train_df, x='Essentials', hue='Decision', palette='pastel', ax=
sns.kdeplot(data=train_df, x='Cooking', hue='Decision', palette='pastel', ax=ax[
sns.kdeplot(data=train_df, x='Price', hue='Decision', palette='pastel', ax=ax[3,
sns.kdeplot(data=train_df, x='Number_of_reviews', hue='Decision', palette='paste
sns.kdeplot(data=train_df, x='Review_scores_rating', hue='Decision', palette='pa
#fig.tight_layout()
fig.savefig("figure/eda_continuous.pdf")
```



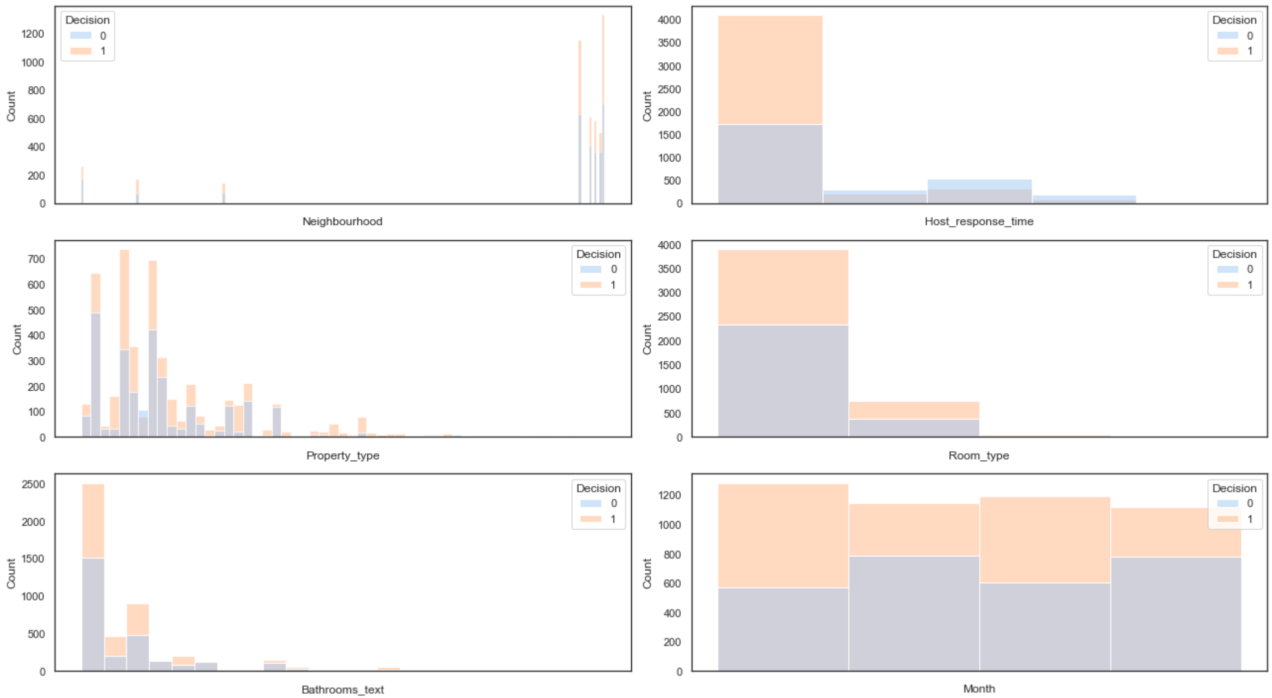
In [131…
```
plt.close()
```

In [132…
```
#then, visualize the categorical features. Firstly, preprocess the feature with
#fill nan value
train_df["Host_response_time"] = train_df["Host_response_time"].fillna(value="mi
test_df["Host_response_time"] = test_df["Host_response_time"].fillna(value="miss
```

In [133…
```
#categorical features: draw histograms
categorical = ['Neighbourhood', 'Host_response_time','Property_type', 'Room_type
fig, ax = plt.subplots(3, 2, figsize=(18, 10))
h1 = sns.histplot(data=train_df, x='Neighbourhood', hue='Decision', palette='pas
h2 = sns.histplot(data=train_df, x='Host_response_time', hue='Decision', palette
h3 = sns.histplot(data=train_df, x='Property_type', hue='Decision', palette='pas
h4 = sns.histplot(data=train_df, x='Room_type', hue='Decision', palette='pastel'
h5 = sns.histplot(data=train_df, x='Bathrooms_text', hue='Decision', palette='pa
h6 = sns.histplot(data=train_df, x='Month', hue='Decision', palette='pastel', ax
```

```
h1.set(xticklabels=[])
h2.set(xticklabels=[])
h3.set(xticklabels=[])
h4.set(xticklabels=[])
h5.set(xticklabels=[])
h6.set(xticklabels=[])
h6.set(xticklabels=[])

fig.tight_layout()
fig.savefig("figure/eda_categorical.pdf")
```



In [134…
```
plt.close()
```

From figures, we can see that features:"Property_type" have too many labels. Considering generalization

In [135…
```
train_df.Property_type.value_counts()
```

Out[135…
```
Entire house                          1132
Entire guest suite                    1118
Entire residential home               1082
Entire apartment                       550
Entire rental unit                     536
Private room in house                  355
Private room in residential home       331
Entire cottage                         269
Entire guesthouse                      246
Entire cabin                           216
Entire condominium                     192
Entire condominium (condo)             192
Entire bungalow                        187
Private room in bed and breakfast      146
Entire townhouse                       137
Tiny house                              98
Entire loft                             98
Private room in bungalow                77
Private room in guest suite             70
```

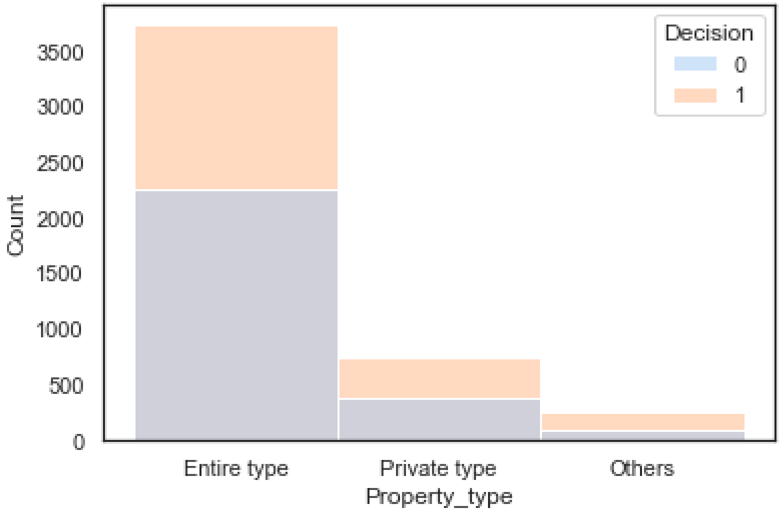
```
Room in bed and breakfast                   56
Room in boutique hotel                      34
Campsite                                    34
Entire chalet                               32
Camper/RV                                   32
Private room                                20
Private room in apartment                   16
Farm stay                                   16
Private room in treehouse                   16
Shared room in hostel                       16
Private room in rental unit                 15
Bus                                         12
Private room in farm stay                   12
Private room in hostel                      12
Treehouse                                   12
Yurt                                        11
Entire place                                 8
Private room in townhouse                    8
Private room in guesthouse                   8
Private room in cabin                        8
Private room in camper/rv                    8
Tent                                         8
Room in hotel                                7
Casa particular (Cuba)                       4
Private room in castle                       4
Shipping container                           4
Entire villa                                 4
Private room in hut                          4
Private room in cottage                      4
Private room in condominium (condo)          2
Shared room in apartment                     2
Shared room in rental unit                   2
Shared room in residential home              2
Private room in condominium                  2
Shared room in house                         2
Casa particular                              2
Name: Property_type, dtype: int64
```

In [136…

