airbnb

Navigating the NYC AirBnb Landscape

Insights for Hosts and Renters

Data Science Bootcamp - Spring 2024

Satyajit Sriram | Saurabh Parrikar | Bradley Guirand | Mofei Wang

Introduction

Table of Contents

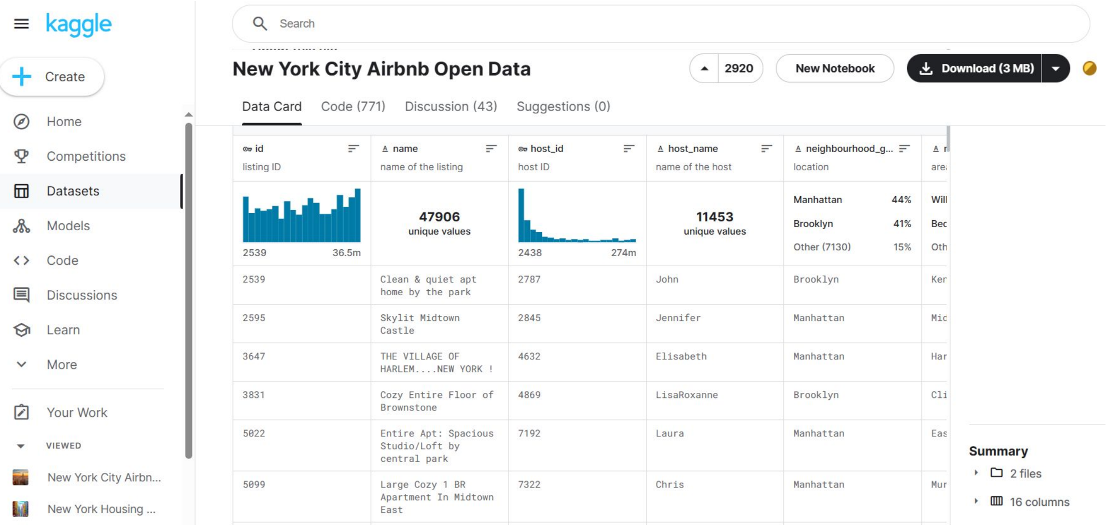
The Dataset

https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data/data

- Since 2008, guests and hosts have used Airbnb to expand on traveling possibilities and present more unique, personalized way of experiencing the world. This dataset describes the listing activity and metrics in NYC, NY for **2019**.

Fields to Analyze:

- List_id
- Listing_name
- Host_id
- Host_name
- Neighborhood
- Borough
- Room_type
- Lat-Long
- Price
- Reviews



Problem Statement

Predicting Rental Prices:

- Develop a **model to predict Airbnb rental prices in NYC** based on features such as neighborhood, room type, minimum nights, and number of reviews.

Overview

- In this report/presentation, we will delve into the world of Airbnb in NYC, **focusing on the development of a model to predict rental prices** based on these key features.
- By analyzing the NYC Airbnb dataset and leveraging machine learning techniques, we **aim to provide valuable insights into the factors influencing rental prices** in one of the world's most vibrant cities.

- With its **diverse neighborhoods** and array of listings, **NYC has offered a dynamic Airbnb market** where **rental prices vary widely** based on factors such as neighborhood, room type, minimum nights, and number of reviews.
- **Understanding and predicting these rental prices is essential** for hosts looking to optimize their listings and for travelers seeking the best value for their stay.

Recap from Comments

Please provide any additional comments regarding this group's project design.	Please provide any additional comments about how the group addressed the stated problem in their project.	Please share any additional comments for the presentation and demo here.	Please provide any additional feedback or suggestions for this group regarding their project here.
- Need more detail on how they are going to solve the stated problem - Should reconsider why they are deleting missing values, there could be value there - Histogram is actually a bar chart - Use Log(price), Log(# of reviews) before visualizing and taking average	Good job here	- Shouldn't flip back and forth through the presentation - Introduce yourselves at the beginning then the problem - Add a section for "Next Steps"	N/A
Values above 10k and below 10 were thrown out but no methodological reason was given. Not clear overall what the project wants to accomplish, analytics-wise.	There was no clear problem statement or what the team is trying to work towards.	Should do a better job of keeping audience in mind and practice your flow of working through the slides and speaker parts. Lots of hesitating and flipping between slides.	NA

P1 - Data Preprocessing & Cleaning

P1 - Data Preprocessing & Cleaning

- Data preprocessing & Cleaning is crucial because it ensures that the dataset is **accurate and ready for analysis.**
- This step helps to **remove errors, inconsistencies, and irrelevant information**, leading to more reliable and meaningful insights.

