Airbnb Kaggle Competition Writeup

CompSci 671

Yujie(Johnny) Ye

2023.12.3

1. Exploratory Analysis

1.1 Preliminary Data Selection

The dataset provides us with information about each homestay, such as geographic location, living conditions, and information about the hosts. Our goal is to predict the price range of the homestay (from 0 to 5, with 0 being the cheapest and 5 being the most expensive). First, we exclude some information that we do not need:

- id: This is just a sequential number assigned to each record.
- 'scrape_id', 'last_scraped', 'calendar_last_scraped': The date of data scraping and the same scraping ID, irrelevant to predicting homestay prices.
- name: The living condition information in the name has already been extracted into separate variables. And the rating information in the name is often missing, with some observations lacking ratings or being new with few reviews and therefore no rating.
- description: This variable contains some information about living conditions. Further exploration would require an NLP model and is quite complex.
- picture_url: Requires CNN or other image processing models.

1.2 Variable Processing

We processed the 'amenities' variable, extracting the number of amenities into a numerical variable, 'number_amenities', as the number of amenities might affect the Airbnb housing's pricing; generally, more amenities may indicate a higher price. Additionally, we converted categorical variables into numerical variables, which is helpful for our collinearity tests and subsequent modeling. Notably, I retained 'bathrooms_text' without extracting the specific number of bathrooms, as it includes distinctions between shared and private bathrooms, so it was encoded numerically. Other categorical variables were similarly encoded with unique numbers for each category.

1.3 Missing Values

Next, we checked for the presence of missing values. (figure 1) We chose to delete observations with missing values for two reasons. Firstly, the proportion of missing values is not high, with a maximum of about 3.3% missing. Secondly, we do not have information about how Airbnb internally evaluates whether a host is a "superhost". Therefore, we can opt to directly exclude observations with missing values.

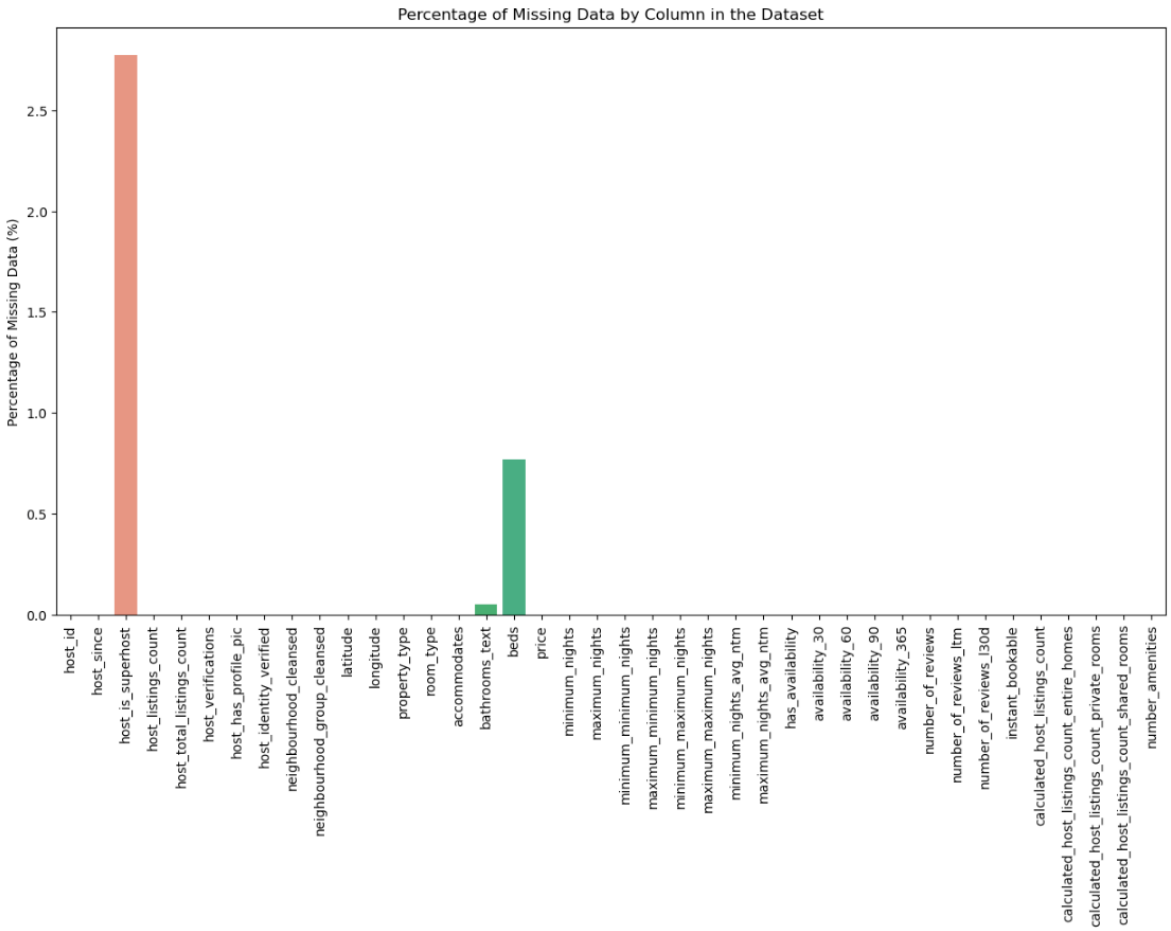


Figure 1 Missing Value Check

1.4 Summary Statistics

We select some relevant variables to conduct summary statistics.

Table 1 Summary Statistics of 'price', 'minimum_nights', 'number_of_reviews','availability_365'

	price	minimum_nights	number_of_reviews	availability_365
count	14886.000000	14886.000000	14886.000000	14886.000000
mean	1.961911	16.810359	30.773008	204.242308
std	1.622594	30.550330	67.731314	135.317802
min	0.000000	1.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	82.000000
50%	2.000000	5.000000	5.000000	217.000000
75%	3.000000	30.000000	28.000000	344.000000
max	5.000000	1124.000000	1118.000000	365.000000

The table 1 is a statistical summary showing columns for price, minimum nights, number of reviews, and availability over a year for 14,886 listings. Prices range from 0 to 5, minimum nights required range up to 1124, the number of reviews per listing goes up to 1118, and availability varies from 0 to 365 days. The data shows variability in each category, as indicated by the standard deviation values.

We also created histograms for these variables to examine their distribution.

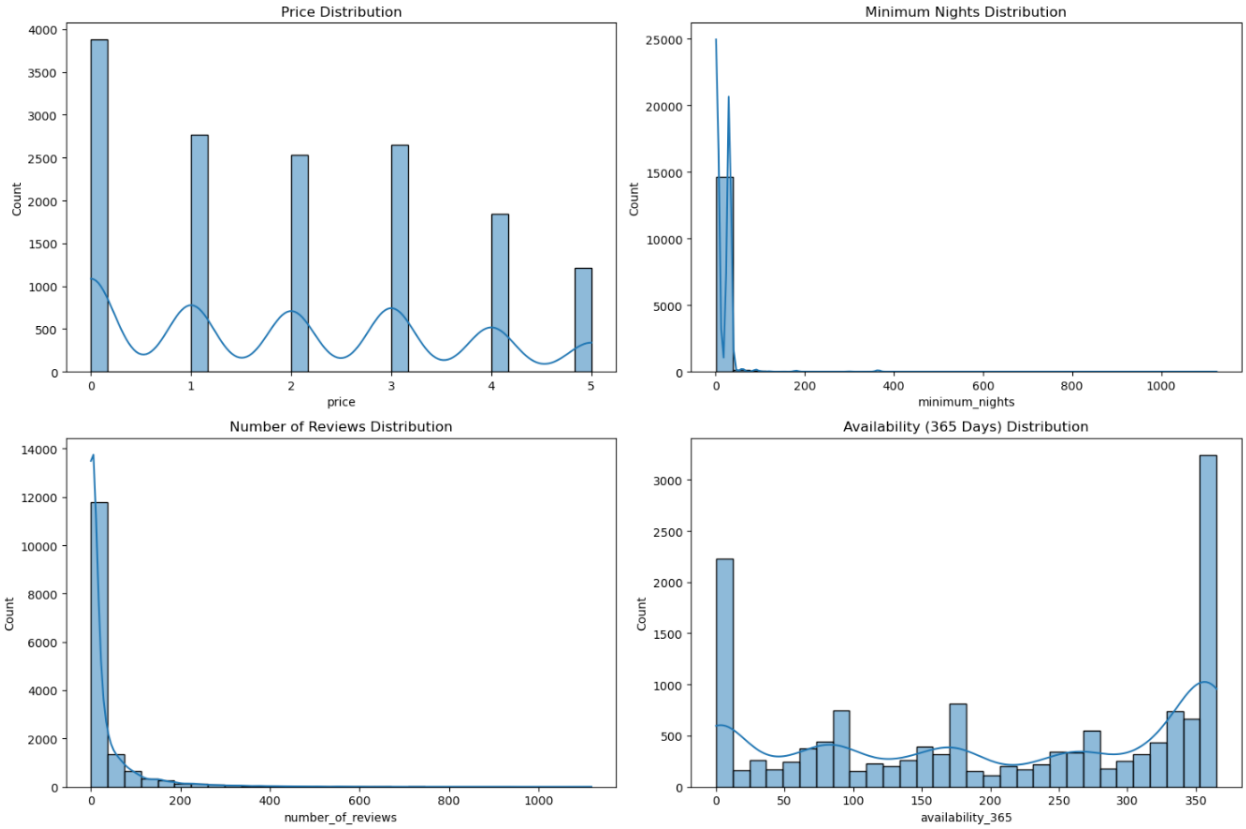


Figure 2 Histograms of 'price', 'minimum_nights', 'number_of_reviews', 'availability_365'

Figure 2 displays four histograms, each representing the distribution of different variables. The price distribution histogram indicates a concentration of listings in the lower price range, primarily between 0 to 1, and progressively fewer listings at higher prices. In the minimum nights distribution, there is a pronounced spike at the lower end, suggesting that most listings require only a few nights' stay, with the frequency decreasing for longer minimum stays. The number of reviews histogram is heavily right-skewed, showing that most listings have very few reviews, with a rapid drop-off as review counts increase. Finally, the availability histogram illustrates that listings are commonly available for very few days or the entire year, with fewer listings in the mid-range of availability. All histograms display right-skewed distributions, indicating that for each variable, the median is lower than the mean.

Additionally, we examined box plots to check for the presence of any outliers in the distribution.

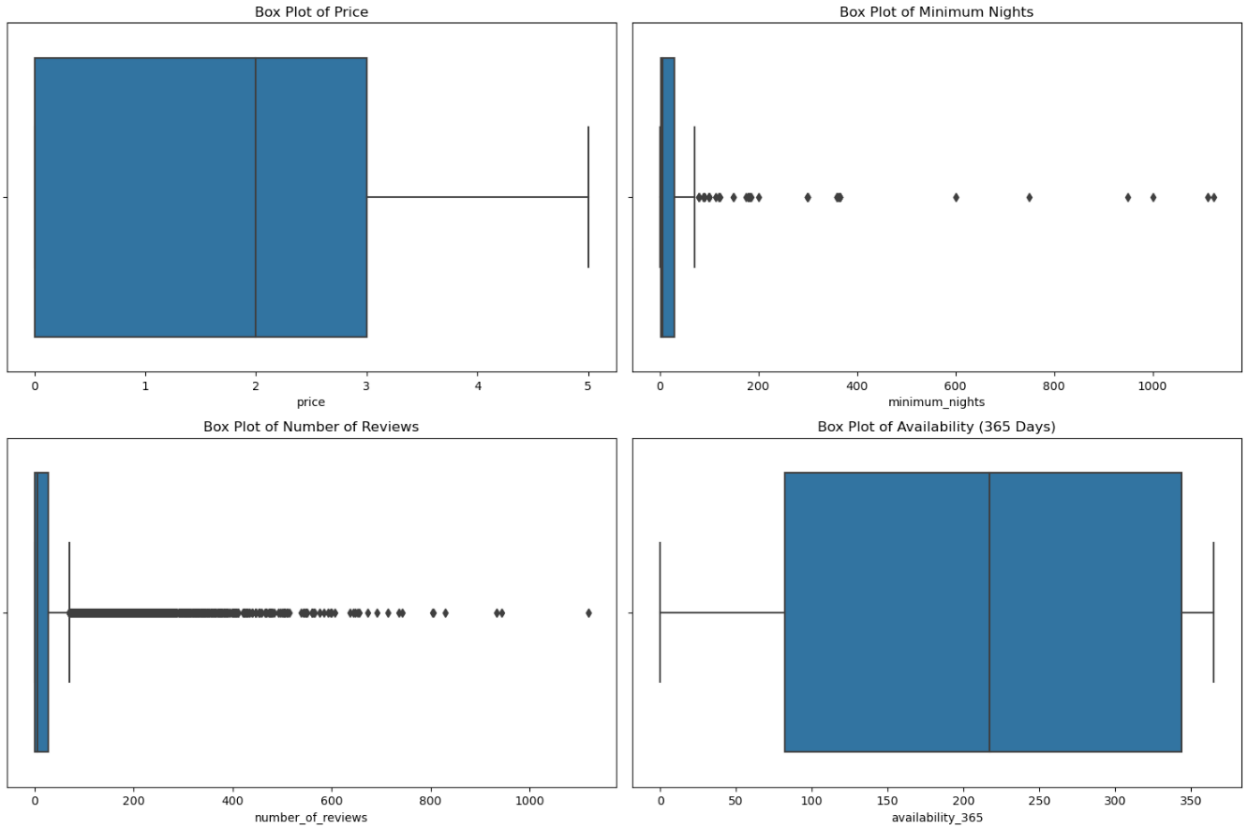


Figure 3 Box-plot of 'price', 'minimum_nights', 'number_of_reviews', 'availability_365'

Figure 3 contains four box plots, each visualizing the spread and distribution of a different variable. The first box plot of price suggests a concentration of listing prices at the lower end of the scale with a few outliers on the higher side, indicating that most listings are in the lower price bracket. The box plot for minimum nights shows a similar pattern with a tight interquartile range and many high-value outliers, implying that while most listings require only a few nights' stay, there are some with a much higher requirement. The number of reviews box plot also depicts a concentration of data points at the lower end, with outliers stretching towards the higher number of reviews. Lastly, the availability box plot suggests a broad spread in the number of days listings are available throughout the year, with some outliers indicating listings that are either rarely or almost always available. The long whiskers and outliers in the plots indicate significant variability and skewness in the data for these variables.

Furthermore, we created plots to examine the relationship between some categorical variables and price.

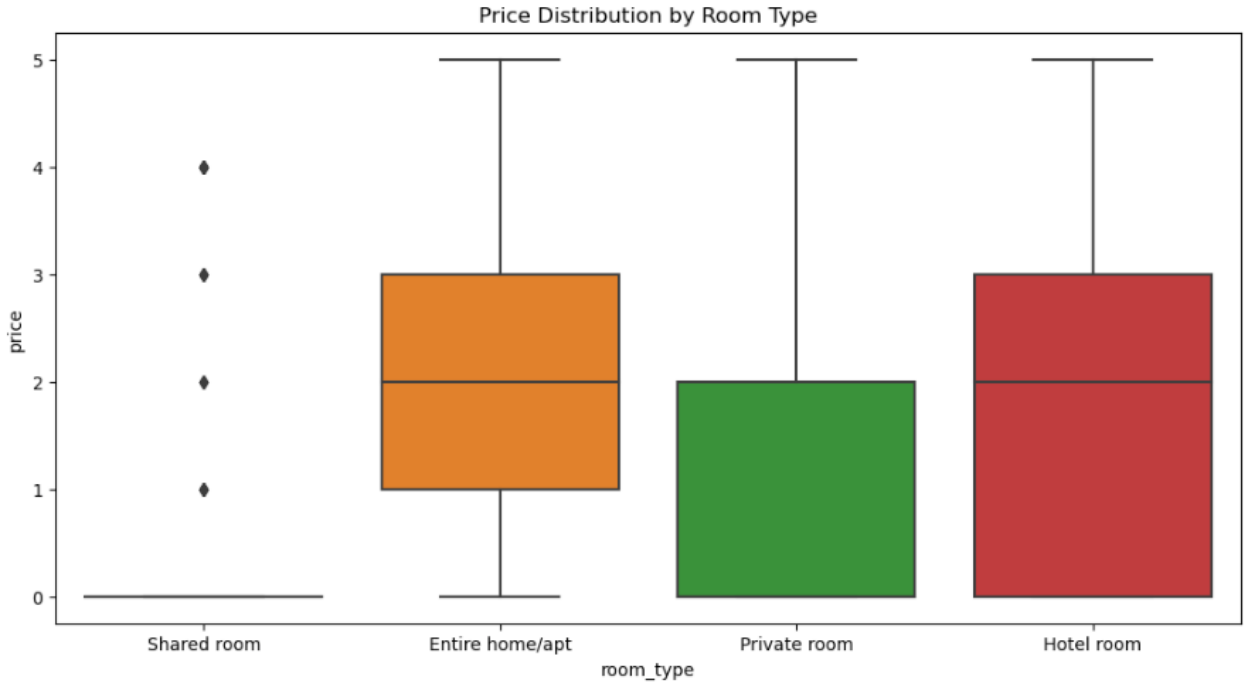


Figure 4 Price Distribution by Room Type

Figure 4 shows a bar graph representing the comparative pricing for different types of rental accommodations. There are four types of rooms: shared room, entire home/apt, private room, and hotel room, each depicted by bars of different colors—orange, green, and red, respectively, with the fourth type not color-coded in the description. The y-axis measures price on a scale of 0 to 5, and each bar's length represents the median price for that room type. The bars also have "whiskers" indicating the range of prices, including outliers, which are depicted as individual points above the whiskers. Shared rooms appear to have the lowest median price and a tight price range, while entire homes/apartments and hotel rooms show a higher median price and a wider range of prices. Private rooms have a median price lower than entire homes and hotel rooms but higher than shared rooms. The presence of outliers for shared rooms suggests some variation in pricing beyond the typical range.

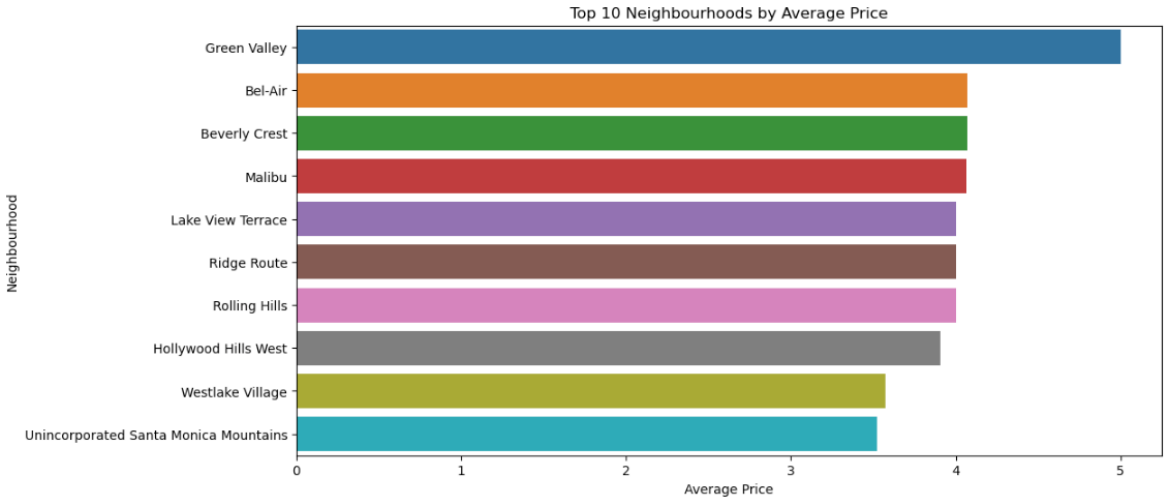


Figure 5 Top 10 Neighborhoods by Average Price

Figure 5 displays a horizontal bar chart, which ranks neighborhoods based on their average price, which is measured on the horizontal axis from 0 to 5. Each neighborhood is represented by a different color bar, with ten neighborhoods listed in total. The bars extend to varying lengths, indicating the average price in each neighborhood. At a glance, the chart shows that Green Valley has the highest average price among the listed neighborhoods, closely followed by Bel Air and Beverly Crest. The shortest bar represents the Unincorporated Santa Monica Mountains, suggesting it has the lowest average price among the top ten neighborhoods displayed. This visualization aids in comparing the average prices across different neighborhoods at a glance.

In addition to this, we also generated a heatmap related to collinearity, which revealed that many variables exhibit issues of multicollinearity. (figure 6)

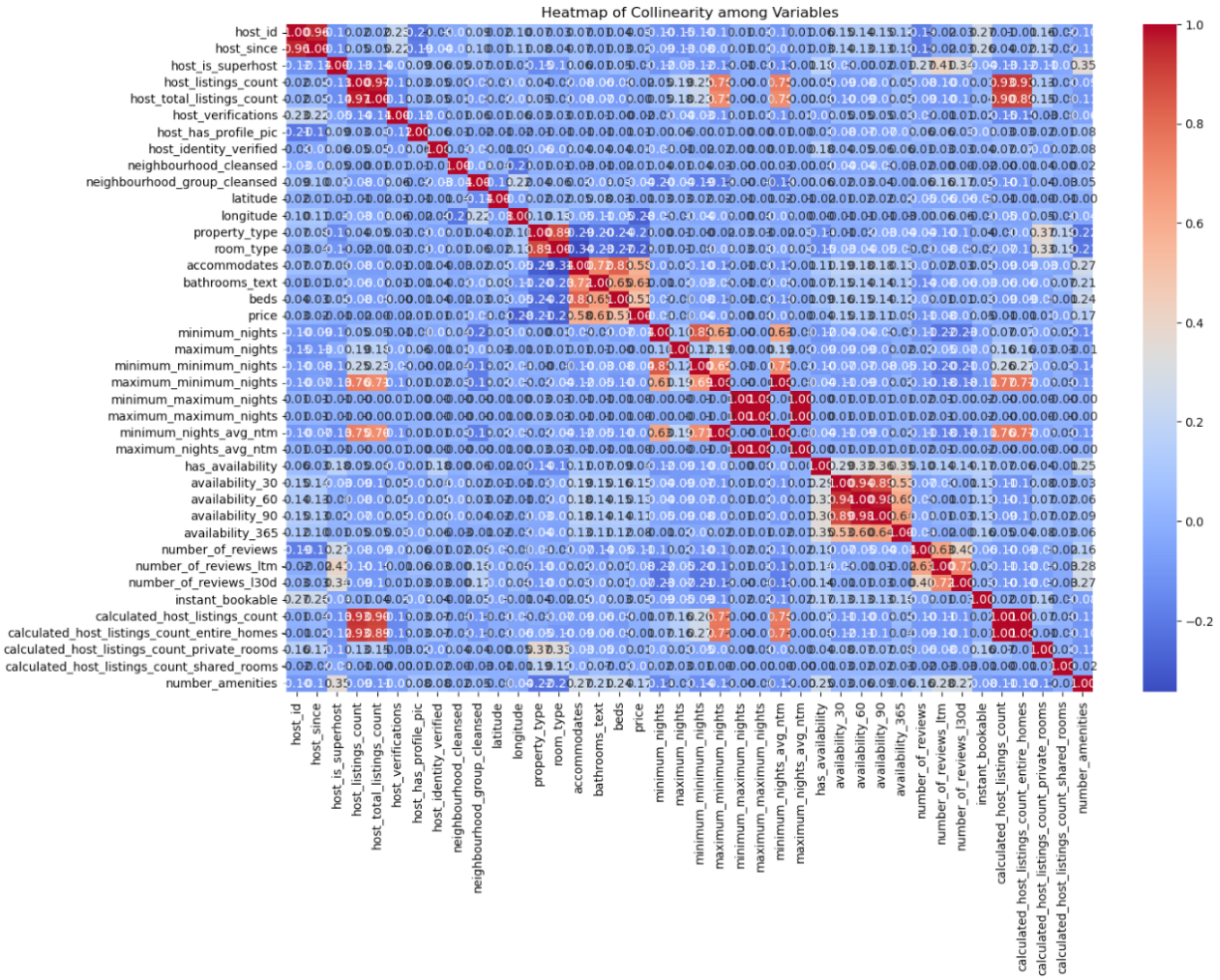


Figure 6 Heatmap of Collinearity among Variables

The heatmap is a statistical tool used to identify relationships between variables, which can be important for understanding patterns within the data and for building predictive models. We will discuss collinearity further in Section 3.

2. Models

2.1.Algorithms Choice

We used four alternative algorithms: Decision Tree Classifier, Random Forest Classifier, AdaBoosting Classifier and gradient boosting (XGBoosting Classifier), to select the best two. First, we split the training data to conduct accuracy checks, and we observed that the Random Forest Classifier and XGBoosting Classifier performed better:

- **Random Forest Accuracy：0.56212**
- **XGBoosting Accuracy: 0.55809**
- Decision Tree Accuracy: 0.47079
- AdaBoosting Accuracy: 0.45937

Additionally, we can also consider other advantages of the algorithms.
Random Forest Classifier:

- Handling Categorical Data: Random Forests are adept at handling categorical data. (e.g., room type, property type)
- Robustness: This algorithm is less prone to overfitting and can handle outliers and non-linear data effectively, which are common in real-world datasets like property listings.
- Feature Importance: Random Forest can rank the importance of various features for the prediction, which is beneficial for understanding which features (like location, number of rooms, etc.) most influence the price.
- Ease of Use: It is relatively easy to implement and has less hyperparameters to tune, making it a good choice for a balanced approach between performance and complexity.

XGBoost Classifier:

- Performance: XGBoost is known for delivering high performance and accuracy in predictive modeling, especially for structured data like the Airbnb dataset.
- Speed and Efficiency: It is computationally efficient and works faster than many other ensemble classifiers.
- Handling Missing Values: XGBoost can handle missing data in the dataset, which is common in real-world datasets.
- Regularization: It includes L1 and L2 regularization which helps in preventing overfitting.

Both algorithms are available in high-quality libraries like scikit-learn (for Random Forest) and the XGBoost library. These libraries are well-documented and widely used in the machine learning community, offering robust support and ease of use.

2.2 References for Libraries

Scikit-learn for Random Forest: Developed by various contributors, it's an open-source machine learning library in Python. XGBoost: This model was utilized via the XGBoost library, originally developed by Tianqi Chen and Carlos Guestrin.

3. Training

Before training the models, we opted to address the multicollinearity issue present in the data. Based on the multicollinearity heatmap (figure 6), as well as the ranking of variable importance within the random forest algorithm, we decided to remove the following variables: 'host_since', 'host_listings_count', 'room_type', 'calculated_host_listings_count', 'maximum_maximum_nights', 'maximum_minimum_nights', 'host_total_listings_count', 'availability_60', 'availability_90', 'minimum_maximum_nights' and plotted a new heatmap (figure 7)

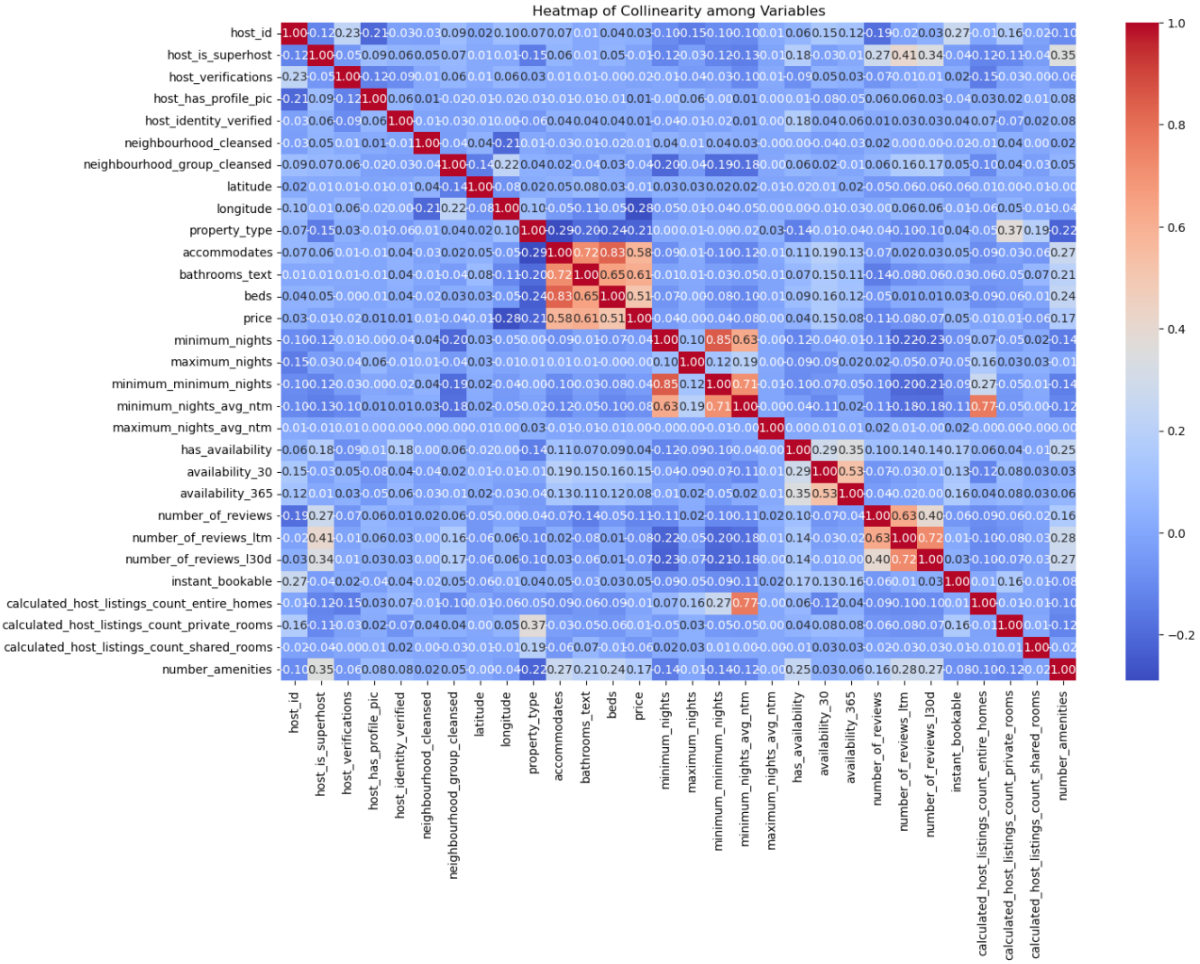


Figure 7 Heatmap after filtering out some variables

3.1 Random Forest Classifier

The Random Forest algorithm constructs numerous decision trees during training and outputs the mode of the classes (classification) of individual trees. It uses bagging and feature randomness when building each tree to create an uncorrelated forest of trees whose prediction by committee is more accurate than any individual tree. The training involves selecting random subsets of the features at each split, which makes the model robust and reduces overfitting. We also did Cross-Validation and Hyperparameter Tuning to prevent overfitting.