```
train_df.Property_type = train_df.Property_type.apply(lambda x: 'Private type' i
train_df.Property_type = train_df.Property_type.apply(lambda x: 'Entire type' if
train_df.loc[~train_df.Property_type.isin(['Entire type', 'Private type']), 'Pro
```

In [138…

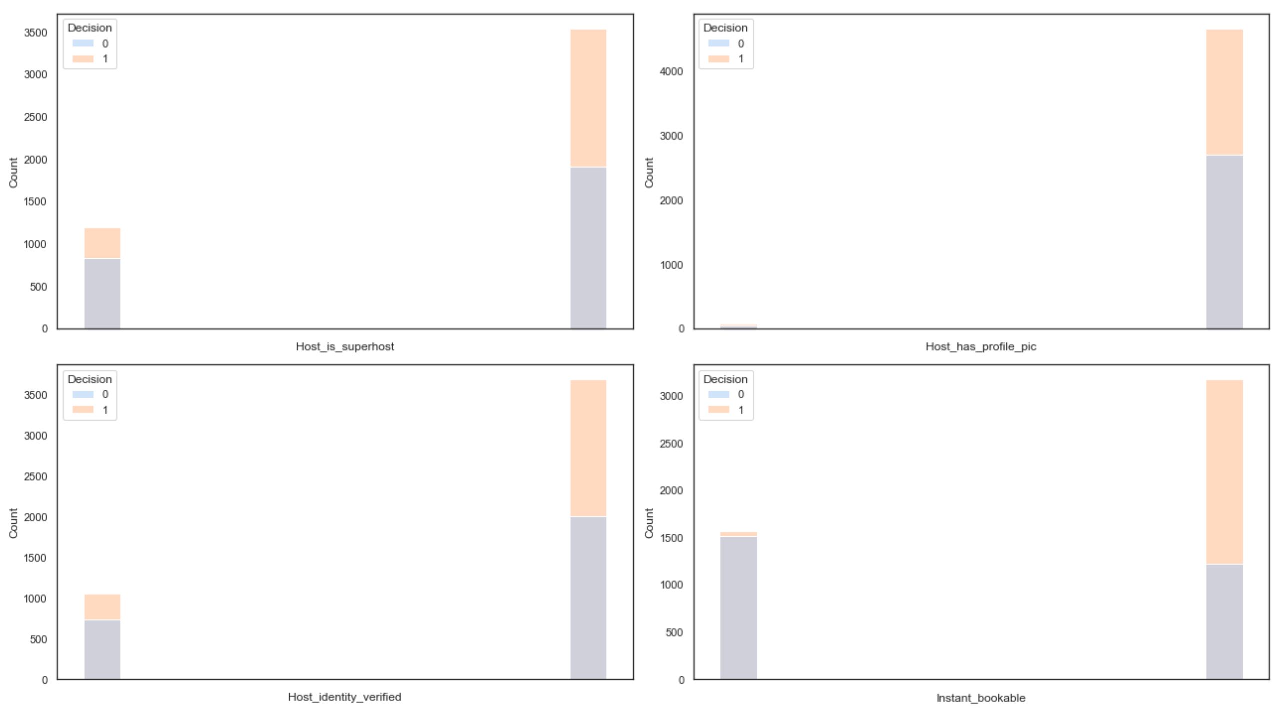
```
#train_df['Property_type'].hist()
fig = plt.figure()
h1 = sns.histplot(data=train_df, x='Property_type', hue='Decision', palette='pas
fig.savefig("figure/eda_categorical_property_type_transform.pdf")
```



In [139…
```
#binary festures: histograms
fig, ax = plt.subplots(2, 2, figsize=(18, 10))
h1 = sns.histplot(data=train_df, x='Host_is_superhost', hue='Decision', palette=
h2 = sns.histplot(data=train_df, x='Host_has_profile_pic', hue='Decision', palet
h3 = sns.histplot(data=train_df, x='Host_identity_verified', hue='Decision', pal
h4 = sns.histplot(data=train_df, x='Instant_bookable', hue='Decision', palette='

h1.set(xticklabels=[])
h2.set(xticklabels=[])
h3.set(xticklabels=[])
h4.set(xticklabels=[])

fig.tight_layout()
fig.savefig("figure/eda_binary.pdf")
```



In [140…
```
plt.close()
```

In [141…
```
#drop features"Host_has_profile_pic"
train_df = train_df.drop(columns=['Host_has_profile_pic'])
```

