Joseph Carluccio
BisonBnB Consulting
Lewisburg, PA 17837
April 2024

Dear Alia Stanciu:

On behalf of BisonBnB Consulting, we thank you for this opportunity. The goal was to accurately predict the pricing of an Airbnb Listing in New York City, using property and listing attributes. As you know, this prediction can be crucial for hosts and guests, as it aids hosts in setting competitive prices and assists guests in making informed booking decisions.

Given the provided data, we have completed an exploratory data analysis prepped the data, and then built and tested the performance of 4 different prediction models. Based upon our analysis, we would recommend implementing the gradient boosting model to most accurately predict the listing price as this one had the best performance based on various metrics.

DATA: SUMMARY AND GENERAL PREPARATION

The dataset consisted of information about 48,895 Airbnb listings across the 5 boroughs of New York City. The information about each listing was 16 different attributes - 11 quantitative and 5 categorical.

The information about each listing:

listing ID

name - name of the listing

host ID

host_name - name of the host

neighbourhood_group - which of the 5 boroughs is the listing located in

neighbourhood - neighborhood the listing is located in

latitude - latitude coordinates

longitude - longitude coordinates

room_type - listing space type

price - price in dollars

minimum_nights - the amount of nights minimum

number_of_reviews - number of reviews

last_review - date the last review was given reviews_per_month - number of reviews per month

calculated_host_listings_count - the number of listings per host

availability_365 - number of days when the listing is available for booking

Missing Data:

Of the data provided, 10052 listings had at least one null data field.

Of the fields missing, they were all in fields relating to reviews or the name of the host. So, we filled these missing values with 0 as they were not important to our Regression Models.

We addressed these missing values before building any of the models. We thought this would give the best comparison of performance across them.

Outliers:

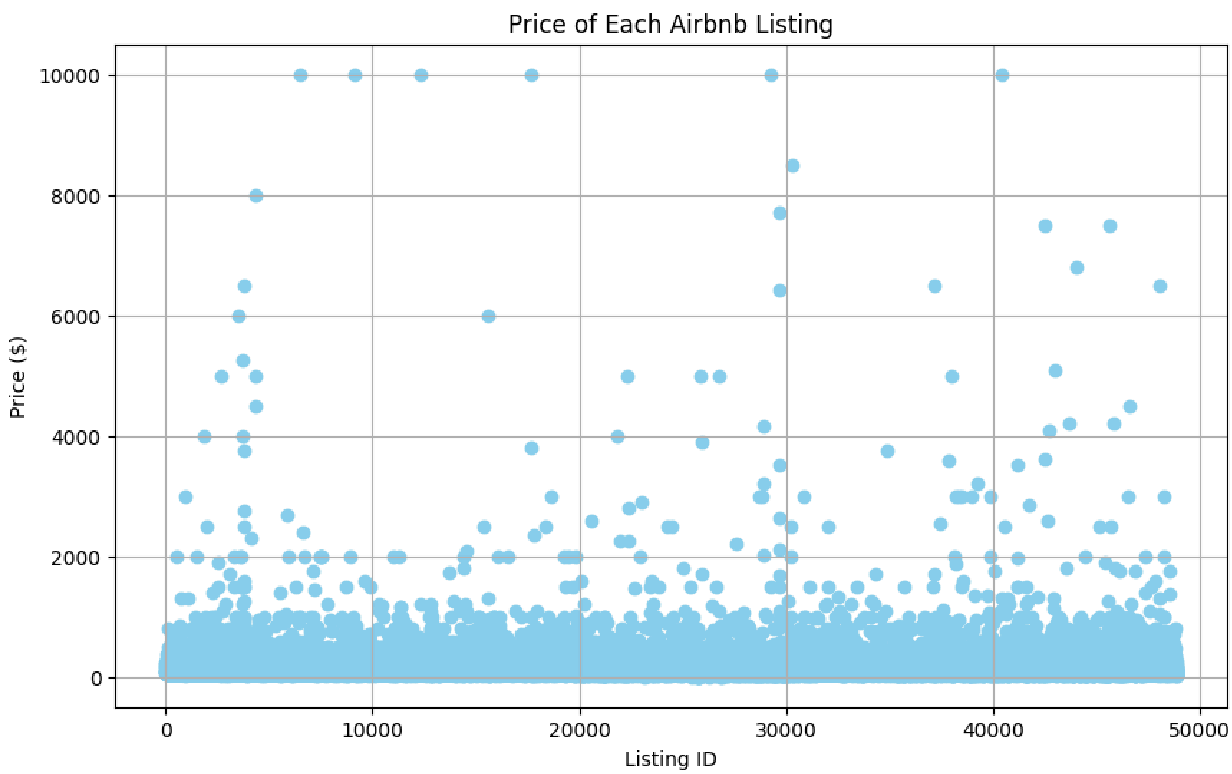


Figure 1.

Scatterplot of Airbnb Prices

Of the 48,895 listings, 6021 were identified as outliers. These outliers can be seen in Figure 1 with some Airbnbs being priced as high as $10,000. Since this is quite a significant portion of our dataset, these outliers were left in the data. Also, since 2 of our implemented models are tree-based (which are naturally less likely to be affected by outliers), these outliers should have very minimal effects on the quality of the analysis and the performance of these models.

Training and Validation Data:

For each of the 4 models, the data was split into 80% training data (39116 listings) and 20% testing or validation data (9779 listings)

<u>MODELING:</u>

4 different models were implemented all with the same parameters and target value

After the initial EDA, we determined which parameters had the most effect on price: neighborhood group, latitude, longitude, room type, minimum nights, number of reviews, last review, and availability 365. The target value for all of these models was price.

Selected Models

1. Decision Tree Regression:

> Performed Grid Search to tune max_depth, min_samples_split, and min_samples_leaf. Found that the optimal values for each were:
>
> max depth, which controls the maximum number of levels in the tree = 7
>
> minimal samples per leaf, which controls the minimum number of samples required to consider splitting a node further = 10
>
> minimal samples per split, which controls the minimum number of samples required to consider a node as a leaf (i.e., not split further) = 2
>
> Tested the performance of the model using 5-fold cross-validation with negative mean absolute error as the scoring criterion. The output showing the Decision Tree can be seen in Figure 1.

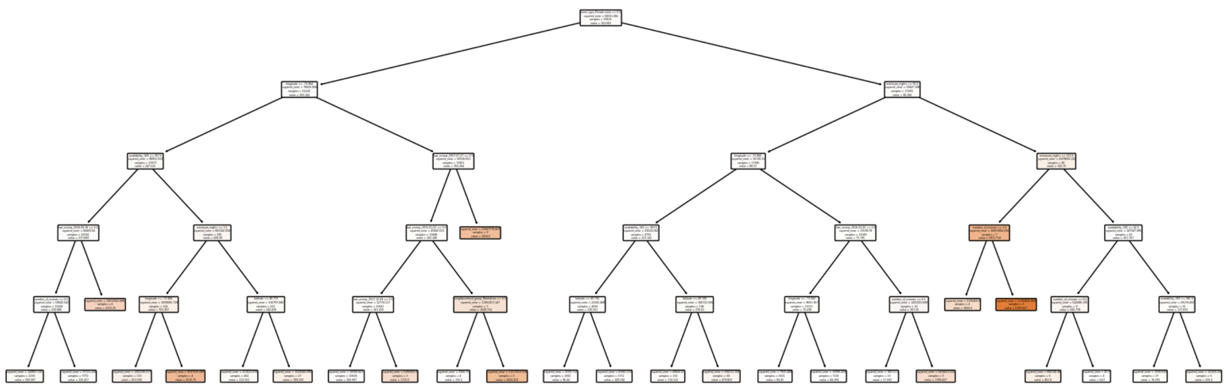


Figure 2.

Decision Tree Regression Model

2. LassoCV (Lasso Regression with Cross-Validation):

Implemented using basic sci-kit learn functionality. To be used in comparing Decision Tree Regressor performance. Grid Search CV was not needed for LassoCV, as it handles hyperparameter tuning internally

3. Gradient Boosting Regression:

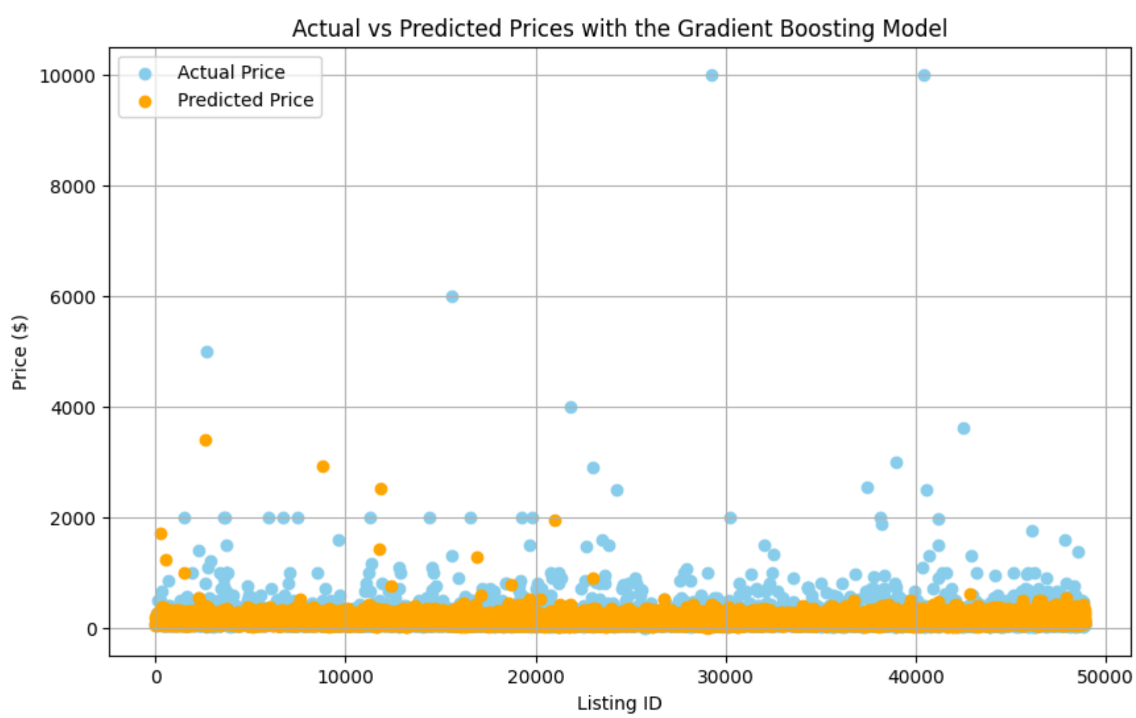


Figure 3.

Scatterplot Showing the Actual Vs. Predicted Price using Gradient Boosting Model

The Gradient Boosting Model is similar to a Decision Tree Regressor. By fitting smaller decision trees to the residuals of the previous trees, gradient boosting can gradually improve the predictive performance of the model.