CPU times: user 1.59 s, sys: 16.8 ms, total: 1.61 s. Wall time: 1.64 s

3.2 XGBoosting Classifier

XGBoost is an implementation of gradient-boosted decision trees designed for speed and performance. It uses a gradient descent algorithm to minimize the loss when adding new models. This process involves creating new models that predict the residuals or errors of prior models and then adding them together to make the final prediction. We also did Cross-Validation and Hyperparameter Tuning to prevent overfitting.

CPU times: user 4.12 s, sys: 2.3 s, total: 6.42 s. Wall time: 18.2 s

4. Hyperparameter Selection

Random Forest Classifier:

1. **Hyperparameters**:
   - The hyperparameters chosen for tuning were **n_estimators** (number of trees in the forest) and **max_depth** (maximum depth of each tree).
   - **n_estimators**: [100, 150, 200]:

     This parameter specifies the number of trees in the forest. Generally, more trees increase the model's performance and stability. Each tree contributes with its own prediction, and the final outcome is determined by averaging these predictions by a majority vote (for classification). This aggregation helps in reducing variance and avoiding overfitting.
   - **max_depth**: [10, 15, 20]:

     The depth of the tree is a measure of how many splits it makes before making a prediction. A deeper tree can model more complex patterns, but it can also lead to overfitting as it might start to model noise and outliers in the data. Limiting the depth of the trees makes the model simpler and can improve the generalization to new data. A shallower tree is less prone to capturing noise.

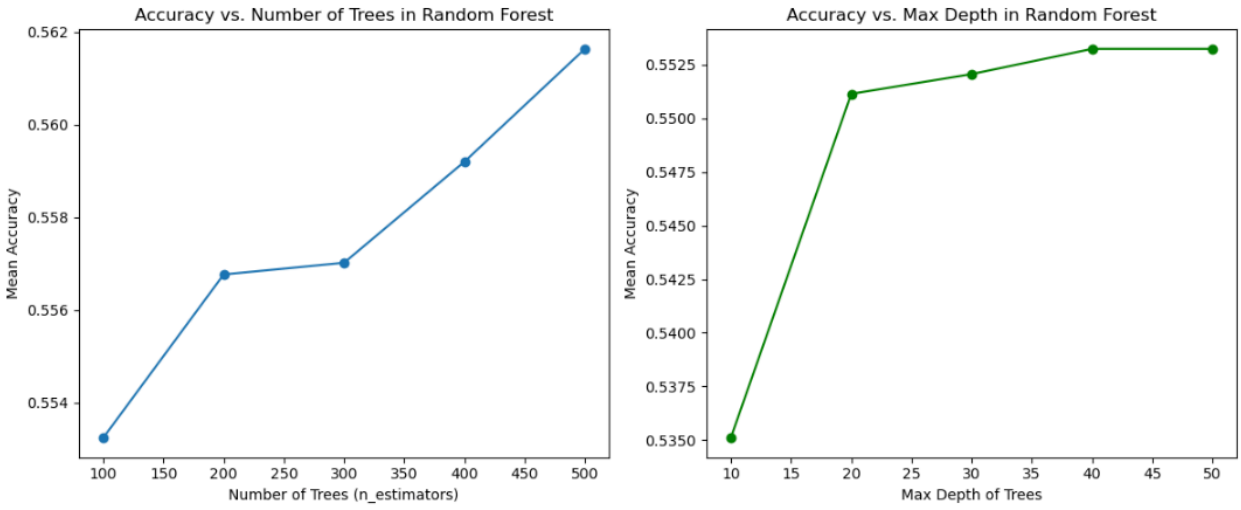


Figure 8 Relation between accuracy and hyperparameters in Random Forest

2. **Grid Search with Cross-Validation**:
    - Cross-validation was used to evaluate each model. This method splits the training data into several subsets, trains the model on some subsets, and validates it on the others, thereby providing a robust estimate of the model's performance. We set Cross-Validation = 5.
    - Grid Search with Cross-Validation gave us a result that ['max_depth': 20, 'n_estimators': 200]
3. **Scoring Metric**:
    - The scoring metric used was 'accuracy', which measures the proportion of correctly predicted instances. We will choose hyperparameters making a higher accuracy score.
4. **Variable Importance Selection:**
    - We grew a random forest of decision stumps on the training dataset with K = 1, . . . , 5 randomly selected variables available for each stump. Use M = 1000 stumps and B = 0.8 × n bootstrap samples.
    - We counted the number of each variable used for the best split and the top 20 variables most frequently selected as the best splits were selected.
    - Since we had less features, we use **n_estimators**: [500, 600, 700] and **max_depth**: [20, 30, 40] to prevent underfitting. Grid Search with Cross-Validation gave us a result that ['max_depth': 20, 'n_estimators': 700]

XGBoosting Classifier:

1. **Hyperparameters**:
    - The hyperparameters chosen for tuning were **learning_rate** (step size shrinkage used in update to prevents overfitting.), **max_depth** (maximum depth of each tree), **n_estimators** (number of trees in the forest), **subsample** (controls the fraction of the training data to be randomly sampled for each tree), **colsample_bytree** (specifies the fraction of features to be randomly sampled for each tree.)
    - **n_estimators**: [100, 200, 300]
    - **max_depth**: [3, 4, 5]
    - **learning_rate**: [0.01, 0.1, 0.2]:
        By tuning the learning rate, we can find a balance between the speed of convergence and the risk of overfitting. It's one of the most sensitive parameters in gradient boosting models.
    - **subsample**: [0.8, 0.9, 1.0]:
        This parameter defines the fraction of samples to be used for fitting the individual base learners. It is used for reducing overfitting and introducing randomness into the model.
    - **colsample_bytree**: [0.8, 0.9, 1.0]:
        Similar to 'subsample', this introduces randomness into the model, helping prevent overfitting. It ensures that each tree in the ensemble is slightly different, increasing model diversity. Tuning this parameter helps in optimizing the feature

sampling strategy, which can be crucial for high-dimensional data.

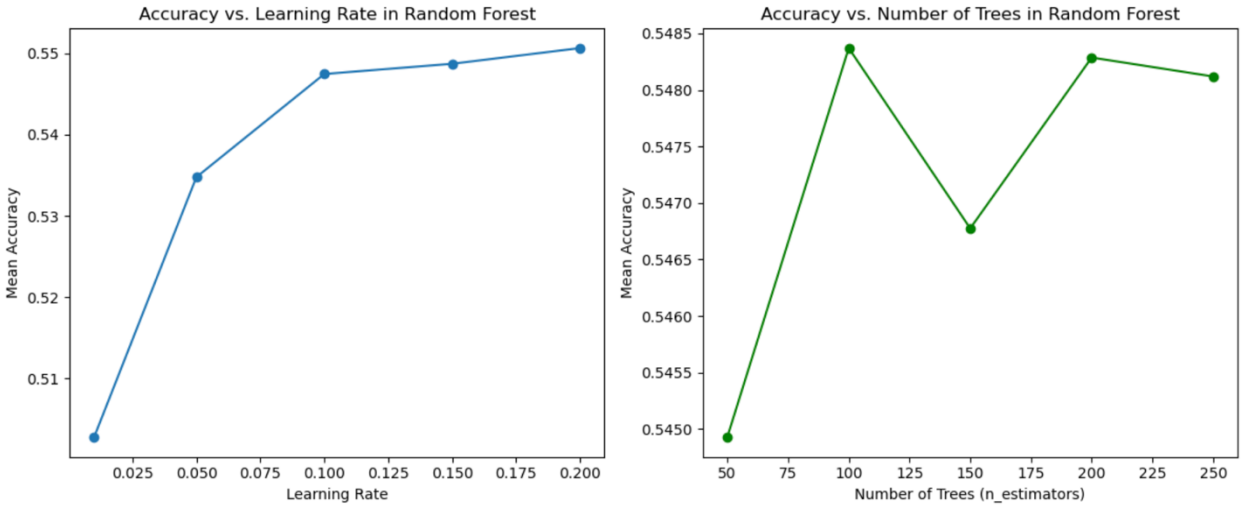


Figure 9 Relation between accuracy and hyperparameters in XGBoosting

2. **Random Search with Cross-Validation**:
    - Random Search: Randomly selects combinations within specified ranges, which can be more efficient than grid search, especially in high-dimensional spaces.
    - Use cross-validation to evaluate each combination of hyperparameters. This reduces the risk of overfitting and ensures that the model generalizes well to unseen data. We set Cross-Validation = 5.
    - Random Search with Cross-Validation suggested:
      ['subsample': 1.0, 'n_estimators': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0]
3. **Scoring Metric**:
    - The scoring metric used was 'accuracy', which measures the proportion of correctly predicted instances. We will choose hyperparameters making a higher accuracy score.
4. **Variable Selection:**
    - We plotted features importance figure and selected variables whose importance is greater than 0.2.
    - Model after variable Selection:
      ['subsample': 0.9, 'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.2, 'colsample_bytree': 0.8]

Because the prediction of Random Forest Classifier Model on testing dataset is not very good in both cases above. We tried to use all data features and didn't do any filtering and train a new Random Forest Classifier Model again. All missing values would be replaced by the median value in the same column and we also did the encoding part in this case as before.

New Random Forest Classifier:

1. **Hyperparameters**:
    - The hyperparameters chosen for tuning were **n_estimators** (number of trees in the forest) and **max_depth** (maximum depth of each tree), **min_samples_split** (minimum number of samples required to split an internal node) and **min_samples_leaf** (minimum number of samples required to be at a leaf node). Both min_samples_split and min_samples_leaf can help to control overfitting.
    - **n_estimators**: [100, 120, 150, 180, 200]
    - **max_depth**: [10, 15, 20]
    - **min_samples_split**: [2, 4, 6]
    - **min_samples_leaf**: [1, 2, 3]
2. **Grid Search with Cross-Validation**:
    - We set Cross-Validation = 10.
    - Grid Search with Cross-Validation gave us a result that ['max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200]

5. Data Splits

Firstly, we use `from sklearn.model_selection import train_test_split` to split the training dataset into "sub-training" and "sub-testing" datasets:

```
X = data.drop('price', axis=1)
y = data['price']
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,random_state=42)
```

Then we used 5-fold Cross-Validation to split the "sub-training" dataset. The dataset is split into 5 parts. In each iteration, 4 parts are used for training and 1 part for validation. The model's performance metric (accuracy) is calculated for each of the 5 iterations. This provides a more robust measure of the model's performance compared to using a single train-test split.

6. Error and Mistakes

The steps that slowed down my process were mainly focused on adjusting hyperparameters and dealing with the issue of overfitting. After addressing multicollinearity, the model achieved an accuracy of about 0.57 on the training set, but only about 0.53 in the Kaggle test, indicating that the model has an overfitting problem. I used the variable importance from the random forest for feature selection, which did not yield particularly good results.

Judging solely from the results, we find that the models trained without any variable selection perform better. However, not applying any variable selection can lead to noise affecting the model, resulting in overfitting. But after performing variable selection earlier, the results worsened, suggesting there might have been issues in the process of handling the variables.

7. Predictive Accuracy

Our best model is a Random Forest Classifier with hyperparameters set to ['max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200].

Kaggle username: Johnny Ye

Table 2 Model Accuracy in Training Dataset

	precision	recall	f1-score	support
0.0	0.73	0.85	0.78	797
1.0	0.48	0.47	0.47	595
2.0	0.44	0.31	0.37	529
3.0	0.44	0.56	0.49	544
4.0	0.52	0.44	0.48	349
5.0	0.83	0.67	0.74	275
accuracy			0.57	3089
macro avg	0.57	0.55	0.56	3089
weighted avg	0.57	0.57	0.56	3089

Figure 10 Prediction of Final Model (Test Dataset Accuracy: 0.56329)

8. Code

1. Exploratory Analysis

In [1]:
```
# Pakcages
import pandas as pd
import re
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
from sklearn.utils import resample
```

```
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of S
ciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Load the dataset
file_path = './train.csv'
dataset = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
dataset.head()
```

Out[2]:

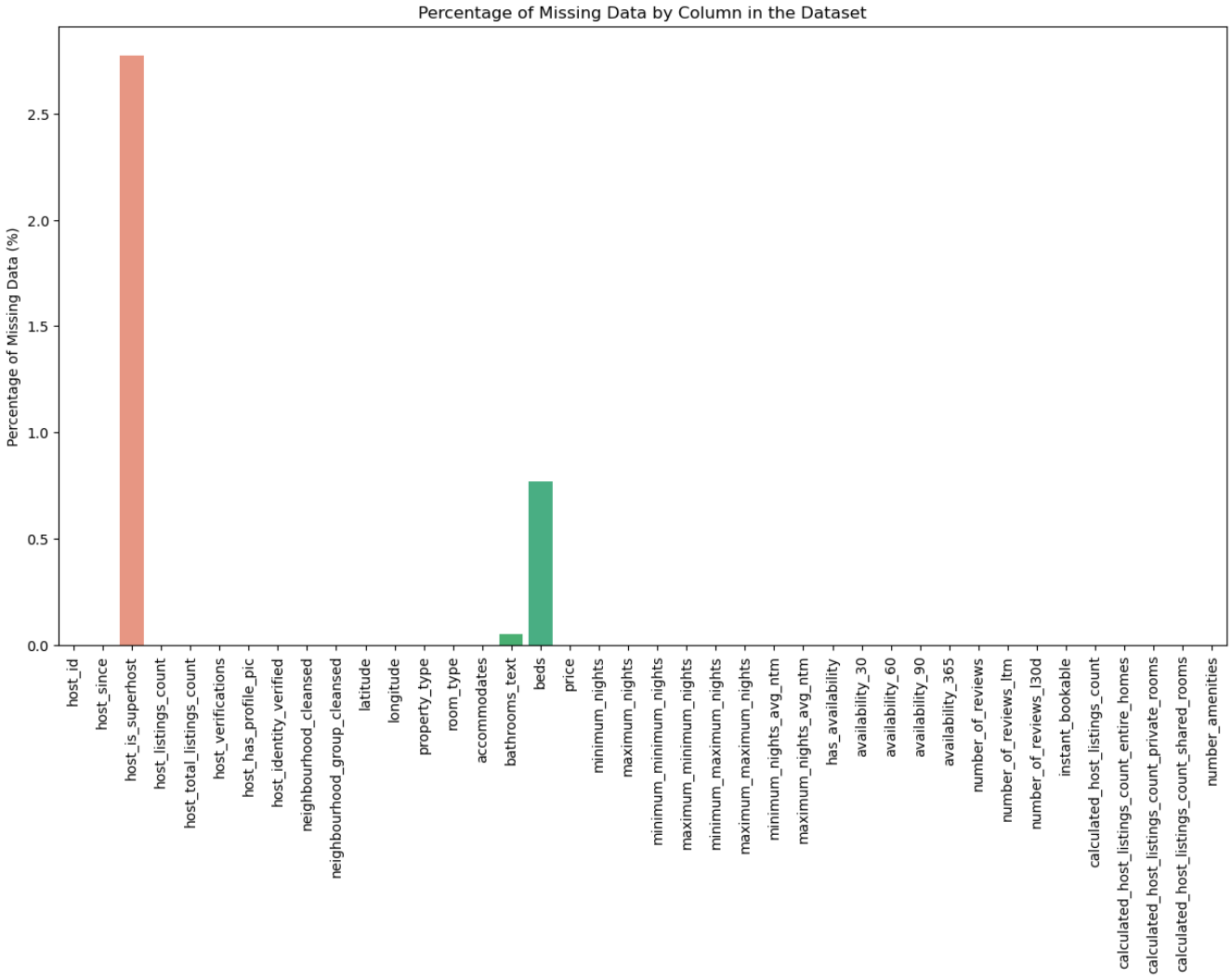
	id	scrape_id	last_scraped	name	description	picture_url	host_id	host_name	host_since	host_is_superhost	...	availability_365	calendar_
0	0	20230903194229	2023-09-04	Home in Torrance · 1 bedroom · 1 bed · 1 share...	Beautifully designed decor, lighting and enter...	https://a0.muscache.com/pictures/0f019985-8295...	205539927	Leonardo	2018-07-27	f	...	363	
1	1	20230903194229	2023-09-04	Rental unit in Venice · ★4.74 · 2 bedrooms · 2...	The location is great - the apt is HUGH 1877 S...	https://a0.muscache.com/pictures/2d5c133d-a280...	7599550	Donald	2013-07-19	t	...	296	
2	2	20230903194229	2023-09-04	Home in Avalon · ★5.0 · 3 bedrooms · 4 beds · ...	315 Eucalyptus Lower Spacious, Remodeled ...	https://a0.muscache.com/pictures/prohost-api/H...	271118401	Catalina Vacations	2019-06-24	f	...	201	
3	3	20230903194229	2023-09-04	Home in Santa Clarita · 4 bedrooms · 5 beds · ...	Close to almost everything when you stay at th...	https://a0.muscache.com/pictures/miso/Hosting-...	429951162	Kellie	2021-11-01	f	...	364	
4	4	20230903194229	2023-09-04	Hotel in Los Angeles · ★New · 1 bedroom · 1 be...	Enjoy the thrill of staying in a hotel located...	https://a0.muscache.com/pictures/prohost-api/H...	501999278	RoomPicks	2023-02-20	f	...	269	

5 rows × 48 columns

In [3]:
```
dataset_cleaned = dataset.drop(['id','name','scrape_id','last_scraped','description',
                                'picture_url','host_name','calendar_last_scraped'],axis=1)
```

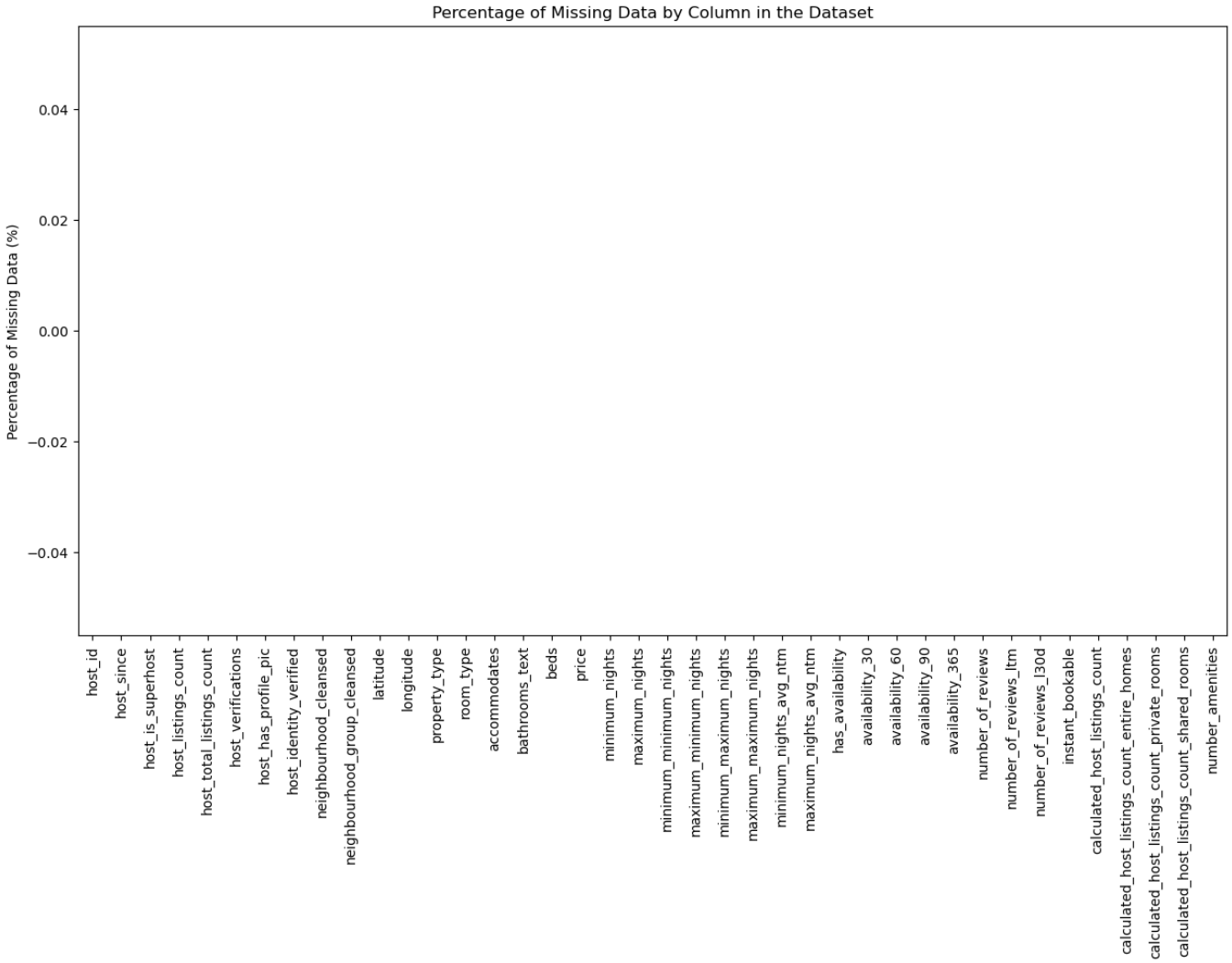
In [4]:
```
# Counting the number of amenities for each row
# Assuming amenities are separated by commas in the 'amenities' column
dataset_cleaned['number_amenities'] = dataset_cleaned['amenities'].str.split(',').apply(lambda x: len(x))

# Drop amenities
dataset_cleaned = dataset_cleaned.drop(['amenities'],axis=1)
```

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the percentage of missing data in each column
missing_data = dataset_cleaned.isnull().mean() * 100

# Plotting
plt.figure(figsize=(15, 8))
sns.barplot(x=missing_data.index, y=missing_data)
plt.xticks(rotation=90)
plt.ylabel('Percentage of Missing Data (%)')
plt.title('Percentage of Missing Data by Column in the Dataset')
plt.show()
```

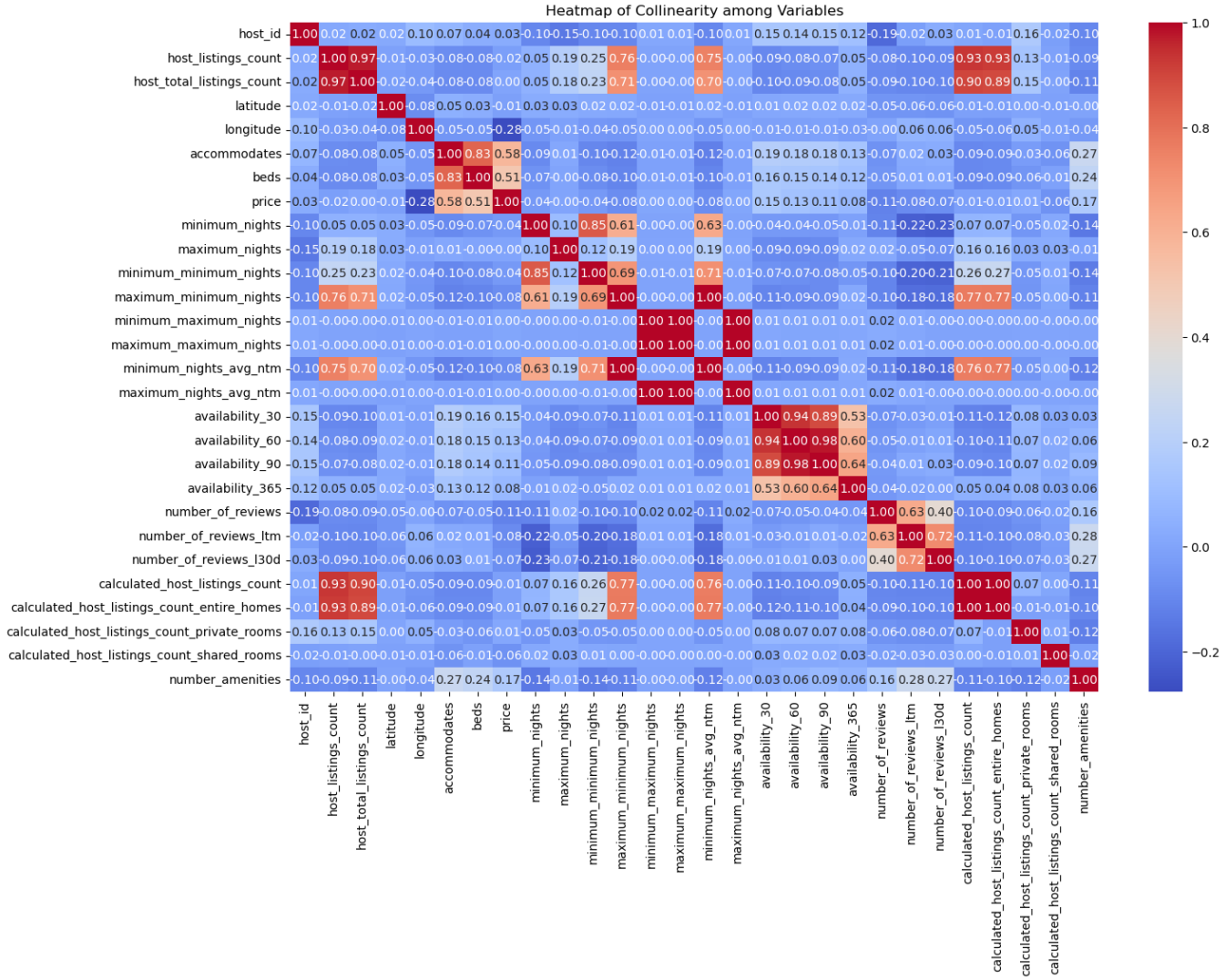


In [6]:
```
# Dropping off rows with missing values in 'host_is_superhost' column
dataset_cleaned = dataset_cleaned.dropna(subset=['host_is_superhost'])
# Dropping off rows with missing values in 'host_is_superhost' column
dataset_cleaned = dataset_cleaned.dropna(subset=['bathrooms_text'])
# Dropping off rows with missing values in 'host_is_superhost' column
dataset_cleaned = dataset_cleaned.dropna(subset=['beds'])
```

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the percentage of missing data in each column
missing_data = dataset_cleaned.isnull().mean() * 100

# Plotting
plt.figure(figsize=(15, 8))
sns.barplot(x=missing_data.index, y=missing_data)
plt.xticks(rotation=90)
plt.ylabel('Percentage of Missing Data (%)')
plt.title('Percentage of Missing Data by Column in the Dataset')
plt.show()
```



In [8]:
```
# Selecting columns for checking collinearity, excluding specified columns
cols_for_collinearity = dataset_cleaned.select_dtypes(include=[np.number])

# Calculating correlation matrix
corr_matrix = cols_for_collinearity.corr()

# Plotting the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Heatmap of Collinearity among Variables')
plt.show()
```

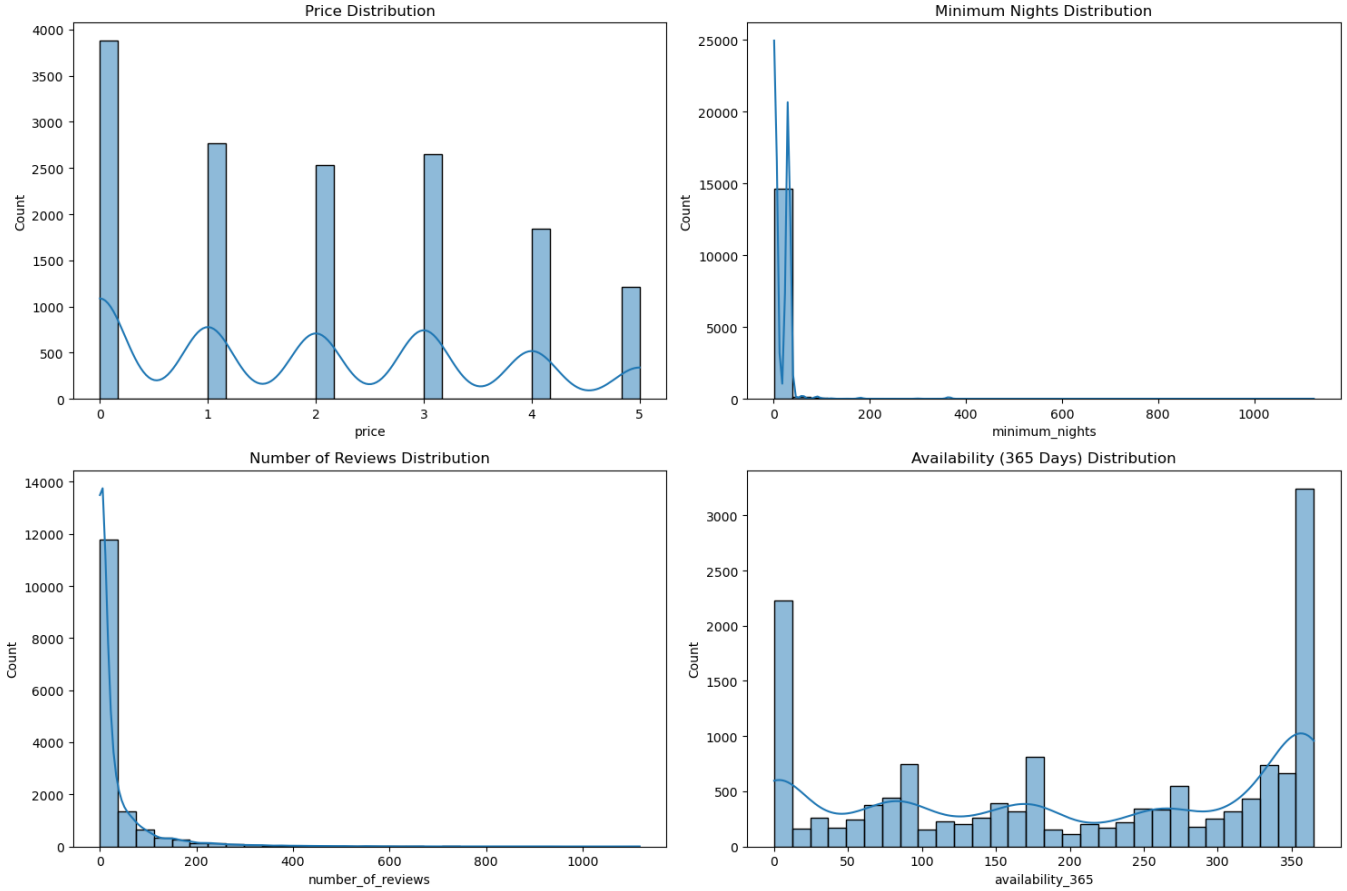


In [9]:
```
# Summary statistics for selected columns
summary_stats = dataset_cleaned[['price', 'minimum_nights', 'number_of_reviews', 'availability_365']].describe()
summary_stats
```

Out[9]:

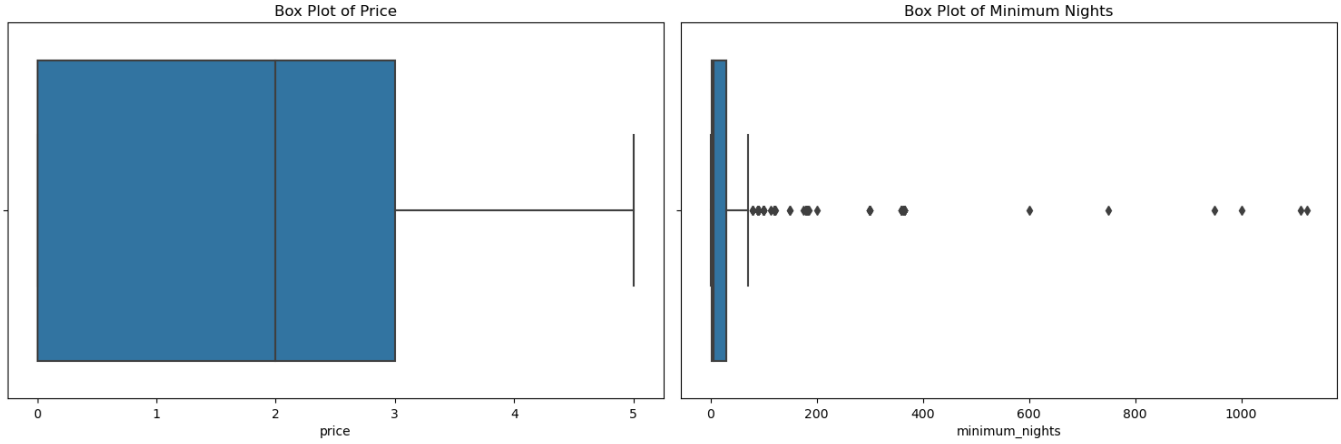
	price	minimum_nights	number_of_reviews	availability_365
count	14886.000000	14886.000000	14886.000000	14886.000000
mean	1.961911	16.810359	30.773008	204.242308
std	1.622594	30.550330	67.731314	135.317802
min	0.000000	1.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	82.000000
50%	2.000000	5.000000	5.000000	217.000000
75%	3.000000	30.000000	28.000000	344.000000
max	5.000000	1124.000000	1118.000000	365.000000

In [10]:
```
# Plotting histograms for these columns
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

sns.histplot(dataset_cleaned['price'], bins=30, ax=axes[0, 0], kde=True)
axes[0, 0].set_title('Price Distribution')

sns.histplot(dataset_cleaned['minimum_nights'], bins=30, ax=axes[0, 1], kde=True)
axes[0, 1].set_title('Minimum Nights Distribution')

sns.histplot(dataset_cleaned['number_of_reviews'], bins=30, ax=axes[1, 0], kde=True)
axes[1, 0].set_title('Number of Reviews Distribution')

sns.histplot(dataset_cleaned['availability_365'], bins=30, ax=axes[1, 1], kde=True)
axes[1, 1].set_title('Availability (365 Days) Distribution')

plt.tight_layout()
plt.show()
```

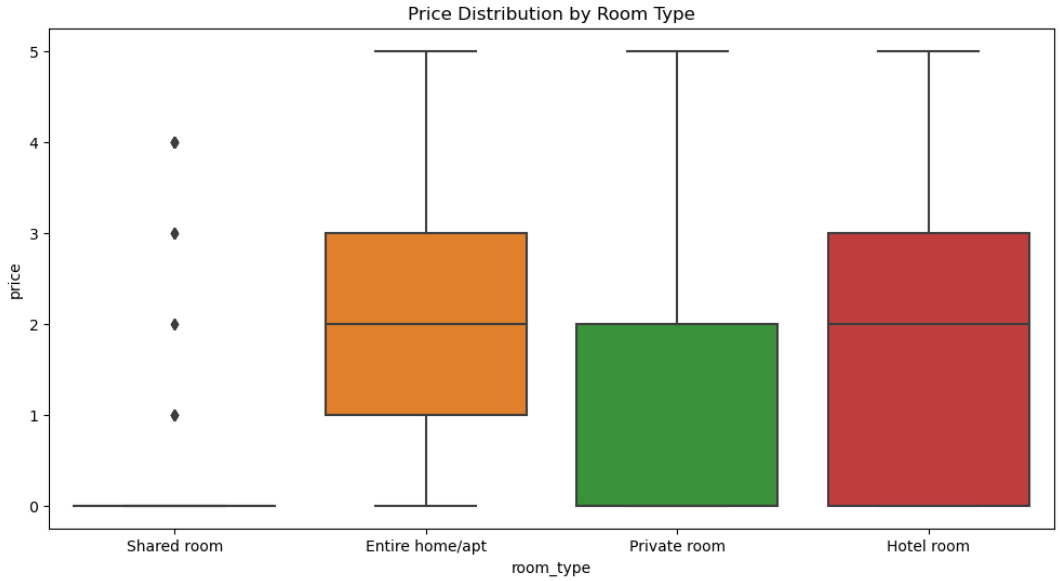


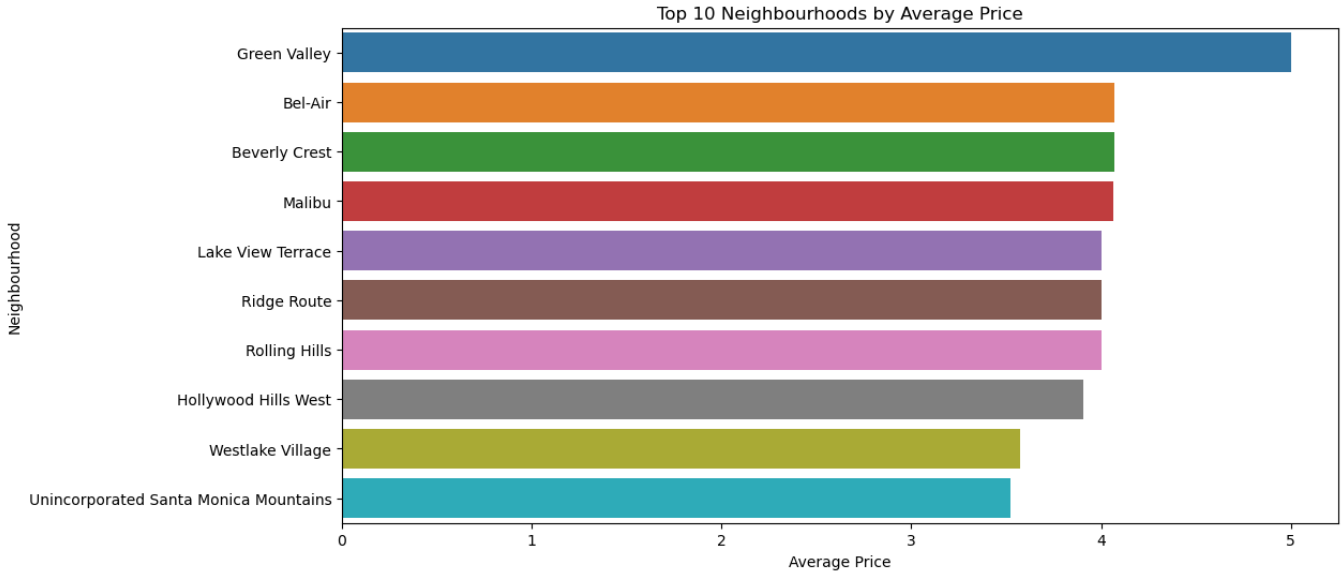
In [11]:

```
# Box Plots for Numerical Features
plt.figure(figsize=(15, 10))

# Box plot for 'price'
plt.subplot(2, 2, 1)
sns.boxplot(x=dataset_cleaned['price'])
plt.title('Box Plot of Price')

# Box plot for 'minimum_nights'
plt.subplot(2, 2, 2)
sns.boxplot(x=dataset_cleaned['minimum_nights'])
plt.title('Box Plot of Minimum Nights')

# Box plot for 'number_of_reviews'
plt.subplot(2, 2, 3)
sns.boxplot(x=dataset_cleaned['number_of_reviews'])
plt.title('Box Plot of Number of Reviews')

# Box plot for 'availability_365'
plt.subplot(2, 2, 4)
sns.boxplot(x=dataset_cleaned['availability_365'])
plt.title('Box Plot of Availability (365 Days)')

plt.tight_layout()
plt.show()
```



In [12]:
```
# Box plot for 'price' by 'room_type'
plt.figure(figsize=(12, 6))
sns.boxplot(x='room_type', y='price', data=dataset_cleaned)
plt.title('Price Distribution by Room Type')
plt.show()

# Average price per 'neighbourhood_cleansed'
avg_price_neighbourhood = dataset_cleaned.groupby('neighbourhood_cleansed')['price'].mean().sort_values(ascending=False).reset_index()

# Plotting the top 10 neighbourhoods by average price
plt.figure(figsize=(12, 6))
sns.barplot(x='price', y='neighbourhood_cleansed', data=avg_price_neighbourhood.head(10))
plt.title('Top 10 Neighbourhoods by Average Price')
plt.xlabel('Average Price')
plt.ylabel('Neighbourhood')
plt.show()
```





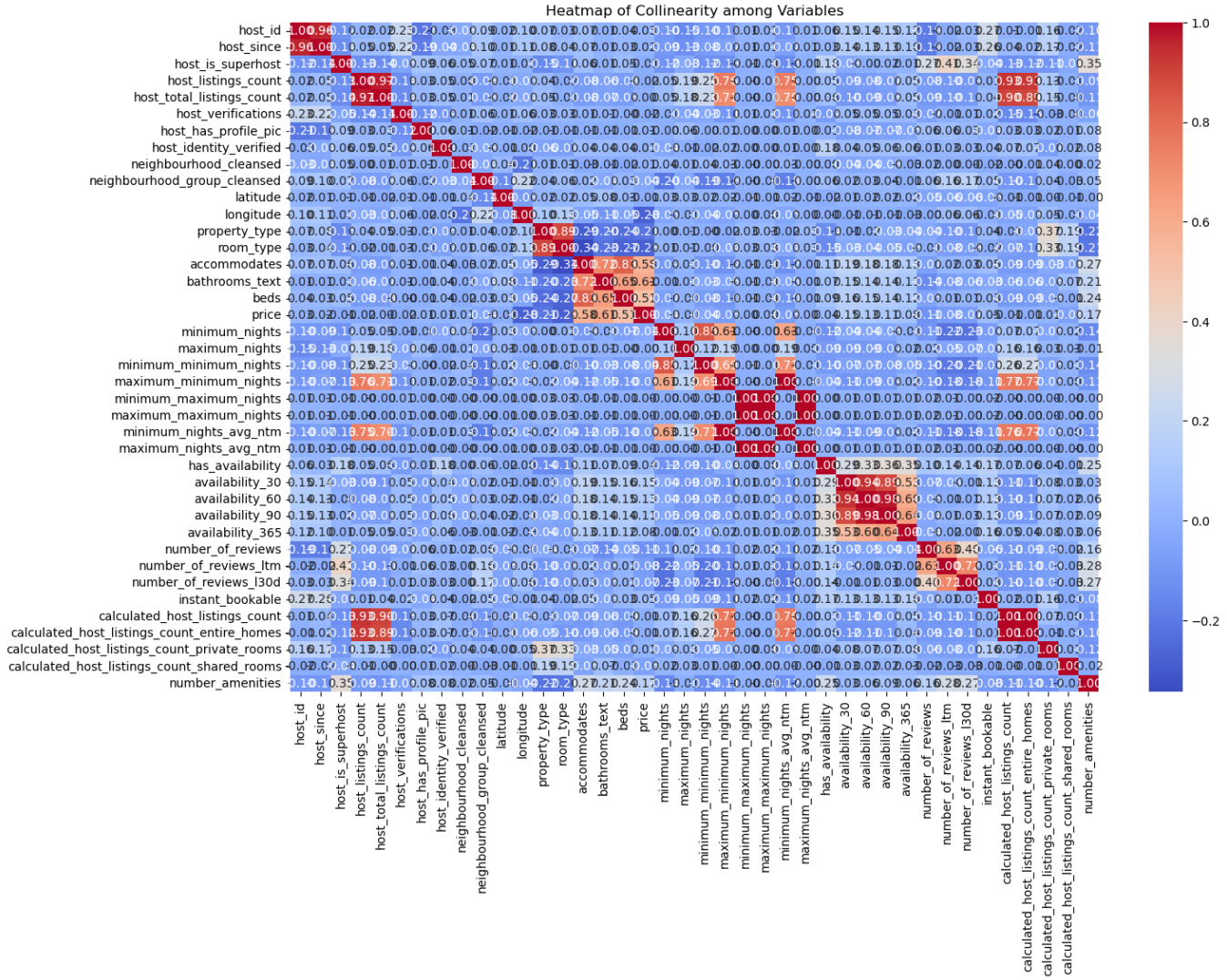
2. Model

In [13]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
```

In [14]:
```
train_df = dataset_cleaned
test_df = pd.read_csv('./test.csv')

test_df = test_df.drop(['id','name','scrape_id','last_scraped','description',
                        'picture_url','host_name','calendar_last_scraped'],axis=1)
# Counting the number of amenities for each row
# Assuming amenities are separated by commas in the 'amenities' column
test_df['number_amenities'] = test_df['amenities'].str.split(',').apply(lambda x: len(x))

# Drop amenities
test_df = test_df.drop(['amenities'],axis=1)
```

In [15]:
```
# Function to encode categorical variables with handling unseen data in test set
def encode_with_unseen(train_series, test_series):
    """Encodes train and test series, handling unseen categories in test set."""
    unique_values = np.union1d(train_series.unique(), test_series.unique())
    le = LabelEncoder().fit(unique_values)
    return le.transform(train_series), le.transform(test_series)

# Encoding categorical variables
categorical_columns = train_df.select_dtypes(include=['object']).columns
for col in categorical_columns:
    train_df[col], test_df[col] = encode_with_unseen(train_df[col].astype(str), test_df[col].astype(str))
```

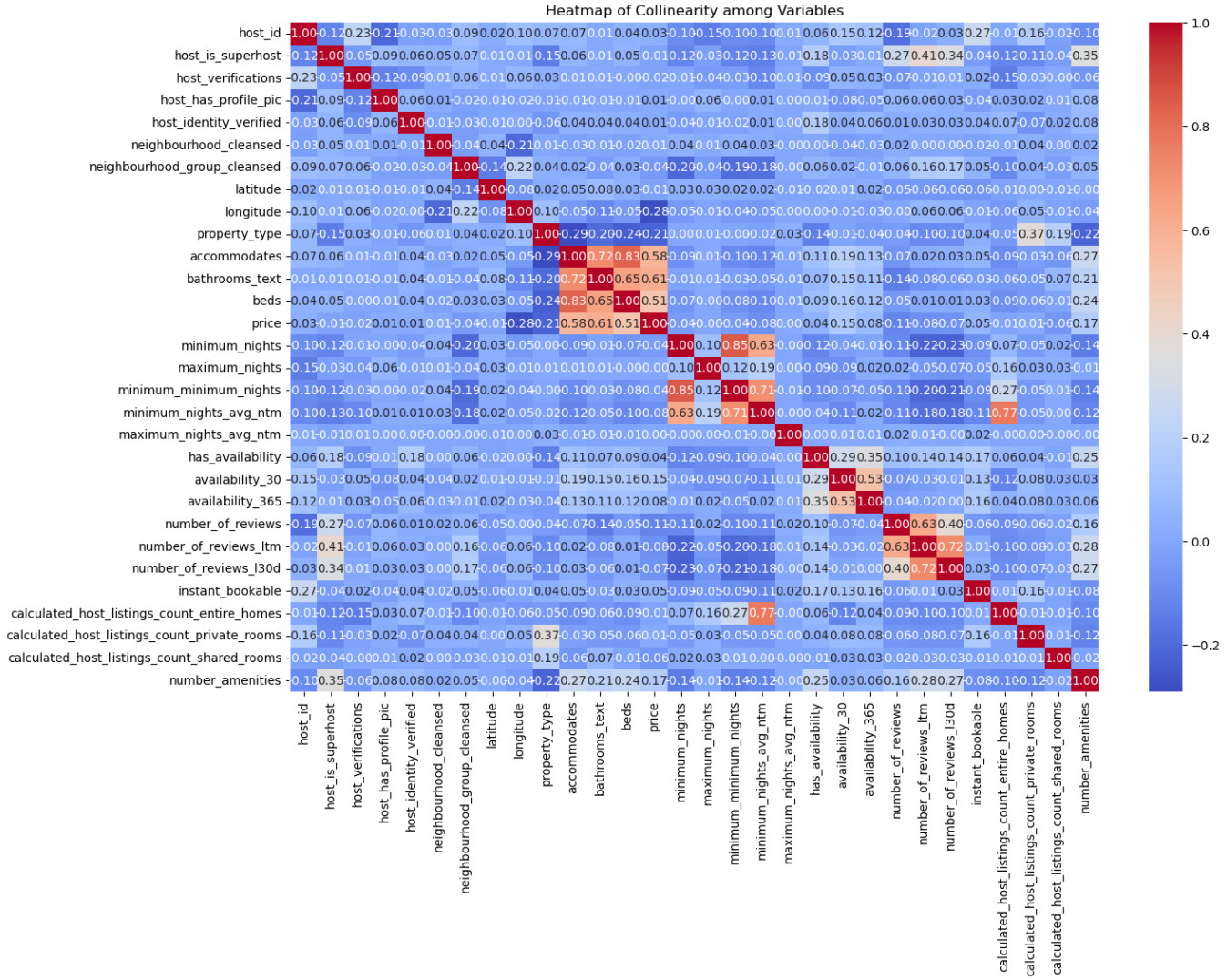
In [16]:
```
# Selecting columns for checking collinearity, excluding specified columns
cols_for_collinearity = train_df.select_dtypes(include=[np.number])

# Calculating correlation matrix
corr_matrix = cols_for_collinearity.corr()

# Plotting the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Heatmap of Collinearity among Variables')
plt.show()
```



In [17]:
```
def clean_dataset(dataset):

    columns_to_drop = ['host_since','host_listings_count','room_type',
                       'calculated_host_listings_count','maximum_maximum_nights',
                      'maximum_minimum_nights','host_total_listings_count',
                      'availability_60','availability_90','minimum_maximum_nights']

    # Drop the columns if they exist in the dataset
    dataset_cleaned = dataset.drop(columns=[col for col in columns_to_drop if col in dataset.columns], axis=1)

    return dataset_cleaned
```

In [18]:
```
train_df = clean_dataset(train_df)
test_df = clean_dataset(test_df)
```

In [19]:
```
# Selecting columns for checking collinearity, excluding specified columns
cols_for_collinearity = train_df.select_dtypes(include=[np.number])

# Calculating correlation matrix
corr_matrix = cols_for_collinearity.corr()

# Plotting the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Heatmap of Collinearity among Variables')
plt.show()
```



Decision Tree Classifier

In [20]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

In [21]:
```
# Preparing the features and target variable

X = train_df.drop('price', axis=1)
y = train_df['price']

X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,random_state=42)
```

In [22]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the DecisionForestClassifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Fit the model on the training data
dt_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_dt = dt_classifier.predict(X_test)

dt_accuracy = accuracy_score(y_test, y_pred_dt)
dt_report = classification_report(y_test, y_pred_dt)

# Display Decision Forest results
print("Classifier Accuracy:", dt_accuracy)
print("Classification Report:\n", dt_report)
```

```
Classifier Accuracy: 0.470785762256548
Classification Report:
              precision    recall  f1-score   support

         0.0       0.71      0.71      0.71       779
         1.0       0.37      0.37      0.37       558
         2.0       0.32      0.28      0.30       536
         3.0       0.35      0.37      0.36       527
         4.0       0.39      0.42      0.40       333
         5.0       0.66      0.63      0.65       245

    accuracy                           0.47      2978
   macro avg       0.47      0.46      0.46      2978
weighted avg       0.47      0.47      0.47      2978
```

Random Forest

In [23]:
```
%%time
from sklearn.ensemble import RandomForestClassifier

# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

# Fit the model on the training data
rf_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_rf = rf_classifier.predict(X_test)

rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_report = classification_report(y_test, y_pred_rf)

# Display results
print("Classifier Accuracy:", rf_accuracy)
print("Classification Report:\n", rf_report)
```

```
Classifier Accuracy: 0.5621222296843519
Classification Report:
              precision    recall  f1-score   support

         0.0       0.71      0.85      0.77       779
         1.0       0.48      0.45      0.46       558
         2.0       0.41      0.30      0.35       536
         3.0       0.43      0.51      0.47       527
         4.0       0.53      0.49      0.51       333
         5.0       0.85      0.69      0.76       245

    accuracy                           0.56      2978
   macro avg       0.57      0.55      0.55      2978
weighted avg       0.56      0.56      0.55      2978

CPU times: user 1.57 s, sys: 10.7 ms, total: 1.58 s
Wall time: 1.6 s
```

Ada Boosting

In [24]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, f1_score

# Initialize the AdaBoostClassifier
adaboost_classifier = AdaBoostClassifier(random_state=42)

# Fit the model on the training data
adaboost_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_ad = adaboost_classifier.predict(X_test)

ad_accuracy = accuracy_score(y_test, y_pred_ad)
ad_report = classification_report(y_test, y_pred_ad)

# Display results
print("Classifier Accuracy:", ad_accuracy)
print("Classification Report:\n", ad_report)
```

```
Classifier Accuracy: 0.45936870382807254
Classification Report:
              precision    recall  f1-score   support

         0.0       0.62      0.78      0.69       779
         1.0       0.38      0.36      0.37       558
         2.0       0.33      0.19      0.24       536
         3.0       0.35      0.47      0.40       527
         4.0       0.33      0.19      0.24       333
         5.0       0.56      0.60      0.58       245

    accuracy                           0.46      2978
   macro avg       0.43      0.43      0.42      2978
weighted avg       0.44      0.46      0.44      2978
```

XGBoosting

In [25]:
```
%%time
from xgboost import XGBClassifier

# Initialize the DecisionForestClassifier
xgboost_classifier = XGBClassifier(random_state=42)

# Fit the model on the training data
xgboost_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_xg = xgboost_classifier.predict(X_test)

xg_accuracy = accuracy_score(y_test, y_pred_xg)
xg_report = classification_report(y_test, y_pred_xg)
```

```
# Display results
print("Classifier Accuracy:", xg_accuracy)
print("Classification Report:\n", xg_report)
```

```
Classifier Accuracy: 0.5580926796507724
Classification Report:
               precision    recall  f1-score   support

         0.0       0.74      0.84      0.79       779
         1.0       0.45      0.45      0.45       558
         2.0       0.38      0.31      0.34       536
         3.0       0.44      0.48      0.46       527
         4.0       0.49      0.50      0.50       333
         5.0       0.87      0.70      0.77       245

    accuracy                           0.56      2978
   macro avg       0.56      0.55      0.55      2978
weighted avg       0.55      0.56      0.55      2978

CPU times: user 4.12 s, sys: 2.3 s, total: 6.42 s
Wall time: 18.2 s
```

Hyperparameter Tuning for Random Forest and XGBoosting

RF

In [26]:

```
from sklearn.model_selection import GridSearchCV
# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

# Hyperparameter grid for Random Forest Regressor
param_grid_rf = {
    'n_estimators': [100,150,200],
    'max_depth': [10,15,20],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3],
}

# Grid search with cross-validation for Random Forest
grid_search_rf = GridSearchCV(rf_classifier, param_grid_rf, cv=5, n_jobs = -1)
grid_search_rf.fit(X_train, y_train)

# Best parameters and score for Random Forest
best_params_rf = grid_search_rf.best_params_
best_params_rf
```

```
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Out[26]:

```
{'max_depth': 20,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

In [27]:

```
final_model = RandomForestClassifier(**best_params_rf, random_state=42)
final_model.fit(X_train, y_train)
final_predictions = final_model.predict(X_test)
final_accuracy = accuracy_score(y_test, final_predictions)
final_report = classification_report(y_test, final_predictions)
print("Classifier Accuracy:", final_accuracy)
print("Classification Report:\n", final_report)
```

```
Classifier Accuracy: 0.568502350570853
Classification Report:
               precision    recall  f1-score   support

         0.0       0.71      0.85      0.77       779
         1.0       0.47      0.46      0.47       558
         2.0       0.42      0.28      0.34       536
         3.0       0.44      0.55      0.49       527
         4.0       0.56      0.51      0.54       333
         5.0       0.85      0.68      0.76       245

    accuracy                           0.57      2978
   macro avg       0.58      0.55      0.56      2978
weighted avg       0.56      0.57      0.56      2978
```

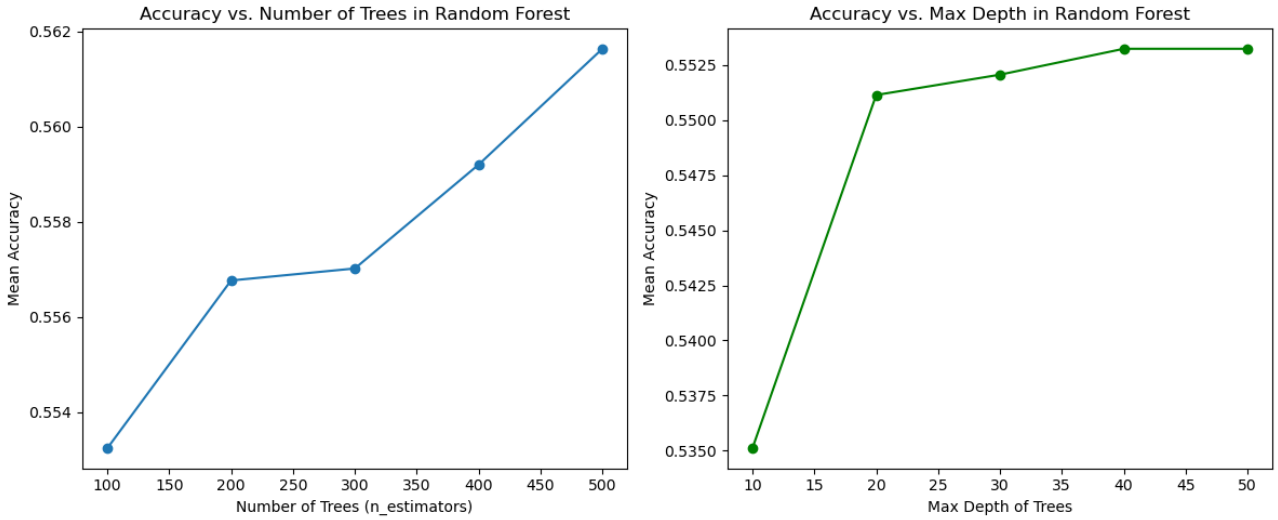
Make prediction in test dataset

In [28]:

```
predictions_rf = final_model.predict(test_df)
predictions_rf
```

```
submission_format_file_path = './sample_submission.csv'
submission_format = pd.read_csv(submission_format_file_path)

submission_ensemble1 = submission_format.copy()
submission_ensemble1['price'] = predictions_rf

submission_ensemble1.to_csv('submission_rf1.csv', index=False)
```

In [29]:
```
from sklearn.model_selection import cross_val_score
# We will use a smaller range of hyperparameters for demonstration due to time constraints
n_estimators_range = [100, 200, 300, 400,500]
max_depth_range = [10, 20, 30, 40, 50]

# Initialize lists to store mean accuracies for each hyperparameter value
mean_accuracies_estimators = []
mean_accuracies_depth = []

# Loop over n_estimators_range and calculate mean cross-validated accuracy
for n_estimators in n_estimators_range:
    model = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_accuracies_estimators.append(np.mean(scores))

# Loop over max_depth_range and calculate mean cross-validated accuracy
for max_depth in max_depth_range:
    model = RandomForestClassifier(max_depth=max_depth, random_state=42)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_accuracies_depth.append(np.mean(scores))


# Plotting accuracy vs. n_estimators
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(n_estimators_range, mean_accuracies_estimators, marker='o')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Mean Accuracy')
plt.title('Accuracy vs. Number of Trees in Random Forest')

# Plotting accuracy vs. max_depth
plt.subplot(1, 2, 2)
plt.plot(max_depth_range, mean_accuracies_depth, marker='o', color='green')
plt.xlabel('Max Depth of Trees')
plt.ylabel('Mean Accuracy')
plt.title('Accuracy vs. Max Depth in Random Forest')

plt.tight_layout()
plt.show()
```



RF Variable Importance Selection

In [30]:
```
M = 1000
B = int(0.8 * len(train_df))
forests = {}
for K in range(1, 6):
    rf_clf = RandomForestClassifier(n_estimators=M, max_depth=1, max_samples=B,
                                    max_features=K, random_state=42, criterion='gini',oob_score=True)
    rf_clf.fit(X, y)
    forests[K] = rf_clf
```

In [31]:
```
best_splits_counts = {k: {col: 0 for col in X_train.columns} for k in range(1, 6)}

for K in range(1, 6):
    for _ in range(M):
        bootstrap_sample = resample(train_df, n_samples=B,random_state=42)
        X_bootstrap = bootstrap_sample.drop('price', axis=1)
        y_bootstrap = bootstrap_sample['price']

        selected_features = np.random.choice(X_bootstrap.columns, K, replace=False)

        clf = DecisionTreeClassifier(max_depth=1, criterion='gini',random_state=42)
        clf.fit(X_bootstrap[selected_features], y_bootstrap)

        best_split_feature = selected_features[clf.tree_.feature[0]]
        best_splits_counts[K][best_split_feature] += 1

best_splits_counts = pd.DataFrame(best_splits_counts).T
best_splits_counts
```

Out[31]:

	host_id	host_is_superhost	host_verifications	host_has_profile_pic	host_identity_verified	neighbourhood_cleansed	neighbourhood_group_cleansed	latitude	longitude	property_type	...	av
1	39	28	39	40	32	30	27	28	21	43	...	
2	10	28	11	0	5	9	2	28	59	55	...	
3	3	9	6	0	0	4	0	27	60	82	...	
4	0	4	2	0	0	1	0	2	65	85	...	
5	0	0	1	0	0	0	0	5	49	122	...	

5 rows × 29 columns

In [32]:
```
# Sorting the fifth row
fifth_row_sorted = best_splits_counts.iloc[4].sort_values(ascending=False)

# Creating a new DataFrame with 'x' as the sorted values of the fifth row and 'y' as the original column names
sorted_frequency = pd.DataFrame({'Frequency': fifth_row_sorted.values, 'Variable Name': fifth_row_sorted.index})
sorted_frequency
```

Out[32]:

	Frequency	Variable Name
0	167	accommodates
1	135	bathrooms_text
2	130	beds
3	122	property_type
4	84	calculated_host_listings_count_entire_homes
5	74	calculated_host_listings_count_private_rooms
6	49	longitude
7	47	minimum_nights_avg_ntm
8	45	number_amenities
9	34	minimum_nights
10	30	minimum_minimum_nights
11	27	number_of_reviews
12	18	number_of_reviews_ltm
13	11	number_of_reviews_l30d
14	9	availability_30
15	8	calculated_host_listings_count_shared_rooms
16	5	latitude
17	3	maximum_nights_avg_ntm
18	1	maximum_nights
19	1	host_verifications
20	0	has_availability
21	0	availability_365
22	0	host_is_superhost
23	0	neighbourhood_group_cleansed
24	0	neighbourhood_cleansed
25	0	instant_bookable
26	0	host_identity_verified
27	0	host_has_profile_pic
28	0	host_id

In [33]:
```
imp_var = list(sorted_frequency['Variable Name'][0:21])
imp_var.append("price")
print(imp_var)
```

```
['accommodates', 'bathrooms_text', 'beds', 'property_type', 'calculated_host_listings_count_entire_homes', 'calculated_host_listings_count_private_rooms', 'longitude', 'minimum_nights_avg_ntm', 'number_amenities', 'minimum_nights', 'minimum_minimum_nights', 'number_of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d', 'availability_30', 'calculated_host_listings_count_shared_rooms', 'latitude', 'maximum_nights_avg_ntm', 'maximum_nights', 'host_verifications', 'has_availability', 'price']
```

In [34]:
```
train_df1 = train_df[imp_var]
```

In [35]:
```
# Preparing the features and target variable

X1 = train_df1.drop('price', axis=1)
y1 = train_df1['price']

X_train1,X_test1,y_train1,y_test1 = train_test_split(X1,y1, test_size=0.2,random_state=42)
```

In [36]:
```
from sklearn.model_selection import GridSearchCV

# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

# Hyperparameter grid for Random Forest Regressor
param_grid_rf = {
    'n_estimators': [500,600,700],
    'max_depth': [20,30,40],

}

# Grid search with cross-validation for Random Forest
grid_search_rf = GridSearchCV(rf_classifier, param_grid_rf, cv=5, n_jobs = -1)
grid_search_rf.fit(X_train1, y_train1)

# Best parameters and score for Random Forest
best_params_rf = grid_search_rf.best_params_
best_params_rf
```

```
/Users/johnnys/anaconda3/lib/python3.9/site-packages/joblib/externals/loky/process_executor.py:702: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Out[36]:
```
{'max_depth': 40, 'n_estimators': 600}
```

In [37]:
```python
final_model = RandomForestClassifier(**best_params_rf, random_state=42)
final_model.fit(X_train1, y_train1)
```

Out[37]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=40, n_estimators=600, random_state=42)
```

In [38]:
```python
final_predictions = final_model.predict(X_test1)
final_accuracy = accuracy_score(y_test1, final_predictions)
final_report = classification_report(y_test1, final_predictions)
print("Classifier Accuracy:", final_accuracy)
print("Classification Report:\n", final_report)
```

```
Classifier Accuracy: 0.5607790463398253
Classification Report:
               precision    recall  f1-score   support

         0.0       0.72      0.85      0.78       779
         1.0       0.46      0.45      0.46       558
         2.0       0.40      0.28      0.33       536
         3.0       0.44      0.55      0.49       527
         4.0       0.54      0.47      0.50       333
         5.0       0.83      0.67      0.74       245

    accuracy                           0.56      2978
   macro avg       0.57      0.54      0.55      2978
weighted avg       0.55      0.56      0.55      2978
```

Make prediction in test dataset

In [ ]:
```python
imp_var.remove("price")
```

In [42]:
```python
test_df1 = test_df[imp_var]
```

In [43]:
```python
predictions_rf = final_model.predict(test_df1)
predictions_rf

submission_format_file_path = './sample_submission.csv'
submission_format = pd.read_csv(submission_format_file_path)

submission_ensemble2 = submission_format.copy()
submission_ensemble2['price'] = predictions_rf

submission_ensemble2.to_csv('submission_rf2.csv', index=False)
```

In [44]:
```python
submission_ensemble2.head()
```

Out[44]:

	id	price
0	0	3.0
1	1	2.0
2	2	3.0
3	3	2.0
4	4	3.0

XGBoosting

In [45]:
```python
X = train_df.drop('price', axis=1)
y = train_df['price']

X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,random_state=42)
```

In [46]:
```python
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import RandomizedSearchCV, cross_val_score

param = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

random_search = RandomizedSearchCV(xgb, param_distributions=param, n_iter=50,
                                   scoring='accuracy', n_jobs=-1, cv=5, random_state=42)
random_search.fit(X_train, y_train)

# Best hyperparameters
```

```
best_params = random_search.best_params_

# Retrain model with best parameters
xgb_classifier = XGBClassifier(**best_params, use_label_encoder=False, eval_metric='mlogloss')
xgb_classifier.fit(X_train, y_train)
```

```
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Out[46]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=1.0, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=5,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=300,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

In [47]:

```
y_pred_xgb = xgb_classifier.predict(X_test)
print("Detailed classification report:")
print()
print(classification_report(y_test, y_pred_xgb))
print()
```

```
Detailed classification report:

              precision    recall  f1-score   support

         0.0       0.73      0.84      0.78       779
         1.0       0.45      0.45      0.45       558
         2.0       0.41      0.30      0.35       536
         3.0       0.44      0.48      0.46       527
         4.0       0.52      0.51      0.52       333
         5.0       0.85      0.71      0.78       245

    accuracy                           0.56      2978
   macro avg       0.57      0.55      0.56      2978
weighted avg       0.55      0.56      0.55      2978
```

In [48]:

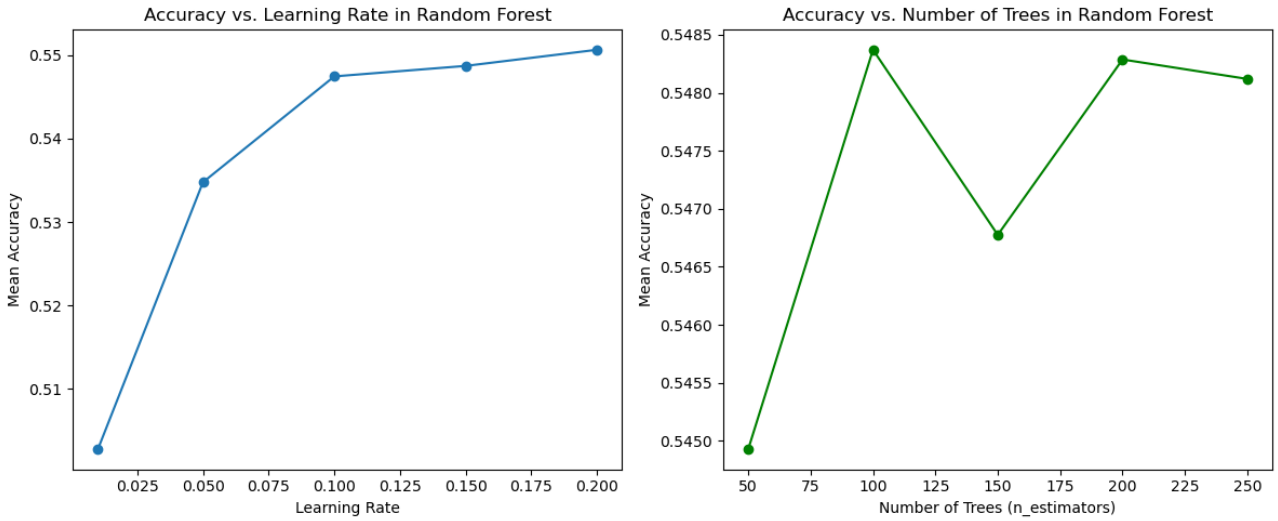
```
predictions_xgb = xgb_classifier.predict(test_df)
predictions_xgb

submission_format_file_path = './sample_submission.csv'
submission_format = pd.read_csv(submission_format_file_path)

submission_ensemble3 = submission_format.copy()
submission_ensemble3['price'] = predictions_xgb

submission_ensemble3.to_csv('submission_xgb1.csv', index=False)
```

In [ ]:

In [49]:

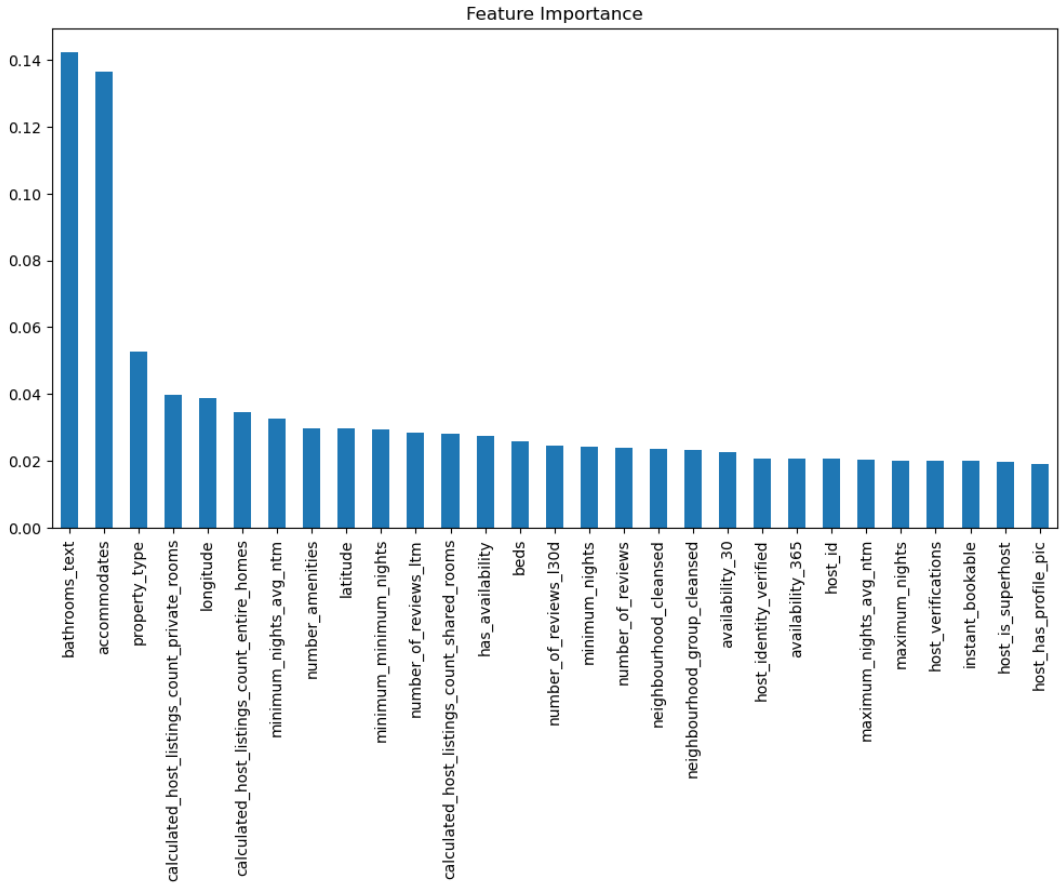
```
learning_rates_range = [0.01,0.05,0.1,0.15,0.2]
n_estimators_range = [50, 100,150,200,250]

# Initialize lists to store mean accuracies for each hyperparameter value
mean_accuracies_learn = []
mean_accuracies_estimator = []

# Loop over n_estimators_range and calculate mean cross-validated accuracy
for learning_rate in learning_rates_range:
    model = XGBClassifier(learning_rate=learning_rate, random_state=42)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_accuracies_learn.append(np.mean(scores))

# Loop over max_depth_range and calculate mean cross-validated accuracy
for n_estimators in n_estimators_range:
    model = XGBClassifier(n_estimators=n_estimators, random_state=42)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_accuracies_estimator.append(np.mean(scores))


# Plotting accuracy vs. n_estimators
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
```

```
plt.plot(learning_rates_range, mean_accuracies_learn, marker='o')
plt.xlabel('Learning Rate')
plt.ylabel('Mean Accuracy')
plt.title('Accuracy vs. Learning Rate in Random Forest')

# Plotting accuracy vs. max_depth
plt.subplot(1, 2, 2)
plt.plot(n_estimators_range, mean_accuracies_estimator, marker='o', color='green')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Mean Accuracy')
plt.title('Accuracy vs. Number of Trees in Random Forest')

plt.tight_layout()
plt.show()
```



XGBoosting Variable Selection

In [50]:
```
# Feature Importance
importances = xgb_classifier.feature_importances_
feature_names = X_train.columns
feature_importances = pd.Series(importances, index=feature_names).sort_values(ascending=False)

# Plotting Feature Importances
plt.figure(figsize=(12, 6))
feature_importances.plot(kind='bar')
plt.title('Feature Importance')
plt.show()
```



In [51]:

```
imp_var_xgb = list(feature_importances[feature_importances > 0.02].index)
imp_var_xgb.append('price')
# Printing the index labels
print(imp_var_xgb)
```

```
['bathrooms_text', 'accommodates', 'property_type', 'calculated_host_listings_count_private_rooms', 'longitude', 'calculated_host_listings_count_entire_homes', 'minimum_nights_avg_ntm', 'number_amenities', 'latitude', 'minimum_minimum_nights', 'number_of_reviews_ltm', 'calculated_host_listings_count_shared_rooms', 'has_availability', 'beds', 'number_of_reviews_l30d', 'minimum_nights', 'number_of_reviews', 'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'availability_30', 'host_identity_verified', 'availability_365', 'host_id', 'maximum_nights_avg_ntm', 'maximum_nights', 'host_verifications', 'price']
```

In [52]:

```
train_df2 = train_df[imp_var_xgb]
X2 = train_df2.drop('price', axis=1)
y2 = train_df2['price']

X_train2,X_test2,y_train2,y_test2 = train_test_split(X2,y2, test_size=0.2,random_state=42)
```

In [53]:

```
param = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

xgb2 = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

random_search2 = RandomizedSearchCV(xgb, param_distributions=param, n_iter=50,
                                   scoring='accuracy', n_jobs=-1, cv=5, random_state=42)
random_search2.fit(X_train2, y_train2)

# Best hyperparameters
best_params2 = random_search2.best_params_

# Retrain model with best parameters
xgb_classifier2 = XGBClassifier(**best_params2, use_label_encoder=False, eval_metric='mlogloss')
xgb_classifier2.fit(X_train2, y_train2)
```

```
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/Users/johnnys/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Out[53]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.2, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=5,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=100,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

In [54]:

```
best_params2
```

Out[54]:

```
{'subsample': 0.9,
 'n_estimators': 100,
 'max_depth': 5,
 'learning_rate': 0.2,
 'colsample_bytree': 0.8}
```

In [55]:

```
y_pred_xgb2 = xgb_classifier2.predict(X_test2)
print("Detailed classification report:")
print()
print(classification_report(y_test2, y_pred_xgb2))
print()
```

```
Detailed classification report:

              precision    recall  f1-score   support

         0.0       0.72      0.84      0.77       779
         1.0       0.45      0.44      0.45       558
         2.0       0.38      0.27      0.31       536
         3.0       0.42      0.49      0.46       527
         4.0       0.53      0.52      0.52       333
         5.0       0.88      0.69      0.78       245

    accuracy                           0.55      2978
   macro avg       0.56      0.54      0.55      2978
weighted avg       0.55      0.55      0.55      2978
```

In [56]:
```
imp_var_xgb.remove('price')
```

In [57]:
```
test_df2 = test_df[imp_var_xgb]
```

In [58]:
```
predictions_xgb = xgb_classifier2.predict(test_df2)
predictions_xgb

submission_format_file_path = './sample_submission.csv'
submission_format = pd.read_csv(submission_format_file_path)

submission_ensemble4 = submission_format.copy()
submission_ensemble4['price'] = predictions_xgb

submission_ensemble4.to_csv('submission_xgb2.csv', index=False)
```

In [59]:
```
# Count how many times each price appears in the DataFrame to plot the frequency of each price category.

# Calculate the counts of each price
price_counts = submission_ensemble1['price'].value_counts().sort_index()

# Plot bar plot with price on the x-axis and counts on the y-axis
plt.figure(figsize=(8, 4))
plt.bar(price_counts.index, price_counts.values, color='lightgreen')
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Count of Prices')
plt.xticks(price_counts.index)
plt.grid(True)
plt.show()
```

In [60]:
```
# Count how many times each price appears in the DataFrame to plot the frequency of each price category.

# Calculate the counts of each price
price_counts = submission_ensemble2['price'].value_counts().sort_index()

# Plot bar plot with price on the x-axis and counts on the y-axis
plt.figure(figsize=(8, 4))
plt.bar(price_counts.index, price_counts.values, color='red')
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Count of Prices')
plt.xticks(price_counts.index)
plt.grid(True)
plt.show()
```

In [61]:
```
# Count how many times each price appears in the DataFrame to plot the frequency of each price category.

# Calculate the counts of each price
price_counts = submission_ensemble3['price'].value_counts().sort_index()

# Plot bar plot with price on the x-axis and counts on the y-axis
plt.figure(figsize=(8, 4))
plt.bar(price_counts.index, price_counts.values, color='lightblue')
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Count of Prices')
plt.xticks(price_counts.index)
plt.grid(True)
plt.show()
```

In [62]:
```
# Count how many times each price appears in the DataFrame to plot the frequency of each price category.

# Calculate the counts of each price
price_counts = submission_ensemble4['price'].value_counts().sort_index()

# Plot bar plot with price on the x-axis and counts on the y-axis
plt.figure(figsize=(8, 4))
plt.bar(price_counts.index, price_counts.values, color='gray')
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Count of Prices')
plt.xticks(price_counts.index)
plt.grid(True)
plt.show()
```

Try to use all features

In [63]:
```
# model-new
import pandas as pd
```

```
# Load the datasets. Try to use all features
train_df = pd.read_csv('./train.csv')
test_df = pd.read_csv('./test.csv')
sample_submission_df = pd.read_csv('./sample_submission.csv')

# Display the first few rows of each dataset to understand their structure
train_head = train_df.head()
test_head = test_df.head()
sample_submission_head = sample_submission_df.head()
```

In [64]:
```
def encode_with_unseen(train_series, test_series):
    """Encodes train and test series, handling unseen categories in test set."""
    unique_values = np.union1d(train_series.unique(), test_series.unique())
    le = LabelEncoder().fit(unique_values)
    return le.transform(train_series), le.transform(test_series)
```

In [65]:
```
# Encoding categorical variables
categorical_columns = train_df.select_dtypes(include=['object']).columns
for col in categorical_columns:
    train_df[col], test_df[col] = encode_with_unseen(train_df[col].astype(str), test_df[col].astype(str))

# Splitting the train data into features and target
X = train_df.drop(['price', 'id'], axis=1)
y = train_df['price']

# Handling missing values - imputing with median
X.fillna(X.median(), inplace=True)
test_df.fillna(test_df.median(), inplace=True)

# Split the training data for validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_val)

# Evaluate Random Forest
rf_accuracy = accuracy_score(y_val, rf_predictions)
rf_report = classification_report(y_val, rf_predictions)

# Gradient Boosting Classifier (XGBoost)
xgb_classifier = XGBClassifier(random_state=42)
xgb_classifier.fit(X_train, y_train)
xgb_predictions = xgb_classifier.predict(X_val)

# Evaluate XGBoost
xgb_accuracy = accuracy_score(y_val, xgb_predictions)
xgb_report = classification_report(y_val, xgb_predictions)

# Display Random Forest results
print("Random Forest Classifier Accuracy:", rf_accuracy)
print("Classification Report:\n", rf_report)

# Display XGBoost results (if available)
print("XGBoost Classifier Accuracy:", xgb_accuracy)
print("Classification Report:\n", xgb_report)
```

```
Random Forest Classifier Accuracy: 0.5613467141469731
Classification Report:
               precision    recall  f1-score   support

         0.0       0.73      0.85      0.78       797
         1.0       0.47      0.45      0.46       595
         2.0       0.43      0.33      0.37       529
         3.0       0.44      0.54      0.49       544
         4.0       0.49      0.41      0.45       349
         5.0       0.81      0.66      0.73       275

    accuracy                           0.56      3089
   macro avg       0.56      0.54      0.54      3089
weighted avg       0.56      0.56      0.55      3089

XGBoost Classifier Accuracy: 0.5500161864681127
Classification Report:
               precision    recall  f1-score   support

         0.0       0.75      0.80      0.78       797
         1.0       0.45      0.46      0.46       595
         2.0       0.38      0.32      0.35       529
         3.0       0.44      0.51      0.47       544
         4.0       0.49      0.42      0.45       349
         5.0       0.77      0.70      0.73       275

    accuracy                           0.55      3089
   macro avg       0.55      0.54      0.54      3089
weighted avg       0.55      0.55      0.55      3089
```

In [66]:
```
from sklearn.model_selection import GridSearchCV

# Grid of parameters to search
param_grid = {
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3],
    'n_estimators': [100, 120, 150, 180,200]
}

# Create a base model
rf = RandomForestClassifier(random_state=42)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=10, n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Best parameters
best_params = grid_search.best_params_
```

```
# Train the final model with the best parameters
final_model = RandomForestClassifier(**best_params, random_state=42)
final_model.fit(X_train, y_train)

# Evaluate the final model on the validation set
final_predictions = final_model.predict(X_val)
final_accuracy = accuracy_score(y_val, final_predictions)
final_report = classification_report(y_val, final_predictions)
```

```
Fitting 10 folds for each of 135 candidates, totalling 1350 fits
```

In [67]:
```
best_params
```

Out[67]:
```
{'max_depth': 20,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

In [68]:
```
print(best_params,final_accuracy)
print(final_report)
```

```
{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200} 0.5704111362900615
              precision    recall  f1-score   support

         0.0       0.73      0.85      0.78       797
         1.0       0.48      0.47      0.47       595
         2.0       0.44      0.31      0.37       529
         3.0       0.44      0.56      0.49       544
         4.0       0.52      0.44      0.48       349
         5.0       0.83      0.67      0.74       275

    accuracy                           0.57      3089
   macro avg       0.57      0.55      0.56      3089
weighted avg       0.57      0.57      0.56      3089
```

In [69]:
```
# Train the final model with the best parameters
final_model = RandomForestClassifier(**best_params, random_state=42)
final_model.fit(X, y)
```

Out[69]:
```
RandomForestClassifier(max_depth=20, n_estimators=200, random_state=42)
```

In [70]:
```
# During prediction
test_features = test_df.drop('id', axis=1)  # Exclude 'id' from the features used for prediction
predictions = final_model.predict(test_features)

# Prepare the submission file
submission_df = pd.DataFrame({'id': test_df['id'], 'price': predictions})
submission_df.to_csv('submission_rf_best3.csv', index=False)
```

In [71]:
```
# Count how many times each price appears in the DataFrame to plot the frequency of each price category.

# Calculate the counts of each price
price_counts = submission_df['price'].value_counts().sort_index()

# Plot bar plot with price on the x-axis and counts on the y-axis
plt.figure(figsize=(8, 4))
plt.bar(price_counts.index, price_counts.values, color='black')
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Count of Prices')
plt.xticks(price_counts.index)
plt.grid(True)
plt.show()
```