```python
test_df = test_df.drop(columns=['Host_has_profile_pic'])
```

In [142…

```python
#normalization contibuous features
scaler = StandardScaler()
scaler.fit(train_df[continuous])
train_df[continuous] = scaler.transform(train_df[continuous])
test_df[continuous] = scaler.transform(test_df[continuous])
```

In [143…

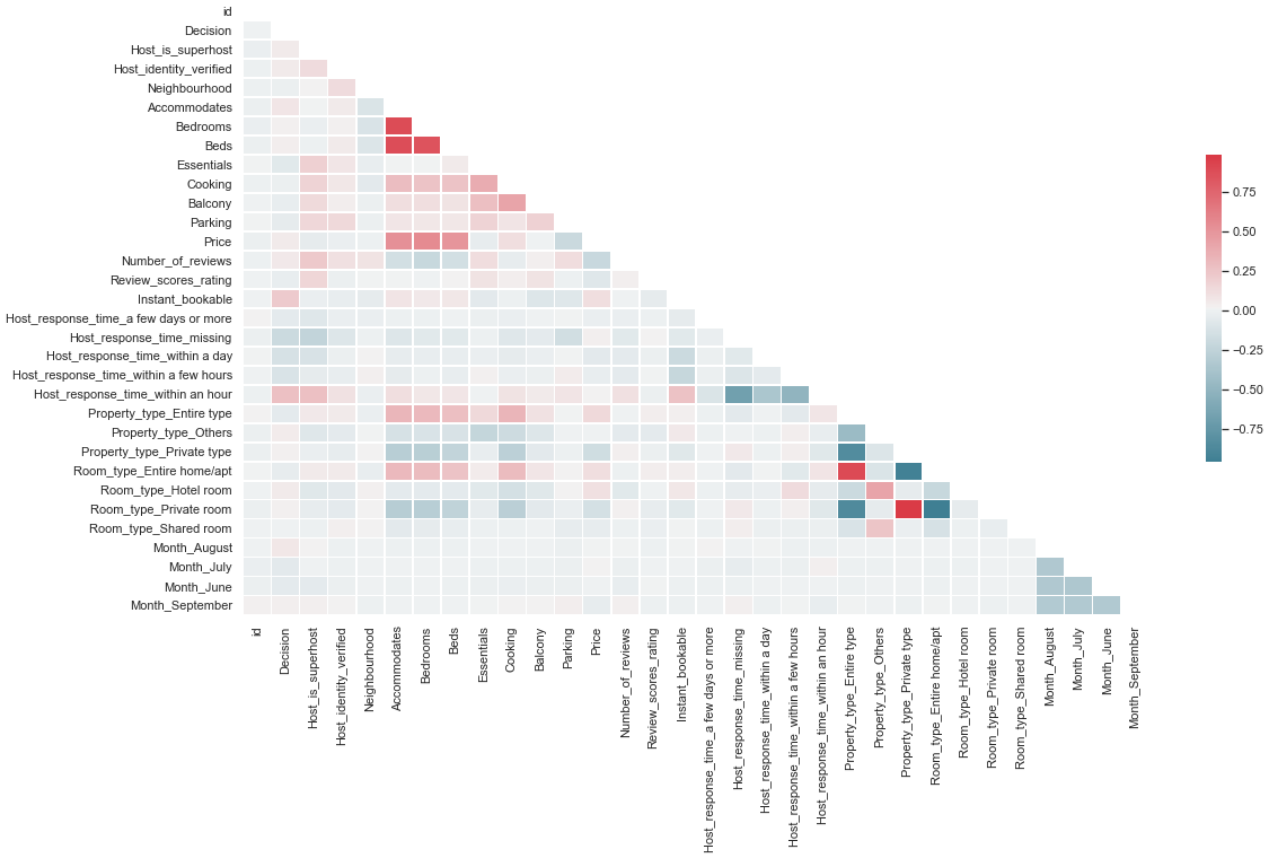
```python
#draw heatmap to see multi colinearity correlation between features, doing featu
def draw_heatmap(df,save_name):
    sns.set(style="white")
    corr = df.corr()
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    #f, ax = plt.subplots(figsize=figsize)
    fig = plt.figure(figsize=(18,10))
    cmap = sns.diverging_palette(220, 10, as_cmap=True)
    sns.heatmap(corr, mask=mask, cmap=cmap,linewidths=0.5,cbar_kws={"shrink": .5
    plt.savefig("figure/"+save_name)
```

In [144…

```python
transformed_df = pd.get_dummies(train_df)
draw_heatmap(transformed_df,'heatmap_1.pdf')
```

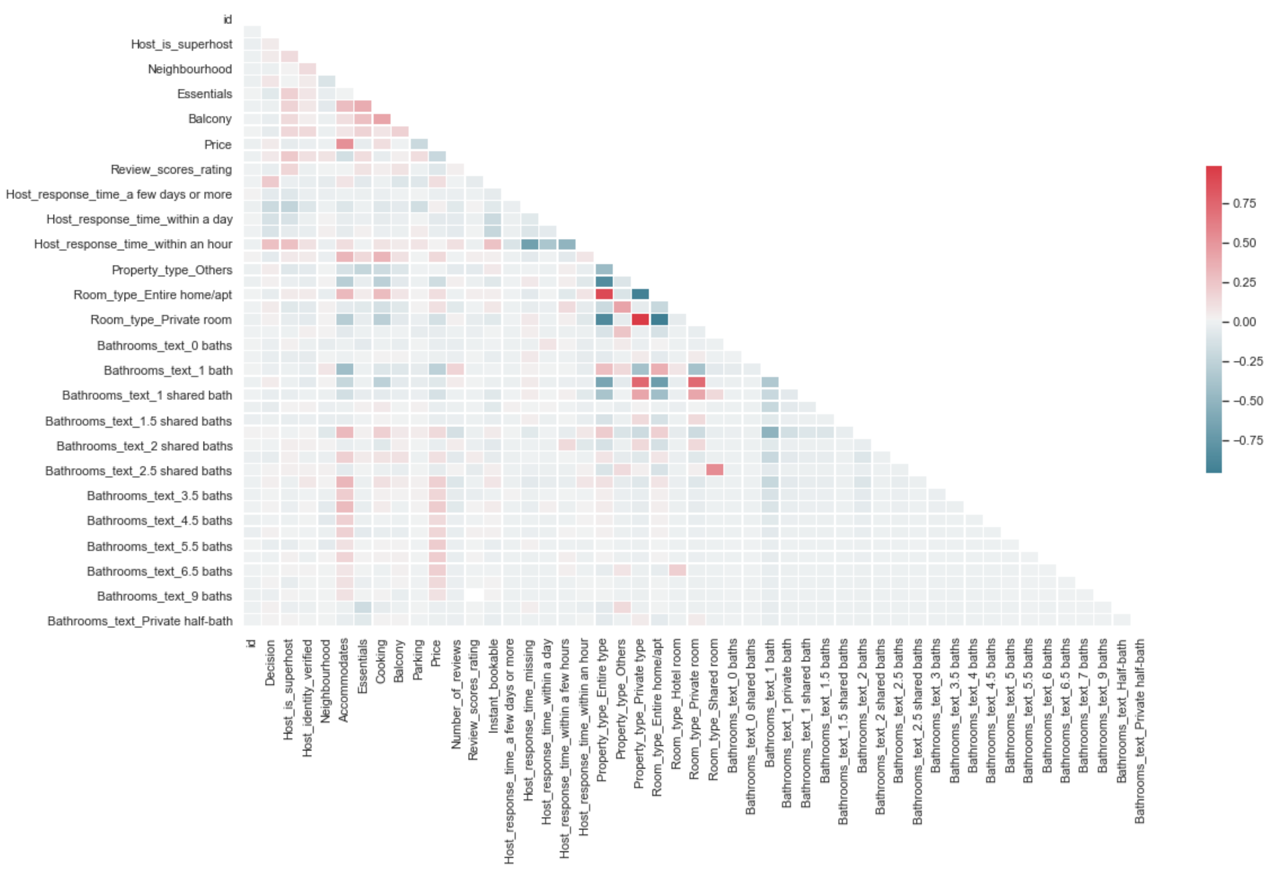
In [145…

```python
train_df2 = train_df.drop(columns=['Bathrooms_text'])
transformed_df = pd.get_dummies(train_df2)
draw_heatmap(transformed_df,"heatmap_2.pdf")
```

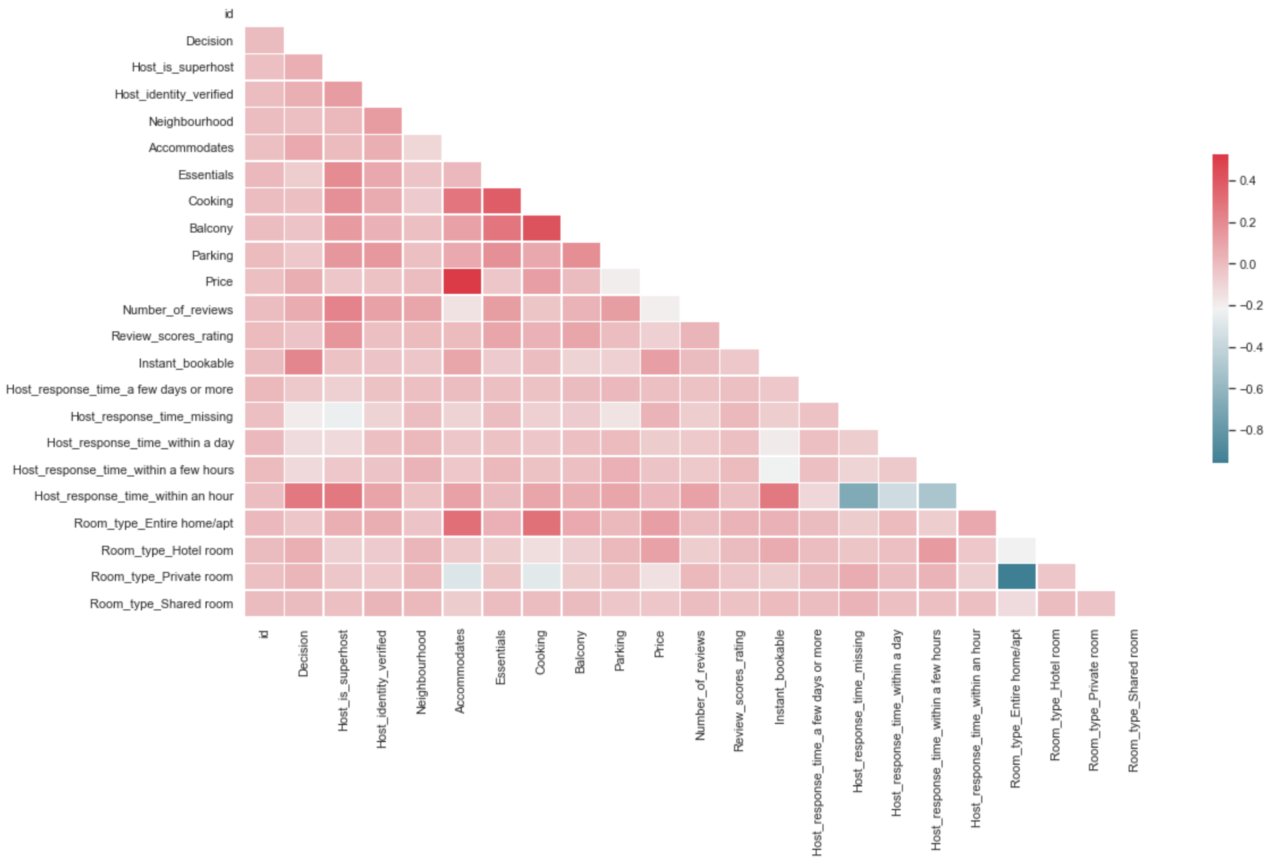


In [146…

```python
#delete: Bedrooms, Beds,Property_type,Month
train_df3 = train_df.drop(columns=['Bedrooms', 'Beds','Month'])
transformed_df = pd.get_dummies(train_df3)
draw_heatmap(transformed_df,"heatmap_3.pdf")
```



In [147…
```
#delete: Bedrooms, Beds,Property_type
train_df4 = train_df.drop(columns=['Bedrooms', 'Beds','Month','Property_type','B
transformed_df = pd.get_dummies(train_df4)
draw_heatmap(transformed_df,"heatmap_4.pdf")
```



In [148…
```
#so it is improper to drop "property_type"
```

In [149…
```
#Therefore we drop 4 features in total: 'Bedrooms', 'Beds','Month''Host_has_prof
train_df = train_df.drop(columns = ['Bedrooms', 'Beds','Month'])
test_df = test_df.drop(columns = ['Bedrooms', 'Beds','Month'])
```

In [150…
```
train_df.isna().sum()
```

Out[150…
```
id                        0
Decision                  0
Host_response_time        0
Host_is_superhost         0
Host_identity_verified    0
Neighbourhood             0
Property_type             0
Room_type                 0
Accommodates              0
Bathrooms_text            0
Essentials                0
Cooking                   0
Balcony                   0
Parking                   0
Price                     0
Number_of_reviews         0
Review_scores_rating    395
Instant_bookable          0
dtype: int64
```

In [151…