Checking for duplicates

```
In [5]: df.duplicated().sum()
Out[5]: 0
```

No duplicates present in the dataset so we dont need to act on it

```
In [6]: df.isnull().sum()
Out[6]: 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
```

P1 - Data Preprocessing & Cleaning

- After checking for duplicates & Null values, we move on to **remove entries with price = 0** as this is probably an anomaly.

```
In [7]: # We remove entries with price set to 0 as this is probably an anomaly
        #df = df[df['price'] >= 10]
```

This is how df looks for now

```
In [8]: df.head()
```

Out[8]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitu
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.972
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.983
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.941
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.959
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.943

```
In [9]: df.shape
Out[9]: (48895, 16)
```

We tried to look repeating host_id for entries where host_name is present so we could pick the name from another occourance but there was no repeat

P1 - Data Preprocessing & Cleaning

- Remove entries with Null

  host_name and name as these are

  0.042% of the values.

Here we remove entries with null host name- 21 entries

```
In [10]: nan_count = df['host_name'].isna().sum()
         print("Number of entries with NaN host names:", nan_count)
```

```
Number of entries with NaN host names: 21
```

```
In [11]: # Remove rows with NaN values in the 'host_name' column
         df = df.dropna(subset=['host_name'])

         # Check the number of entries after removing NaN values
         print("Number of entries after removing entries with NaN host name values:", len(df))
```

```
Number of entries after removing entries with NaN host name values: 48874
```

Here we remove entries with null name- 16 entries

```
In [12]: nan_count = df['name'].isna().sum()
         print("Number of entries with NaN names:", nan_count)
```

```
Number of entries with NaN names: 16
```

```
In [13]: # Remove rows with NaN values in the 'host_name' column
         df = df.dropna(subset=['name'])

         # Check the number of entries after removing NaN values
         print("Number of entries after removing entries with NaN name values:", len(df))
```

```
Number of entries after removing entries with NaN name values: 48858
```

P1 - Data Preprocessing & Cleaning

- Converting values from ***object*** to ***datetime***.

```
In [17]: df.dtypes
Out[17]: id                                    int64
         name                                 object
         host_id                               int64
         host_name                            object
         neighbourhood_group                  object
         neighbourhood                        object
         latitude                            float64
         longitude                           float64
         room_type                            object
         price                                 int64
         minimum_nights                        int64
         number_of_reviews                     int64
         last_review                  datetime64[ns]
         reviews_per_month                   float64
         calculated_host_listings_count        int64
         availability_365                      int64
         dtype: object
```

Converting last_review to date time

```
In [15]: df['last_review']
Out[15]: 0        2018-10-19
         1        2019-05-21
         2               NaN
         3        2019-07-05
         4        2018-11-19
                    ...
         48890           NaN
         48891           NaN
         48892           NaN
         48893           NaN
         48894           NaN
         Name: last_review, Length: 48858, dtype: object
```

P1 - Data Preprocessing & Cleaning

- We then **convert all Null values** in reviews_per_month **into 0** using *fillna()* as this would actually indicate zero reviews for that property

In [18]:
```
df['reviews_per_month'].fillna(0, inplace=True)
```

In [19]:
```
df.head()
```

Out[19]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_revie
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	1	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	1	
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	3	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	1	
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	10	

P2 - EDA & Data Visualization

P2 - Exploratory Data Analysis & Visualization

Exploratory Data Analysis (EDA) and Visualization are essential as they **help to uncover patterns, trends, and relationships in the data.** This process provides valuable insights into the dataset, aiding in the selection of appropriate modeling techniques and variables for predictive modeling.

```
In [20]: print("Total number of hosts: ", df['host_id'].nunique())

         Total number of hosts:  37425

In [21]: listings_by_host=df.host_id.value_counts()
         print("Top 20 hosts with most listings: ")
         listings_by_host.head(20)

         Top 20 hosts with most listings:

Out[21]: 219517861    327
         107434423    232
         30283594     121
         137358866    103
         16098958      96
         12243051      96
         61391963      91
         22541573      87
         200380610     65
         7503643       52
         1475015       52
         120762452     50
         2856748       49
         205031545     49
         190921808     47
         26377263      43
         2119276       39
         19303369      37
         25237492      34
         119669058     34
         Name: host_id, dtype: int64
```

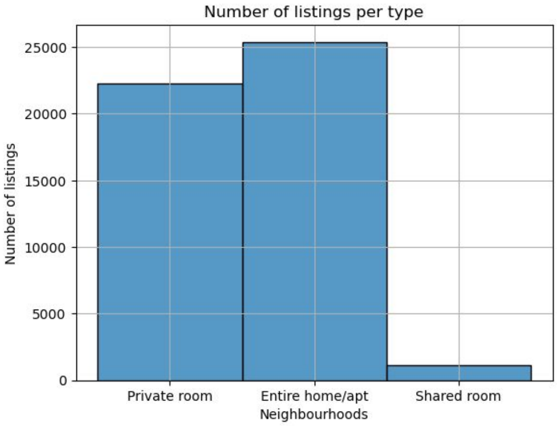
```
In [22]: print("Areas with most listings: ")
         df.neighbourhood.value_counts().head(20)

         Areas with most listings:

Out[22]: Williamsburg          3917
         Bedford-Stuyvesant    3713
         Harlem                2655
         Bushwick              2462
         Upper West Side       1969
         Hell's Kitchen        1954
         East Village          1852
         Upper East Side       1797
         Crown Heights         1563
         Midtown               1545
         East Harlem           1116
         Greenpoint            1113
         Chelsea               1112
         Lower East Side        911
         Astoria                900
         Washington Heights     898
         West Village           768
         Financial District     744
         Flatbush               619
         Clinton Hill           572
         Name: neighbourhood, dtype: int64
```

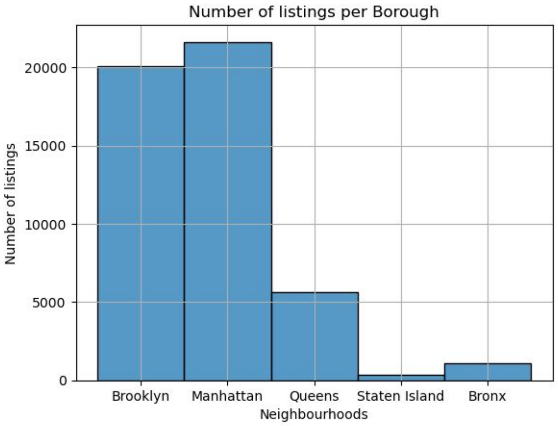
P2 - Exploratory Data Analysis & Visualization

Borough-wise Study

Borough wise study

In [24]:
```python
# Grouping and aggregating data
boroughwise = df.groupby('neighbourhood_group').agg({
    'price': 'mean',
    'neighbourhood': 'nunique',
    'id': 'nunique',
    'availability_365': 'mean',
    'minimum_nights': 'mean',  # Additional: Average minimum nights
    'number_of_reviews': 'mean',  # Additional: Average number of reviews
    'room_type': lambda x: x.value_counts()  # Additional: Frequency of each room type
})

# Renaming columns
boroughwise.rename(columns={'id': 'Number of Listings',
                            'price': 'Price',
                            'neighbourhood': 'Number of Neighbourhoods',
                            'availability_365': 'Days available /365',
                            'minimum_nights': 'Average Minimum Nights',
                            'number_of_reviews': 'Average Number of Reviews',
                            'room_type': 'Room Type Frequencies'
                            }, inplace=True)

# Splitting room type frequencies into separate columns
boroughwise[['Private Room', 'Entire home/apt', 'Shared Room']] = boroughwise['Room Type Frequencies'].apply(pd.Series)

# Dropping the original 'Room Type Frequencies' column
boroughwise.drop(columns=['Room Type Frequencies'], inplace=True)

boroughwise
```





Out[24]:

neighbourhood_group	Price	Number of Neighbourhoods	Number of Listings	Days available /365	Average Minimum Nights	Average Number of Reviews	Private Room	Entire home/apt	Shared Room
Bronx	87.469238	48	1089	165.704316	4.564738	26.018365	652	378	59
Brooklyn	124.410523	47	20089	100.235801	6.057693	24.201006	10123	9553	413
Manhattan	196.897473	32	21643	112.013445	8.538188	20.982581	13190	7973	480
Queens	99.536017	51	5664	144.487288	5.182910	27.701624	3370	2096	198
Staten Island	114.812332	43	373	199.678284	4.831099	30.941019	188	176	9

P2 - Exploratory Data Analysis & Visualization

Room Type-wise Study

Room type wise distribution

In [27]:

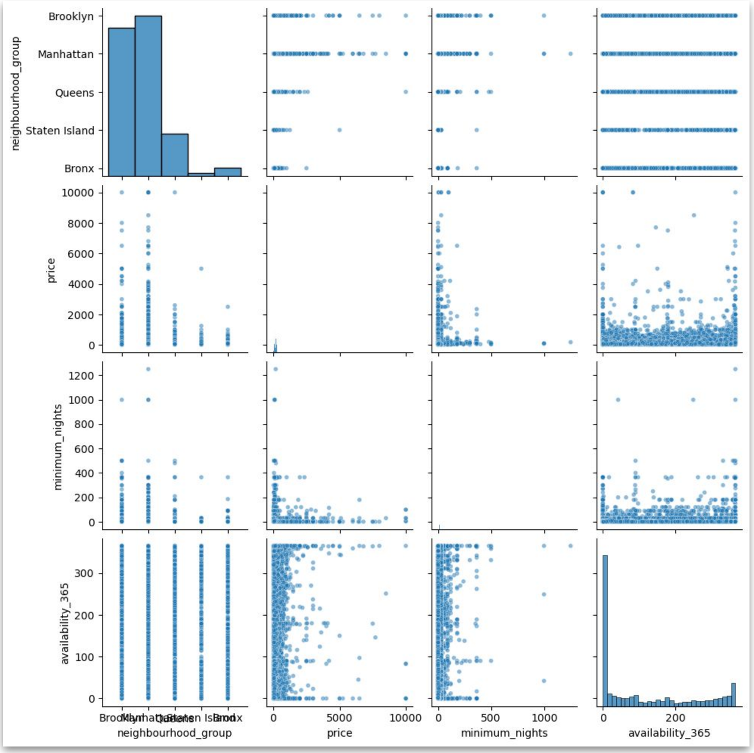
```
import pandas as pd

# Group by room type and calculate averages
room_type_stats = df.groupby('room_type').agg({
    'price': 'mean',
    'availability_365': 'mean',  # Average days available
    # Add other metrics as needed
})

# Rename columns for clarity
room_type_stats.rename(columns={'price': 'Average Price', 'availability_365': 'Average Days Available'}, inplace=True)

# Print the table

room_type_stats
```

Out[27]:

room_type	Average Price	Average Days Available
Entire home/apt	211.806994	111.914110
Private room	89.794360	111.264279
Shared room	70.075928	161.825712

P2 - Exploratory Data Analysis & Visualization

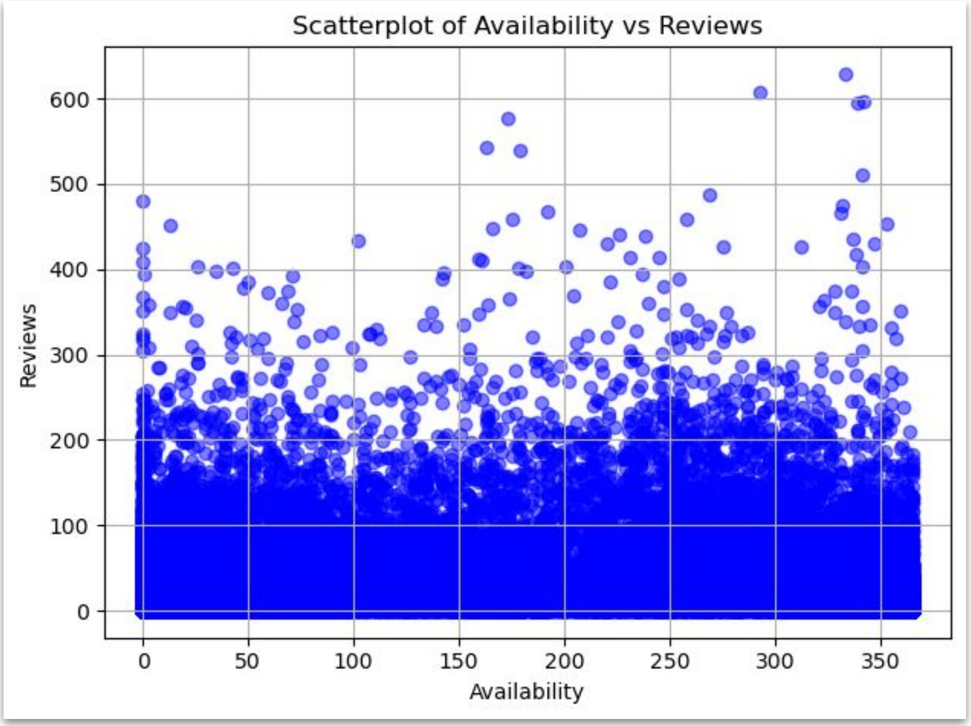
Pairplot



P2 - Exploratory Data Analysis & Visualization

- There is not an obvious relationship between availability and reviews

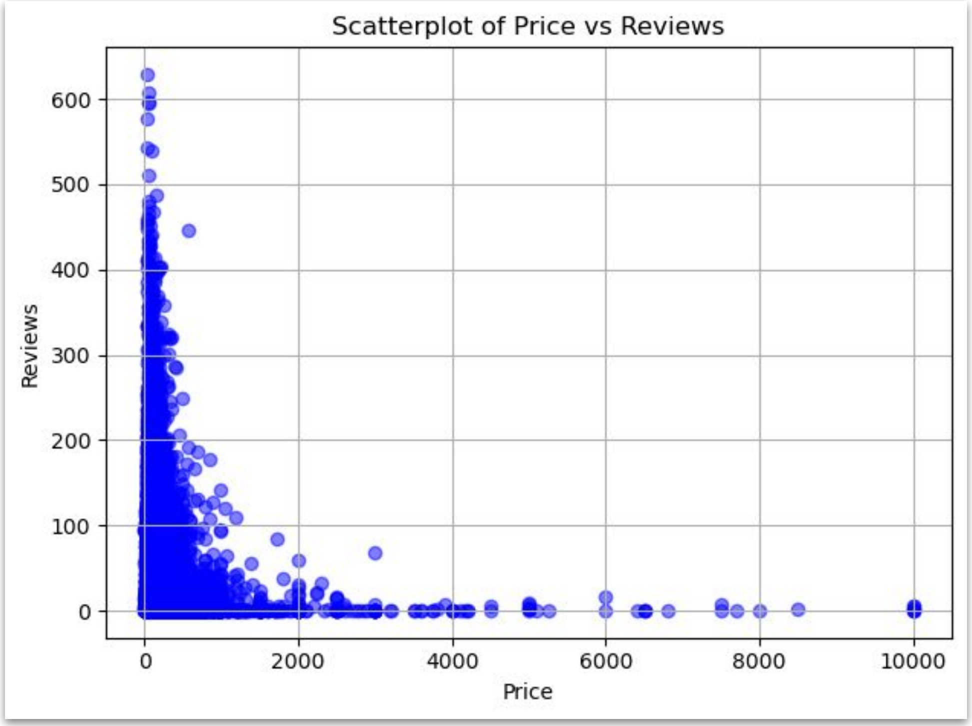
- As availability increases, the number of reviews does not change by a clear pattern



P2 - Exploratory Data Analysis & Visualization

- There is an inverse correlation relationship between the price and number of reviews

- The cheaper properties, also being more common, tend to have a higher amount of reviews

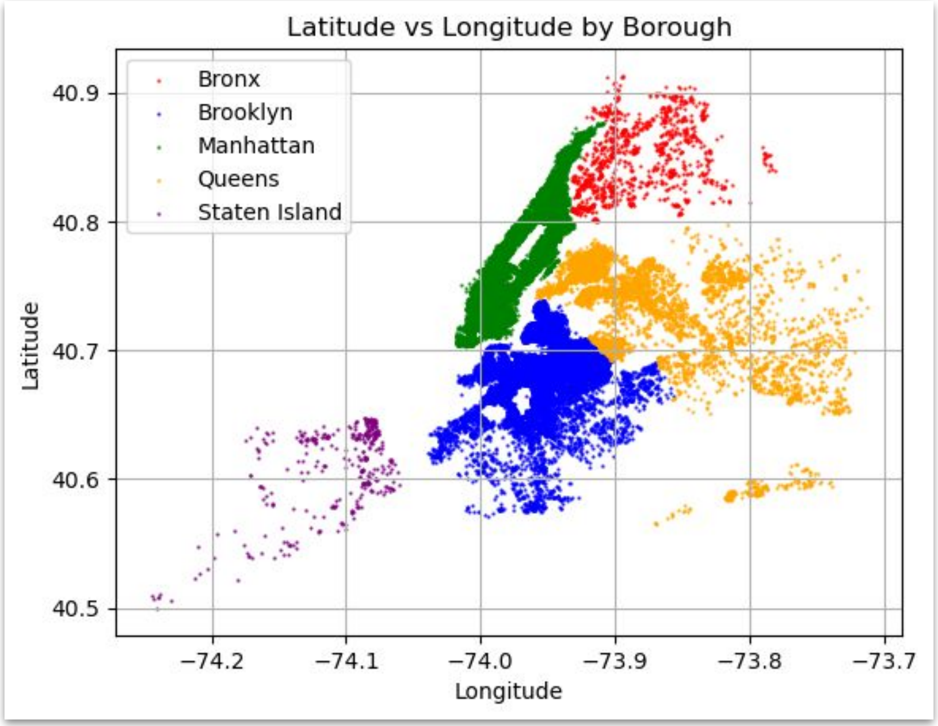


P2 - Exploratory Data Analysis & Visualization

- Boroughs are separated by color and density is shown through the dots

- Manhattan has the highest density, followed by Brooklyn

- Staten Island has the lowest density



P2 - Exploratory Data Analysis & Visualization

- Entire home/apt cost the most, on average ($220)

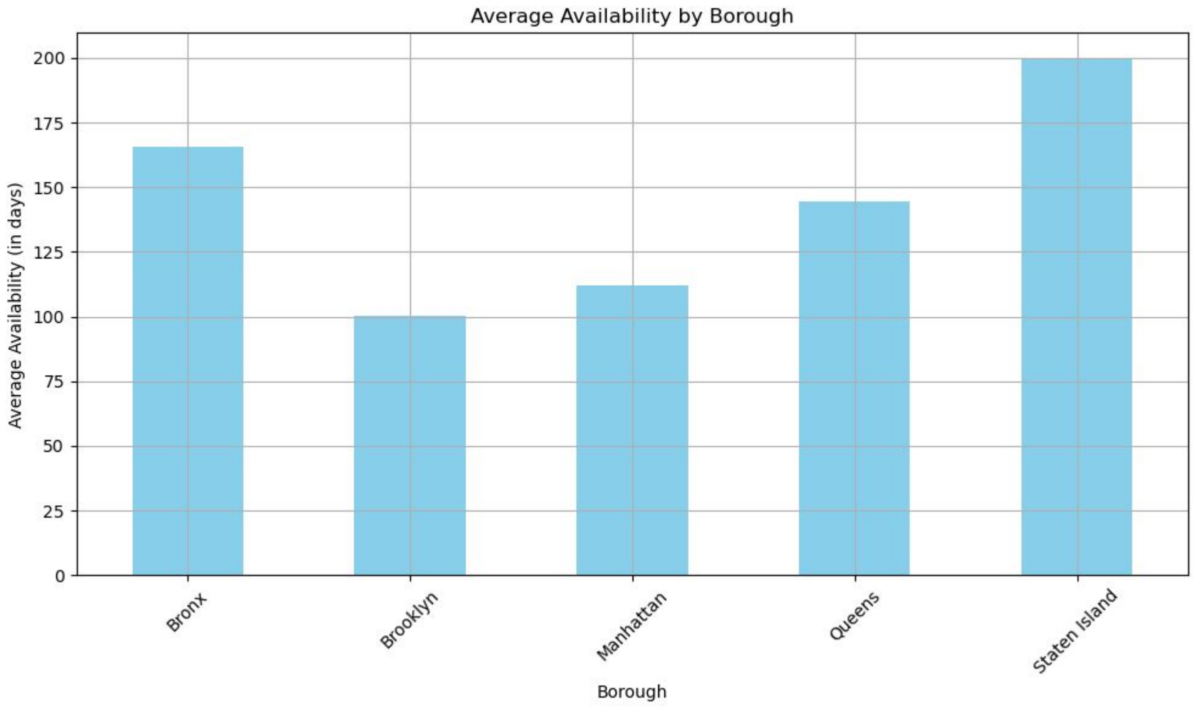
- Private rooms come in second ($90)

- Shared rooms are on average the cheapest room type ($70)

P2 - Exploratory Data Analysis & Visualization

- Availability is highest in Staten Island and the Bronx, on average

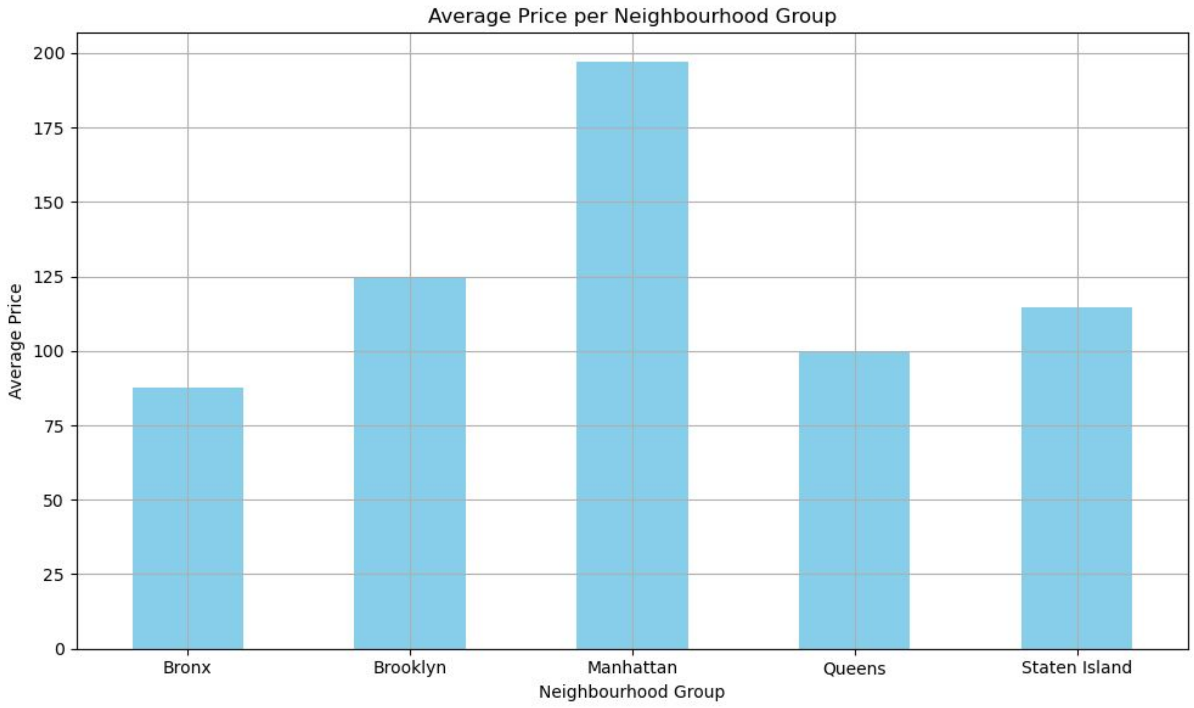
- Lowest in Brooklyn followed by Manhattan, on average



P2 - Exploratory Data Analysis & Visualization

- Manhattan has the highest average price ($195)

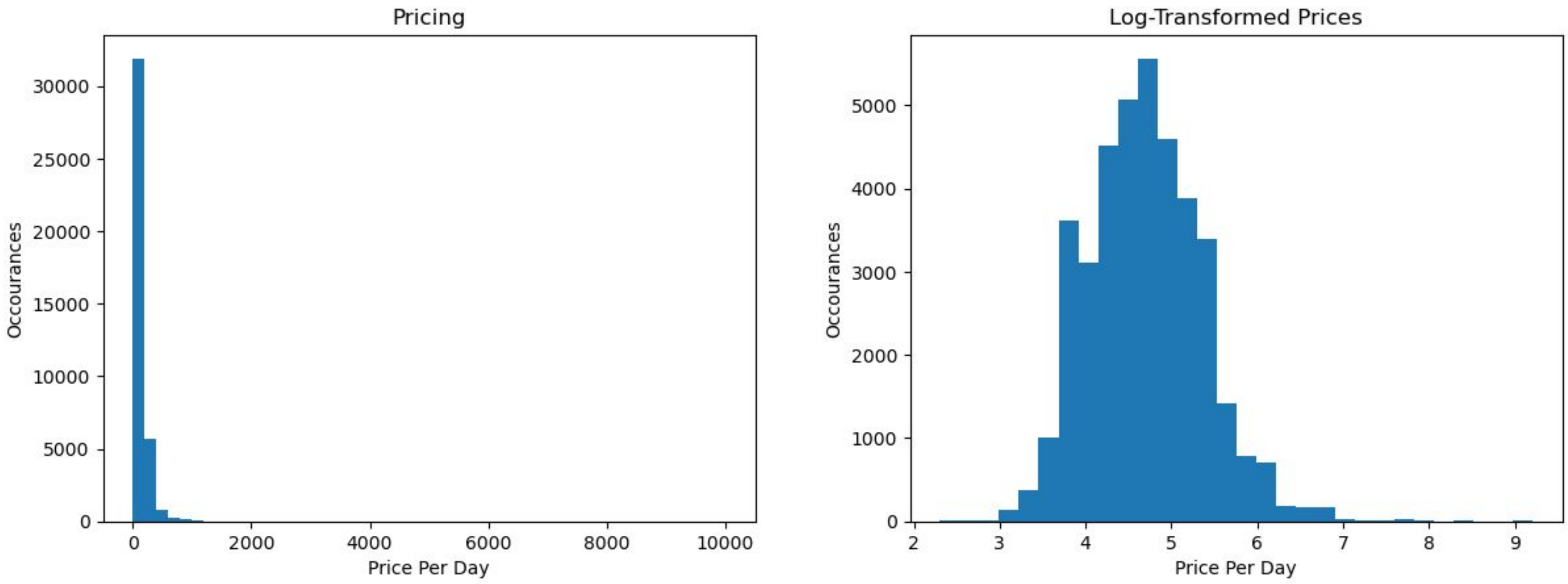
- The Bronx has the lowest average price ($85)



P3 - The Price Prediction Model

P3 - The Model (Price Prediction)

Step 1- Taking log value of price

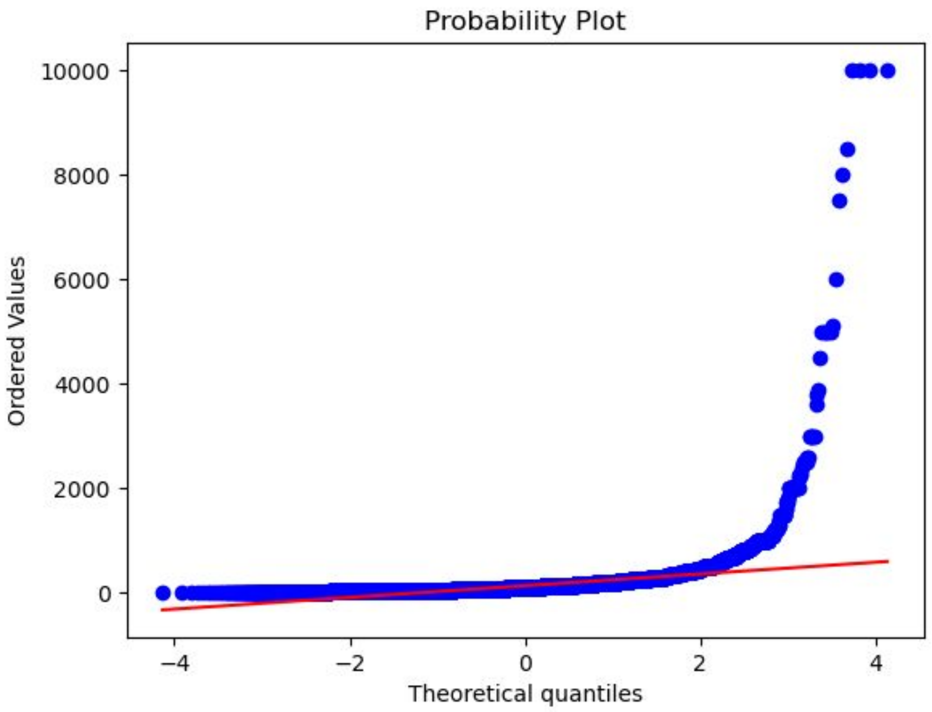


P3 - The Model (Price Prediction)

Step 2- Probability plot analysis

Close Fit at Lower Tail: The closeness of the blue dots to the red line at the lower tail suggests that the lower values closely follow a normal distribution.

Deviation in Upper Tail: As the values increase, the blue dots start to deviate significantly from the red line, particularly at the upper tail (right side). This indicates a heavy right skew, meaning there are more high values than what a normal distribution would predict.



P3 - The Model (Price Prediction)

Step 3- Encoding

1. Encode categorical data
2. Have hot encoded columns
3. Adjust for Target variable (Price)

In [47]:
```
neighbourhood_dummies = pd.get_dummies(df['neighbourhood'])
room_dummies = pd.get_dummies(df['room_type'])

df = pd.concat((df.drop(['neighbourhood', 'room_type'], axis=1),
                neighbourhood_dummies.astype(int),
                room_dummies.astype(int)),
               axis=1)

print('New number of columns after encoding:', len(df.columns))

cols = list(df.columns.values)
idx = cols.index('price')
rearrange_cols = cols[:idx] + cols[idx+1:] + [cols[idx]]
df = df[rearrange_cols]
```

```
New number of columns after encoding: 233
```

P3 - The Model (Price Prediction)

Step 4 - Hyperparameter tuning and Random Forest Results & Outcomes

1. R squared value of close to **99%**
2. Mean Squared Error (MSE) of around **175**

```
Random Forest with Hyperparameter Tuning - R-squared: 0.995747613643827
Random Forest with Hyperparameter Tuning - Mean Squared Error: 176.06267927783327
```

```
In [38]: param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)

rf_random = RandomizedSearchCV(estimator = rf, param_distributions = param_grid,
                               n_iter = 1, cv = 3, verbose=2, random_state=42, n_jobs = -1)

rf_random.fit(X_train, y_train)

best_params = rf_random.best_params_

rf_best = RandomForestRegressor(n_estimators=best_params['n_estimators'],
                                max_depth=best_params['max_depth'],
                                min_samples_split=best_params['min_samples_split'],
                                min_samples_leaf=best_params['min_samples_leaf'],
                                random_state=42)

rf_best.fit(X_train, y_train)

y_pred_rf_best = rf_best.predict(X_test)

r2_rf_best = r2_score(y_test, y_pred_rf_best)
mse_rf_best = mean_squared_error(y_test, y_pred_rf_best)

print("Random Forest with Hyperparameter Tuning - R-squared:", r2_rf_best)
print("Random Forest with Hyperparameter Tuning - Mean Squared Error:", mse_rf_best)
```

Conclusions from the Model

Benefits & Possible scope of improvement in the future

1. Very **accurate** in given data
2. Can help new owners/ people new to Airbnb and **suggest a price for their listing**- not too high or low
3. Data could include features about the listing to **help better predict price over larger geographical areas**
4. We c**ould have more temporal data** to judge the traffic of airbnbs over time.

Moving forward, this predictive model can be a **valuable tool for hosts** to set competitive prices and maximize their earnings, while **also helping travelers** find affordable and suitable accommodations in the vibrant city of NYC.

As the Airbnb market continues to evolve, understanding these pricing dynamics will be essential for both hosts and travelers to make informed decisions.

Thank You