We attempted to perform hyperparameter tuning on this model but ultimately ran into extensive run times with no foreseeable outcome. We attempted to first perform a Grid Search with all the possible hyperparameters of the model but found this to run for an extended and unknown amount of time. We shortened the possible range of hyperparameters while now only focusing on the maximum depth of each tree and the minimum samples to split the leaf further. This once against resulted in extensive run times with no end. We then tried to use a random search with 100 iterations followed by only 10 iterations. Both of these processes ran for a total of 1.5 hours. perhaps this was due to an error somewhere in the code or to a lack of computing power.

Since we are unable to see the possible hundreds of trees that the model creates, we wanted to visualize the performance of the model using Figure 3. As we can see, this model does a good job of predicting the price of the listings, even with some of the outliers.

4. Dummy Model:

The Dummy model predicts the price of the Airbnb Listing by simply taking the average price of all the Airbnb listings included in the dataset. This is heavily skewed by any outliers and clearly does not take into account any characteristics of the listings.

MODEL PERFORMANCE

In the context of our goal,

MAE is the average value that our model's prediction is off from the actual price. This can be in either the negative or positive direction.

MSE is the average difference between predicted and actual prices, squared.

RMSE is the square root of the MSE. This metric gives us a better idea of the actual performance of the model as it minimizes the effects of outliers.

Performance of Models

Model	DT Regression	LassoCV	Gradient Boosting	Dummy (mean)
MAE	**69.73**	**74.92**	**69.02**	**93.34**
MSE	**49996.24**	**49722.40**	**50282.01**	**55028.32**
RMSE	**223.60**	**222.24**	**224.24**	**234.60**

Table 1.

Performance of Models

From Table 1, we can see that the best-performing model was the gradient boosting model with Our decision tree regression performing second best, LassoCV performing third best, and our dummy model performing the worst.

Our best model was the Gradient boosting model with a prediction that was on average, off by $69.73 of the actual price.

This was a significant improvement over our worst model, the dummy model which on average, was off by $93.354 of the actual price.

<u>IMPLEMENTED MODEL FOR PERFORMANCE</u>

From Table 1, we can see that the Gradient Boosting model performed the best, and as such, that is the one that we would recommend implementing. Our Gradient Boosting model can accurately predict the nightly rate of the Airbnb Listing within $70. obviously, in practice, this is not a good tool to give you an exact number to include in the listing, but does offer value for the host to have an estimate of the value of their listing, and even more value for the guest as they can examine the listing and judge whether it is worth the extra $70 (or $70 less!).

One thing that we would suggest to increase the performance of the model is hyperparameter tuning to further optimize the model. We attempted to do so with a Grid Search and Random Search but ultimately ran into a limit with computing power. In the near future, we are upgrading our computers here at BisonBnB and would love the opportunity to continue working on the model.

<u>SECTION 5. APPENDIX</u>

In [ ]:

```
# import packages
import warnings
warnings.filterwarnings('ignore')

import os

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import randint

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, cross_va
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
from sklearn.linear_model import LassoCV
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.dummy import DummyRegressor




# Set the display options to print the dataframe
# pd.options.display.float_format = '{:,.3f}'.format
```

We are going to try and predict air bnb prices in NY based off of information about the airbnb

In [ ]:

```
# Readin the data
air_df = pd.read_csv('Data/AB_NYC_2019.csv')
air_df.head()
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem

In [ ]:

```
air_df.info()
```

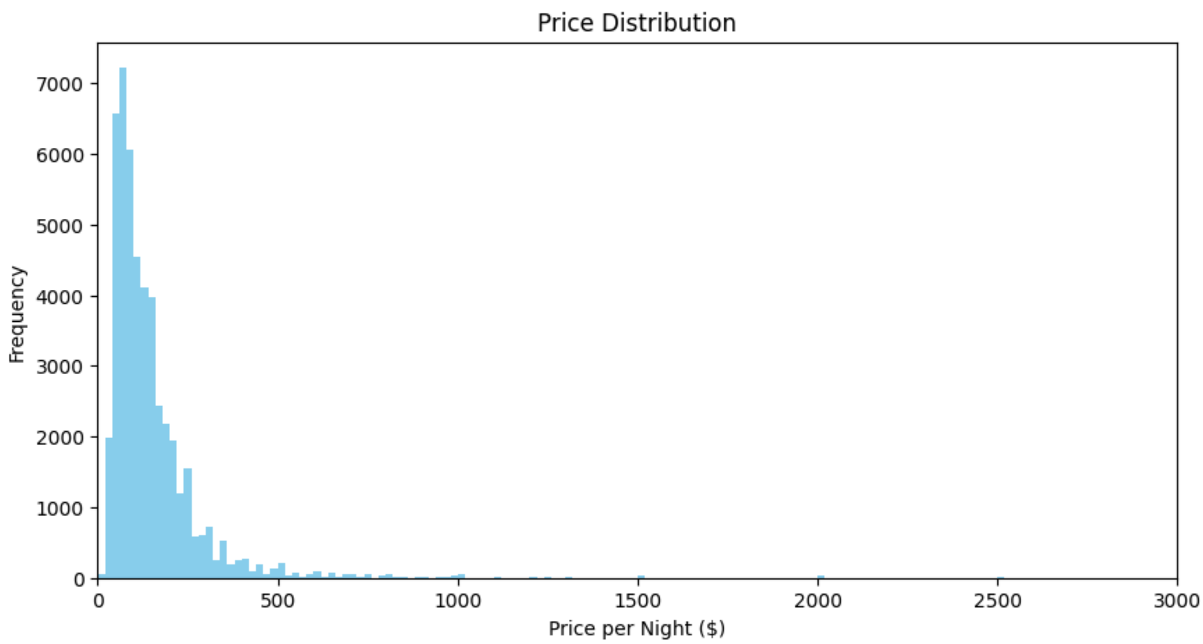
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   name                            48879 non-null  object 
 2   host_id                         48895 non-null  int64  
 3   host_name                       48874 non-null  object 
 4   neighbourhood_group             48895 non-null  object 
 5   neighbourhood                   48895 non-null  object 
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object 
 9   price                           48895 non-null  int64  
 10  minimum_nights                  48895 non-null  int64  
 11  number_of_reviews               48895 non-null  int64  
 12  last_review                     38843 non-null  object 
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64  
 15  availability_365                48895 non-null  int64  
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

After the quick look at all of our data's features and their types, lets now examine the distribution for our target value, price:

In [ ]:

```
# Plot Histogram of the price distirbution

# Make the figure
fig, ax = plt.subplots(figsize = (10,5))
```

```python
# set labels
ax.set_xlabel("Price per Night ($)")
ax.set_ylabel("Frequency")
ax.set_title("Price Distribution")

# change x axis to only show values between 0 and 3000
plt.xlim(0, 3000)

plt.hist(air_df['price'], bins = 500, color = 'skyblue')
plt.show()
```



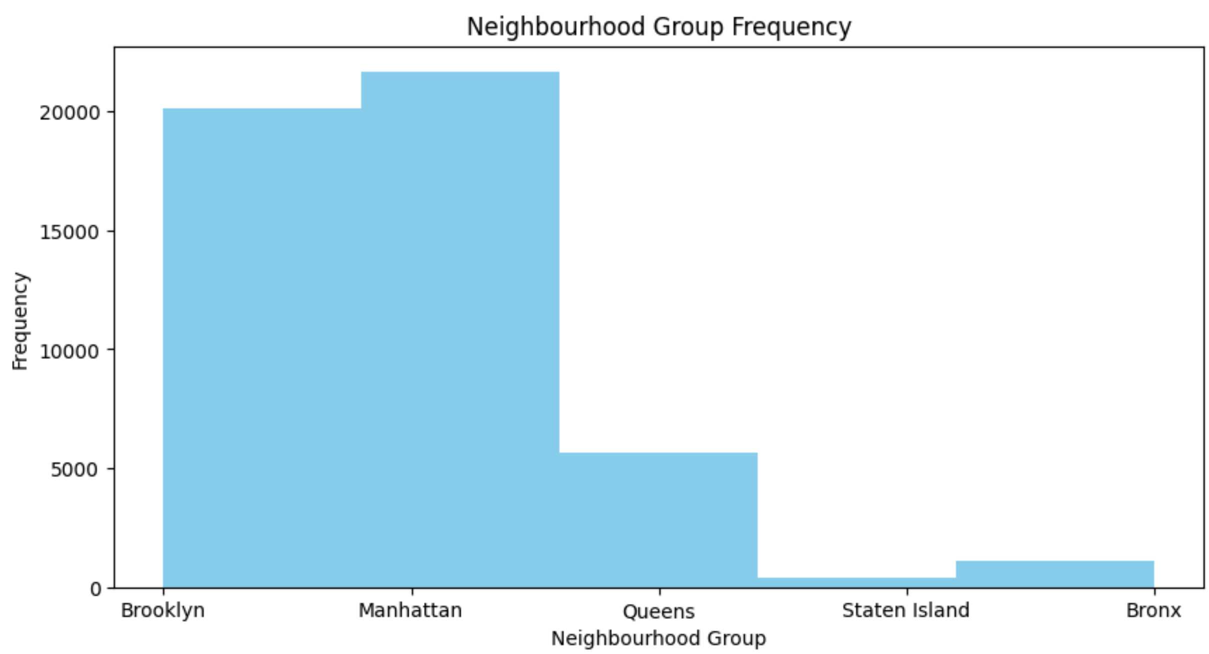
Find the minimum, maximum, median, and mean price in our data

In [ ]:
```python
# we can get most of our required values from the .describe() function in pa
price_summary = air_df['price'].describe()

# Extracting median separately
median_price = air_df['price'].median()

print("Summary statistics for price:")
print(price_summary)
print("Median price:", median_price)
```

```
Summary statistics for price:
count    48895.000000
mean       152.720687
std        240.154170
min          0.000000
25%         69.000000
50%        106.000000
75%        175.000000
max      10000.000000
Name: price, dtype: float64
Median price: 106.0
```

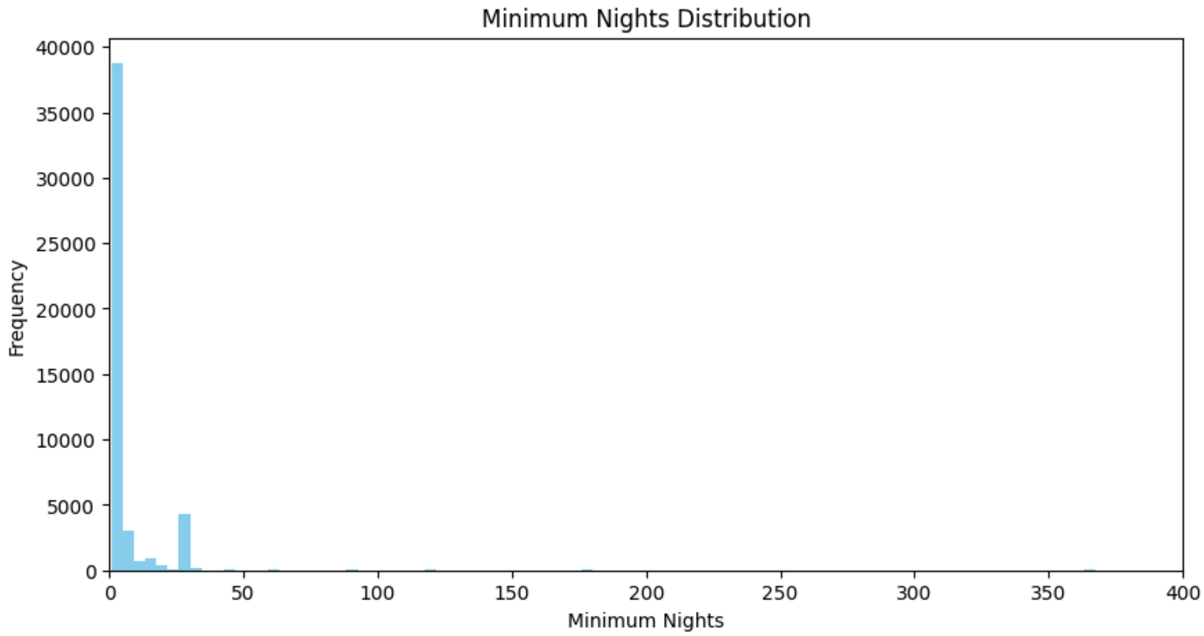
Now lets examine the distribution for other features of the data. lets try to focus on ones that wil help us best predict the value of price. From real world experience, my assumptions for two features that will give us the most insight are neighborhood, minimum_nights, and room_type.

In [ ]:
```python
# Plot Histogram of the neighbourhood_group

# Make the figure
fig, ax = plt.subplots(figsize = (10,5))

# set labels
ax.set_xlabel("Neighbourhood Group")
ax.set_ylabel("Frequency")
ax.set_title("Neighbourhood Group Frequency")

# change x axis to only show values between 0 and 3000
#plt.xlim(0, 3000)

plt.hist(air_df['neighbourhood_group'], bins = 5, color = 'skyblue')
plt.show()
```



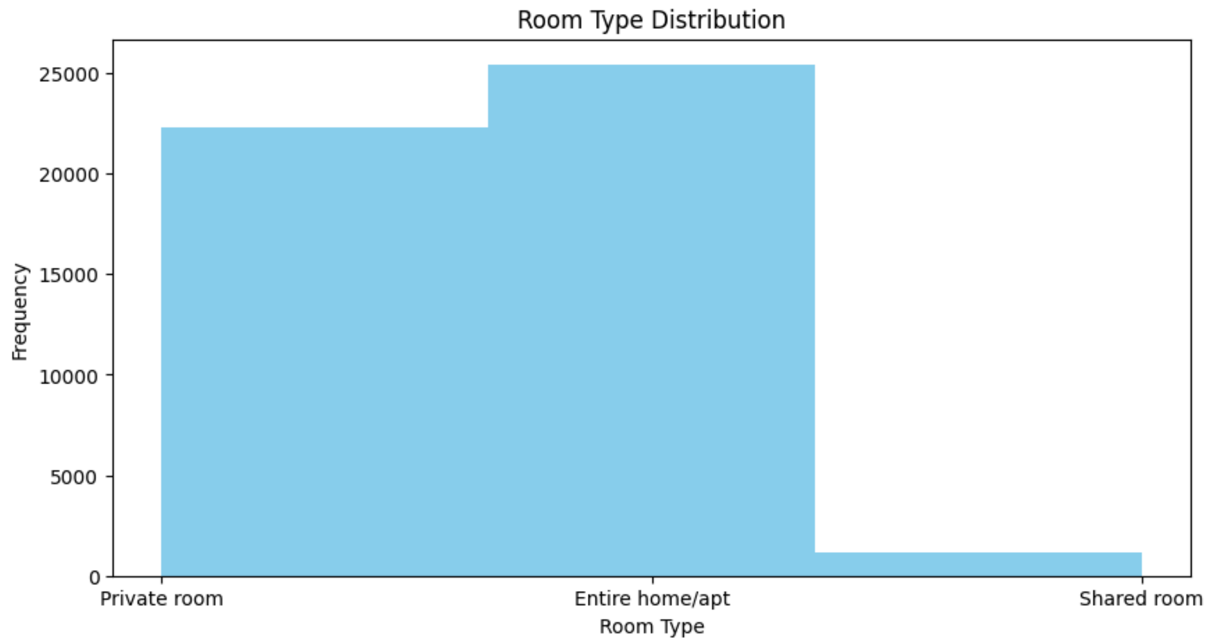
In [ ]:
```python
# Plot Histogram of the minimum_nights

# Make the figure
fig, ax = plt.subplots(figsize = (10,5))

# set labels
ax.set_xlabel("Minimum Nights")
ax.set_ylabel("Frequency")
ax.set_title("Minimum Nights Distribution")

# change x axis to only show values between 0 and 3000
plt.xlim(0, 400)
```

```
plt.hist(air_df['minimum_nights'], bins = 300, color = 'skyblue')
plt.show()
```



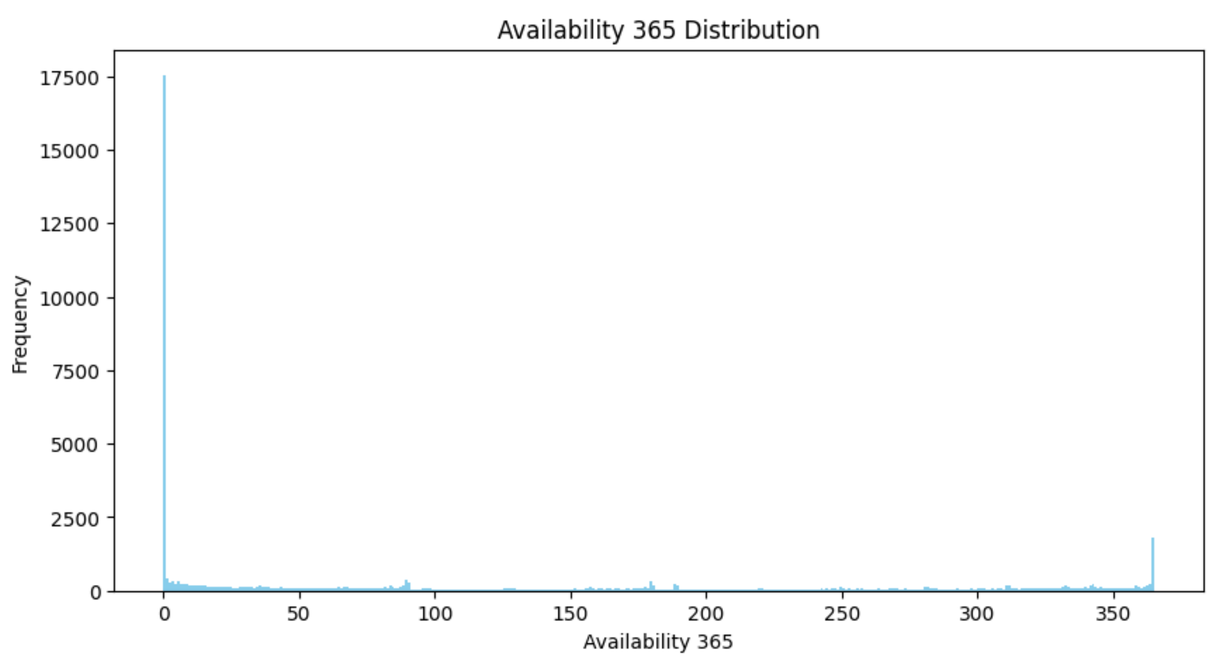
In [ ]:

```
# Plot Histogram of the room_type

# Make the figure
fig, ax = plt.subplots(figsize = (10,5))

# set labels
ax.set_xlabel("Room Type")
ax.set_ylabel("Frequency")
ax.set_title("Room Type Distribution")

# change x axis to only show values between 0 and 3000
#plt.xlim(0, 400)

plt.hist(air_df['room_type'], bins = 3, color = 'skyblue')
plt.show()
```



In [ ]:

```
# Plot Histogram of the availability_365

# Make the figure
fig, ax = plt.subplots(figsize = (10,5))

# set labels
ax.set_xlabel("Availability 365")
ax.set_ylabel("Frequency")
ax.set_title("Availability 365 Distribution")

# change x axis to only show values between 0 and 3000
#plt.xlim(0, 400)

plt.hist(air_df['availability_365'], bins = 365, color = 'skyblue')
plt.show()
```



Now, lets see if we have any null values in our dataset and how many we have

In [ ]:
```
import pandas as pd

# Assuming your DataFrame is named 'airbnb_data'
missing_values = air_df.isnull().sum()

# Calculate the percentage of missing values for each column
total_rows = len(air_df)
missing_percentage = (missing_values / total_rows) * 100

print("Missing values by column:")
print(missing_values)
print("\nPercentage of missing values by column:")
print(missing_percentage)
```

```
Missing values by column:
id                                    0
name                                 16
host_id                               0
host_name                            21
neighbourhood_group                   0
neighbourhood                         0
latitude                              0
longitude                             0
room_type                             0
price                                 0
minimum_nights                        0
number_of_reviews                     0
last_review                       10052
reviews_per_month                 10052
calculated_host_listings_count        0
availability_365                      0
dtype: int64

Percentage of missing values by column:
id                                 0.000000
name                               0.032723
host_id                            0.000000
host_name                          0.042949
neighbourhood_group                0.000000
neighbourhood                      0.000000
latitude                           0.000000
longitude                          0.000000
room_type                          0.000000
price                              0.000000
minimum_nights                     0.000000
number_of_reviews                  0.000000
last_review                       20.558339
reviews_per_month                 20.558339
calculated_host_listings_count     0.000000
availability_365                   0.000000
dtype: float64
```

Lets calculate the percentage of missing data of our ENTIRE data set

In [ ]:
```
total_entries = air_df.size
missing_values_total = air_df.isnull().sum().sum()

# Calculate the percentage of missing values in the entire dataset
missing_percentage_total = (missing_values_total / total_entries) * 100

print("Total missing values:", missing_values_total)
print("Total entries:", total_entries)
print("Percentage of missing values in the entire dataset:", missing_percent
```

```
Total missing values: 20141
Total entries: 782320
Percentage of missing values in the entire dataset: 2.574521934758155
```

Now, from our histograms alone we can see that we do have outliers. Lets check which features have outliers based on IQR

In [ ]:
```
# Specify the features you want to check for outliers
features_to_check = ['price','minimum_nights','number_of_reviews']

for feature in features_to_check:
    # Calculate the first quartile (Q1) and third quartile (Q3)
    Q1 = air_df[feature].quantile(0.25)
    Q3 = air_df[feature].quantile(0.75)

    # Calculate the interquartile range (IQR)
    IQR = Q3 - Q1

    # Determine the outlier step (1.5 times the IQR)
    outlier_step = 1.5 * IQR

    # Identify outliers
    outliers = air_df[(air_df[feature] < (Q1 - outlier_step)) | (air_df[feat

    print("Outliers identified in", feature, ":", outliers)
```

```
Outliers identified in price :              id
name    host_id  \
61        15396                Sunny & Spacious Chelsea Apartment      6027
8
85        19601               perfect for a family or small group      7430
3
103       23686  2000 SF 3br 2bath West Village private  townhouse      9379
0
114       26933  2 BR / 2 Bath Duplex Apt with patio! East Village      7206
2
121       27659                      3 Story Town House in Park Slope     11958
8
...         ...                                               ...
...
48758  36420289    Rustic Garden House Apt, 2 stops from Manhattan   7321139
3
48833  36450896   Brand New 3-Bed Apt in the Best Location of FiDi   2974181
3
48839  36452721  Massage Spa. Stay overnight. Authors Artist dr...  27407996
4
48842  36453160  LUXURY MANHATTAN PENTHOUSE+HUDSON RIVER+EMPIRE...  22417137
1
48856  36457700  Large 3 bed, 2 bath , garden , bbq , all you need   6699339
5

                     host_name neighbourhood_group       neighbourhood  \
61                       Petra           Manhattan             Chelsea
85                      Maggie            Brooklyn    Brooklyn Heights
103                        Ann           Manhattan        West Village
114                      Bruce           Manhattan        East Village
121                       Vero            Brooklyn         South Slope
...                        ...                 ...                 ...
48758                LaGabrell              Queens    Long Island City
48833                      Yue           Manhattan  Financial District
48839                  Richard            Brooklyn      Sheepshead Bay
48842  LuxuryApartmentsByAmber           Manhattan             Chelsea
48856                   Thomas            Brooklyn  Bedford-Stuyvesant

       latitude  longitude        room_type  price  minimum_nights  \
61     40.74623  -73.99530  Entire home/apt    375             180
85     40.69723  -73.99268  Entire home/apt    800               1
103    40.73096  -74.00319  Entire home/apt    500               4
114    40.72540  -73.98157  Entire home/apt    350               2
121    40.66499  -73.97925  Entire home/apt    400               2
...         ...        ...              ...    ...             ...
48758  40.75508  -73.93258  Entire home/apt    350               2
48833  40.70605  -74.01042  Entire home/apt    475               2
48839  40.59866  -73.95661     Private room    800               1
48842  40.75204  -74.00292  Entire home/apt    350               1
48856  40.68886  -73.92879  Entire home/apt    345               4

       number_of_reviews last_review  reviews_per_month  \
61                     5  2018-11-03               0.12
85                    25  2016-08-04               0.24
103                   46  2019-05-18               0.55
114                    7  2017-08-09               0.06
```

```
121                  16  2018-12-30               0.24
...                 ...         ...                ...
48758                 0         NaN                NaN
48833                 0         NaN                NaN
48839                 0         NaN                NaN
48842                 0         NaN                NaN
48856                 0         NaN                NaN

       calculated_host_listings_count  availability_365
61                                  1               180
85                                  1                 7
103                                 2               243
114                                 4               298
121                                 2               216
...                               ...               ...
48758                               1               364
48833                               1                64
48839                               1                23
48842                               1                 9
48856                               3               354

[2972 rows x 16 columns]
Outliers identified in minimum_nights :              id
name    host_id  \
6          5121                               BlissArtsSpace!       735
6
14         6090                   West Village Nest - Superhost      1197
5
29         9657                Modern 1 BR / NYC / EAST VILLAGE      2190
4
36        11452                    Clean and Quiet in Brooklyn       735
5
45        12627  Entire apartment in central Brooklyn neighborh...      4967
0
...         ...                                               ...
...
48810  36445121               UWS Spacious Master Bedroom Sublet  27401445
3
48843  36453642     ☆  HUGE, SUNLIT Room - 3 min walk from Train !   5396611
5
48871  36475746      A LARGE ROOM - 1 MONTH MINIMUM - WASHER&DRYER  14400870
1
48879  36480292  Gorgeous 1.5 Bdr with a private yard- Williams...     54033
5
48882  36482231                        Bushwick _ Myrtle-Wyckoff   6605889
6

       host_name neighbourhood_group              neighbourhood  latitude  \
6          Garon            Brooklyn         Bedford-Stuyvesant  40.68688
14         Alina           Manhattan               West Village  40.73530
29          Dana           Manhattan               East Village  40.72920
36            Vt            Brooklyn         Bedford-Stuyvesant  40.68876
45          Rana            Brooklyn  Prospect-Lefferts Gardens  40.65944
...          ...                 ...                        ...       ...
48810    Dagmara           Manhattan             Upper West Side  40.79952
48843       Nora            Brooklyn         Bedford-Stuyvesant  40.69635
```

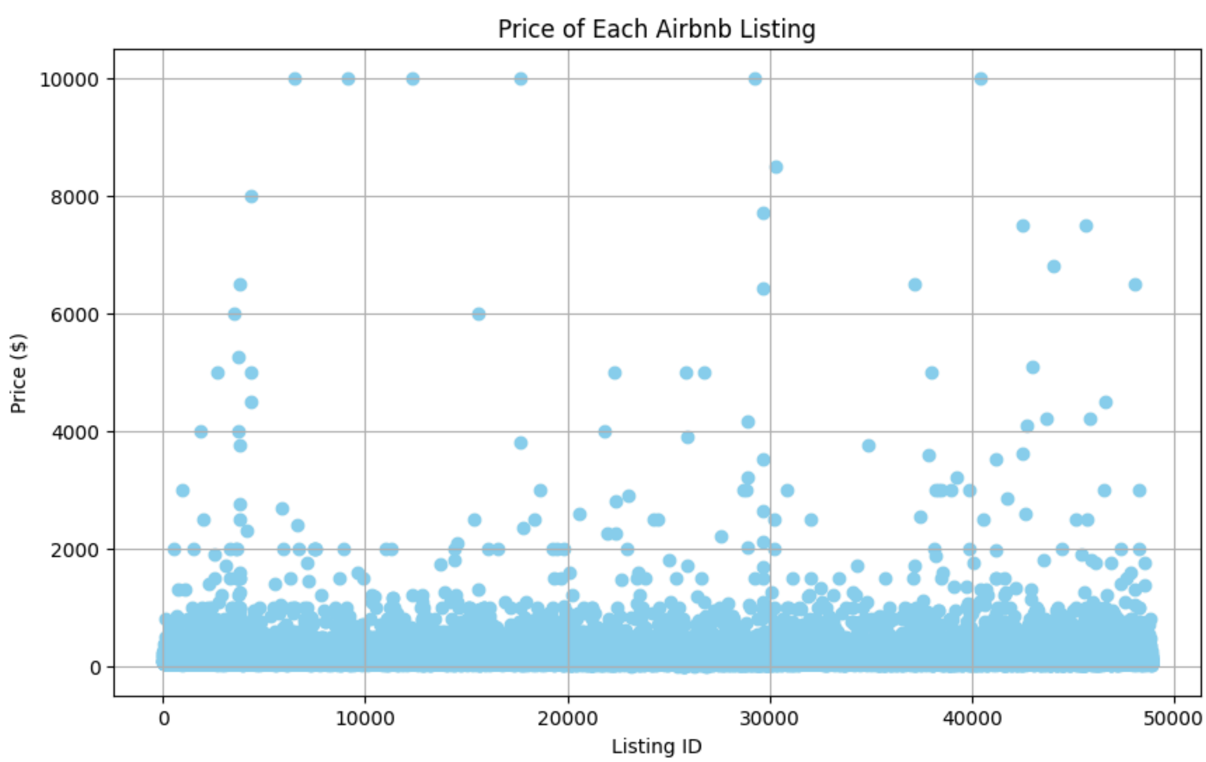
```
48871  Ozzy Ciao          Manhattan                    Harlem  40.82233
48879        Lee           Brooklyn              Williamsburg  40.71728
48882      Luisa           Brooklyn                  Bushwick  40.69652

       longitude        room_type  price  minimum_nights  number_of_reviews
\
6      -73.95596     Private room     60              45                 49
14     -74.00525  Entire home/apt    120              90                 27
29     -73.98542  Entire home/apt    180              14                 29
36     -73.94312     Private room     35              60                  0
45     -73.96238  Entire home/apt    150              29                 11
...          ...              ...    ...             ...                ...
48810  -73.96003     Private room     75              30                  0
48843  -73.93743     Private room     45              29                  0
48871  -73.94687     Private room     35              29                  0
48879  -73.94394  Entire home/apt    120              20                  0
48882  -73.91079     Private room     40              20                  0

      last_review  reviews_per_month  calculated_host_listings_count  \
6      2017-10-05               0.40                               1
14     2018-10-31               0.22                               1
29     2019-04-19               0.24                               1
36            NaN                NaN                               1
45     2019-06-05               0.49                               1
...           ...                ...                             ...
48810         NaN                NaN                               1
48843         NaN                NaN                               2
48871         NaN                NaN                               2
48879         NaN                NaN                               1
48882         NaN                NaN                               1

       availability_365
6                     0
14                    0
29                   67
36                  365
45                   95
...                 ...
48810                90
48843               341
48871                31
48879                22
48882                31

[6607 rows x 16 columns]
Outliers identified in number_of_reviews :              id
name    host_id  \
3         3831                    Cozy Entire Floor of Brownstone       486
9
5         5099          Large Cozy 1 BR Apartment In Midtown East       732
2
7         5178                     Large Furnished Room Near B'way       896
7
8         5203                   Cozy Clean Guest Room - Family Apt       749
0
9         5238                   Cute & Cozy Lower East Side 1 bdrm       754
```

```
9
...         ...                                                ...
...
40104  31123611     JFK Airport Great place to stay 6 minutes away  23225188
1
40297  31249784        Studio Apartment 6 minutes from JFK Airport  23225188
1
40424  31336245           Jfk crash pad 1-2persons in SHARED space  23225188
1
42075  32678719  Enjoy great views of the City in our Deluxe Room!  24436158
9
42076  32678720           Great Room in the heart of Times Square!  24436158
9

         host_name neighbourhood_group     neighbourhood  latitude  longitud
e  \
3      LisaRoxanne            Brooklyn      Clinton Hill  40.68514  -73.9597
6
5            Chris           Manhattan       Murray Hill  40.74767  -73.9750
0
7         Shunichi           Manhattan    Hell's Kitchen  40.76489  -73.9849
3
8        MaryEllen           Manhattan   Upper West Side  40.80178  -73.9672
3
9              Ben           Manhattan         Chinatown  40.71344  -73.9903
7
...            ...                 ...               ...       ...
...
40104     Lakshmee              Queens           Jamaica  40.66823  -73.7837
4
40297     Lakshmee              Queens           Jamaica  40.66793  -73.7845
2
40424     Lakshmee              Queens           Jamaica  40.66715  -73.7834
6
42075      Row NYC           Manhattan  Theater District  40.75918  -73.9880
1
42076      Row NYC           Manhattan  Theater District  40.75828  -73.9887
6

             room_type  price  minimum_nights  number_of_reviews last_review
\
3      Entire home/apt     89               1                270  2019-07-05
5      Entire home/apt    200               3                 74  2019-06-22
7         Private room     79               2                430  2019-06-24
8         Private room     79               2                118  2017-07-21
9      Entire home/apt    150               1                160  2019-06-09
...                ...    ...             ...                ...         ...
40104      Shared room     40               1                 65  2019-07-06
40297     Private room     67               1                 95  2019-07-05
40424      Shared room     39               1                 65  2019-07-07
42075     Private room    100               1                156  2019-07-07
42076     Private room    199               1                 82  2019-07-07

       reviews_per_month  calculated_host_listings_count  availability_365
3                   4.64                               1               194
5                   0.59                               1               129
```

```
7                3.47                      1          220
8                0.99                      1            0
9                1.33                      4          188
...               ...                    ...          ...
40104           10.00                      8          346
40297           15.32                      8          145
40424           10.60                      8          320
42075           58.50                      9          299
42076           27.95                      9          299

[6021 rows x 16 columns]
```

Let's try and visualize some of the outliers using a scatterplot of the prices of all the listings

In [ ]:
```
# Plotting the price of each Airbnb listing
plt.figure(figsize=(10, 6))
plt.scatter(range(len(air_df)), air_df['price'], color='skyblue')
plt.title('Price of Each Airbnb Listing')
plt.xlabel('Listing ID')
plt.ylabel('Price ($)')
plt.grid(True)
plt.show()
```



Now, lets look for correlations. We are going to do this by plotting the correlation matrix for all features in our data

In [ ]:
```
columns_except_price = air_df.drop(columns=['price', 'id', 'host_name', 'nam
X = pd.get_dummies(columns_except_price, drop_first=True)

X.info()
```

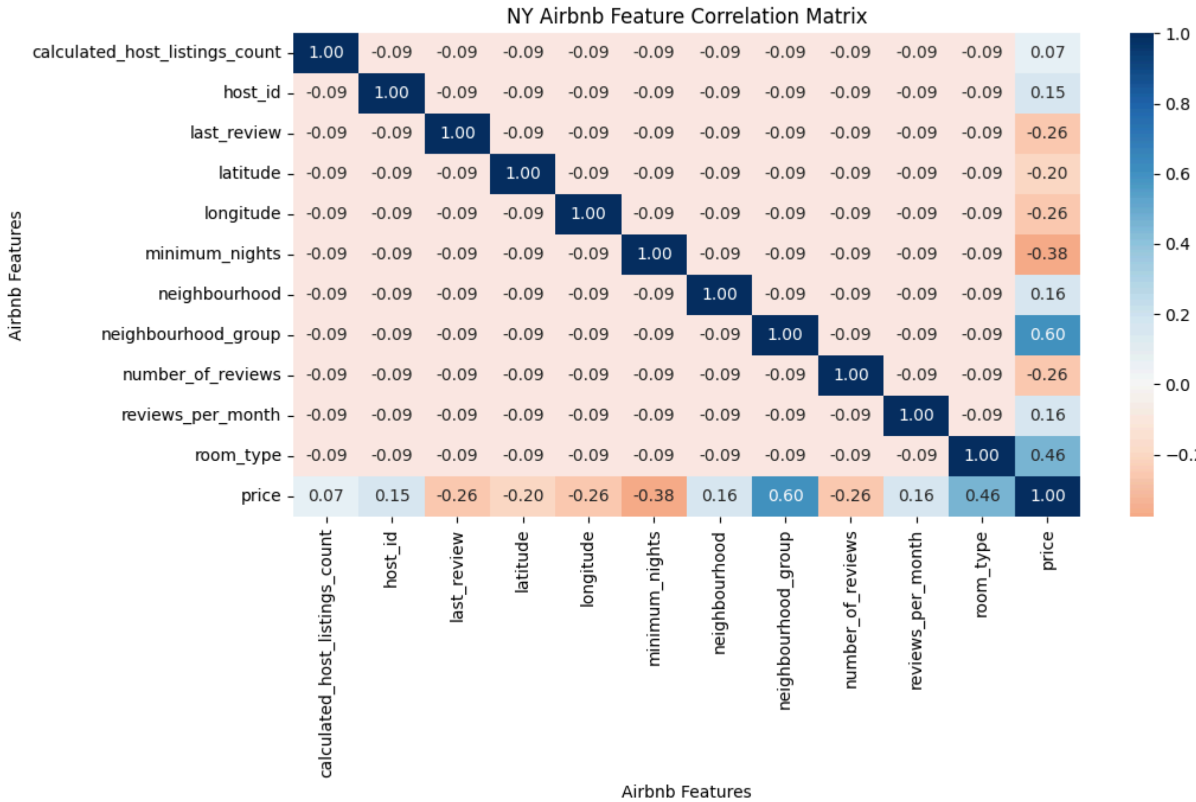
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   calculated_host_listings_count  12 non-null     uint8
 1   host_id                         12 non-null     uint8
 2   last_review                     12 non-null     uint8
 3   latitude                        12 non-null     uint8
 4   longitude                       12 non-null     uint8
 5   minimum_nights                  12 non-null     uint8
 6   neighbourhood                   12 non-null     uint8
 7   neighbourhood_group             12 non-null     uint8
 8   number_of_reviews               12 non-null     uint8
 9   reviews_per_month               12 non-null     uint8
 10  room_type                       12 non-null     uint8
dtypes: uint8(11)
memory usage: 264.0 bytes
```

In [ ]:

```
# Create the target (or outcome) field
y = air_df['price']
```

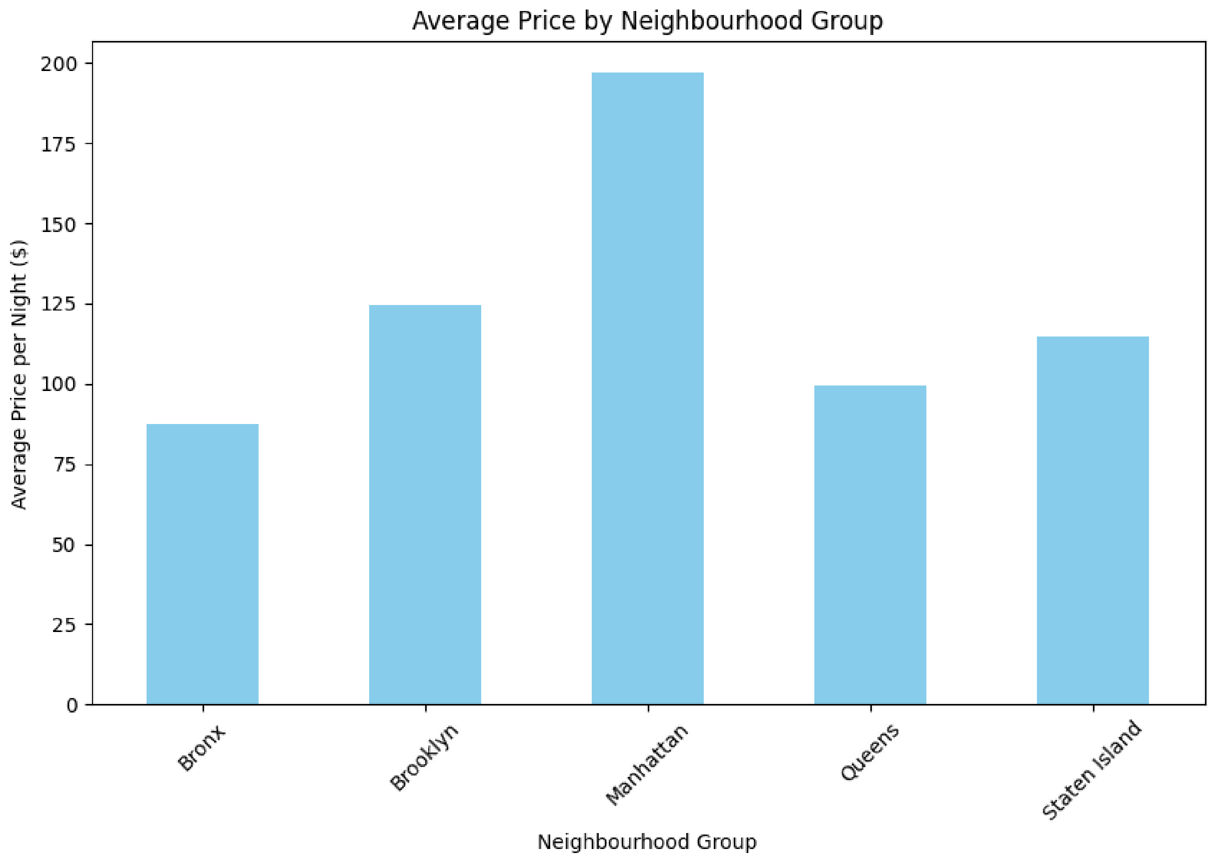
In [ ]:

```
corr = pd.concat([X, y], axis=1).corr()

# Include information about values
fig, ax = plt.subplots(ncols=1,nrows=1,figsize=(11, 7))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="RdBu", center=0, ax=ax);
ax.set_title('NY Airbnb Feature Correlation Matrix')
ax.set_xlabel('Airbnb Features')
ax.set_ylabel('Airbnb Features')

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



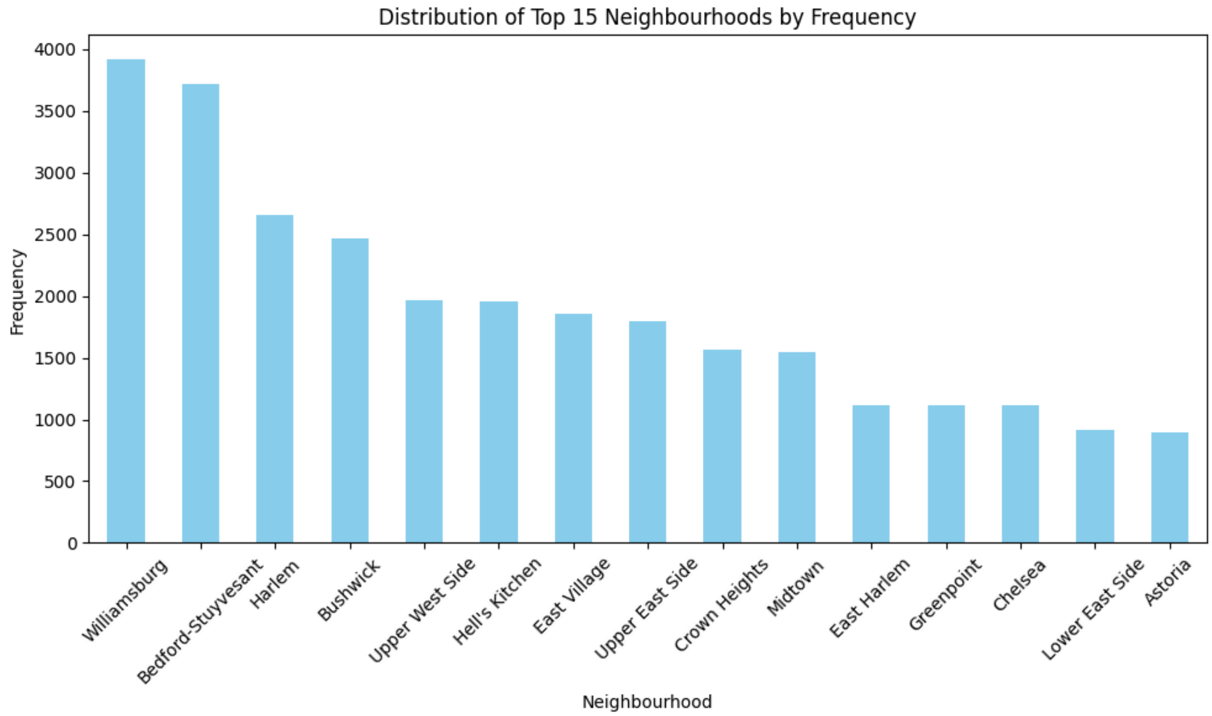
Now, lets examine some of the correlations more closely using histograms, box plots, and scatter plots

In [ ]:
```
# Group data by 'neighbourhood_group' and calculate mean
average_price_by_neighbourhood_group = air_df.groupby('neighbourhood_group')

# Plot the average price by neighbourhood_group
average_price_by_neighbourhood_group.plot(kind='bar', figsize=(10, 6), color

# Set the title and labels
plt.title('Average Price by Neighbourhood Group')
plt.xlabel('Neighbourhood Group')
plt.ylabel('Average Price per Night ($)')

# Rotate x axis labels by 45 degrees
plt.xticks(rotation=45)

# Show plot
plt.show()
```



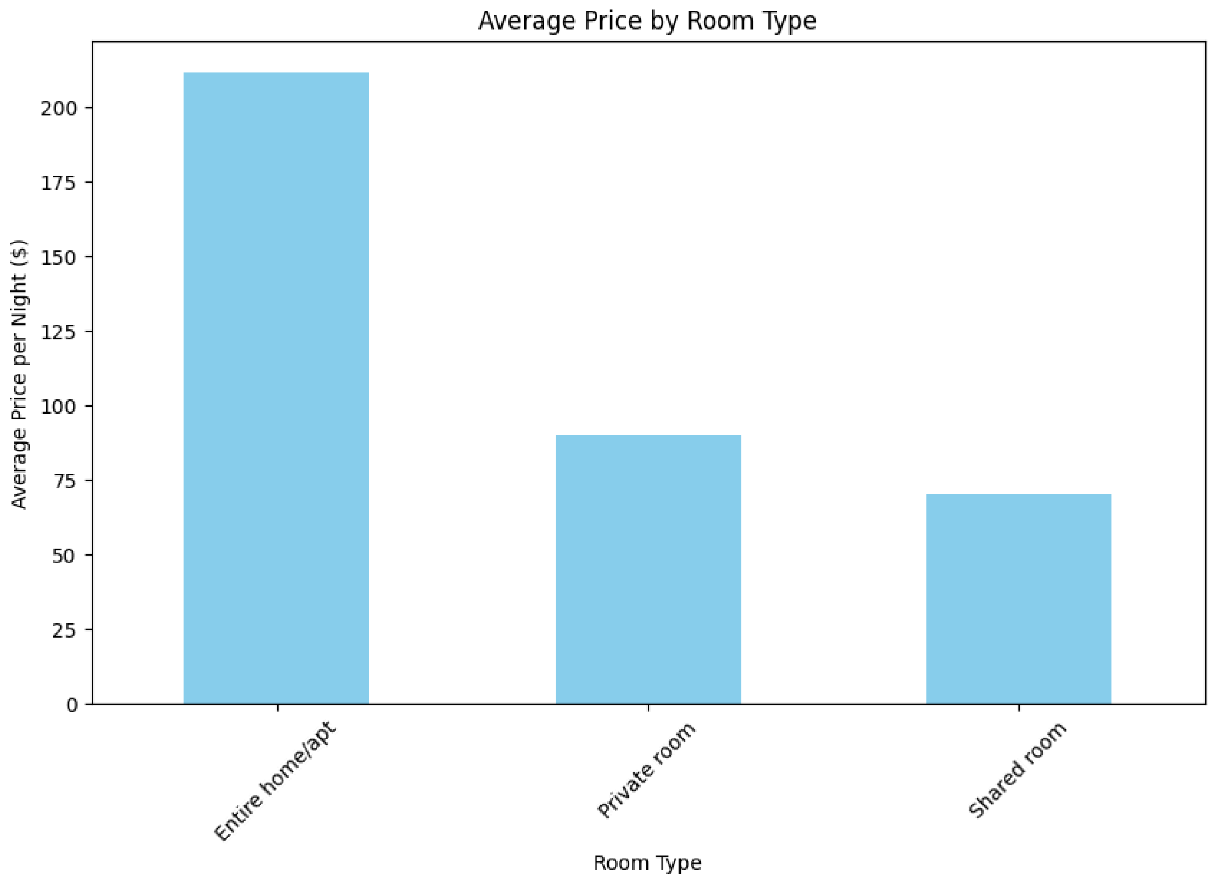
Lets plot the Histogram for the top 15 neighborhoods

In [ ]:
```
# Get the counts of each neighbourhood
neighbourhood_counts = air_df['neighbourhood'].value_counts()

# Select the top 15 most common neighbourhoods
top_neighbourhoods = neighbourhood_counts.head(15)

# Plot the distribution of neighbourhood
plt.figure(figsize=(10, 6))
top_neighbourhoods.plot(kind='bar', color='skyblue')
plt.title('Distribution of Top 15 Neighbourhoods by Frequency')
plt.xlabel('Neighbourhood')
plt.ylabel('Frequency')

# Rotate x axis labels by 45 degrees
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



In [ ]:

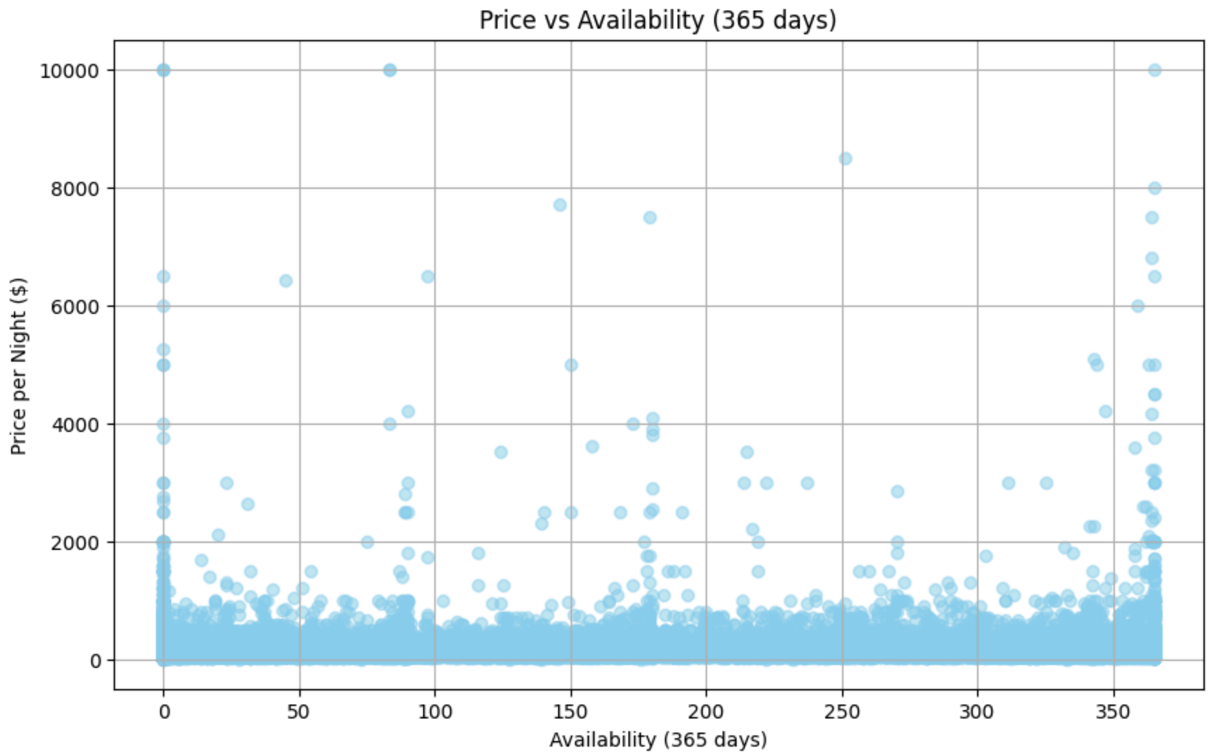
```
# Group data by 'room_type' and calculate mean
average_price_by_room_type = air_df.groupby('room_type')['price'].mean()

# Plot the average price by room_type
average_price_by_room_type.plot(kind='bar', figsize=(10, 6), color='skyblue'

# Set the title and labels
plt.title('Average Price by Room Type')
plt.xlabel('Room Type')
plt.ylabel('Average Price per Night ($)')

# Rotate x axis labels by 45 degrees
plt.xticks(rotation=45)

# Show plot
plt.show()
```



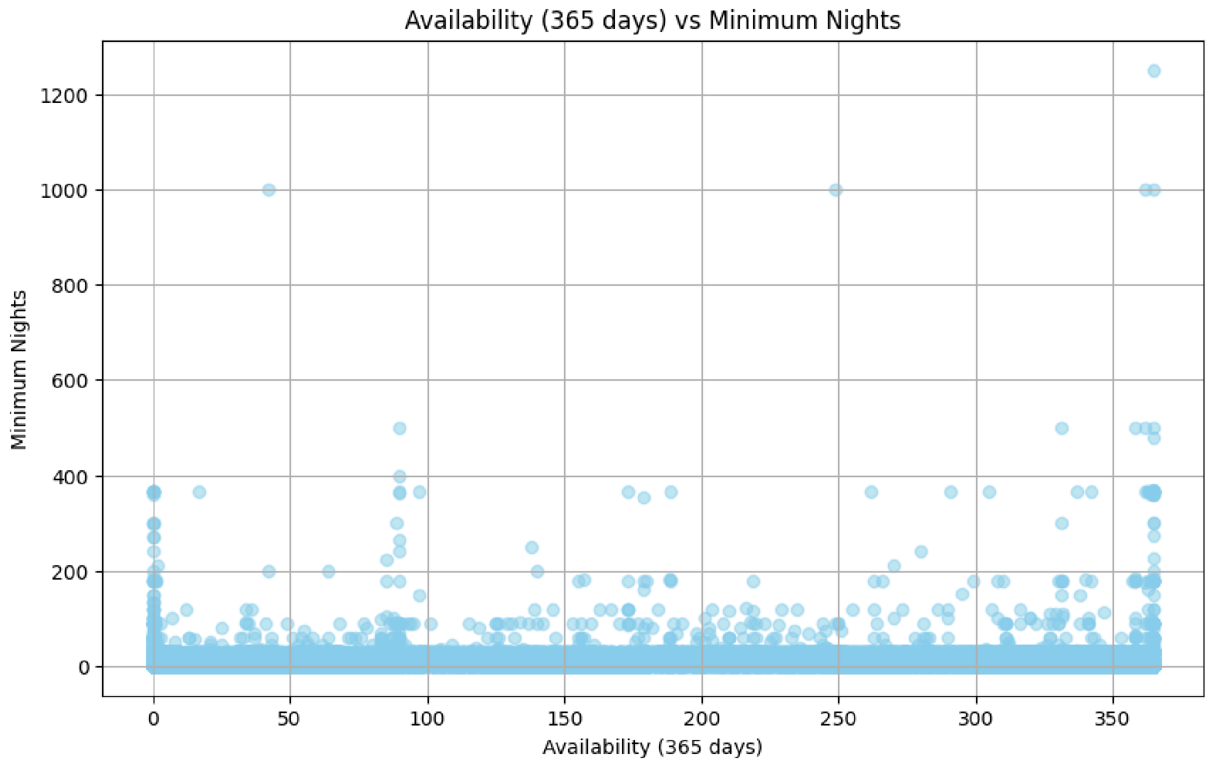
In [ ]:
```
# Get the top 15 most common neighbourhoods
top_neighbourhoods = air_df['neighbourhood'].value_counts().head(15).index.t

# Filter the DataFrame to include only data for the top 15 neighbourhoods
top_neighbourhood_data = air_df[air_df['neighbourhood'].isin(top_neighbourho

# Group the data by 'neighbourhood' and calculate the mean price within each
average_price_by_neighbourhood = top_neighbourhood_data.groupby('neighbourho

# Plot the average price for the top 15 neighbourhoods
plt.figure(figsize=(10, 6))
average_price_by_neighbourhood.plot(kind='bar', color='skyblue')
plt.title('Average Price for Top 15 Neighbourhoods by Price')
plt.xlabel('Neighbourhood')
plt.ylabel('Average Price per Night ($)')

# Rotate x axis labels by 45 degrees
plt.xticks(rotation=45)

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

In [ ]:

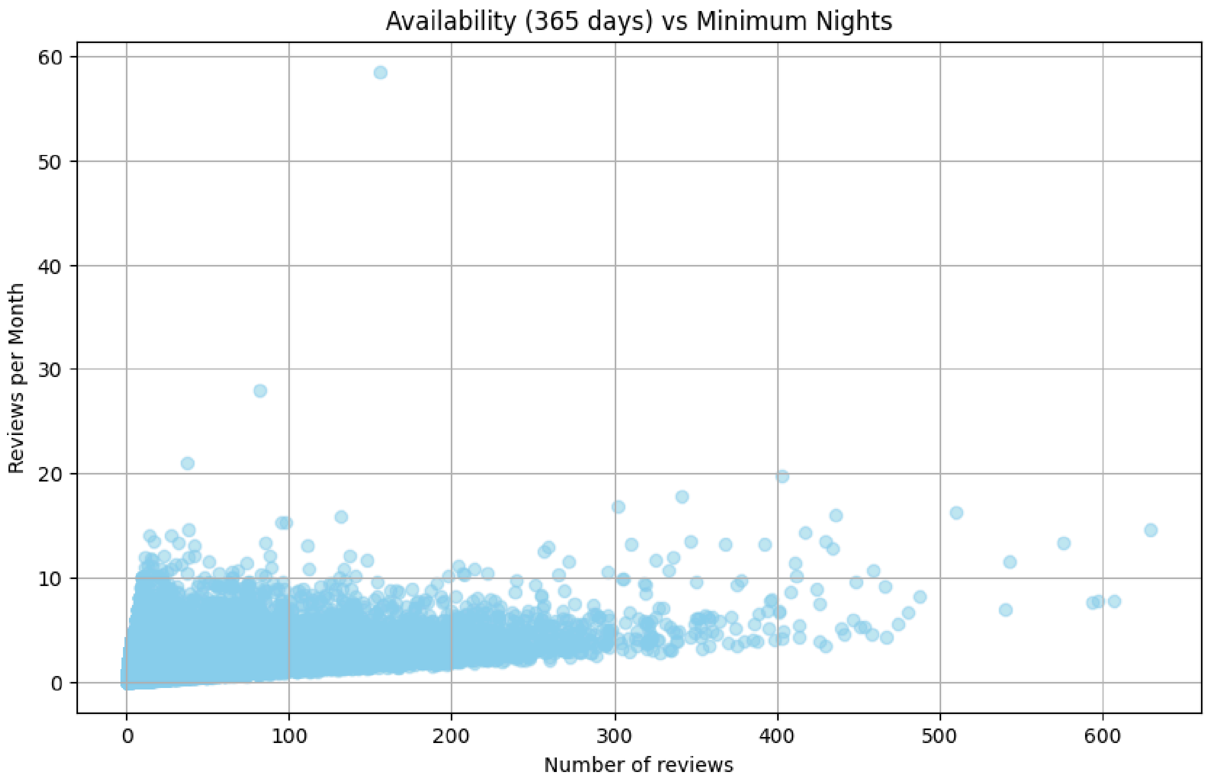
```
# Plot scatter plot of price vs availability_365
plt.figure(figsize=(10, 6))

plt.scatter(air_df['availability_365'], air_df['price'], alpha=0.5, color='s

plt.title('Price vs Availability (365 days)')
plt.xlabel('Availability (365 days)')
plt.ylabel('Price per Night ($)')
plt.grid(True)

plt.show()
```



In [ ]:
```
# Plot scatter plot of availability_365 vs minimum_nights
plt.figure(figsize=(10, 6))

plt.scatter(air_df['availability_365'], air_df['minimum_nights'], alpha=0.5,

plt.title('Availability (365 days) vs Minimum Nights')
plt.xlabel('Availability (365 days)')
plt.ylabel('Minimum Nights')
plt.grid(True)

plt.show()
```



In [ ]:
```
# Plot scatter plot of availability_365 vs minimum_nights
plt.figure(figsize=(10, 6))

plt.scatter(air_df['number_of_reviews'], air_df['reviews_per_month'], alpha=

plt.title('Availability (365 days) vs Minimum Nights')
plt.xlabel('Number of reviews')
plt.ylabel('Reviews per Month')
plt.grid(True)

plt.show()
```



In [ ]:
```
def report(df):
    col = []
    d_type = []
    uniques = []
    n_uniques = []

    for i in df.columns:
        col.append(i)
        d_type.append(df[i].dtypes)
        uniques.append(df[i].unique()[:5])
        n_uniques.append(df[i].nunique())

    return pd.DataFrame({'Column': col, 'd_type': d_type, 'unique_sample': u
```

In [ ]:
```
report(air_df)
```

Out[ ]:

	Column	d_type	unique_sample	n_uniques
0	id	int64	[2539, 2595, 3647, 3831, 5022]	48895
1	name	object	[Clean & quiet apt home by the park, Skylit Mi...	47905
2	host_id	int64	[2787, 2845, 4632, 4869, 7192]	37457
3	host_name	object	[John, Jennifer, Elisabeth, LisaRoxanne, Laura]	11452
4	neighbourhood_group	object	[Brooklyn, Manhattan, Queens, Staten Island, B...	5
5	neighbourhood	object	[Kensington, Midtown, Harlem, Clinton Hill, Ea...	221
6	latitude	float64	[40.64749, 40.75362, 40.80902, 40.68514, 40.79...	19048
7	longitude	float64	[-73.97237, -73.98377, -73.9419, -73.95976, -7...	14718
8	room_type	object	[Private room, Entire home/apt, Shared room]	3
9	price	int64	[149, 225, 150, 89, 80]	674
10	minimum_nights	int64	[1, 3, 10, 45, 2]	109
11	number_of_reviews	int64	[9, 45, 0, 270, 74]	394
12	last_review	object	[2018-10-19, 2019-05-21, nan, 2019-07-05, 2018...	1764
13	reviews_per_month	float64	[0.21, 0.38, nan, 4.64, 0.1]	937
14	calculated_host_listings_count	int64	[6, 2, 1, 4, 3]	47
15	availability_365	int64	[365, 355, 194, 0, 129]	366

Lets change some object types to Categorical as it will make later analysis easier.

More specifically, we are going to change neighbourhood_group and room_type to categorical

In [ ]:

```
air_df['neighbourhood_group'] = air_df['neighbourhood_group'].astype('catego
air_df['room_type'] = air_df['room_type'].astype('category')
```

In [ ]:

```
air_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  category
 5   neighbourhood                   48895 non-null  object
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  category
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: category(2), float64(3), int64(7), object(4)
memory usage: 5.3+ MB
```

We know that we have null values for 1052 entries, lets fill these null values with the mean of the column

In [ ]:

```
air_df.fillna(air_df.mean(), inplace=True)
air_df.info()
air_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  category
 5   neighbourhood                   48895 non-null  object
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  category
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               48895 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: category(2), float64(3), int64(7), object(4)
memory usage: 5.3+ MB
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem

Now, let's develope our first model, a Decision Tree Regression model

In [ ]:

```
# Based on the correlations, set predictors and outcome
predictors = ['neighbourhood_group', 'latitude', 'longitude', 'room_type', '
outcome = 'price'

# Create dummy variables for categorical predictors
X = pd.get_dummies(air_df[predictors], drop_first=True)
y = air_df[outcome]

# select the columns that are objects and typecast them to category
cat_cols = X.select_dtypes(include='object').columns
X[cat_cols] = X[cat_cols].astype('category')
```

Create our training and testing data:

In [ ]:

```
# partition 80% of the data into training and 20% into testing sets
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8, ra
```

Lets fine-tune the hyperparameters of the model using GridSearchCV.

In [ ]:

```
# Define the Decision Tree Regression model
dt_model = DecisionTreeRegressor()

# Define the parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1,3,5,7,10]
}

# Perform GridSearchCV
```

```
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid, cv=5)
grid_search.fit(train_X, train_y)

# Print the best parameters
print("Best parameters found by GridSearchCV:")
print(grid_search.best_params_)

# Make predictions with the best model
best_model = grid_search.best_estimator_
test_pred_y = best_model.predict(test_X)

# Calculate mean absolute error on the test set
mae = mean_absolute_error(test_y, test_pred_y)
print("Mean Squared Error on Test Set:", mae)

# Compare with LassoCV model
lasso_model = LassoCV(cv=5)
lasso_model.fit(train_X, train_y)
lasso_test_pred_y = lasso_model.predict(test_X)

lasso_mae = mean_absolute_error(test_y, lasso_test_pred_y)
print("Mean Squared Error with LassoCV:", lasso_mae)
```

```
Best parameters found by GridSearchCV:
{'max_depth': 7, 'min_samples_leaf': 10, 'min_samples_split': 2}
Mean Squared Error on Test Set: 69.73328037182786
Mean Squared Error with LassoCV: 74.92954822821314
```

Now, lets create the actual model with the best hyperparameters we found in the previous step.

In [ ]:

```
# Initialize the DecisionTreeRegressor model with the best parameters
best_dt_model = DecisionTreeRegressor(criterion="squared_error", max_depth=5

# Fit the model to the training data
best_dt_model.fit(train_X, train_y)
```

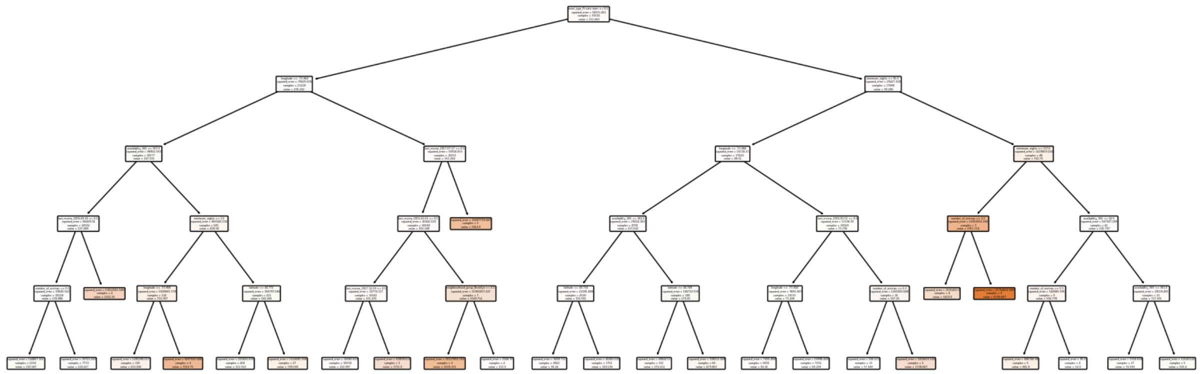
Out[ ]:

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, min_samples_leaf=3)
```

Lets see the tree to verify the results.

In [ ]:

```
plt.figure(figsize=(20, 7))
plot_tree(best_dt_model, feature_names=train_X.columns.to_list(), filled = T
plt.show()
```



Now, let's begin quickly testing the model. First, lets use MAE followed by cross validation with MAPE as the scoring.

In [ ]:
```
# Make predictions on the validation data using the best DT Regressor model
val_pred_y = best_dt_model.predict(test_X)

# Calculate the MAE to quickly evaluate performance
mae = mean_absolute_error(test_y, val_pred_y)

print("Mean Absolute Percentage Error on Validation Data:", mae)
```

```
Mean Absolute Percentage Error on Validation Data: 74.35689405333234
```

In [ ]:
```
# Perform cross-validation to evaluate the model's performance
cv_scores = cross_val_score(best_dt_model, X, y, cv=5, scoring='neg_mean_abs

# Take the negative mean of the cv scores to get MAE
cv_mae = -cv_scores.mean()

print("Cross-Validation Mean Absolute Error:", cv_mae)
```

```
Cross-Validation Mean Absolute Error: 73.88596354290772
```

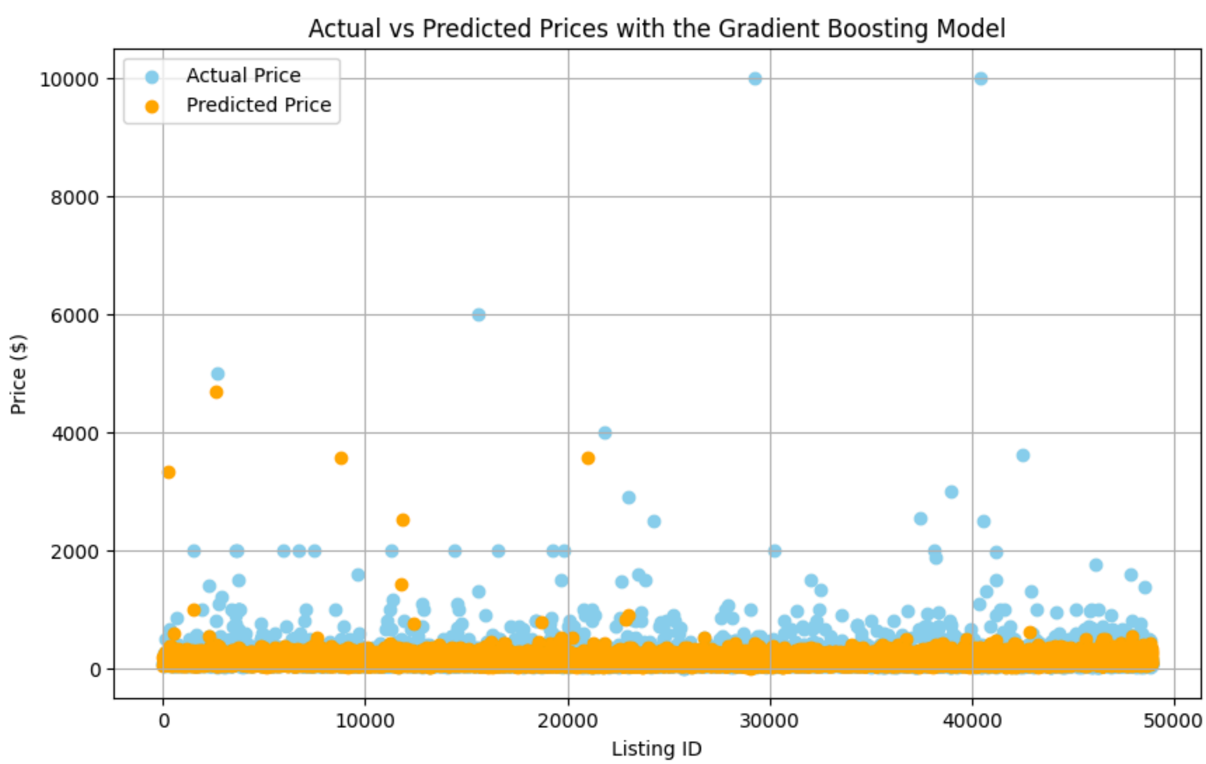
Lets implement a Gradient Boosting Regression model! This builds smaller DT's off of the previous ones and should account for small errors within them, thus increasing performance.

In [ ]:
```
# Create the Gradient Boosting Regression model
gb_model = GradientBoostingRegressor()

# Fit the model to the training data
gb_model.fit(train_X, train_y)

# Make predictions on the test data
y_pred = gb_model.predict(test_X)

# Quickly evaluate the model
mae = mean_absolute_error(test_y, y_pred)
print("Mean Absolute Error (MAE):", mae)
```

```
Mean Absolute Error (MAE): 69.45034711502922
```

Since we can't see all the trees that the model creates, lets try and visualize the performance using a scatter plot of the actual and predicted values using the Gradient Boosting Model.

In [ ]:
```
# Save the predicted values
predicted_values = gb_model.predict(test_X)

# Plotting the actual prices
plt.figure(figsize=(10, 6))
plt.scatter(test_X.index, test_y, color='skyblue', label='Actual Price')

# Plotting the predicted prices
plt.scatter(test_X.index, predicted_values, color='orange', label='Predicted

plt.title('Actual vs Predicted Prices with the Gradient Boosting Model')
plt.xlabel('Listing ID')
plt.ylabel('Price ($)')
plt.legend()
plt.grid(True)
plt.show()
```



Now, lets compare the accuracy to a dummy model, For the sake of simplicity and understanding, lets use the mean dummy model.

In [ ]:
```
# Define the Dummy model using the mean value as the strategy (predict the m
dummy_model = DummyRegressor(strategy='mean')

# Train the Dummy model
dummy_model.fit(train_X, train_y)
```

```
# Make predictions with the Dummy model
dummy_test_pred_y = dummy_model.predict(test_X)

# Calculate mean absolute error for the Dummy model
dummy_mae = mean_absolute_error(test_y, dummy_test_pred_y)
print("Mean Absolute Error with Dummy model (mean strategy):", dummy_mae)
```

```
Mean Absolute Error with Dummy model (mean strategy): 93.34945129158264
```

Now, lets compare accuracy and error metrics for each model (Including LassoCV so we can compare it to the Decision Tree Regression model)

In [ ]:
```
# Define a function to calculate multiple evaluation metrics
def calculate_metrics(true_y, pred_y):
    mae = mean_absolute_error(true_y, pred_y)
    mse = mean_squared_error(true_y, pred_y)
    rmse = np.sqrt(mse)
    r2 = r2_score(true_y, pred_y)
    n = len(true_y)
    p = test_X.shape[1]
    adj_r2 = 1 - (1 - r2) * ((n - 1) / (n - p - 1))  # Compute Adjusted R-sq
    return mae, mse, rmse, r2, adj_r2

# Evaluate Decision Tree model
dt_mae, dt_mse, dt_rmse, dt_r2, dt_adj_r2 = calculate_metrics(test_y, test_p

# Evaluate LassoCV model
lasso_mae, lasso_mse, lasso_rmse, lasso_r2, lasso_adj_r2 = calculate_metrics

# Evaluate Gradient Boosting model
gb_mae, gb_mse, gb_rmse, gb_r2, gb_adj_r2 = calculate_metrics(test_y, y_prec

# Evaluate Dummy model
dummy_mae, dummy_mse, dummy_rmse, dummy_r2, dummy_adj_r2 = calculate_metrics

# Print the evaluation metrics for Decision Tree, Gradient Boosting, and Dum
print("Decision Tree Model:")
print("MAE:", dt_mae)
print("MSE:", dt_mse)
print("RMSE:", dt_rmse)
print("R-squared:", dt_r2)
print("Adjusted R-squared:", dt_adj_r2)
print()

print("LassoCV Model:")
print("MAE:", lasso_mae)
print("MSE:", lasso_mse)
print("RMSE:", lasso_rmse)
print("R-squared:", lasso_r2)
print("Adjusted R-squared:", lasso_adj_r2)
print()

print("Gradient Boosting Model:")
print("MAE:", gb_mae)
print("MSE:", gb_mse)
print("RMSE:", gb_rmse)
```

```python
print("R-squared:", gb_r2)
print("Adjusted R-squared:", gb_adj_r2)
print()

print("Dummy Model:")
print("MAE:", dummy_mae)
print("MSE:", dummy_mse)
print("RMSE:", dummy_rmse)
print("R-squared:", dummy_r2)
print("Adjusted R-squared:", dummy_adj_r2)
print()
```

```
Decision Tree Model:
MAE: 69.73328037182786
MSE: 49996.24858786161
RMSE: 223.59840918007805
R-squared: 0.09160877191822858
Adjusted R-squared: -0.10972631536526234

LassoCV Model:
MAE: 74.92954822821314
MSE: 49722.400746358224
RMSE: 222.98520297624734
R-squared: 0.09658436476923304
Adjusted R-squared: -0.10364793619270851

Gradient Boosting Model:
MAE: 69.45034711502922
MSE: 53073.18201817909
RMSE: 230.376175022894
R-squared: 0.03570339068587447
Adjusted R-squared: -0.1780225194744527

Dummy Model:
MAE: 93.34945129158264
MSE: 55038.31691017777
RMSE: 234.60246569500876
R-squared: -1.5141481737313e-06
Adjusted R-squared: -0.2216410301525289
```

Gradient Boosting performed the best with the basic DT Regressor not far behind. LassoCV did a pretty good job for the ease of implementation. The Dummy Model performed drastically worse than any of the other models as expected.