```
test_df.isna().sum()
```

Out[151…

```
id                           0
Host_response_time           0
Host_is_superhost            0
Host_identity_verified       0
Neighbourhood                0
Property_type                0
Room_type                    0
Accommodates                 0
Bathrooms_text               0
Essentials                   0
Cooking                      0
Balcony                      0
Parking                      0
Price                        0
Number_of_reviews            0
Review_scores_rating       274
Instant_bookable             0
dtype: int64
```

In [104…

In [ ]:

Random Forest

In [75]:
```
#to process data
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
import time
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(42)
```

In [76]:
```
#import data and convert bool features to numerical
bool_converter = lambda x: 1 if x == 't' else 0
train_df = pd.read_csv(
    "train.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    },
)

test_df = pd.read_csv(
    "test.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    }
)
```

In [77]:
```
#We drop 4 features in total: 'Bedrooms', 'Beds','Month''Host_has_profile_pic'
train_df = train_df.drop(columns = ['Bedrooms', 'Beds','Month','Host_has_profile
test_df = test_df.drop(columns = ['Bedrooms', 'Beds','Month','Host_has_profile_p
```

In [78]:
```
columns = ['id', 'Decision', 'Host_response_time',
           'Host_is_superhost', 'Host_has_profile_pic', 'Host_identity_verified'
           'Property_type', 'Room_type', 'Accommodates','Bathrooms_text',
           'Bedrooms', 'Beds',
           'Essentials', 'Cooking',
           'Balcony', 'Parking',
           'Price',
           'Number_of_reviews',
           'Review_scores_rating',
           'Instant_bookable',
           'Month']
#we can split features like below:
categorical = ['Neighbourhood', 'Host_response_time','Property_type', 'Room_type
continuous = ['Accommodates','Balcony', 'Parking','Essentials', 'Cooking',
              'Price','Review_scores_rating','Number_of_reviews']
binary = ['Host_is_superhost', 'Host_identity_verified','Instant_bookable']
```

In [79]:
```
#fill nan values
train_df.isna().sum()
```

Out[79]:
```
id                          0
Decision                    0
Host_response_time        858
Host_is_superhost           0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      395
Instant_bookable            0
dtype: int64
```

In [80]:
```
test_df.isna().sum()
```

Out[80]:
```
id                          0
Host_response_time        293
Host_is_superhost           0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      274
Instant_bookable            0
dtype: int64
```

In [81]:
```
train_df["Host_response_time"] = train_df["Host_response_time"].fillna(value="mi
test_df["Host_response_time"] = test_df["Host_response_time"].fillna(value="miss
```

In [82]:
```
#still have a feature with nan values: fill it with median
train_df["Review_scores_rating"].median()
```

Out[82]:
```
4.93
```

In [83]:
```
train_df["Review_scores_rating"] = train_df["Review_scores_rating"].fillna(value
test_df["Review_scores_rating"] = test_df["Review_scores_rating"].fillna(value=4
```

In [84]:
```
#price features: str to numerical
train_df['Price'] = train_df['Price'].replace({'\$':'',',':''},regex = True)
train_df['Price'] = train_df['Price'].astype('float')

test_df['Price'] = test_df['Price'].replace({'\$':'',',':''},regex = True)
test_df['Price'] = test_df['Price'].astype('float')
```

In [85]:
```
# continuous features require normalization
# normalize the continuous features to zero mean and unit varianceе
scaler = StandardScaler()
scaler.fit(train_df[continuous])
train_df[continuous] = scaler.transform(train_df[continuous])
test_df[continuous] = scaler.transform(test_df[continuous])
```

In [86]:
```
#split labels and features
y_train_full = train_df['Decision']
X_train_full = train_df.drop(['Decision'], axis=1)
```

In [87]:
```
# convert categorical features to one-hot representations
len_train = len(X_train_full)

total_X = X_train_full.append(test_df, ignore_index=True)
one_hot_X = pd.get_dummies(total_X, columns=categorical)
#split train and test dataset
X_test = one_hot_X[len_train:]
X_train_full = one_hot_X[:len_train]
```

In [88]:
```
X_test
```

Out[88]:

	id	Host_is_superhost	Host_identity_verified	Accommodates	Essentials	Cooking	E
7471	1	1	1	-1.192110	-1.171512	-0.385235	0.:
7472	2	1	1	-0.830126	0.447498	0.843931	0.:
7473	3	1	1	-0.830126	0.447498	0.843931	0.:
7474	4	1	1	-0.830126	0.447498	0.843931	0.:
7475	5	1	1	-0.830126	0.447498	0.843931	0.:
...	...	...	...	...	...	...	
9906	2436	0	0	0.617810	-6.028541	-2.228985	-1.
9907	2437	1	1	1.703762	0.447498	0.843931	0.:
9908	2438	1	1	3.513681	-1.171512	-1.614402	-1.
9909	2439	1	1	-0.830126	0.447498	-0.999819	-1.
9910	2440	1	1	-0.830126	-1.171512	-1.614402	-1.

2440 rows × 111 columns

In [89]:
```
X_train_full = X_train_full.drop(['id'], axis=1)
test_index =X_test['id']
#ids = ids.astype(np.int64)
X_test = X_test.drop(['id'], axis=1)
```

In [90]:
```
#split validation set
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, te
```

random forest

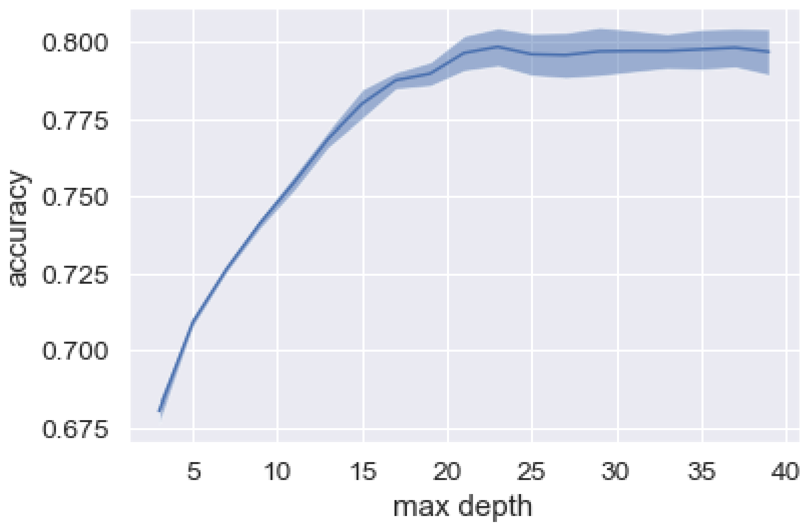
```
from sklearn.ensemble import RandomForestClassifier
```

In [91]:
```
from sklearn.model_selection import GridSearchCV
```

In [92]:
```
#hyperparameter selection
param_test1 = {
 'max_depth':range(3,40,2)
}
gsearch1 = GridSearchCV(estimator = RandomForestClassifier(n_estimators = 1100,r
 param_grid = param_test1, scoring='accuracy',n_jobs=4,cv=3)
gsearch1.fit(X_train_full, y_train_full)
print(gsearch1.best_params_)
print(gsearch1.best_score_)
```

```
{'max_depth': 23}
0.798420552274668
```

In [97]:
```
cv_results = gsearch1.cv_results_
params = list(range(3, 40, 2))
mean_test_score = cv_results["mean_test_score"]
std_test_score = cv_results["std_test_score"]
sns.set(font_scale = 1.25)
ax = sns.lineplot(x=params, y=mean_test_score)
ax.fill_between(params, y1=mean_test_score - std_test_score, y2=mean_test_score
ax.set_xlabel("max depth")
ax.set_ylabel("accuracy")
plt.savefig("rf_depth.pdf", bbox_inches="tight")
```

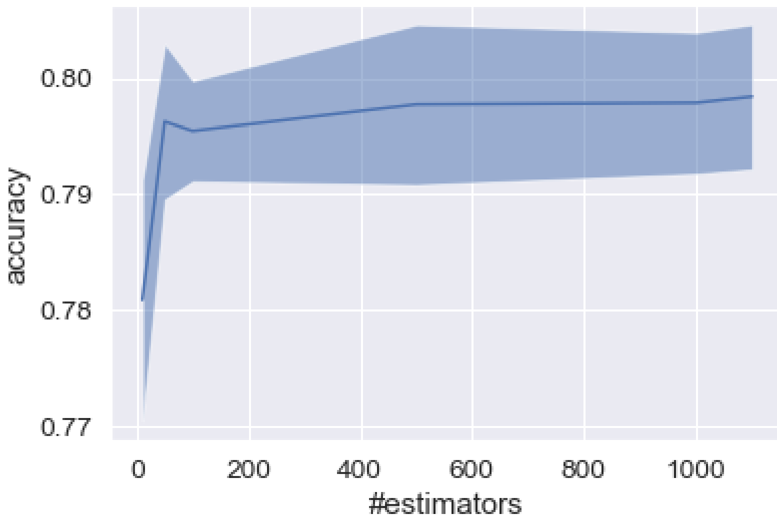


In [95]:
```
#hyperparameter selection
param_test2 = {
 'n_estimators':[10, 50, 100, 500, 1000, 1100]
}
gsearch2 = GridSearchCV(estimator = RandomForestClassifier(max_depth=23,random_s
 param_grid = param_test2, scoring='accuracy',n_jobs=4,cv=3)
gsearch2.fit(X_train_full, y_train_full)
print(gsearch2.best_params_)
print(gsearch2.best_score_)
```

```
{'n_estimators': 1100}
0.798420552274668
```

In [96]:
```
cv_results = gsearch2.cv_results_
params = [10, 50, 100, 500, 1000, 1100]
mean_test_score = cv_results["mean_test_score"]
std_test_score = cv_results["std_test_score"]
```

```
sns.set(font_scale = 1.25)
ax = sns.lineplot(x=params, y=mean_test_score)
ax.fill_between(params, y1=mean_test_score - std_test_score, y2=mean_test_score
ax.set_xlabel("#estimators")
ax.set_ylabel("accuracy")
plt.savefig("rf_num_estimators.pdf", bbox_inches="tight")
```



In [42]:
```
#score on validation dataset
clf_rf = RandomForestClassifier(n_estimators = 1100, max_depth=23,random_state=4
clf_rf.fit(X_train,y_train)
acc_train = clf_rf.score(X_val,y_val)
print("train acc: {:.4f}".format(acc_train))
```

```
train acc: 0.8155
```

In [98]:
```
#train on full train dataset
clf_rf = RandomForestClassifier(n_estimators = 1100, max_depth=23,random_state=4
t_start = time.time()
clf_rf.fit(X_train_full,y_train_full)
t_end = time.time()
acc_train = clf_rf.score(X_train_full,y_train_full)
print("train acc: {:.4f}".format(acc_train))
print("training time: {:.2f}".format(t_end - t_start))
```

```
train acc: 0.9653
training time: 11.82
```

In [101…
```
#do prefictiom
preds = clf_rf.predict(X_test)
results = pd.Series(preds, index=test_index)
results.to_csv("results/rf_1209.csv", header=['Decision'], index=True, index_lab
```

SVM

In [20]:
```
#to process data
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
import time

np.random.seed(42)
```

In [21]:
```
#import data and convert bool features to numerical
bool_converter = lambda x: 1 if x == 't' else 0
train_df = pd.read_csv(
    "train.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    },
)

test_df = pd.read_csv(
    "test.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    }
)
```

In [22]:
```
#We drop 4 features in total: 'Bedrooms', 'Beds','Month''Host_has_profile_pic'
train_df = train_df.drop(columns = ['Bedrooms', 'Beds','Month','Host_has_profile
test_df = test_df.drop(columns = ['Bedrooms', 'Beds','Month','Host_has_profile_p
```

In [23]:
```
columns = ['id', 'Decision', 'Host_response_time',
           'Host_is_superhost', 'Host_has_profile_pic', 'Host_identity_verified'
           'Property_type', 'Room_type', 'Accommodates','Bathrooms_text',
           'Bedrooms', 'Beds',
           'Essentials', 'Cooking',
           'Balcony', 'Parking',
           'Price',
           'Number_of_reviews',
           'Review_scores_rating',
           'Instant_bookable',
           'Month']
#we can split features like below:
categorical = ['Neighbourhood', 'Host_response_time','Property_type', 'Room_type
continuous = ['Accommodates','Balcony', 'Parking','Essentials', 'Cooking',
              'Price','Review_scores_rating','Number_of_reviews']
binary = ['Host_is_superhost', 'Host_identity_verified','Instant_bookable']
```

In [24]:
```
#fill nan values
```

```
train_df.isna().sum()
```

Out[24]:
```
id                          0
Decision                    0
Host_response_time        858
Host_is_superhost           0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      395
Instant_bookable            0
dtype: int64
```

In [25]:
```
test_df.isna().sum()
```

Out[25]:
```
id                          0
Host_response_time        293
Host_is_superhost           0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      274
Instant_bookable            0
dtype: int64
```

In [26]:
```
train_df["Host_response_time"] = train_df["Host_response_time"].fillna(value="mi
test_df["Host_response_time"] = test_df["Host_response_time"].fillna(value="miss
```

In [27]:
```
#still have a feature with nan values: fill it with median
train_df["Review_scores_rating"].median()
```

Out[27]:
```
4.93
```

In [28]:
```
train_df["Review_scores_rating"] = train_df["Review_scores_rating"].fillna(value
test_df["Review_scores_rating"] = test_df["Review_scores_rating"].fillna(value=4
```

In [29]:
```
#price features: str to numerical
train_df['Price'] = train_df['Price'].replace({'\$':'',',':''},regex = True)
train_df['Price'] = train_df['Price'].astype('float')

test_df['Price'] = test_df['Price'].replace({'\$':'',',':''},regex = True)
test_df['Price'] = test_df['Price'].astype('float')
```

In [30]:
```
# continuous features require normalization
# normalize the continuous features to zero mean and unit varianceе
scaler = StandardScaler()
scaler.fit(train_df[continuous])
train_df[continuous] = scaler.transform(train_df[continuous])
test_df[continuous] = scaler.transform(test_df[continuous])
```

In [31]:
```
#split labels and features
y_train_full = train_df['Decision']
X_train_full = train_df.drop(['Decision'], axis=1)
```

In [32]:
```
# convert categorical features to one-hot representations
len_train = len(X_train_full)

total_X = X_train_full.append(test_df, ignore_index=True)
one_hot_X = pd.get_dummies(total_X, columns=categorical)
#split train and test dataset
X_test = one_hot_X[len_train:]
X_train_full = one_hot_X[:len_train]
```

In [33]:
```
X_test
```

Out[33]:

	id	Host_is_superhost	Host_identity_verified	Accommodates	Essentials	Cooking	E
7471	1	1	1	-1.192110	-1.171512	-0.385235	0.:
7472	2	1	1	-0.830126	0.447498	0.843931	0.:
7473	3	1	1	-0.830126	0.447498	0.843931	0.:
7474	4	1	1	-0.830126	0.447498	0.843931	0.:
7475	5	1	1	-0.830126	0.447498	0.843931	0.:
...	...	...	...	...	...	...	
9906	2436	0	0	0.617810	-6.028541	-2.228985	-1.
9907	2437	1	1	1.703762	0.447498	0.843931	0.:
9908	2438	1	1	3.513681	-1.171512	-1.614402	-1.
9909	2439	1	1	-0.830126	0.447498	-0.999819	-1.
9910	2440	1	1	-0.830126	-1.171512	-1.614402	-1.

2440 rows × 111 columns

In [34]:
```
X_train_full = X_train_full.drop(['id'], axis=1)
test_index =X_test['id']
#ids = ids.astype(np.int64)
X_test = X_test.drop(['id'], axis=1)
```

In [35]:
```
#split validation set
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, te
```

SVM

In [43]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

In [45]:
```
#hyperparameter selection
param_test1 = {
 'C': [10e-3, 10e-2, 10e-1, 1, 10, 100]
}
gsearch1 = GridSearchCV(estimator = SVC(kernel='rbf', random_state=42),
 param_grid = param_test1, scoring='accuracy',n_jobs=4,cv=3)
gsearch1.fit(X_train_full, y_train_full)
print(gsearch1.best_params_)
print(gsearch1.best_score_)
```

```
{'C': 100}
0.7736568111063282
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
cv_results = gsearch1.cv_results_
params = [10e-3, 10e-2, 10e-1, 1, 10, 100]
mean_test_score = cv_results["mean_test_score"]
std_test_score = cv_results["std_test_score"]
sns.set(font_scale = 1.25)
ax = sns.lineplot(x=params, y=mean_test_score)
ax.fill_between(params, y1=mean_test_score - std_test_score, y2=mean_test_score
ax.set_xscale("log")
ax.set_xlabel("$C$")
ax.set_ylabel("accuracy")
plt.savefig("svm_c.pdf", bbox_inches="tight")
```

In [61]:
```
classifer = SVC(
    C=100,
    kernel='rbf',
    verbose=False,
    random_state=42
)
t_start = time.time()
classifer.fit(X_train_full, y_train_full)
t_end = time.time()
acc_train = classifer.score(X_train_full, y_train_full)
preds = classifer.predict(X_test) + 1
results = pd.Series(preds, index=test_index)
results.to_csv("results/svm.csv", header=None, index=True)
print("train acc: {:.4f}".format(acc_train))
print("training_time: {:.2f}sec".format(t_end - t_start))
```

```
train acc: 0.8921
training_time: 8.44sec
```

Gradient Boosting

In [24]:
```
#to process data
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
import time
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(42)
```

In [25]:
```
#import data and convert bool features to numerical
bool_converter = lambda x: 1 if x == 't' else 0
train_df = pd.read_csv(
    "train.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    },
)

test_df = pd.read_csv(
    "test.csv",
    converters={
        'Host_is_superhost':bool_converter,
        'Host_has_profile_pic': bool_converter,
        'Host_identity_verified':bool_converter,
        'Instant_bookable':bool_converter
    }
)
```

In [26]:
```
#We drop 4 features in total: 'Bedrooms', 'Beds','Month''Host_has_profile_pic'
train_df = train_df.drop(columns = ['Bedrooms', 'Beds','Month','Host_has_profile
test_df = test_df.drop(columns = ['Bedrooms', 'Beds','Month','Host_has_profile_p
```

In [5]:
```
columns = ['id', 'Decision', 'Host_response_time',
           'Host_is_superhost', 'Host_has_profile_pic', 'Host_identity_verified'
           'Property_type', 'Room_type', 'Accommodates','Bathrooms_text',
           'Bedrooms', 'Beds',
           'Essentials', 'Cooking',
           'Balcony', 'Parking',
           'Price',
           'Number_of_reviews',
           'Review_scores_rating',
           'Instant_bookable',
           'Month']
#we can split features like below:
categorical = ['Neighbourhood', 'Host_response_time','Property_type', 'Room_type
continuous = ['Accommodates','Balcony', 'Parking','Essentials', 'Cooking',
              'Price','Review_scores_rating','Number_of_reviews']
binary = ['Host_is_superhost', 'Host_identity_verified','Instant_bookable']
```

In [6]:
```
#fill nan values
train_df.isna().sum()
```

Out[6]:
```
id                          0
Decision                    0
Host_response_time        858
Host_is_superhost           0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      395
Instant_bookable            0
dtype: int64
```

In [7]:
```
test_df.isna().sum()
```

Out[7]:
```
id                          0
Host_response_time        293
Host_is_superhost           0
Host_identity_verified      0
Neighbourhood               0
Property_type               0
Room_type                   0
Accommodates                0
Bathrooms_text              0
Essentials                  0
Cooking                     0
Balcony                     0
Parking                     0
Price                       0
Number_of_reviews           0
Review_scores_rating      274
Instant_bookable            0
dtype: int64
```

In [8]:
```
train_df["Host_response_time"] = train_df["Host_response_time"].fillna(value="mi
test_df["Host_response_time"] = test_df["Host_response_time"].fillna(value="miss
```

In [9]:
```
#still have a feature with nan values: fill it with median
train_df["Review_scores_rating"].median()
```

Out[9]:
```
4.93
```

In [10]:
```
train_df["Review_scores_rating"] = train_df["Review_scores_rating"].fillna(value
test_df["Review_scores_rating"] = test_df["Review_scores_rating"].fillna(value=4
```

In [11]:
```
#price features: str to numerical
train_df['Price'] = train_df['Price'].replace({'\$':'',',':''},regex = True)
train_df['Price'] = train_df['Price'].astype('float')

test_df['Price'] = test_df['Price'].replace({'\$':'',',':''},regex = True)
test_df['Price'] = test_df['Price'].astype('float')
```

In [12]:
```
# continuous features require normalization
# normalize the continuous features to zero mean and unit varianece
scaler = StandardScaler()
scaler.fit(train_df[continuous])
train_df[continuous] = scaler.transform(train_df[continuous])
test_df[continuous] = scaler.transform(test_df[continuous])
```

In [13]:
```
#split labels and features
y_train_full = train_df['Decision']
X_train_full = train_df.drop(['Decision'], axis=1)
```

In [14]:
```
# convert categorical features to one-hot representations
len_train = len(X_train_full)

total_X = X_train_full.append(test_df, ignore_index=True)
one_hot_X = pd.get_dummies(total_X, columns=categorical)
#split train and test dataset
X_test = one_hot_X[len_train:]
X_train_full = one_hot_X[:len_train]
```

In [15]:
```
X_test
```

Out[15]:

	id	Host_is_superhost	Host_identity_verified	Accommodates	Essentials	Cooking	E
7471	1	1	1	-1.192110	-1.171512	-0.385235	0.:
7472	2	1	1	-0.830126	0.447498	0.843931	0.:
7473	3	1	1	-0.830126	0.447498	0.843931	0.:
7474	4	1	1	-0.830126	0.447498	0.843931	0.:
7475	5	1	1	-0.830126	0.447498	0.843931	0.:
...	...	...	...	...	...	...	
9906	2436	0	0	0.617810	-6.028541	-2.228985	-1.
9907	2437	1	1	1.703762	0.447498	0.843931	0.:
9908	2438	1	1	3.513681	-1.171512	-1.614402	-1.
9909	2439	1	1	-0.830126	0.447498	-0.999819	-1.
9910	2440	1	1	-0.830126	-1.171512	-1.614402	-1.

2440 rows × 111 columns

In [16]:
```
X_train_full = X_train_full.drop(['id'], axis=1)
test_index =X_test['id']
#ids = ids.astype(np.int64)
X_test = X_test.drop(['id'], axis=1)
```

In [17]:
```
#split validation set
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, te
```

XGBoost

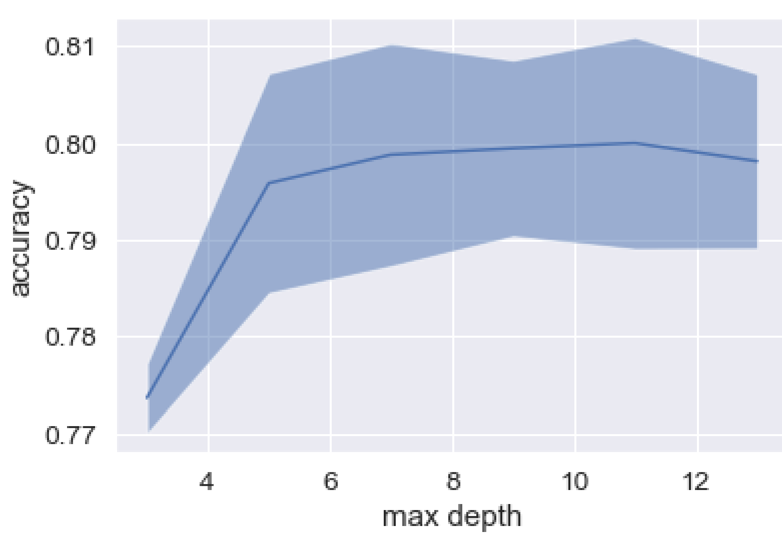
```
from sklearn.ensemble import RandomForestClassifier
```

In [18]:
```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
```

In [19]:
```
#hyperparameter selection
param_test1 = {
 'max_depth': range(3, 15,2 )
}
gsearch1 = GridSearchCV(estimator = xgb.XGBClassifier(learning_rate=0.1, n_estim
 min_child_weight=1, gamma=0.1, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=42),
 param_grid = param_test1, scoring='accuracy',n_jobs=4, cv=3)
gsearch1.fit(X_train_full, y_train_full)
print(gsearch1.best_params_)
print(gsearch1.best_score_)
```

```
/Users/libertyeagle/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_sele
ction/_search.py:847: FutureWarning: The parameter 'iid' is deprecated in 0.22 a
nd will be removed in 0.24.
  warnings.warn(
/Users/libertyeagle/opt/anaconda3/lib/python3.8/site-packages/xgboost/sklearn.p
y:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and
will be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object; a
nd 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [nu
m_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[16:37:35] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Star
ting in XGBoost 1.3.0, the default evaluation metric used with the objective 'bi
nary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
{'max_depth': 11}
0.8000254732297315
```

In [22]:
```
cv_results = gsearch1.cv_results_
params = list(range(3, 15, 2))
mean_test_score = cv_results["mean_test_score"]
std_test_score = cv_results["std_test_score"]
sns.set(font_scale = 1.25)
ax = sns.lineplot(x=params, y=mean_test_score)
ax.fill_between(params, y1=mean_test_score - std_test_score, y2=mean_test_score
ax.set_xlabel("max depth")
ax.set_ylabel("accuracy")
plt.savefig("xgboost_max_depth.pdf", bbox_inches="tight")
```



In [23]:
```
#train on full train dataset
clf_xgb = xgb.XGBClassifier(learning_rate=0.1, n_estimators=1100,
 min_child_weight=1, gamma=0.1,  max_depth=11, subsample=0.8, colsample_bytree=0
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=42)
t_start = time.time()
clf_xgb.fit(X_train_full,y_train_full)
t_end = time.time()
acc_train = clf_xgb.score(X_train_full,y_train_full)
print("train acc: {:.4f}".format(acc_train))
print("training time: {:.2f}".format(t_end - t_start))
```

```
/Users/libertyeagle/opt/anaconda3/lib/python3.8/site-packages/xgboost/sklearn.p
y:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and
will be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object; a
nd 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [nu
m_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:09:17] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Star
ting in XGBoost 1.3.0, the default evaluation metric used with the objective 'bi
nary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
train acc: 0.9916
training time: 23.86
```

In [28]:
```
preds = clf_xgb.predict(X_test)
results = pd.Series(preds, index=test_index)
results.to_csv("results/xgboost.csv", header=['Decision'], index=True, index_lab
```

In [ ]: