—--

Data Intensive Computing 587 B
Project Phase 1 report

—--

Team Number: 35

Sr no.	**Member Name**	**UB IT Name**	**UB Person No.**
1.	**Sarvesh Bhumkar**	**sarveshb**	**50541064**
2.	**Amrut Kulkarni**	**amrutpra**	**50544971**
3.	**Vinit Wadgaonkar**	**vinitwad**	**50545019**

—--

Problem statement -

1. Discuss the background of the problem leading to your objectives. Why is it a significant problem?

Background of the Problem:

The growth of short-term rental platforms like Airbnb has had a great impact on urban housing markets worldwide, including in New York City but this trend has also raised a few issues that need to be addressed and solved.

1. <u>Affordable Housing Crisis:</u> New York City has long grappled with an affordable housing shortage. The rapid expansion of Airbnb listings can exacerbate this problem as residential units are converted into lucrative short-term rentals, reducing the availability of affordable housing for residents.

2. <u>Housing Market Distortion</u>: Airbnb listings can distort the traditional housing market, potentially driving up rental prices and property values in certain neighborhoods, making it even more difficult for residents to find affordable housing.

3. Neighborhood Character and Quality of Life: The proliferation of short-term rentals can change the character and dynamics of neighborhoods. Frequent turnover of short-term guests may disrupt the sense of community and impact the quality of life for long-term residents.

4. Regulatory and Legal Challenges: New York City has enacted strict regulations on short-term rentals, including restrictions on renting entire apartments for less than 30 days without the primary resident present. However, these regulations are challenging to enforce, and violations are common. Analyzing Airbnb data can help identify compliance issues.

Objectives:

1. Identifying Trends: Understand the trends in Airbnb listings, such as the growth rate of listings, types of properties listed, and the distribution of listings across neighborhoods.

2. Impact on Housing Availability: Assess the extent to which Airbnb listings have affected the availability and affordability of long-term rental housing, especially in high-demand neighborhoods.

3. Compliance with Regulations: Analyze data to identify properties that may be in violation of local short-term rental regulations and assess the effectiveness of enforcement measures.

4. Community Impact: Investigate how the presence of Airbnb listings correlates with changes in neighborhood dynamics, community cohesion, and quality of life indicators.

Why is it a Significant Problem?
Analyzing the New York City Airbnb data is significant for several reasons:

1. Housing Crisis: The shortage of affordable housing is a pressing issue in New York City, and the impact of short-term rentals on housing availability is a matter of concern for policymakers, advocates, and residents.

2. Economic Impact: The short-term rental market contributes to the city's economy, but it also presents challenges in terms of taxation and regulation. Understanding its economic impact is vital for informed policymaking.

3. Urban Planning: The data can inform urban planning decisions by identifying areas with high short-term rental activity, helping the city allocate resources and make zoning decisions effectively.

4. Community Well-being: The quality of life for New York City residents is closely tied to their neighborhoods. Analyzing how Airbnb listings affect neighborhood dynamics and community well-being can shed light on the social aspects of the problem.

In conclusion, analyzing the New York City Airbnb data can provide valuable insights into a complex and multifaceted problem. By addressing issues related to affordable housing, neighborhood dynamics, and regulatory compliance, our analysis can contribute to a more informed and data-driven approach to managing short-term rentals in the city.

1b - Explain the potential of your project to contribute to your problem domain. Discuss why this contribution is crucial?

Analyzing the New York City Airbnb data has significant potential to contribute to addressing the problems in the domain of short-term rentals, housing, and urban planning. Here's how our analysis can contribute to the hospitality and accommodation domain.

1. Boosting Local Economy: New York City is a vibrant metropolis that thrives on tourism. By analyzing the NYC Airbnb data, we can identify trends in accommodation preferences, pricing, and availability. This information can help local businesses, from restaurants to tour guides, tailor their offerings to meet the preferences of Airbnb guests, ultimately boosting their revenue and creating job opportunities.

2. Economic Prosperity for Hosts: The NYC Airbnb dataset offers valuable insights into pricing strategies for hosts and short-term rental property owners. By understanding pricing trends and demand fluctuations in different neighborhoods, hosts can optimize their rates, leading to increased income. This not only benefits individual hosts but also contributes to local job creation in the form of cleaning services, maintenance, and property management.

3. Job Creation in the Hospitality Sector: Airbnb has a significant economic impact on cities like NYC. This includes the creation of jobs in the hospitality industry. Hotel staff, cleaning crews, concierge services, and even Airbnb management companies all benefit from the presence of Airbnb. Analyzing the data can help stakeholders understand and promote the job opportunities this industry offers.

4. Market Insights: Real estate professionals and investors can benefit from our analysis by gaining insights into the dynamics of the short-term rental market. This can inform investment decisions and help in adapting to changing market conditions.

Why this Contribution is Crucial:

1. Balancing Economic Growth and Housing: Short-term rentals have the potential to boost the local economy, but they must coexist with policies that ensure access to affordable housing for residents. Striking this balance is essential for the city's overall well-being.

2. Data-Informed Decision Making: In an era of big data, data-driven decision-making is becoming increasingly important. Our analysis provides the empirical evidence needed to make informed choices about short-term rentals.

3.Enhancing Hosts' Income: The NYC Airbnb dataset offers valuable insights into pricing strategies for hosts and short-term rental property owners. By understanding the pricing trends and demand fluctuations in different neighborhoods, hosts can optimize their rates, resulting in increased income. This financial benefit is not limited to individual hosts; it also contributes to local job creation in the form of cleaning services, maintenance, and property management.

2. Data sources -

We have taken the New York City Airbnb Open Dataset from Kaggle.
Kaggle - https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data/data
The dataset has 48895 rows and 13 columns.

Why rows should be greater than column in the dataset:
Having more rows than columns in a dataset is preferred because it offers advantages like better flexibility for analysis, reduced data redundancy, compatibility with many statistical techniques, and easier data visualization. It's especially useful for handling large amounts of data with various attributes. However, the choice between wide and long formats should depend on our specific analysis goals and data structure.

What can be analyzed from the Airbnb data set -

1. What are the different hosts and areas and what can we learn from that information?
   - Host Demographics: We can analyze host information to understand the demographics of Airbnb hosts, such as age, gender, and location.
   - Property Types: We can determine the types of properties (e.g., entire homes, private rooms, shared rooms) that the hosts can offer, which can give insights into the rental market.
   - Host Behavior: Analyze host behavior, such as response times, acceptance rates, and hosting frequency, to understand how hosts interact with guests.
   - Area Insights: By examining property locations, we can identify trends in different neighborhoods, including popular areas, pricing variations, and property availability.

2. What can we learn from predictions? (ex: locations, prices, reviews, etc)
   - Price Predictions: We can build predictive models to estimate property prices based on various factors like location, property type, amenities, and seasonality.
   - Review Sentiment Analysis: Predict the sentiment of reviews to understand guest satisfaction and potential areas for improvement.
   - Occupancy Rates: Use historical data to predict property occupancy rates throughout the year, helping hosts optimize pricing and availability.

3. Which hosts are the busiest and why?
   - Booking Frequency: Identify hosts with the highest booking frequency and analyze the factors contributing to their popularity, such as competitive pricing, exceptional reviews, or unique property features.
   - Seasonal Variations: Determine if some hosts are busier during specific seasons or events and explore the reasons behind this trend.
   - Marketing and Promotion: Analyze whether hosts with active marketing and promotion strategies attract more guests.

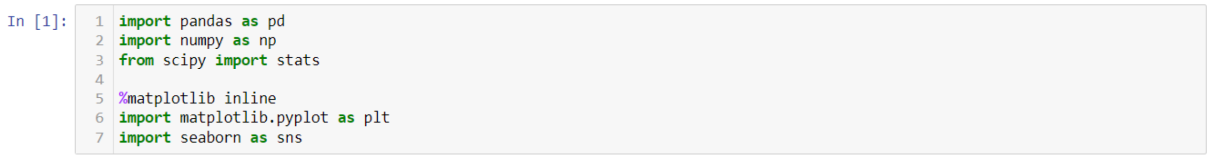
4. Is there any noticeable difference in traffic among different areas, and what could be the reason for it?
   - Demand by Neighborhood: Examine booking data to identify neighborhoods with high and low booking rates. Differences in tourism, attractions, accessibility, and amenities can explain these variations.
   - Pricing Trends: Investigate whether pricing differs significantly between neighborhoods, which could impact booking traffic.
   - Seasonal Factors: Determine if certain areas experience increased traffic during specific seasons or events, such as festivals, conferences, or holidays.
   - Marketing and Reviews: Analyze whether hosts in certain areas are more proactive in marketing their properties or receive better reviews, attracting more guests.

By addressing these questions through data analysis, we can provide valuable insights for hosts, guests, and policymakers alike. Hosts can optimize their listings and operations, guests can make more informed booking decisions, and policymakers can use the information to develop regulations that ensure fair and sustainable short-term rental markets in different areas of New York City.

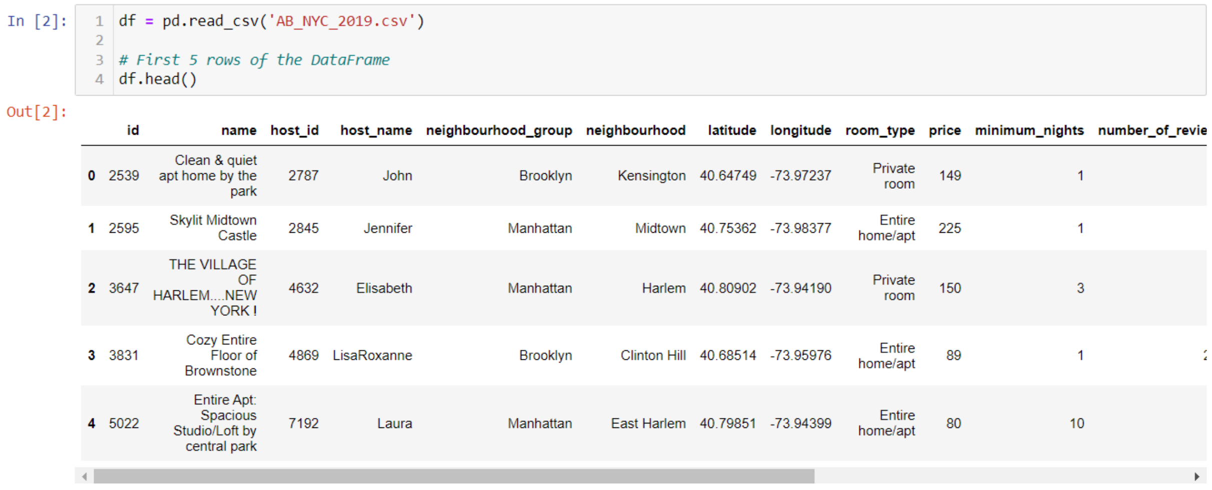
3. Data Cleaning -

Data cleaning is a crucial step in data analysis because the quality of the data directly impacts the accuracy and reliability of the insights and decisions drawn from it. Raw data often contains errors, inconsistencies, missing values, and outliers that can skew the results and lead to erroneous conclusions. Cleaning data involves processes such as identifying and rectifying duplicates, handling missing data, standardizing formats, and removing outliers.

First, we will import all the necessary libraries and read our data csv file.



```
In [1]:
import pandas as pd
import numpy as np
from scipy import stats

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

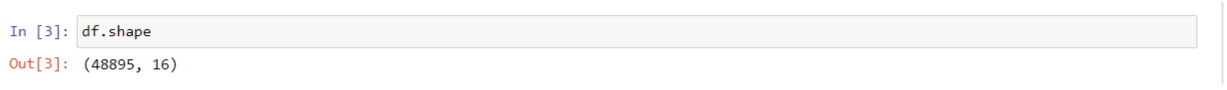


```
In [2]:
df = pd.read_csv('AB_NYC_2019.csv')

# First 5 rows of the DataFrame
df.head()
```

Out[2]:

	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	

We used the pandas library to import our data CSV file 'AB_NYC_2019.csv' and loaded it into the DataFrame named 'df.' The pd.read_csv() function reads the CSV file, while df.head() was used to print the first 5 rows of the DataFrame.
This is a common initial step in data analysis which helps us understand the dataset

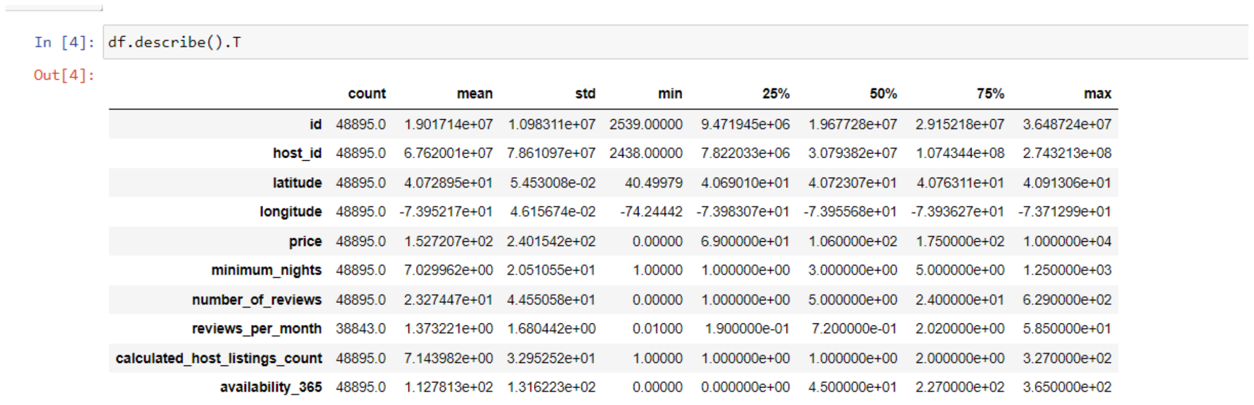


```
In [3]: df.shape
Out[3]: (48895, 16)
```

The dataset's shape is (48895, 13) which suggests that it has substantial size, with almost 49,000 records (rows). Thus we have a vast amount of Airbnb listing data to analyze, which can potentially provide us with valuable insights into the New York City Airbnb market and allow us to explore trends, patterns, and variations across a diverse range of listings.

Step 1 - Using describe method

We will use the df.describe() method to generate descriptive statistics for a DataFrame or Series in pandas.



```
In [4]: df.describe().T
Out[4]:
```

	count	mean	std	min	25%	50%	75%	max
id	48895.0	1.901714e+07	1.098311e+07	2539.00000	9.471945e+06	1.967728e+07	2.915218e+07	3.648724e+07
host_id	48895.0	6.762001e+07	7.861097e+07	2438.00000	7.822033e+06	3.079382e+07	1.074344e+08	2.743213e+08
latitude	48895.0	4.072895e+01	5.453008e-02	40.49979	4.069010e+01	4.072307e+01	4.076311e+01	4.091306e+01
longitude	48895.0	-7.395217e+01	4.615674e-02	-74.24442	-7.398307e+01	-7.395568e+01	-7.393627e+01	-7.371299e+01
price	48895.0	1.527207e+02	2.401542e+02	0.00000	6.900000e+01	1.060000e+02	1.750000e+02	1.000000e+04
minimum_nights	48895.0	7.029962e+00	2.051055e+01	1.00000	1.000000e+00	3.000000e+00	5.000000e+00	1.250000e+03
number_of_reviews	48895.0	2.327447e+01	4.455058e+01	0.00000	1.000000e+00	5.000000e+00	2.400000e+01	6.290000e+02
reviews_per_month	38843.0	1.373221e+00	1.680442e+00	0.01000	1.900000e-01	7.200000e-01	2.020000e+00	5.850000e+01
calculated_host_listings_count	48895.0	7.143982e+00	3.295252e+01	1.00000	1.000000e+00	1.000000e+00	2.000000e+00	3.270000e+02
availability_365	48895.0	1.127813e+02	1.316223e+02	0.00000	0.000000e+00	4.500000e+01	2.270000e+02	3.650000e+02

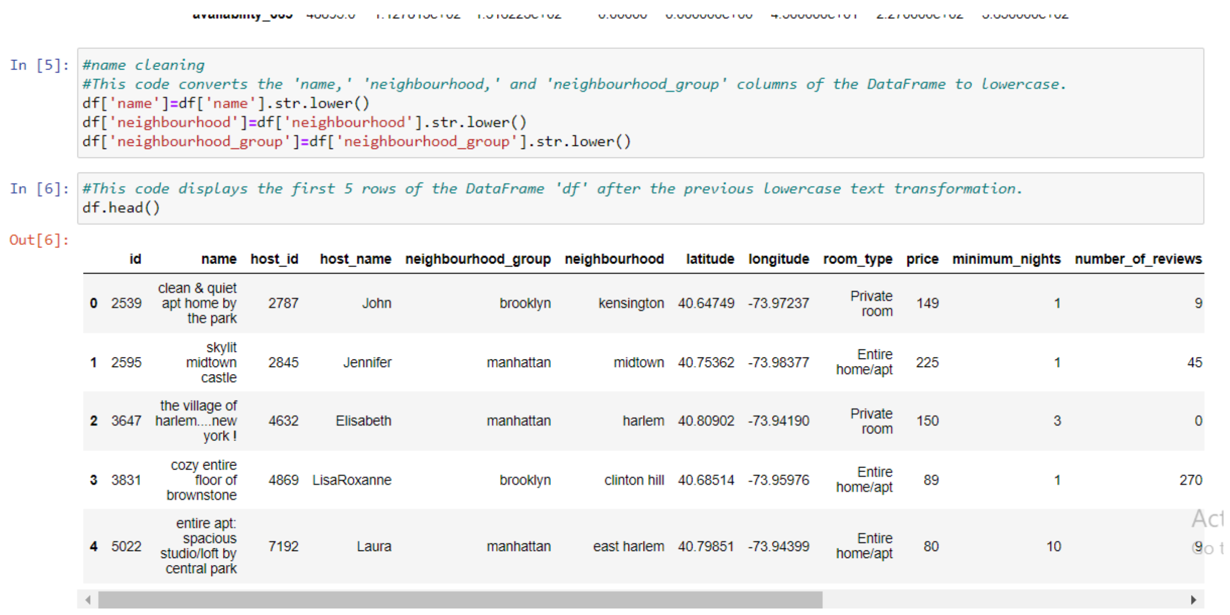
Interesting observations we get after we used describe on our data are -

1. The price column demonstrates a wide range of prices, with a minimum of 0 and a maximum of 10,000. The average price is approximately $152, but the standard deviation is quite high at around $240. This indicates that there is significant price variation among the listings and some outliers might be present that will need further investigation during data cleaning and analysis.
2. The minimum_nights column shows a diverse range of values, with the minimum stay requirement ranging from 1 night to a maximum of 1,250 nights. The average minimum stay is approximately 7 nights, but there is a wide spread as indicated by the standard deviation. Logically, a minimum night stay of 1250 nights does not make much sense, this suggests the presence of outliers.
3. The number_of_reviews column reveals a wide range of review counts, with some listings having no reviews (minimum value of 0) and others having up to 629 reviews. The reviews_per_month column, on average, indicates that most listings receive around 1 review per month. However, there are listings with exceptionally high review rates, with a maximum of 58.5 reviews per month.
4. The calculated_host_listings_count column displays the number of listings managed by a host. While the average is around 7, there are hosts with a significant number of listings, with a maximum count of 327.
5. The availability_365 column shows the number of days a listing is available for booking throughout the year. There is variation in availability, with some listings being available every day (365) and others having no availability (minimum of 0).

Step 2: Converting text data to lowercase

We can convert the text data to lowercase during data cleaning as it is crucial for standardization and consistency in data analysis. It ensures that variations in letter casing do not affect operations like text matching and grouping, making the analysis case-insensitive. Lowercasing also reduces the dimensionality of text data, simplifying analysis while preserving essential semantics.

In our dataset, there are three text columns - ('name,' 'neighbourhood,' and 'neighbourhood_group') for which we can perform conversion of text to lowercase.



```
availability_365  48895.0  1.127813e+02  1.316223e+02  0.000000  0.000000e+00  4.500000e+01  2.270000e+02  3.650000e+02
```

```
In [5]: #name cleaning
        #This code converts the 'name,' 'neighbourhood,' and 'neighbourhood_group' columns of the DataFrame to lowercase.
        df['name']=df['name'].str.lower()
        df['neighbourhood']=df['neighbourhood'].str.lower()
        df['neighbourhood_group']=df['neighbourhood_group'].str.lower()
```

```
In [6]: #This code displays the first 5 rows of the DataFrame 'df' after the previous lowercase text transformation.
        df.head()
```

Out[6]:

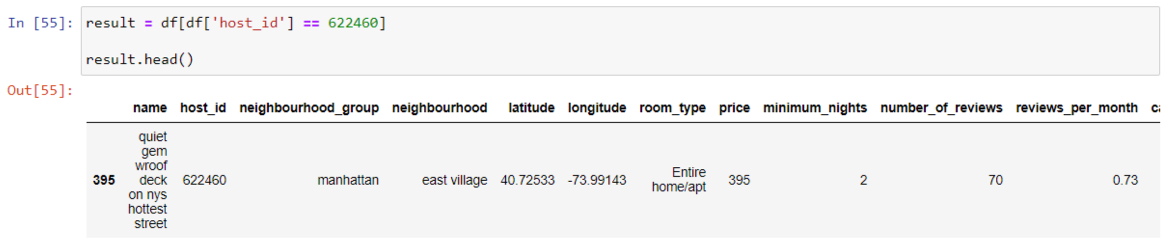
	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews
0	2539	clean & quiet apt home by the park	2787	John	brooklyn	kensington	40.64749	-73.97237	Private room	149	1	9
1	2595	skylit midtown castle	2845	Jennifer	manhattan	midtown	40.75362	-73.98377	Entire home/apt	225	1	45
2	3647	the village of harlem....new york !	4632	Elisabeth	manhattan	harlem	40.80902	-73.94190	Private room	150	3	0
3	3831	cozy entire floor of brownstone	4869	LisaRoxanne	brooklyn	clinton hill	40.68514	-73.95976	Entire home/apt	89	1	270
4	5022	entire apt: spacious studio/loft by central park	7192	Laura	manhattan	east harlem	40.79851	-73.94399	Entire home/apt	80	10	9

str.lower() method was applied to three columns ('name,' 'neighbourhood,' and 'neighbourhood_group') of the DataFrame 'df.' This operation converted all the text in these columns to lowercase.

Step 3 - Removing punctuation from text

Removing punctuation from text during data cleaning is essential for noise reduction, standardizing text, and improving its overall quality. Punctuation can add unnecessary complexity and inconsistency to the data. It aids in tokenization, simplifying text processing, and enhances text matching by ensuring accurate matches. Additionally, removing punctuation can make text more readable and comprehensible for various text analysis tasks.

In the name column, for certain rows, the data has unnecessary punctuations added which needs to be cleaned.

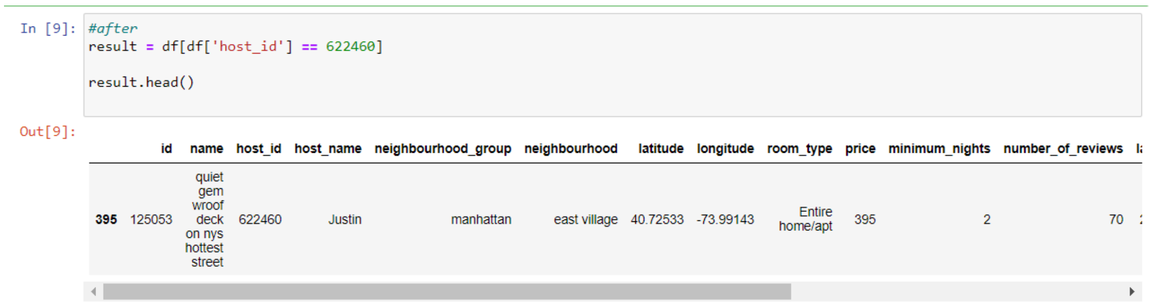


```
In [55]: result = df[df['host_id'] == 622460]

         result.head()
```

Out[55]:

	name	host_id	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	c
395	quiet gem wroof deck on nys hottest street	622460	manhattan	east village	40.72533	-73.99143	Entire home/apt	395	2	70	0.73	

We will use regex to remove the unwanted characters (non-alphanumeric and non-space characters) from the 'name' column



```
In [9]: #after
        result = df[df['host_id'] == 622460]

        result.head()
```

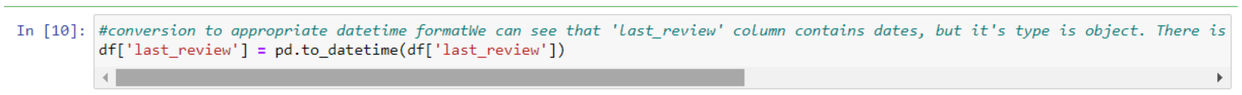
Out[9]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	l
395	125053	quiet gem wroof deck on nys hottest street	622460	Justin	manhattan	east village	40.72533	-73.99143	Entire home/apt	395	2	70	:

We defined a regular expression pattern r'[^a-zA-Z0-9\s]+' which was used to match any characters that were not letters (both uppercase and lowercase), digits, or spaces. The str.replace() method was then applied to the 'name' column of the DataFrame 'df' with this pattern. It replaced any characters that matched the pattern with an empty string, effectively removing them. This is a common data preprocessing step to clean and standardize text data by removing special characters or symbols that may not have been relevant to the analysis or modeling process.

<u>Step 4 - Changing Data type to Datetime</u>

We can see that the 'last_review' column contains dates, but its type is object. So, we need to change its data type to datetime data type. It'll allow us to perform calculations on the dates.

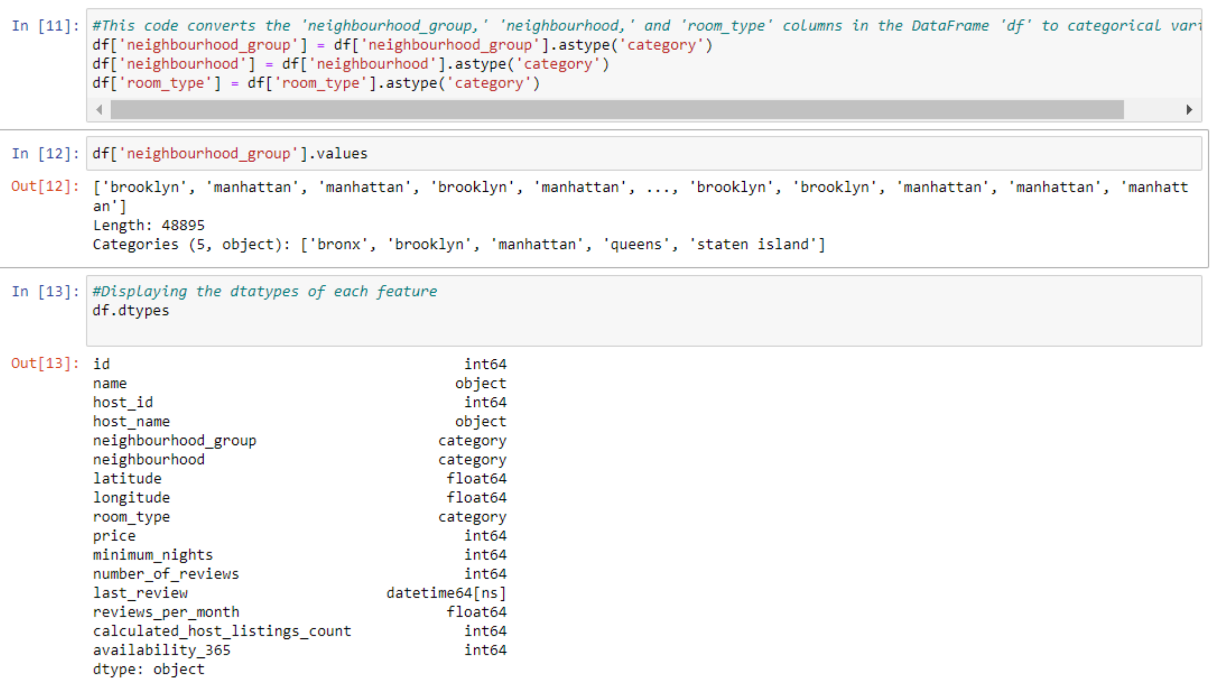


```
In [10]: #conversion to appropriate datetime formatWe can see that 'last_review' column contains dates, but it's type is object. There is
         df['last_review'] = pd.to_datetime(df['last_review'])
```

pd.to_datetime method was used to convert the 'last_review' column in a Pandas DataFrame ('df') into a datetime data type. This transformation allows for easier manipulation and analysis of date and time information within the dataset, enabling operations like sorting and filtering based on dates.

Step 5 - Changing data type to category

Categorical variables are used to represent data that could take on a limited, fixed number of unique values (categories or levels).
We can convert data type of three columns ('neighbourhood_group,' 'neighbourhood,' and 'room_type') in the DataFrame 'df' to categorical variables so that we will be able to handle the data more efficiently, reduce the memory usage, and prepare the data for tasks like grouping, aggregation, and machine learning.

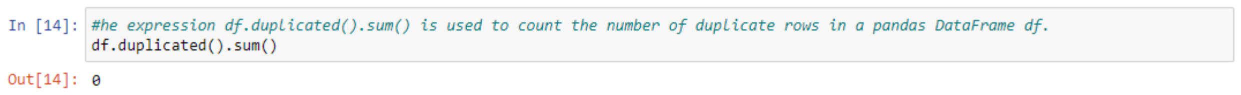


```
In [11]: #This code converts the 'neighbourhood_group,' 'neighbourhood,' and 'room_type' columns in the DataFrame 'df' to categorical vari
         df['neighbourhood_group'] = df['neighbourhood_group'].astype('category')
         df['neighbourhood'] = df['neighbourhood'].astype('category')
         df['room_type'] = df['room_type'].astype('category')

In [12]: df['neighbourhood_group'].values

Out[12]: ['brooklyn', 'manhattan', 'manhattan', 'brooklyn', 'manhattan', ..., 'brooklyn', 'brooklyn', 'manhattan', 'manhattan', 'manhatt
         an']
         Length: 48895
         Categories (5, object): ['bronx', 'brooklyn', 'manhattan', 'queens', 'staten island']

In [13]: #Displaying the dtypes of each feature
         df.dtypes

Out[13]: id                                         int64
         name                                      object
         host_id                                    int64
         host_name                                 object
         neighbourhood_group                     category
         neighbourhood                           category
         latitude                                 float64
         longitude                                float64
         room_type                               category
         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
```

We used the astype('category') method to convert the data type of three columns ('neighbourhood_group,' 'neighbourhood,' and 'room_type') in the DataFrame 'df' to categorical variables.

Step 6a - Check for duplicate rows in the dataset

Checking for duplicate rows during data cleaning is crucial to maintain data quality, improve resource efficiency, ensure analysis accuracy, and enhance model performance. Duplicate rows can introduce errors, waste resources, lead to biased analysis, and affect machine learning model training.

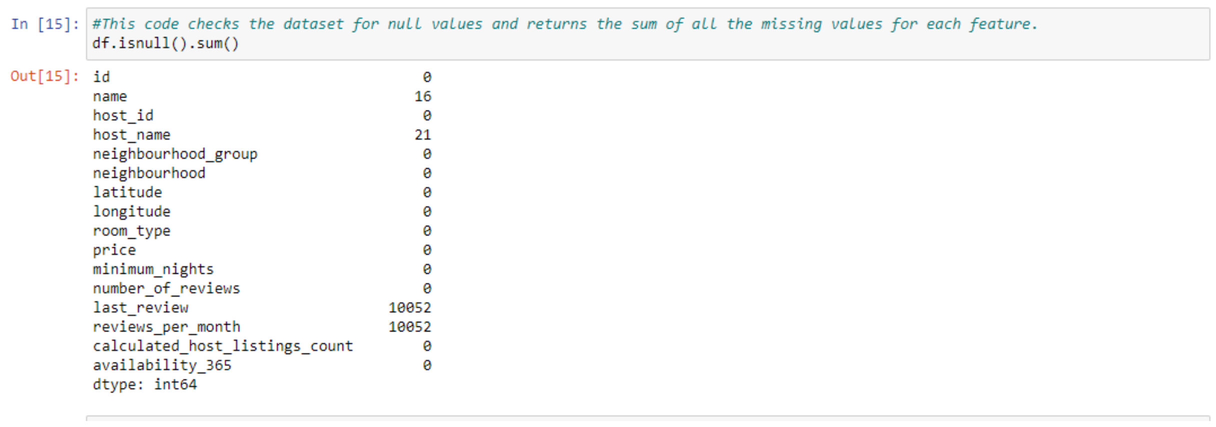


```
In [14]: #he expression df.duplicated().sum() is used to count the number of duplicate rows in a pandas DataFrame df.
         df.duplicated().sum()

Out[14]: 0
```

There are no duplicate rows present in our dataset.

Step 6b - Check null values in the dataset

We will check the null values present in the dataset as checking for null values in a dataset is vital for data quality and analysis. Missing data can lead to errors, affect statistical analysis, and distort visualizations. Addressing missing values through imputation or removal ensures accurate results and reliable model training in machine learning.



```
In [15]: #This code checks the dataset for null values and returns the sum of all the missing values for each feature.
         df.isnull().sum()

Out[15]: 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
```

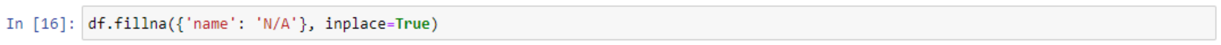
There are missing values in four columns: 'name', 'host_name', 'last_review', and 'reviews_per_month'.

Techniques to deal with missing data:

1. Deletion: We can simply delete the missing values or the entire row or column where they occur. However, this should only be done if the amount of missing data is small and there is no important information in the missing values.
2. Imputation: We can fill in the missing values with some other value or technique. This can be done by replacing missing values with a mean, median, or mode of the data, or by using more sophisticated techniques like regression or machine learning algorithms to predict the missing values.
3. Interpolation: We can use interpolation techniques to estimate the missing values by using the values of neighboring data points. This is particularly useful for time series data.
4. Data augmentation: We can use data augmentation techniques to create new data points based on existing data points. This can be done by using statistical techniques to generate new values or by using machine learning algorithms to generate new data points.

Step 7 - Filling missing values in name column

We will simply replace null values present in the name column with 'N/A for clarity and data consistency. It ensures the dataset is more interpretable, prevents errors, and maintains data integrity.



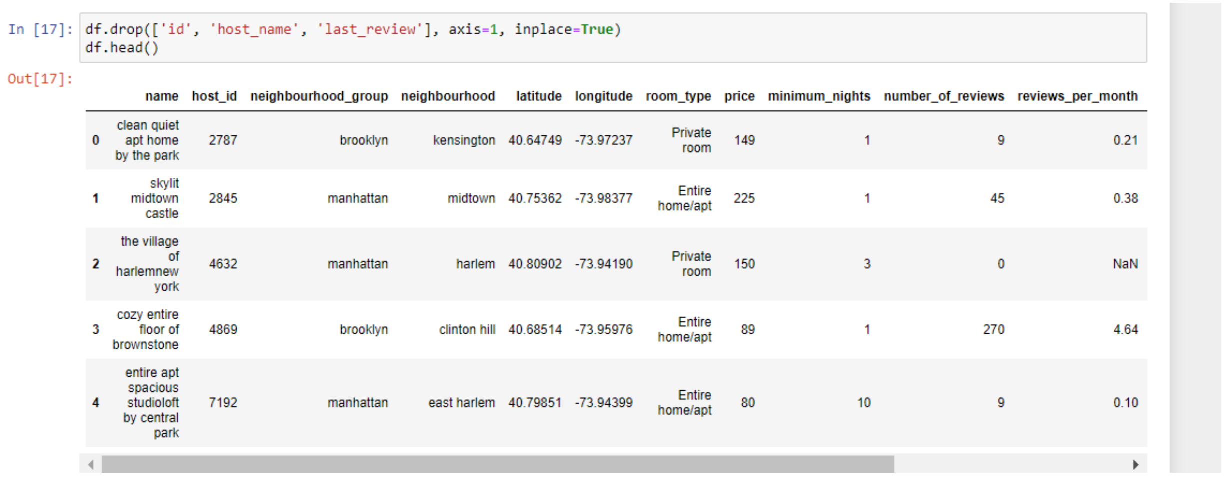
```
In [16]: df.fillna({'name': 'N/A'}, inplace=True)
```

Step 8 - Dropping columns that are irrelevant

The 'host_name' column is irrelevant as there is no need to store people's names. This column can be dropped.

The 'last_review' column contains the date of the last review and missing values simply mean that there are no reviews for that airbnb. This column is also irrelevant for future data analysis and we can also drop it.

The ‘id’ column is also irrelevant because it is unnecessary information and we cannot correlate it with any other data point in the dataset.

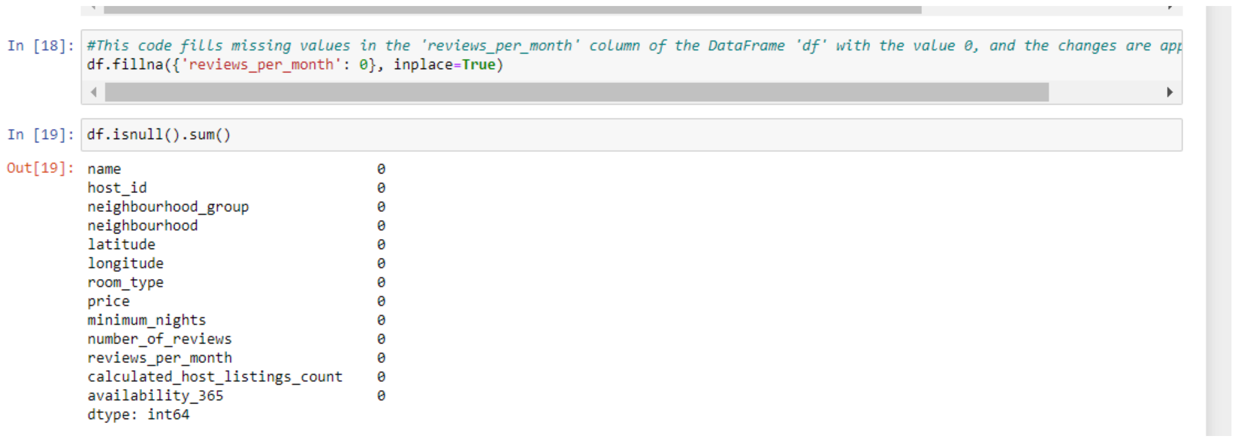


```
In [17]: df.drop(['id', 'host_name', 'last_review'], axis=1, inplace=True)
         df.head()
Out[17]:
```

	name	host_id	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per_month
0	clean quiet apt home by the park	2787	brooklyn	kensington	40.64749	-73.97237	Private room	149	1	9	0.21
1	skylit midtown castle	2845	manhattan	midtown	40.75362	-73.98377	Entire home/apt	225	1	45	0.38
2	the village of harlemnew york	4632	manhattan	harlem	40.80902	-73.94190	Private room	150	3	0	NaN
3	cozy entire floor of brownstone	4869	brooklyn	clinton hill	40.68514	-73.95976	Entire home/apt	89	1	270	4.64
4	entire apt spacious studioloft by central park	7192	manhattan	east harlem	40.79851	-73.94399	Entire home/apt	80	10	9	0.10

Step 9 - Filling missing values for 'reviews_per_month' column

The 'reviews_per_month' column gives information about the number of reviews per month. The missing values in that column means the item/airbnb has no reviews for all time i.e. it has 0 reviews per month.

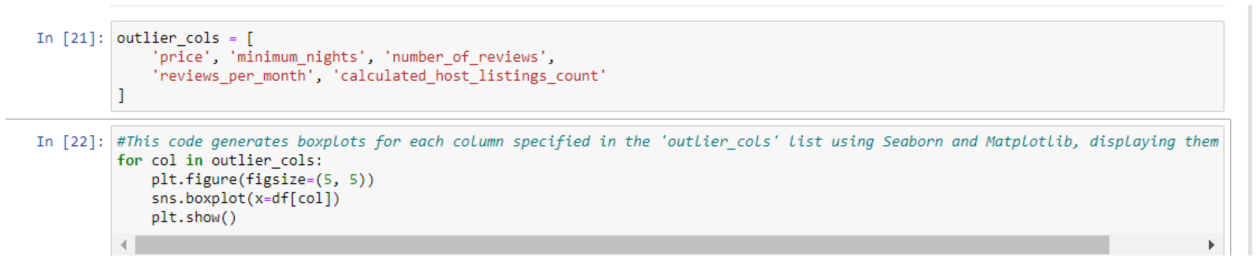


```
In [18]: #This code fills missing values in the 'reviews_per_month' column of the DataFrame 'df' with the value 0, and the changes are app
         df.fillna({'reviews_per_month': 0}, inplace=True)

In [19]: df.isnull().sum()

Out[19]: name                              0
         host_id                           0
         neighbourhood_group               0
         neighbourhood                     0
         latitude                          0
         longitude                         0
         room_type                         0
         price                             0
         minimum_nights                    0
         number_of_reviews                 0
         reviews_per_month                 0
         calculated_host_listings_count    0
         availability_365                  0
         dtype: int64
```

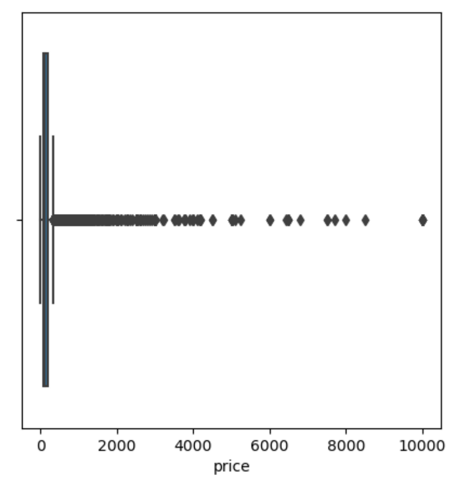
We have filled/handled all the missing data in our dataset.

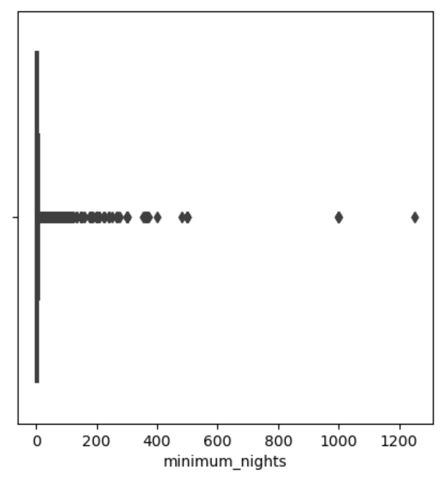
Step 10 - Outlier detection

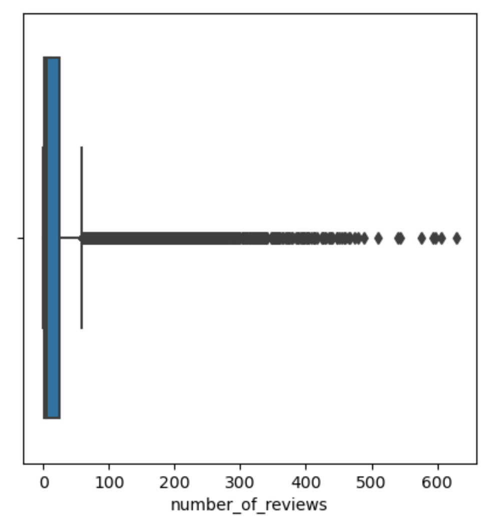
We will define the columns in which there is a presence of potential outliers and store it in a list. Then we will visualize the boxplot of the data distribution and identify the presence of outliers

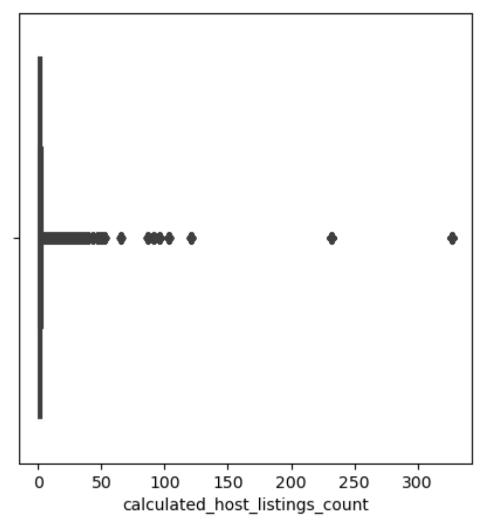


```
In [21]: outlier_cols = [
             'price', 'minimum_nights', 'number_of_reviews',
             'reviews_per_month', 'calculated_host_listings_count'
         ]

In [22]: #This code generates boxplots for each column specified in the 'outlier_cols' list using Seaborn and Matplotlib, displaying them
         for col in outlier_cols:
             plt.figure(figsize=(5, 5))
             sns.boxplot(x=df[col])
             plt.show()
```









We iterated over the list of columns present in the 'outlier_cols.' list. For each column in the list, it created a separate boxplot using Seaborn (sns.boxplot()) to visualize the distribution and identify potential outliers in the data. The plt.show() function was used to display each boxplot one at a time in separate figures with a size of 5x5 inches.

Step 11 - Detecting outliers using z score method

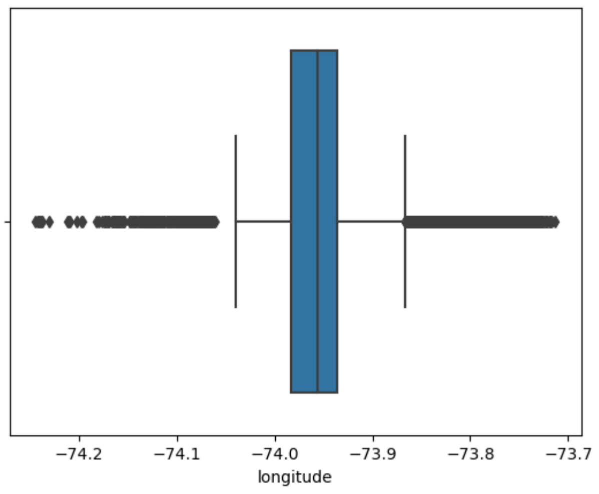
We will use the z score method for outlier detection, in which a z-score above 3 is often considered a threshold for identification of outliers.

```
In [23]: # This code selects numeric columns from the DataFrame 'df,' calculates the z-scores for each numeric column,
         # and identifies and prints the number of outliers in each column where z-scores exceed a threshold of 3.
         # Select numeric columns
         df_numeric = df.select_dtypes(include=[np.number])

         for col in df_numeric.columns:
             # Calculate z-score of the column
             z_score = np.abs(stats.zscore(df[col]))
             outliers_num = len(np.where(z_score > 3)[0])
             if outliers_num:
                 print('{}: {}'.format(col, outliers_num))

         latitude: 99
         longitude: 1236
         price: 388
         minimum_nights: 327
         number_of_reviews: 1221
         reviews_per_month: 847
         calculated_host_listings_count: 680
```

We selected the numeric columns in the DataFrame 'df' and calculated z-scores for each column, and identified and printed the count of outliers in each column where the z-scores was greater than 3

The longitude column has the most outliers, so we will use a boxplot to visualize and identify the outliers present in it

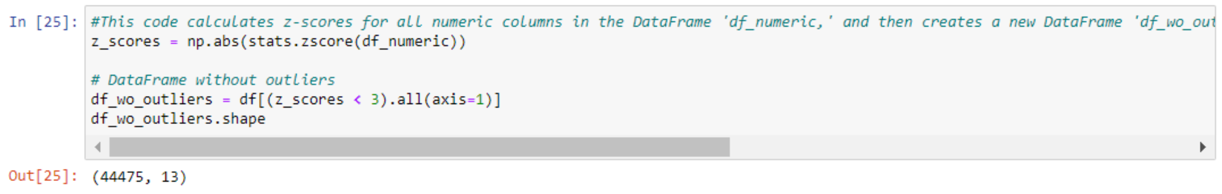
```
In [24]: #This code creates a boxplot using Seaborn to visualize the distribution of the 'longitude' column in the DataFrame 'df.'
         sns.boxplot(x=df['longitude']);
```



The code sns.boxplot(x=df['longitude']) will create a boxplot using Seaborn (sns) for the 'longitude' column in our DataFrame. A boxplot visually represents the distribution of the 'longitude' data, allowing us to identify potential outliers.

Step 12 - Removing outliers

We will remove the outliers from the dataset where z score is greater than 3.



```
In [25]: #This code calculates z-scores for all numeric columns in the DataFrame 'df_numeric,' and then creates a new DataFrame 'df_wo_out
         z_scores = np.abs(stats.zscore(df_numeric))

         # DataFrame without outliers
         df_wo_outliers = df[(z_scores < 3).all(axis=1)]
         df_wo_outliers.shape

Out[25]: (44475, 13)
```

This code removes outliers by calculating z-scores for the numeric columns and then filtering the data based on these z-scores.

np.abs(stats.zscore(df_numeric)) computes the z-scores for all numeric columns in the DataFrame 'df_numeric.' Higher z-scores indicate that the data points are far from the mean and can be considered as potential outliers.

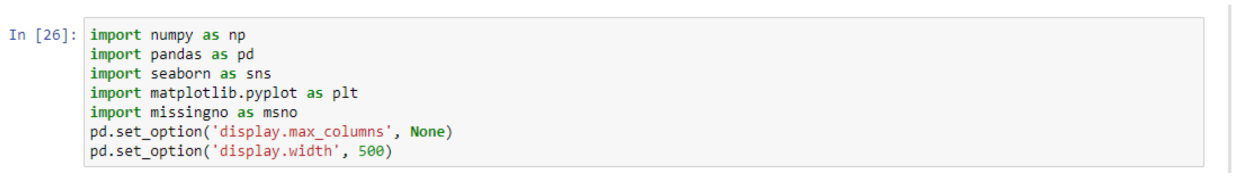
df_wo_outliers = df[(z_scores < 3).all(axis=1)] creates a new DataFrame 'df_wo_outliers' by filtering 'df' using the condition (z_scores < 3).all(axis=1). This condition checks that for each row, all the z-scores in that row (across all numeric columns) are less than 3. A z-score less than 3 is typically used as a threshold for identifying outliers. If all z-scores in a row are less than 3, the row is retained in 'df_wo_outliers.'

df_wo_outliers.shape: This line simply provides the shape (number of rows and columns) of the resulting DataFrame 'df_wo_outliers' to check how many rows remain after removing outliers.

EXPLORATORY DATA ANALYSIS:

Exploratory Data Analysis (EDA) is an essential initial step in the data analysis process, serving as a compass to navigate the intricate landscape of raw data. In this crucial phase, data analysts and scientists dive into the dataset, aiming to unveil its underlying patterns, trends, and peculiarities. EDA involves a spectrum of tasks, from creating informative visualizations that shed light on data distribution and relationships to calculating descriptive statistics, dealing with missing values and outliers, and conducting hypothesis tests. Through EDA, one can gain valuable insights, uncover potential pitfalls, and make informed decisions about how to proceed with more advanced analysis techniques or modeling, ultimately leading to a deeper understanding of the data and its inherent story.

- <u>Importing necessary libraries</u>:



```
In [26]: import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         import missingno as msno
         pd.set_option('display.max_columns', None)
         pd.set_option('display.width', 500)
```

1. numpy (imported as np):

NumPy serves as a foundational library for performing numerical operations in Python. It equips us with essential capabilities for working with arrays and matrices, which are pivotal for mathematical and statistical computations. In our project, we will utilize NumPy for data manipulation and analysis.

2. pandas (imported as pd):

Pandas is a robust library that empowers us to manipulate and analyze data efficiently. It offers versatile data structures like DataFrames and Series, facilitating the handling and exploration of tabular data. Our project will heavily rely on Pandas for loading, cleaning, and analyzing our Airbnb dataset.

3. seaborn (imported as sns):

Seaborn represents a valuable data visualization library that builds upon Matplotlib. It presents a user-friendly interface for crafting aesthetically pleasing statistical graphics. We can leverage Seaborn to generate various types of plots and visualizations, aiding our understanding of our dataset's attributes and relationships.

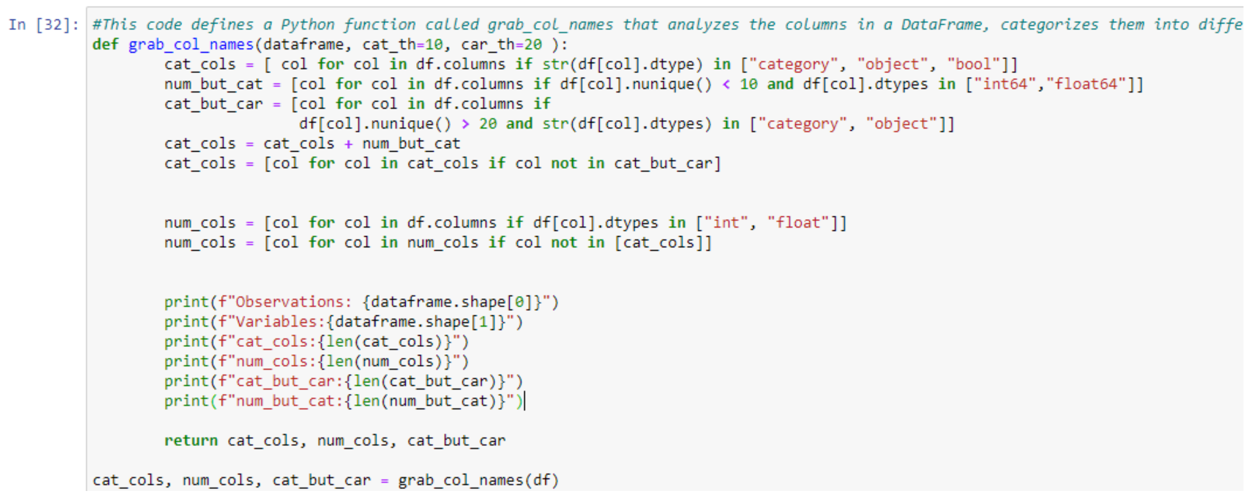
4. matplotlib.pyplot (imported as plt):

Matplotlib stands as a widely-adopted plotting library within Python. The pyplot module offers a straightforward and customizable approach to constructing an array of plots, charts, and graphs. We intend to use Matplotlib to visualize our findings during EDA and craft data visualizations beyond Seaborn's capabilities.

5. missingno (imported as msno):

Missingno emerges as a specialized library tailored to visualizing missing data within datasets. It equips us with tools to visualize the presence and patterns of missing values, which proves indispensable during the data cleaning and EDA phases. Missingno will help us efficiently identify and address missing data within our NYC Airbnb dataset.

We will write a function called grab_col_names that analyzes the columns in a DataFrame, categorizes them into different types (categorical, numerical, etc.), and returns these categorized column lists.



```
In [32]: #This code defines a Python function called grab_col_names that analyzes the columns in a DataFrame, categorizes them into diffe
         def grab_col_names(dataframe, cat_th=10, car_th=20 ):
                 cat_cols = [ col for col in df.columns if str(df[col].dtype) in ["category", "object", "bool"]]
                 num_but_cat = [col for col in df.columns if df[col].nunique() < 10 and df[col].dtypes in ["int64","float64"]]
                 cat_but_car = [col for col in df.columns if
                                   df[col].nunique() > 20 and str(df[col].dtypes) in ["category", "object"]]
                 cat_cols = cat_cols + num_but_cat
                 cat_cols = [col for col in cat_cols if col not in cat_but_car]


                 num_cols = [col for col in df.columns if df[col].dtypes in ["int", "float"]]
                 num_cols = [col for col in num_cols if col not in [cat_cols]]


                 print(f"Observations: {dataframe.shape[0]}")
                 print(f"Variables:{dataframe.shape[1]}")
                 print(f"cat_cols:{len(cat_cols)}")
                 print(f"num_cols:{len(num_cols)}")
                 print(f"cat_but_car:{len(cat_but_car)}")
                 print(f"num_but_cat:{len(num_but_cat)}")

                 return cat_cols, num_cols, cat_but_car

         cat_cols, num_cols, cat_but_car = grab_col_names(df)
```

- grab_col_names function:

The provided code defines a Python function called grab_col_names that categorizes the columns of a given DataFrame into four types: categorical columns (cat_cols), numerical columns that are treated as categorical (num_but_cat), categorical columns with high cardinality (cat_but_car), and other numerical columns (num_cols). It then prints essential statistics about the dataset, including the number of observations (rows) and variables (columns) and the counts of each column type. This code is crucial for EDA because it automates the initial step of understanding the dataset's composition and distribution, facilitating better-informed decisions on how to approach subsequent analysis tasks.

1. Column Categorization:

Categorizing columns into types helps EDA by allowing us to apply appropriate analysis techniques and visualizations to each category. For example, we treat categorical and numerical data differently.

2. Handling High Cardinality:

Identifying high cardinality categorical columns is essential because they often require specialized handling to avoid overwhelming visualizations. Knowing which columns fall into this category helps in planning EDA strategies.

3. Summary Statistics:

Providing summary statistics such as the number of rows and columns gives an immediate understanding of the dataset's size and complexity. It aids in resource allocation and project planning.

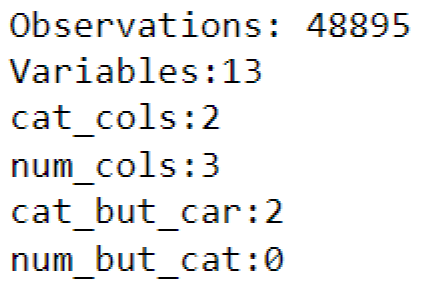
4. Column Counts:

Counting columns in each category offers a quick overview of the dataset's composition. It informs us of the distribution of data types, helping in deciding where to focus our analysis efforts.

5. Usage of Data Types:

Relying on data types for categorization ensures that columns are grouped based on how they are treated in analysis. This is crucial for applying appropriate statistical and visualization techniques.

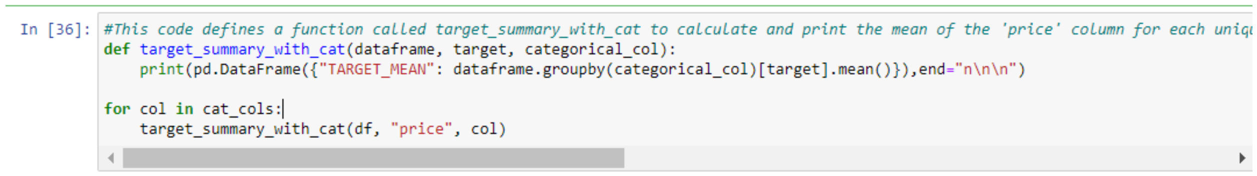
Below, is the output of the code:



```
Observations: 48895
Variables:13
cat_cols:2
num_cols:3
cat_but_car:2
num_but_cat:0
```

The output indicated that the dataset used for the EDA contained 48,895 observations (rows) and 13 variables (columns). Out of these columns, 5 were identified as categorical, 7 as numerical, and none were categorized as categorical with high cardinality. Additionally, one column was considered numerical but treated as categorical due to its relatively low number of unique values. This concise summary helped us understand the dataset's fundamental composition, providing a foundation for structuring and tailoring the subsequent exploratory data analysis (EDA) tasks according to the types of variables present in the data.

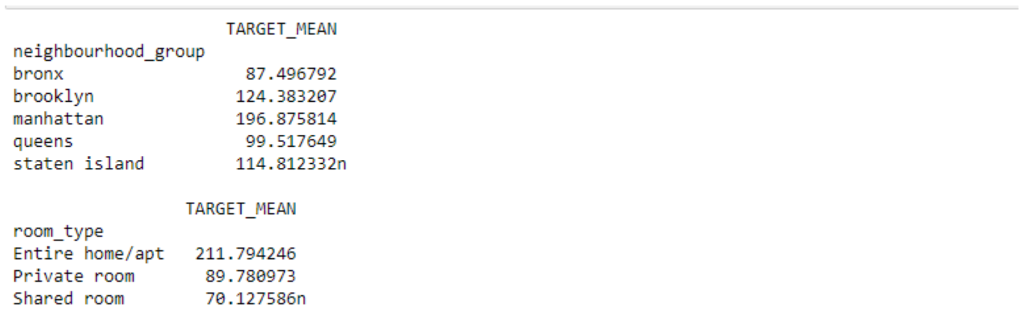
We will write a function called target_summary_with_cat to calculate and print the mean of the 'price' column for each unique value in a specified categorical column. It then applies this function to all categorical columns in the DataFrame 'df' with the target variable 'price.'



```
In [36]: #This code defines a function called target_summary_with_cat to calculate and print the mean of the 'price' column for each uniq
         def target_summary_with_cat(dataframe, target, categorical_col):
             print(pd.DataFrame({"TARGET_MEAN": dataframe.groupby(categorical_col)[target].mean()}),end="n\n\n")

         for col in cat_cols:
             target_summary_with_cat(df, "price", col)
```

- target_summary_with_cat function:

The code defines a function called target_summary_with_cat that takes three parameters: dataframe, target, and categorical_col. Inside this function, it groups the data in the DataFrame by the values in the specified categorical_col and calculates the mean of the target variable within each group. The result is displayed as a Pandas DataFrame, showing the mean value of the target variable for each category in the specified categorical column.

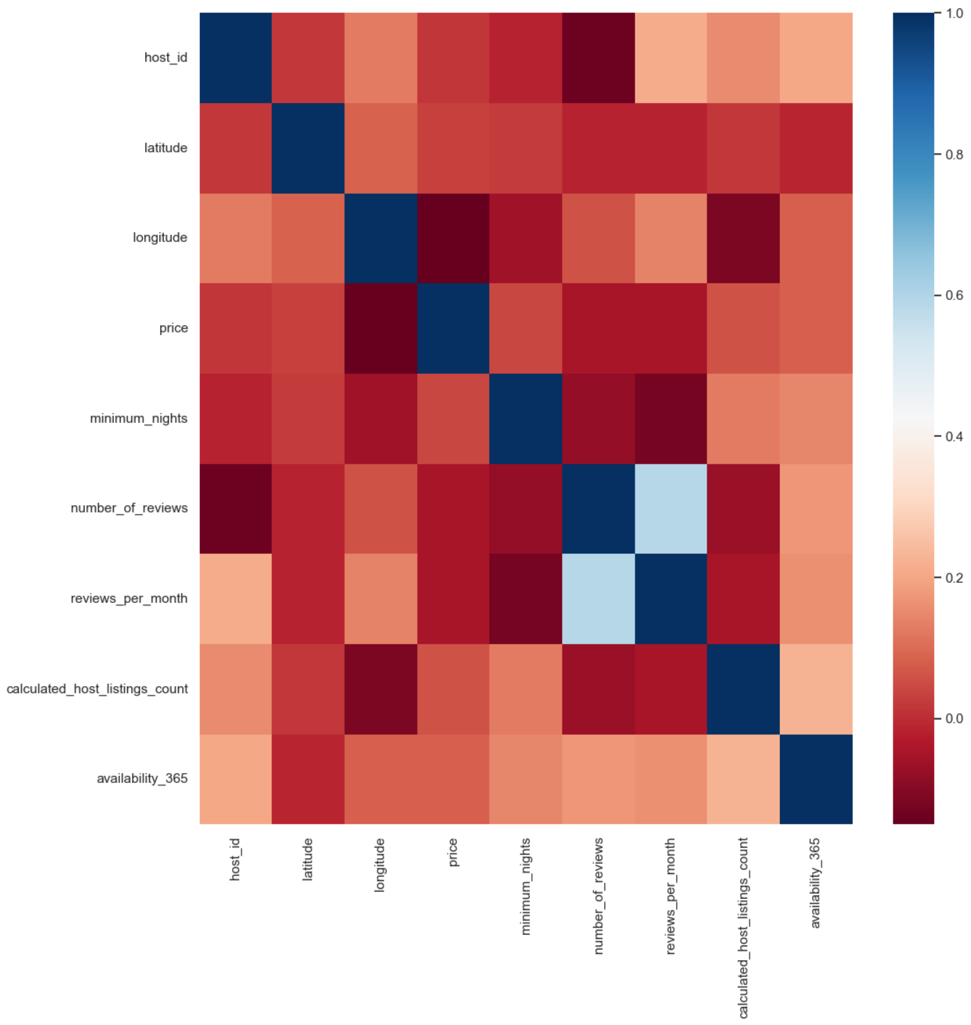


```
                     TARGET_MEAN
neighbourhood_group
bronx                  87.496792
brooklyn              124.383207
manhattan             196.875814
queens                 99.517649
staten island         114.812332n

                 TARGET_MEAN
room_type
Entire home/apt   211.794246
Private room       89.780973
Shared room        70.127586n
```

After defining the function, the code iterates over the categorical columns (cat_cols) and calls the target_summary_with_cat function for each of these columns. For each categorical column, it calculates and displays the mean of the target variable ("price") for each category within that column. This process automates the generation of summary statistics, helping to understand how the target variable varies across different categories within each identified categorical column during exploratory data analysis (EDA).

We will create a heatmap for our dataset.

```
In [42]: corr = df.corr(numeric_only=True)

         #create a heatmap.

         sns.set(rc = {'figure.figsize': (12, 12)})
         sns.heatmap(corr, cmap = "RdBu")
         plt.show()
```



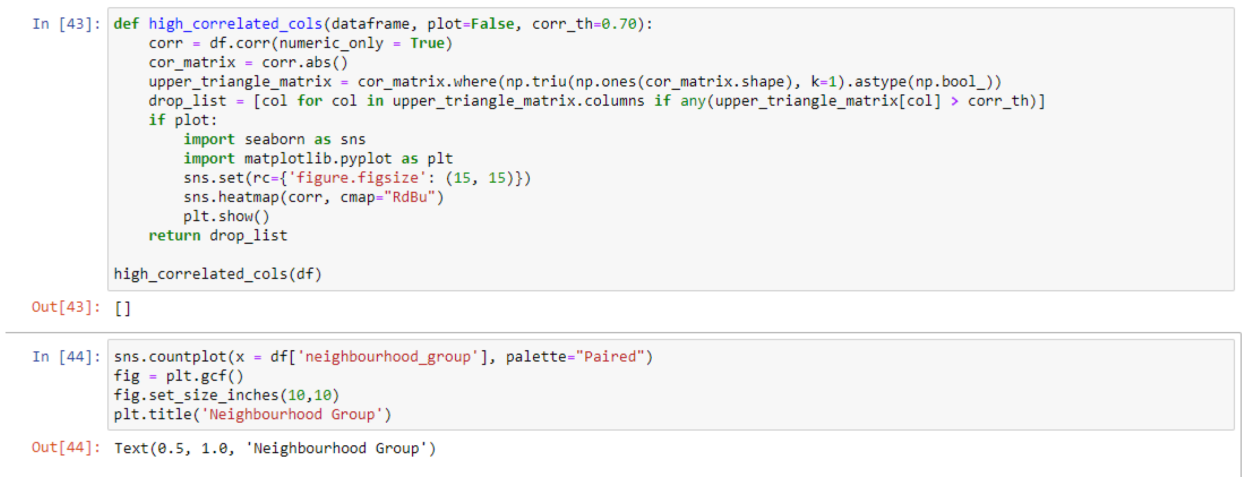
- Heatmap:

Following are our observations based on the heatmap created.
Significant observations from the Airbnb NYC correlation heatmap
1. Price has a positive correlation with availability_365 and longitude.
2. Price has a negative correlation with latitude.
3. Number of reviews has a positive correlation with availability_365, calculated_host_listings_count, and reviews_per_month.

4. Number of reviews has a negative correlation with price.
5. Availability_365 has a positive correlation with calculated_host_listings_count and reviews_per_month.
Below are the insights are conclusions based on our observations:

1. Airbnbs that are available more often tend to be more expensive and have more reviews.
2. Airbnbs in more popular tourist areas (e.g., Manhattan) tend to be more expensive and have fewer reviews.
3. Airbnbs with more reviews tend to be more expensive and available more often.
4. Airbnbs with more host listings tend to be available more often and have more reviews.

We will write a function to create a countplot for the neighborhood_group column to display the count of listed properties in each group.



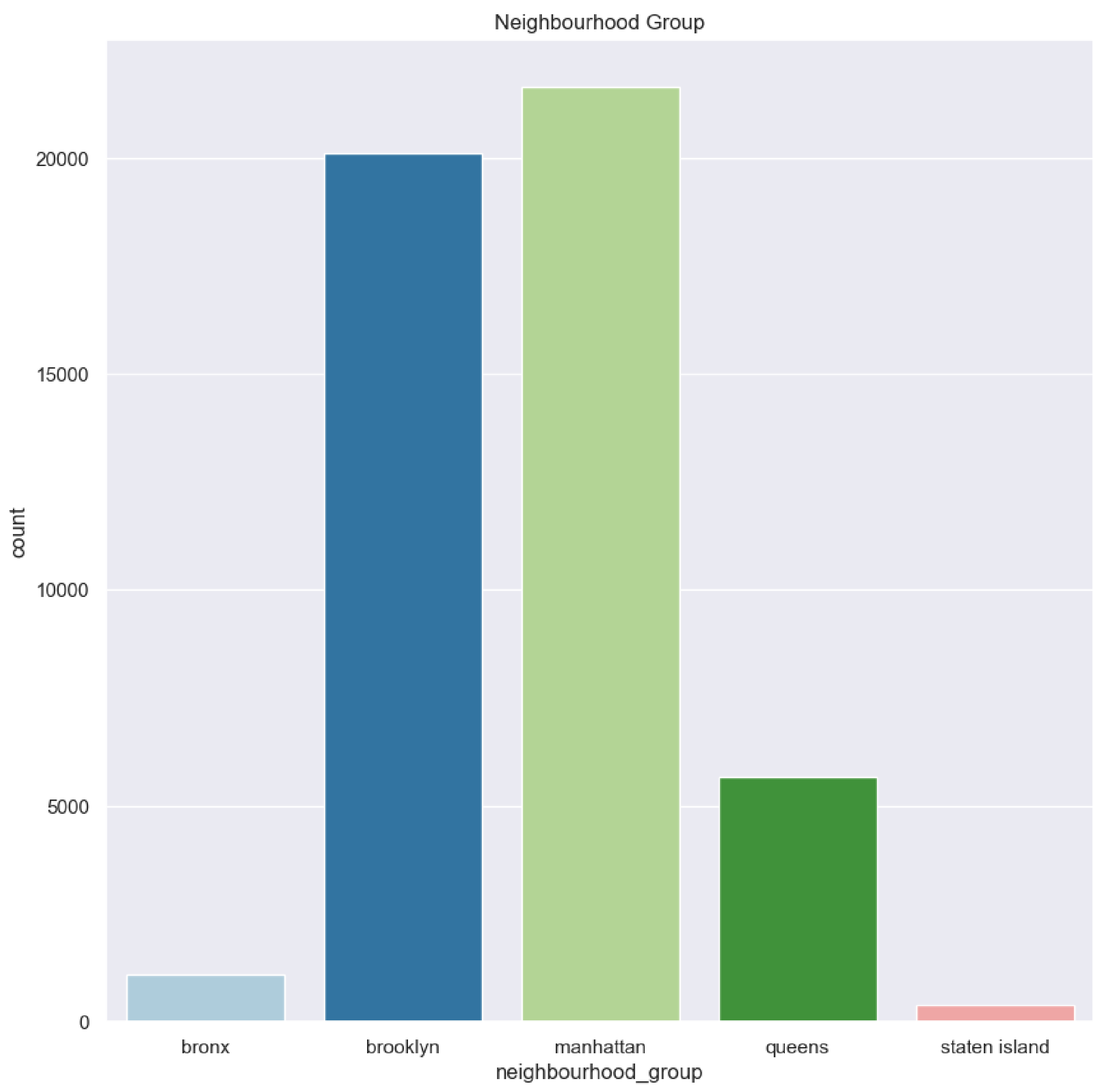
```
In [43]: def high_correlated_cols(dataframe, plot=False, corr_th=0.70):
             corr = df.corr(numeric_only = True)
             cor_matrix = corr.abs()
             upper_triangle_matrix = cor_matrix.where(np.triu(np.ones(cor_matrix.shape), k=1).astype(np.bool_))
             drop_list = [col for col in upper_triangle_matrix.columns if any(upper_triangle_matrix[col] > corr_th)]
             if plot:
                 import seaborn as sns
                 import matplotlib.pyplot as plt
                 sns.set(rc={'figure.figsize': (15, 15)})
                 sns.heatmap(corr, cmap="RdBu")
                 plt.show()
             return drop_list

         high_correlated_cols(df)
Out[43]: []
```

```
In [44]: sns.countplot(x = df['neighbourhood_group'], palette="Paired")
         fig = plt.gcf()
         fig.set_size_inches(10,10)
         plt.title('Neighbourhood Group')
Out[44]: Text(0.5, 1.0, 'Neighbourhood Group')
```

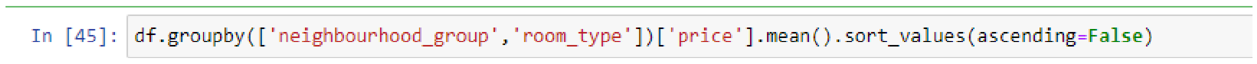
- Countplot:

Observing the countplot we can come up with following observations:

1. Brooklyn is the most populous neighborhood group in New York City, with over 2 million people.
2. Queens is the second most populous neighborhood group, with over 1.5 million people.
3. Manhattan is the third most populous neighborhood group, with over 1 million people.
4. The Bronx and Staten Island are the least populous neighborhood groups, with around 500,000 people each.



- Analysis of Mean Prices by Neighborhood Group and Room Type:



```
In [45]: df.groupby(['neighbourhood_group','room_type'])['price'].mean().sort_values(ascending=False)
```

In this section, we examine the mean (average) prices for different combinations of 'neighbourhood_group' and 'room_type' in the dataset. The analysis is performed as follows:

1. Grouping by Attributes:

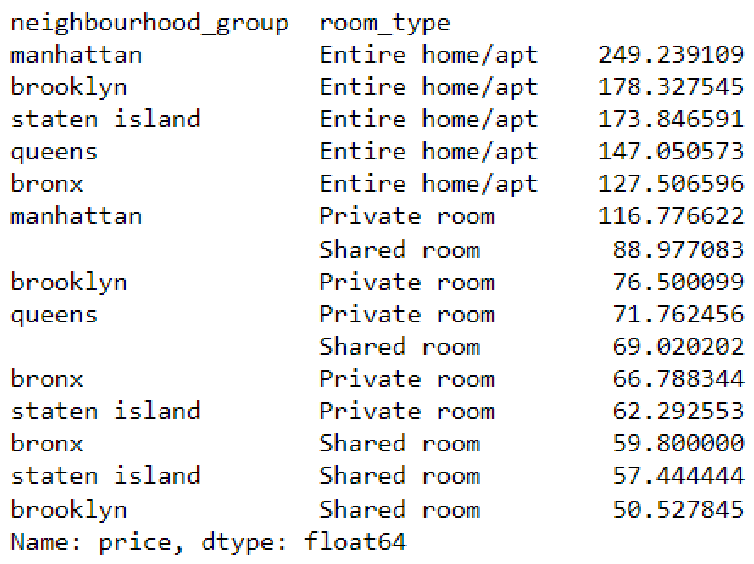
We group the dataset by two categorical attributes, 'neighbourhood_group' and 'room_type.' This grouping divides the data into distinct subgroups based on these two attributes.

2. Mean Price Calculation:

Within each subgroup, we calculate the mean (average) price for the 'price' column. This provides insights into the typical pricing for different types of rooms in various neighborhood groups.

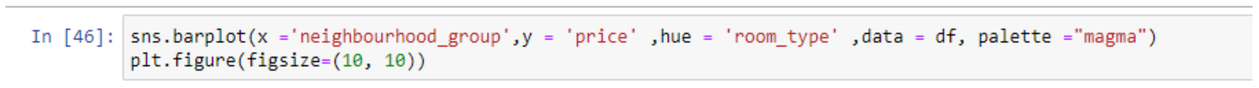
3. Sorting by Mean Price:

The results are then sorted in descending order based on the calculated mean prices. This arrangement places the combinations with the highest mean prices at the top of the list, allowing for easy identification of the most expensive neighborhood group and room type combinations. The output of this analysis helps in understanding the pricing dynamics within different neighborhood groups and room types, providing valuable insights for decision-making and further exploration in the context of the Airbnb dataset.



```
neighbourhood_group  room_type
manhattan            Entire home/apt    249.239109
brooklyn             Entire home/apt    178.327545
staten island        Entire home/apt    173.846591
queens               Entire home/apt    147.050573
bronx                Entire home/apt    127.506596
manhattan            Private room       116.776622
                     Shared room         88.977083
brooklyn             Private room        76.500099
queens               Private room        71.762456
                     Shared room         69.020202
bronx                Private room        66.788344
staten island        Private room        62.292553
bronx                Shared room         59.800000
staten island        Shared room         57.444444
brooklyn             Shared room         50.527845
Name: price, dtype: float64
```

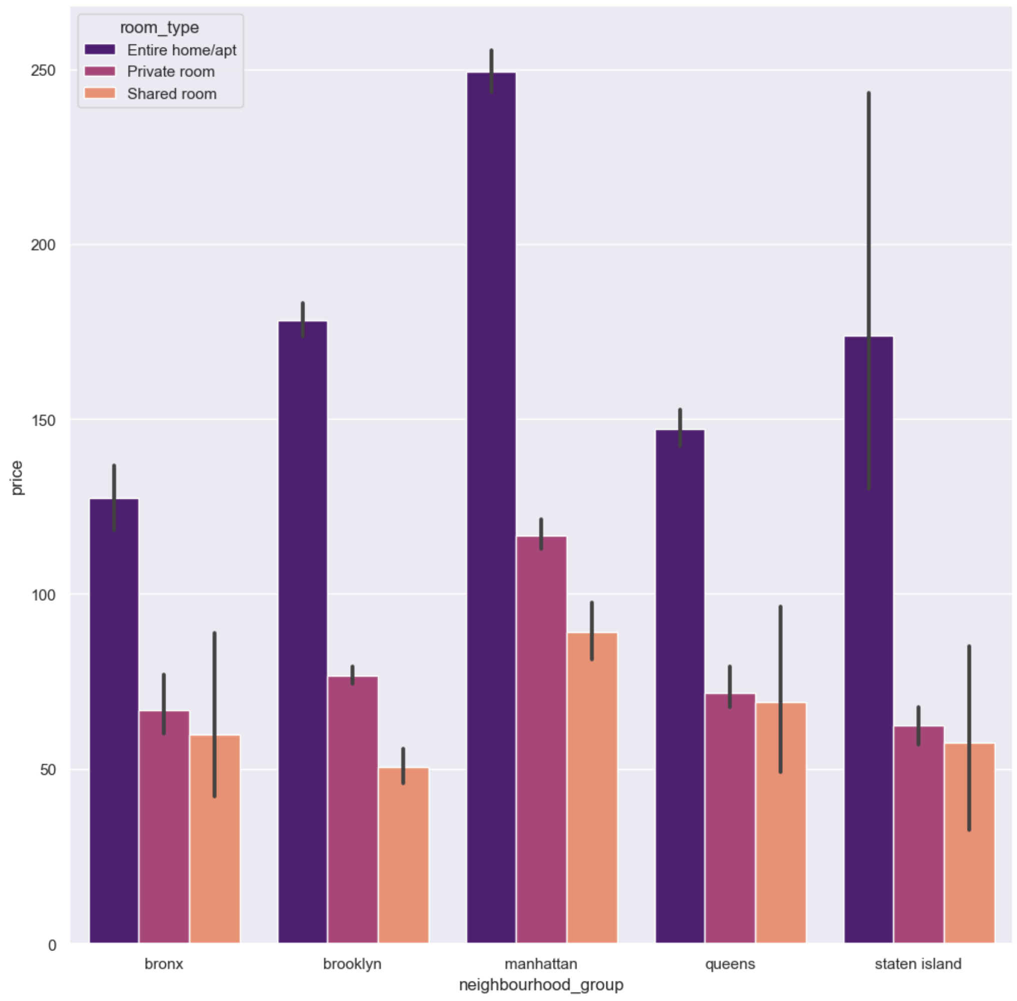
- BarPlot:



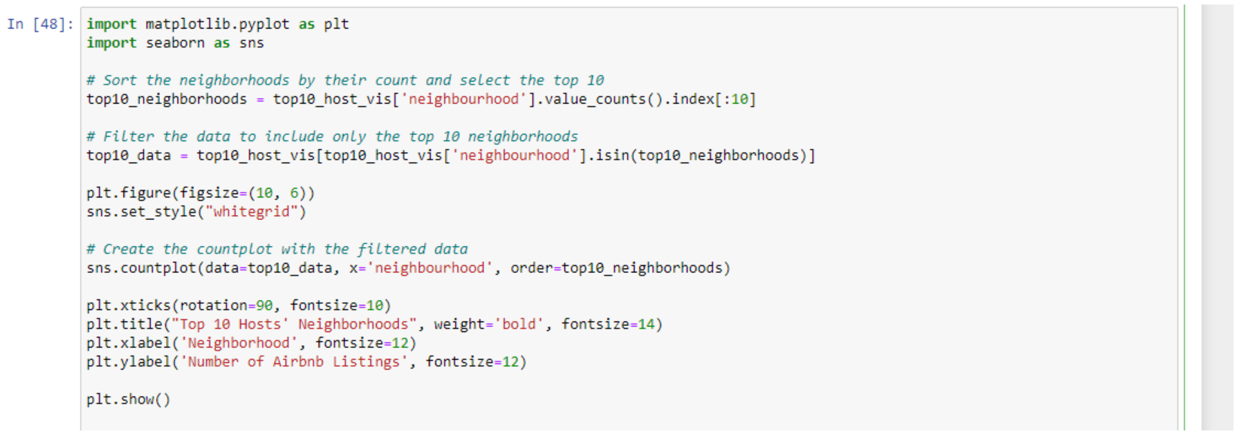
```
In [46]: sns.barplot(x ='neighbourhood_group',y = 'price' ,hue = 'room_type' ,data = df, palette ="magma")
         plt.figure(figsize=(10, 10))
```

Below are the most important insights that we get from barplot:

1. Entire homes/apartments are the most expensive type of room to rent.
2. Private rooms are the second most expensive type of room to rent.
3. Shared rooms are the least expensive type of room to rent.
4. Manhattan is the most expensive borough to rent a room in.
5. Queens is the least expensive borough to rent a room in.
6. The price difference between entire homes/apartments and private rooms is relatively small in Staten Island, compared to other boroughs.
7. The price difference between private rooms and shared rooms is relatively large in Brooklyn, compared to other boroughs.
8. There is a significant price difference between renting a room in Manhattan and renting a room in any other borough.



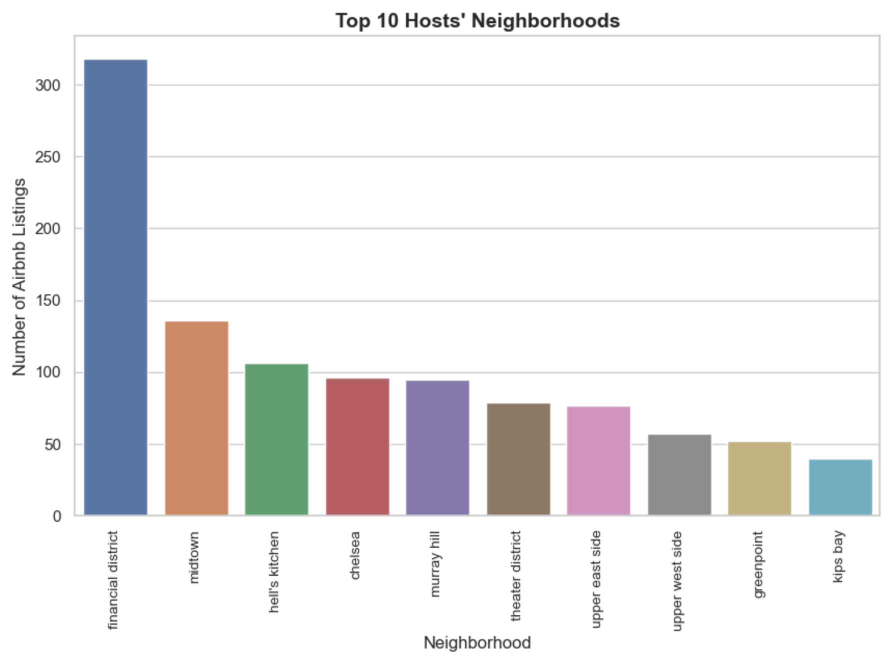
- Countplot for Neighborhood:



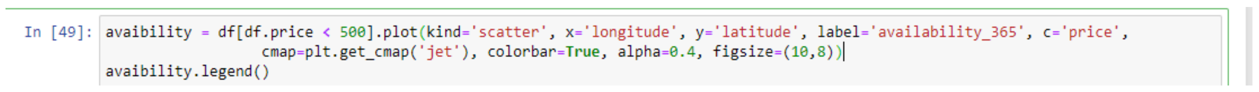
```
In [48]: import matplotlib.pyplot as plt
         import seaborn as sns

         # Sort the neighborhoods by their count and select the top 10
         top10_neighborhoods = top10_host_vis['neighbourhood'].value_counts().index[:10]

         # Filter the data to include only the top 10 neighborhoods
         top10_data = top10_host_vis[top10_host_vis['neighbourhood'].isin(top10_neighborhoods)]

         plt.figure(figsize=(10, 6))
         sns.set_style("whitegrid")

         # Create the countplot with the filtered data
         sns.countplot(data=top10_data, x='neighbourhood', order=top10_neighborhoods)

         plt.xticks(rotation=90, fontsize=10)
         plt.title("Top 10 Hosts' Neighborhoods", weight='bold', fontsize=14)
         plt.xlabel('Neighborhood', fontsize=12)
         plt.ylabel('Number of Airbnb Listings', fontsize=12)

         plt.show()
```

In order to deduce some insights that can help us to learn more about the trends of the dataset we have tried to find other factors not included in the datasets like Shopping malls, Medical school, etc in the vicinity. Most significant insights about the neighborhood are as follows:

1. There is a concentration of Airbnb listings in the central part of the city, with the top 10 neighborhoods all located within a few miles of each other.
2. The Galleria neighborhood, which is home to a large shopping mall and a variety of other businesses and attractions, is also a popular destination for Airbnb guests.
3. The Medical Center neighborhood, which is home to a number of hospitals and medical schools, is also a popular destination for Airbnb guests, particularly those who are visiting the city for medical reasons.



- Scatterplot of latitude and longitude:

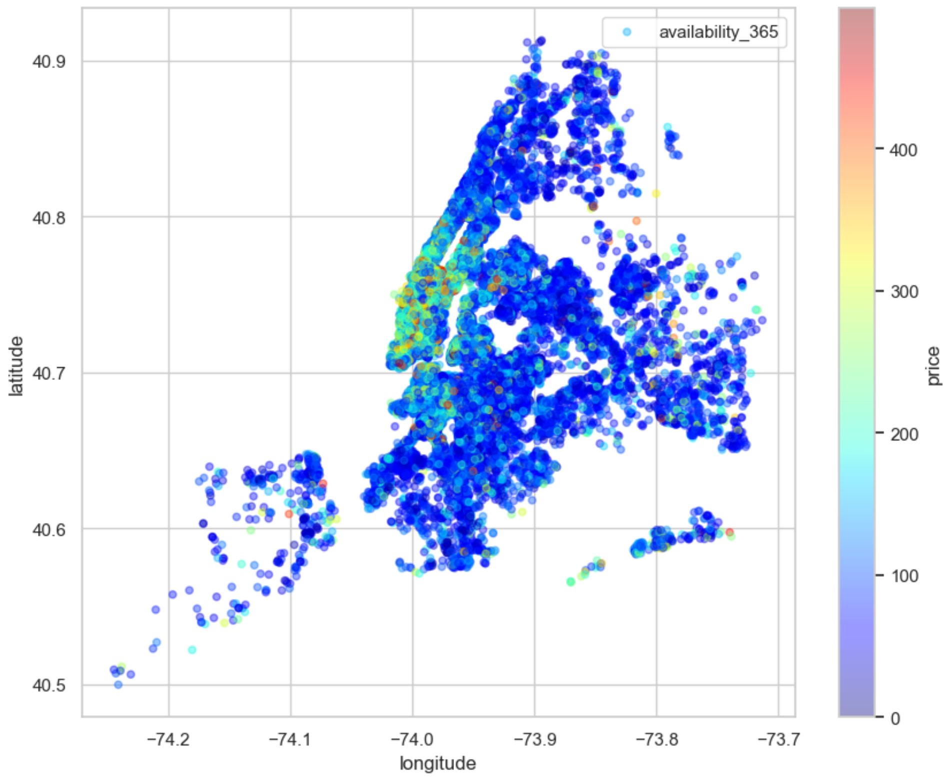


```
In [49]: avaibility = df[df.price < 500].plot(kind='scatter', x='longitude', y='latitude', label='availability_365', c='price',
                    cmap=plt.get_cmap('jet'), colorbar=True, alpha=0.4, figsize=(10,8))
         avaibility.legend()
```

The most significant insights that can be drawn from this scatter plot are:

1. The most expensive Airbnb listings are located in Manhattan. This is likely due to the fact that Manhattan is the most popular tourist destination in New York City and is also home to a number of high-end businesses and attractions.
2. The least expensive Airbnb listings are located in Queens and Brooklyn. These boroughs are less popular with tourists and are also home to a more diverse range of neighborhoods, which means that there are more options for budget-minded travelers.
3. Airbnbs that are located in central Manhattan are generally more expensive than those that are located in more peripheral neighborhoods. This is likely due to the fact that central Manhattan is more convenient for tourists and business travelers.

4. Airbnbs that are available for more nights are generally more expensive than those that are available for fewer nights. This is likely due to the fact that hosts are able to charge a higher price for listings that are in high demand.

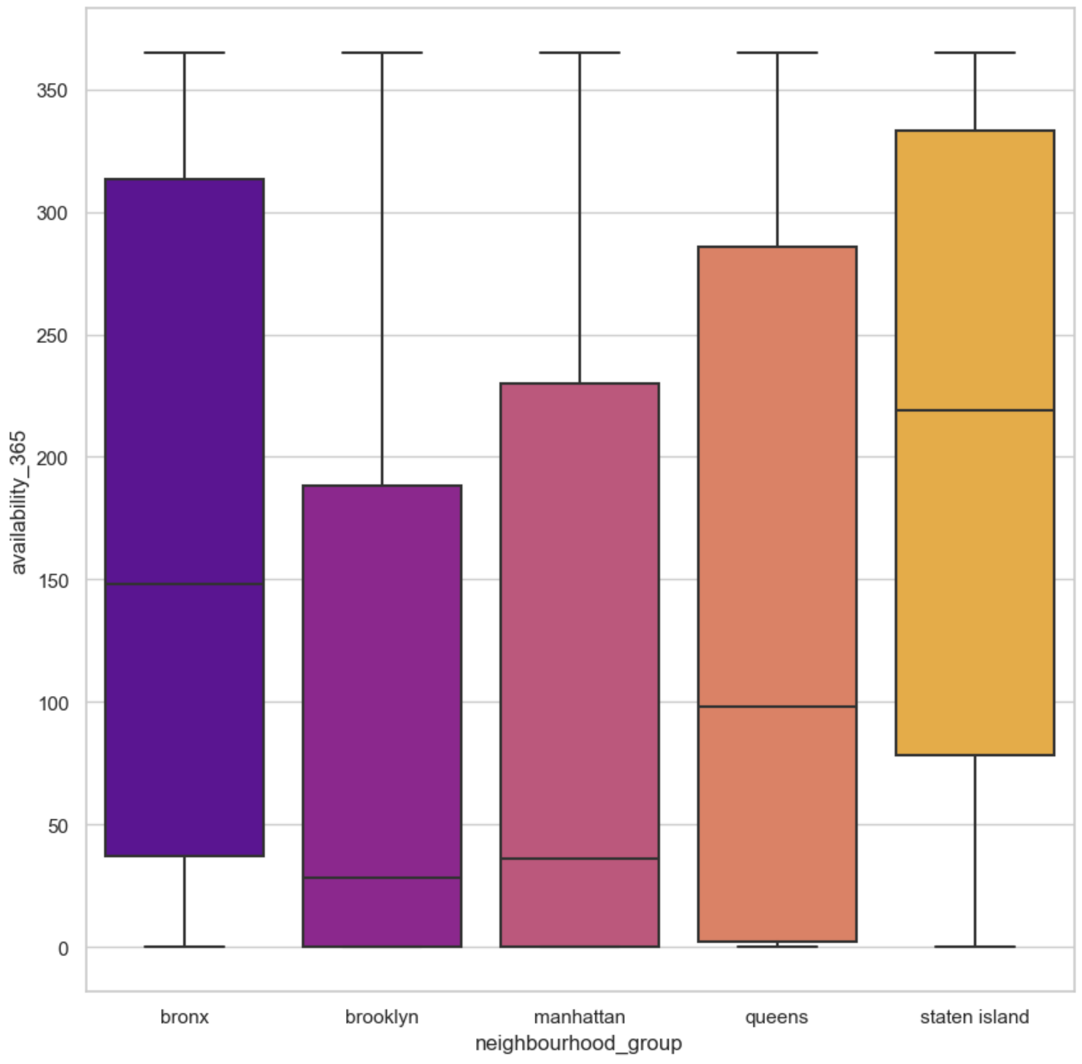


- Boxplot for neighborhood and availability:

```
In [50]: plt.figure(figsize=(10,10))
         ax = sns.boxplot(data=df, x='neighbourhood_group',y='availability_365',palette='plasma')
```

Following are the observations for boxplot of neighborhood and availability:

1. There is a significant difference in the availability of Airbnb listings in different neighborhoods of New York City.
2. Manhattan has the highest median availability, followed by Queens, Brooklyn, and Staten Island. The Bronx has the lowest median availability.
3. The distribution of availability is skewed to the right in all neighborhoods, meaning that there are more listings with lower availability than listings with higher availability.
4. The interquartile range (IQR) is relatively small in Manhattan, indicating that the availability of Airbnb listings is more consistent in this borough than in other boroughs.
5. The IQR is largest in the Bronx, indicating that the availability of Airbnb listings is more variable in this borough than in other boroughs.



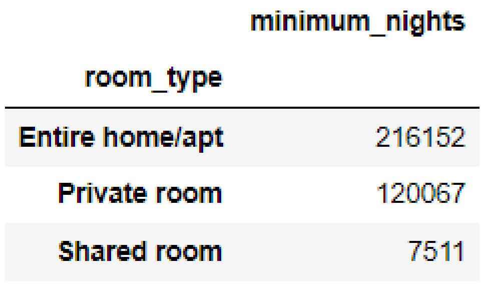
- Key Insights from Minimum Nights Analysis by Room Type:

```
In [52]: df.groupby('room_type').agg( {'minimum_nights': lambda minimum_nights: minimum_nights.sum()})
```

The following are the significant:

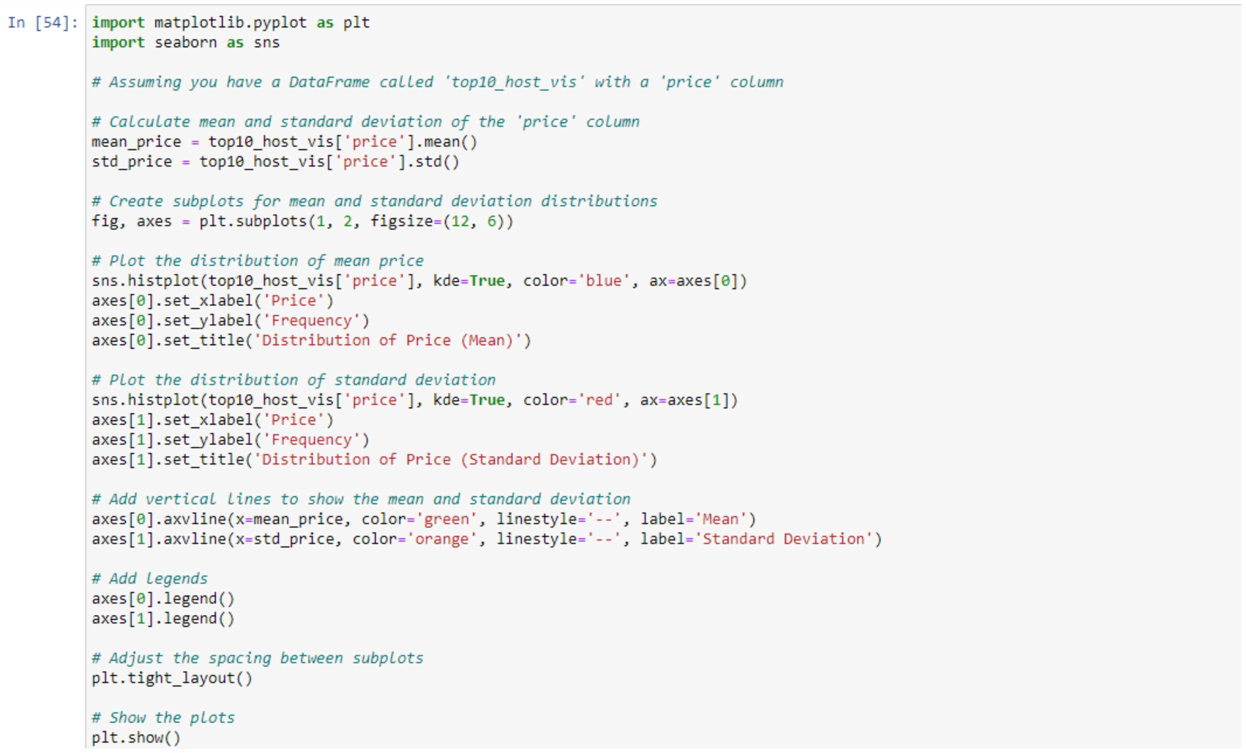
1. Room Type Impact: The analysis reveals that 'Entire home/apartment' listings have the highest total sum of 'minimum_nights,' indicating that, on average, these listings require guests to stay for more nights compared to 'Private room' and 'Shared room' listings. This suggests different expectations for each room type.
2. Diversity in Minimum Nights: While 'Entire home/apartment' listings have the highest total sum of 'minimum_nights,' there is still a significant number of nights required for 'Private room' listings, indicating a wide range of stay duration requirements within this category.

3. Limited Minimum Nights for Shared Rooms: 'Shared room' listings have the lowest total sum of 'minimum_nights,' signifying that they typically offer shorter stays. This highlights the unique nature of shared room listings, often chosen for shorter and more budget-conscious stays.
4. Host and Guest Behavior: The total sum of 'minimum_nights' can offer insights into host and guest behavior and preferences, helping hosts set stay duration requirements and providing guests with an understanding of what to expect when booking different room types.



	minimum_nights
room_type	
Entire home/apt	216152
Private room	120067
Shared room	7511

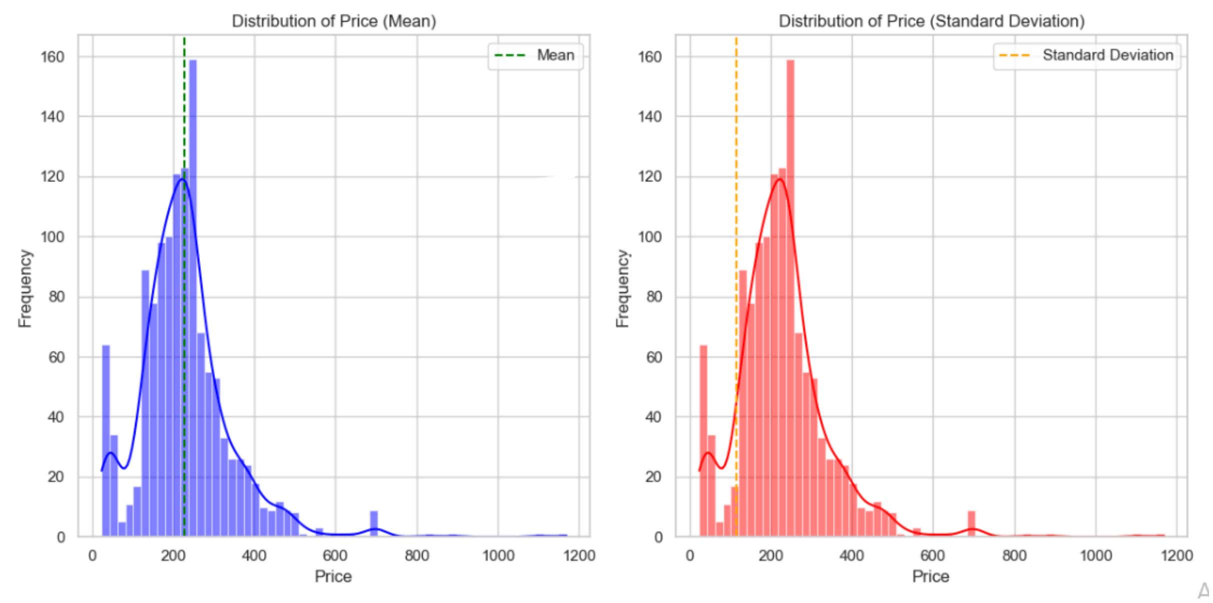
- Histograms for Mean and Standard Deviation:



```
In [54]: import matplotlib.pyplot as plt
         import seaborn as sns

         # Assuming you have a DataFrame called 'top10_host_vis' with a 'price' column

         # Calculate mean and standard deviation of the 'price' column
         mean_price = top10_host_vis['price'].mean()
         std_price = top10_host_vis['price'].std()

         # Create subplots for mean and standard deviation distributions
         fig, axes = plt.subplots(1, 2, figsize=(12, 6))

         # Plot the distribution of mean price
         sns.histplot(top10_host_vis['price'], kde=True, color='blue', ax=axes[0])
         axes[0].set_xlabel('Price')
         axes[0].set_ylabel('Frequency')
         axes[0].set_title('Distribution of Price (Mean)')

         # Plot the distribution of standard deviation
         sns.histplot(top10_host_vis['price'], kde=True, color='red', ax=axes[1])
         axes[1].set_xlabel('Price')
         axes[1].set_ylabel('Frequency')
         axes[1].set_title('Distribution of Price (Standard Deviation)')

         # Add vertical lines to show the mean and standard deviation
         axes[0].axvline(x=mean_price, color='green', linestyle='--', label='Mean')
         axes[1].axvline(x=std_price, color='orange', linestyle='--', label='Standard Deviation')

         # Add legends
         axes[0].legend()
         axes[1].legend()

         # Adjust the spacing between subplots
         plt.tight_layout()

         # Show the plots
         plt.show()
```

Following are the observations gained from the histograms of mean and standard deviation:

1. The distribution of price is skewed towards the mean, while the distribution of price is skewed towards the standard deviation. This suggests that the majority of prices are close to the mean, but there are a few outliers that are much higher or lower than the mean.
2. The standard deviation is relatively high, which suggests that there is a lot of variation in prices. This could be due to a number of factors, such as the location of the property, the amenities it offers, and the time of year.
3. The plots also show that there is a correlation between price and size. This is not surprising, as larger properties are generally more expensive.
4. The price distribution is more skewed for the standard deviation plot than it is for the mean plot. This suggests that there are more outliers in the standard deviation plot, which are likely due to properties that are much larger or smaller than the average.
5. The correlation between price and size is stronger for the standard deviation plot than it is for the mean plot. This suggests that the size of a property is a more important factor in determining its price when the price is significantly above or below the mean.



DASHBOARD:

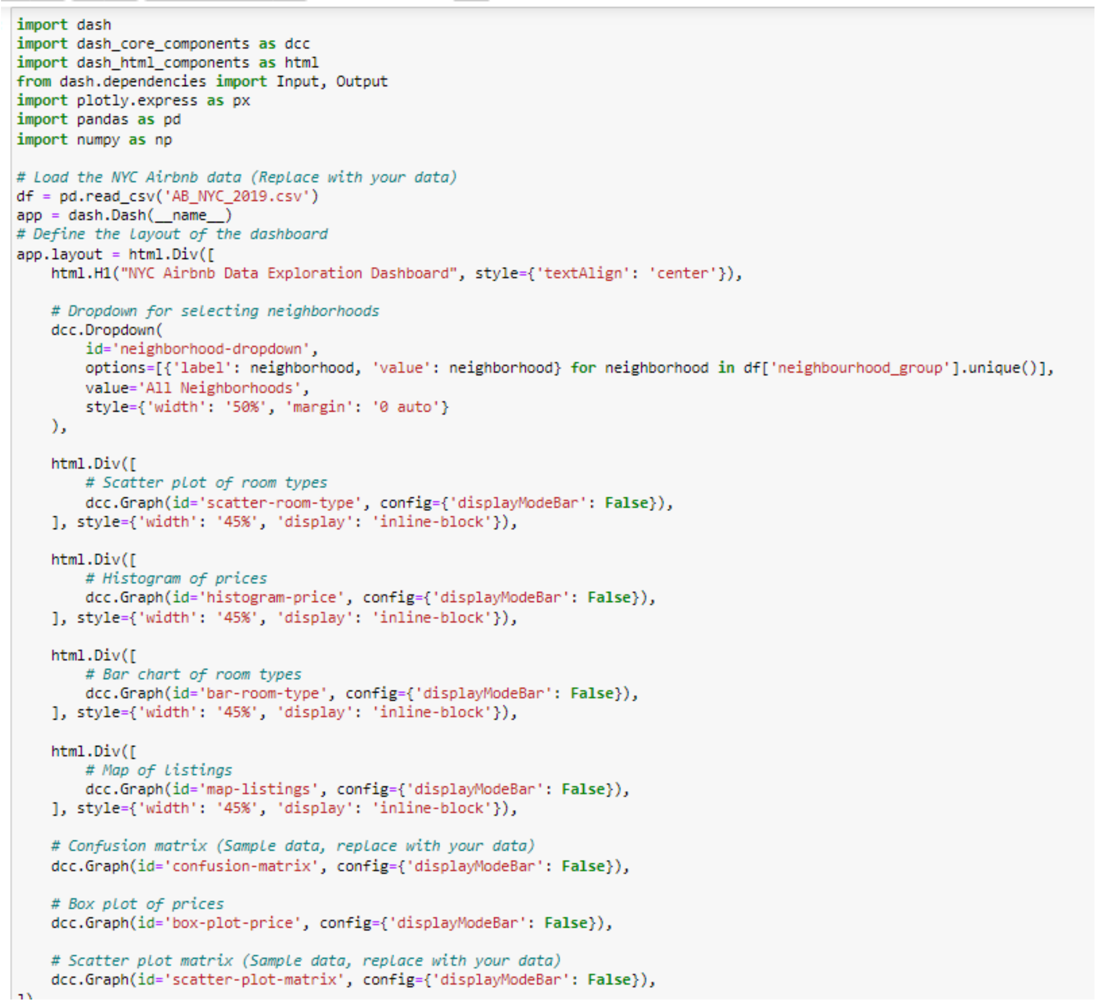
This document presents a Python script that leverages the Dash framework and Plotly Express to construct a web-based dashboard for exploring and visualizing data from the New York City Airbnb dataset. This script loads the dataset, offers user-friendly controls, and presents a range of visualizations, including scatter plots, histograms, bar charts, and maps, to help users gain insights into the data. Whether you're a data analyst, a data scientist, or just someone curious about the world of Airbnb listings in New York City, this dashboard provides a valuable means of interactively exploring and understanding the dataset. Let's dive into the details of this script to see how it all comes together.

Highlights:

Neighborhoods: Pick a neighborhood to see its Airbnb details.
Room Types: Understand the room types, prices, and locations.
Pricing: Explore the distribution of rental prices.
Room Type Counts: Get a count of each room type.
Map View: Visualize listing locations on an interactive map.
Price Trends: Compare prices across different room types.
Scatter Plot Matrix: Discover variable relationships.

Code:

Imported the required libraries and modules and we also installed Dash, Plotly Express, pandas, and numpy.
Loaded our dataset (NYC Airbnb data) into a Pandas DataFrame named 'df'.
Defined the dashboard's layout, which includes an H1 heading, a neighborhood dropdown for selection, and various graphs for data visualization.



```python
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd
import numpy as np

# Load the NYC Airbnb data (Replace with your data)
df = pd.read_csv('AB_NYC_2019.csv')
app = dash.Dash(__name__)
# Define the layout of the dashboard
app.layout = html.Div([
    html.H1("NYC Airbnb Data Exploration Dashboard", style={'textAlign': 'center'}),

    # Dropdown for selecting neighborhoods
    dcc.Dropdown(
        id='neighborhood-dropdown',
        options=[{'label': neighborhood, 'value': neighborhood} for neighborhood in df['neighbourhood_group'].unique()],
        value='All Neighborhoods',
        style={'width': '50%', 'margin': '0 auto'}
    ),

    html.Div([
        # Scatter plot of room types
        dcc.Graph(id='scatter-room-type', config={'displayModeBar': False}),
    ], style={'width': '45%', 'display': 'inline-block'}),

    html.Div([
        # Histogram of prices
        dcc.Graph(id='histogram-price', config={'displayModeBar': False}),
    ], style={'width': '45%', 'display': 'inline-block'}),

    html.Div([
        # Bar chart of room types
        dcc.Graph(id='bar-room-type', config={'displayModeBar': False}),
    ], style={'width': '45%', 'display': 'inline-block'}),

    html.Div([
        # Map of listings
        dcc.Graph(id='map-listings', config={'displayModeBar': False}),
    ], style={'width': '45%', 'display': 'inline-block'}),

    # Confusion matrix (Sample data, replace with your data)
    dcc.Graph(id='confusion-matrix', config={'displayModeBar': False}),

    # Box plot of prices
    dcc.Graph(id='box-plot-price', config={'displayModeBar': False}),

    # Scatter plot matrix (Sample data, replace with your data)
    dcc.Graph(id='scatter-plot-matrix', config={'displayModeBar': False}),
```

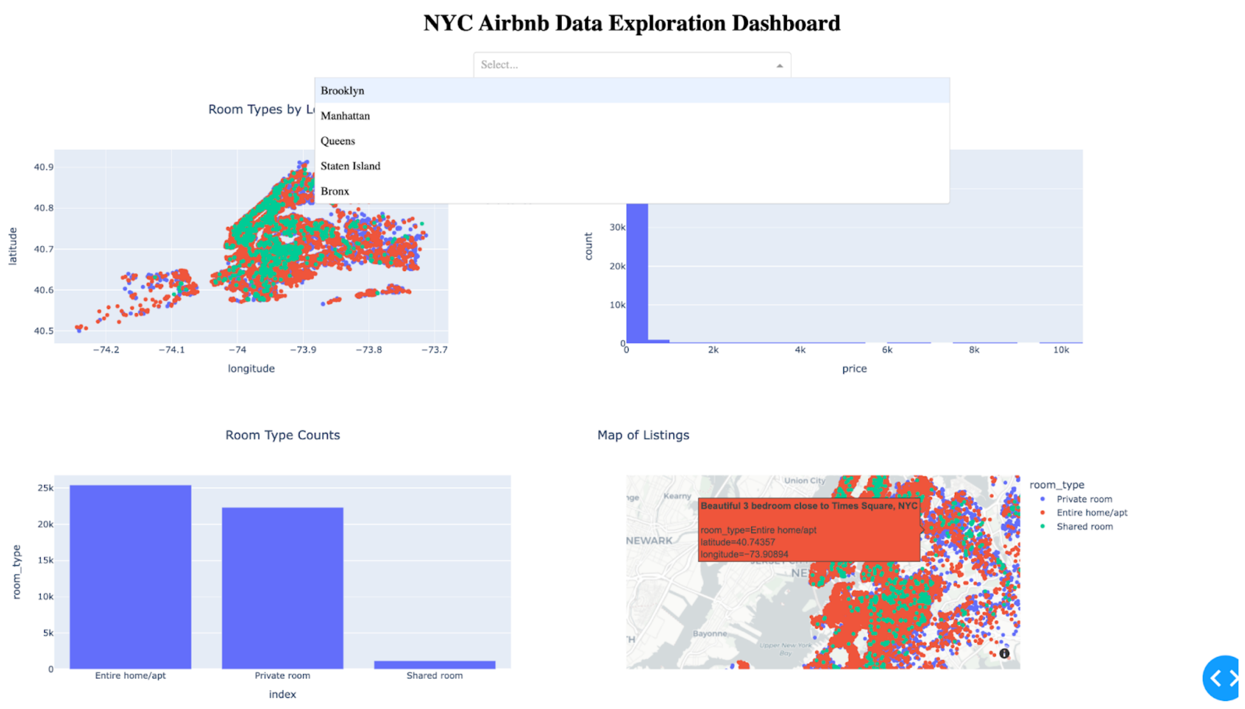
Created callback functions to update the graphs based on the selected neighborhood.

```
# Define callback functions to update the components
@app.callback(
    [Output('scatter-room-type', 'figure'), Output('histogram-price', 'figure'), Output('bar-room-type', 'figure'),
     Output('map-listings', 'figure'), Output('confusion-matrix', 'figure'), Output('box-plot-price', 'figure'),
     Output('scatter-plot-matrix', 'figure')],
    [Input('neighborhood-dropdown', 'value')]
)
def update_graphs(selected_neighbourhood):
    if selected_neighbourhood == 'All Neighborhoods':
        filtered_df = df
    else:
        filtered_df = df[df['neighbourhood_group'] == selected_neighbourhood]

    # Scatter plot of room types
    scatter_room_type = px.scatter(filtered_df, x='longitude', y='latitude', color='room_type', hover_data=['name'], title='Room
    scatter_room_type.update_layout(title_x=0.5)

    # Histogram of prices
    histogram_price = px.histogram(filtered_df, x='price', nbins=30, title='Price Distribution')
    histogram_price.update_layout(title_x=0.5)

    # Bar chart of room types
    bar_room_type = px.bar(filtered_df['room_type'].value_counts().reset_index(), x='index', y='room_type', title='Room Type Cou
    bar_room_type.update_layout(title_x=0.5)

    # Map of listings
    map_listings = px.scatter_mapbox(filtered_df, lat='latitude', lon='longitude', color='room_type', hover_name='name', title='
                                     mapbox_style='open-street-map')
    map_listings.update_layout(mapbox_style="carto-positron", mapbox_zoom=10, mapbox_center={"lat": 40.7128, "lon": -74.0060})

    # Sample confusion matrix (Replace with your data)
    confusion_matrix = np.array([[50, 10], [5, 80]])  # Example confusion matrix

    # Create a heatmap for the confusion matrix
    confusion_matrix_fig = px.imshow(confusion_matrix, labels=dict(x="Predicted", y="Actual"), x=['A', 'B'], y=['A', 'B'],
                                     color_continuous_scale='Blues', color_continuous_midpoint=np.average(confusion_matrix))
    confusion_matrix_fig.update_xaxes(side="top")

    # Box plot of prices
    box_plot_price = px.box(filtered_df, x='room_type', y='price', title='Price Distribution by Room Type')
    box_plot_price.update_layout(title_x=0.5)

    # Sample scatter plot matrix (Replace with your data)
    scatter_plot_matrix = px.scatter_matrix(filtered_df, dimensions=['price', 'minimum_nights', 'number_of_reviews'], title='Scat

    return scatter_room_type, histogram_price, bar_room_type, map_listings, confusion_matrix_fig, box_plot_price, scatter_plot_ma
```

Run the dashboard with the command app.run_server(debug=True) to start hosting it locally. We can access it by opening a web browser and navigating to "http://localhost:8050" (or the port we specified during the run command).

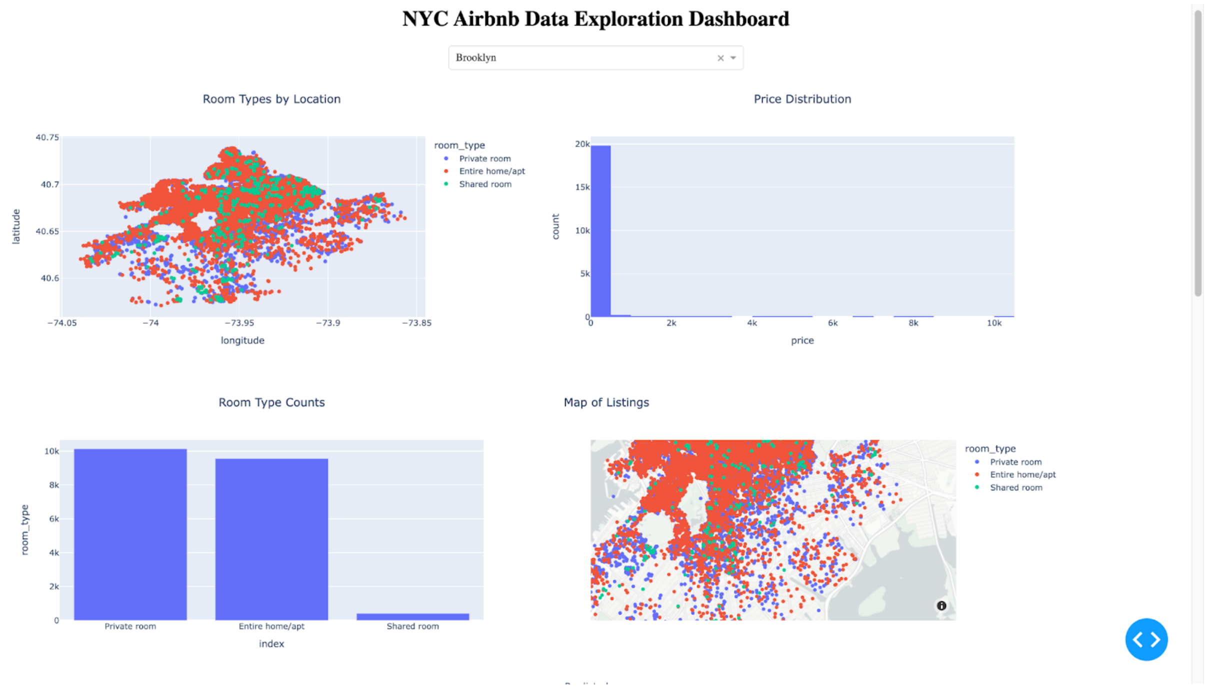
```
if __name__ == '__main__':
    app.run_server(debug=True)
```

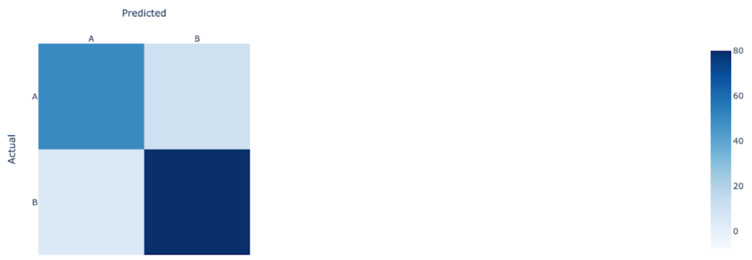
The dashboard includes analysis for each of the areas present in New York City mentioned in the dataset. These Graphs can be viewed for every neighbour_hood group from the drop down menu as below

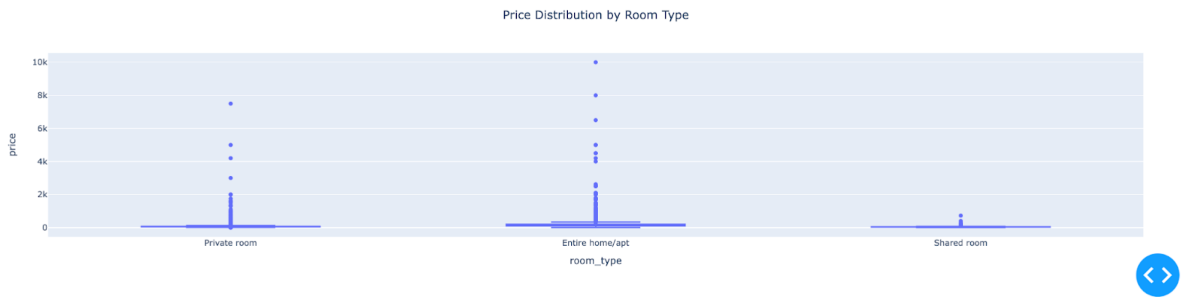


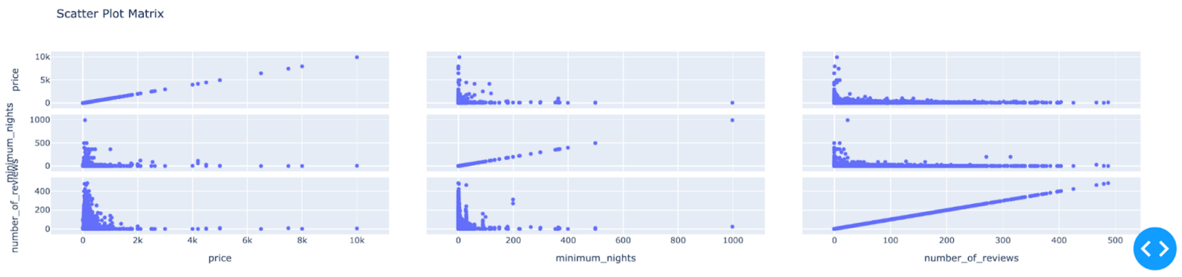
This is how our dynamic dashboard looks like upon selecting city Brooklyn

Following are the images for one of the areas mentioned in the drop down. Below includes images for Brooklyn:









References:

1. https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/
2. https://www.kaggle.com/datasets
3. https://numpy.org/
4. https://pandas.pydata.org/
5. https://matplotlib.org/
6. https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm