Analytical Report

Insights from Airbnb Rentals in New York City

1. Introduction

This report aims to explore the results of an analysis conducted on New York City Airbnb rental data from 2019, in which three key business questions were answered through in-depth exploratory data analysis (EDA) and the use of machine learning (ML). The insights gained will also be discussed in this report, as well as recommendations. For reproducibility, a simplified version of the CRISP-DM process was used in this analysis.

2. Data understanding

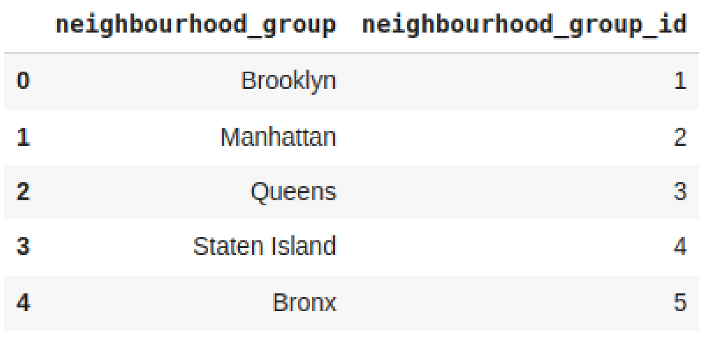
The "New York City Airbnb Open Data" dataset was used in this analysis, which offers a snapshot of Airbnb listings and metrics in New York City in 2019. It is available at https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data.

This dataset contains 16 columns, with 10 of them being numerical (notably, 2 of them are ID-based) and 6 being categorical. This dataset covers diverse aspects of each listing, such as host name, latitude and longitude, reviews per month, room type, neighbourhood, rental availability in the next year and more, providing a solid basis for understanding the city’s Airbnb market.

3. Data preparation

An initial analysis of this dataset revealed a total of 10,074 null rows (20.6% of all records) that were concentrated in the columns that represented the rental name, host name, last review and reviews per month. As removing these rows from the dataset would have resulted in data loss, it was decided to fill those null values with appropriate replacements, such as “Unknown” and “Never” for null names and dates.

The next step involved the transformation of the categorical columns. Conducting numerical operations on text data is not very efficient, which led to the creation of a mapping structure to assign numerical values to categorical data.



	neighbourhood_group	neighbourhood_group_id
0	Brooklyn	1
1	Manhattan	2
2	Queens	3
3	Staten Island	4
4	Bronx	5

Figure 1: The mapping process applied to the neighbourhood group column

4. **Exploratory data analysis**

Considering the scope of the data, uncovering rental patterns would be of interest to Airbnb. As such, the following three guiding questions were posed:

- How does location and room type affect the cost of the rental?
- How does availability over the next year correlate to room type and location?
- What is the distribution of listings between hosts and property types?

The first step to determine underlying relationships in the data was to calculate the dataset’s correlation. Before doing so, it was important to determine whether the data followed a normal distribution, given correlation methods depend on this property.

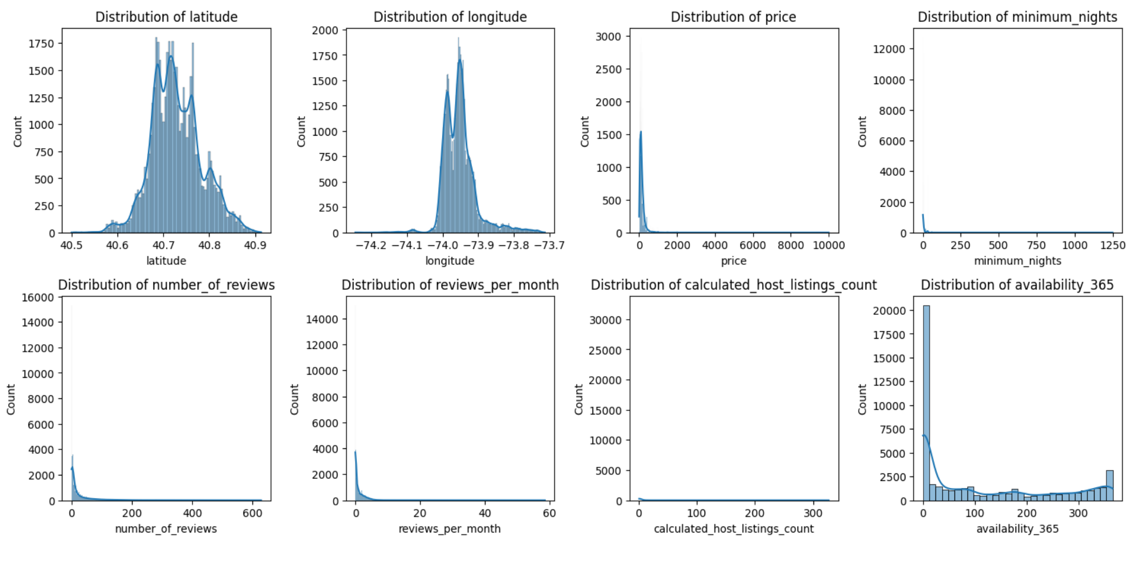


Figure 2: Numerical data distribution (excluding ID data)

As seen in the figure, the data did not follow a normal distribution, which meant that correlation methods like Pearson’s correlation were unsuited to this data (Schober et al., 2018), thus necessitating a different approach.

4.1 **Correlation analysis**

Point-biserial correlation was used, which is the value of Pearson’s product correlation when one of the variables is dichotomous (i.e., a categorical variable with two possible values) and the other variable is metric (Kornbrot, 2005). Before applying it, the categorical columns were first encoded via one-hot encoding, which converts them into binary features.

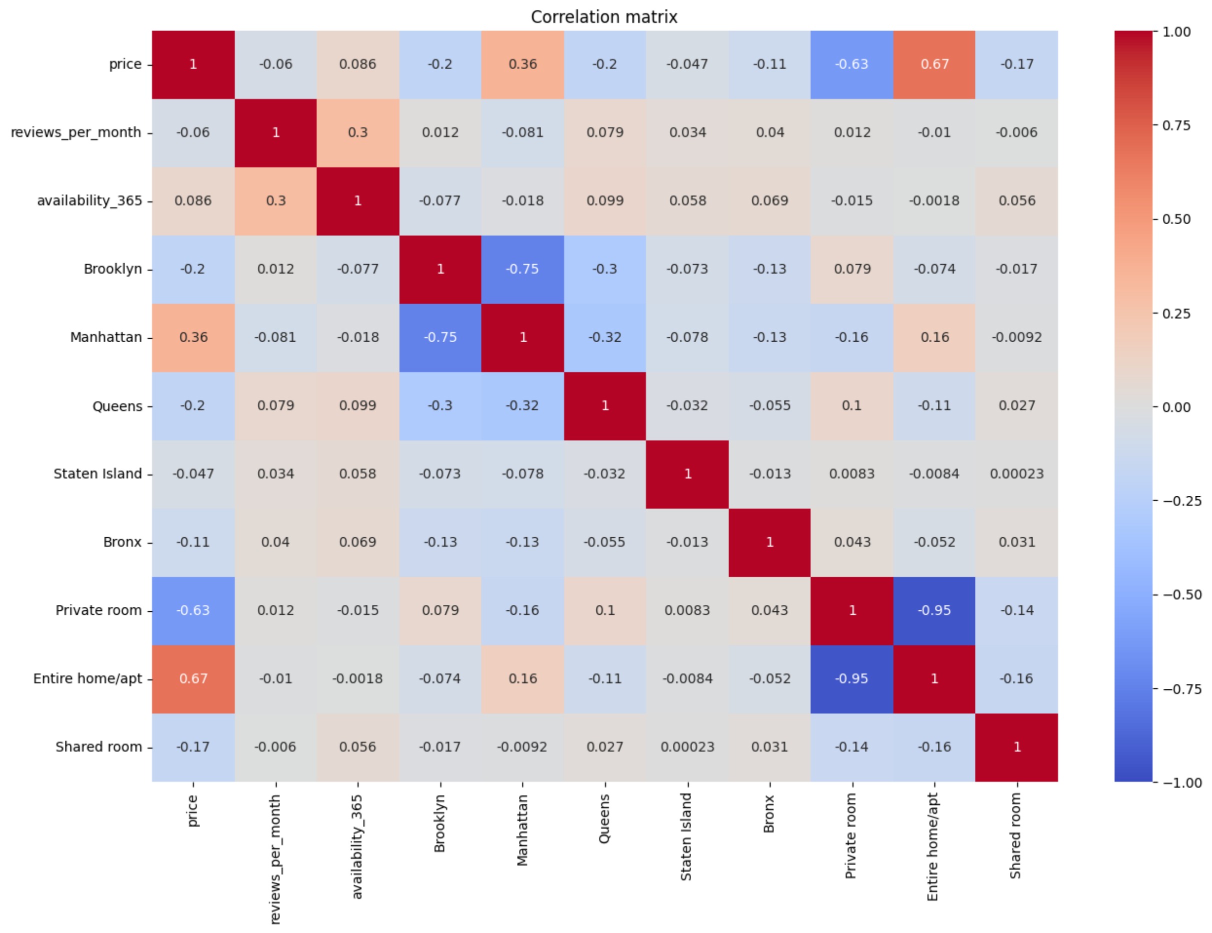


Figure 3: Correlation matrix, showing the various relationships between the dataset's neighbourhood groups, price and room types

The most important records for this analysis were focused upon. Price is seen to correlate with all neighbourhood groups and has a particularly strong positive relationship with Manhattan. It also has a correlation with all rental types, thus hinting that these variables share a mutual relationship.

4.2 **Rental patterns**

The first insights were gained on rental prices and its influencing variables. Using a visual approach was instrumental in evidencing the relationships shown in the correlation analysis.

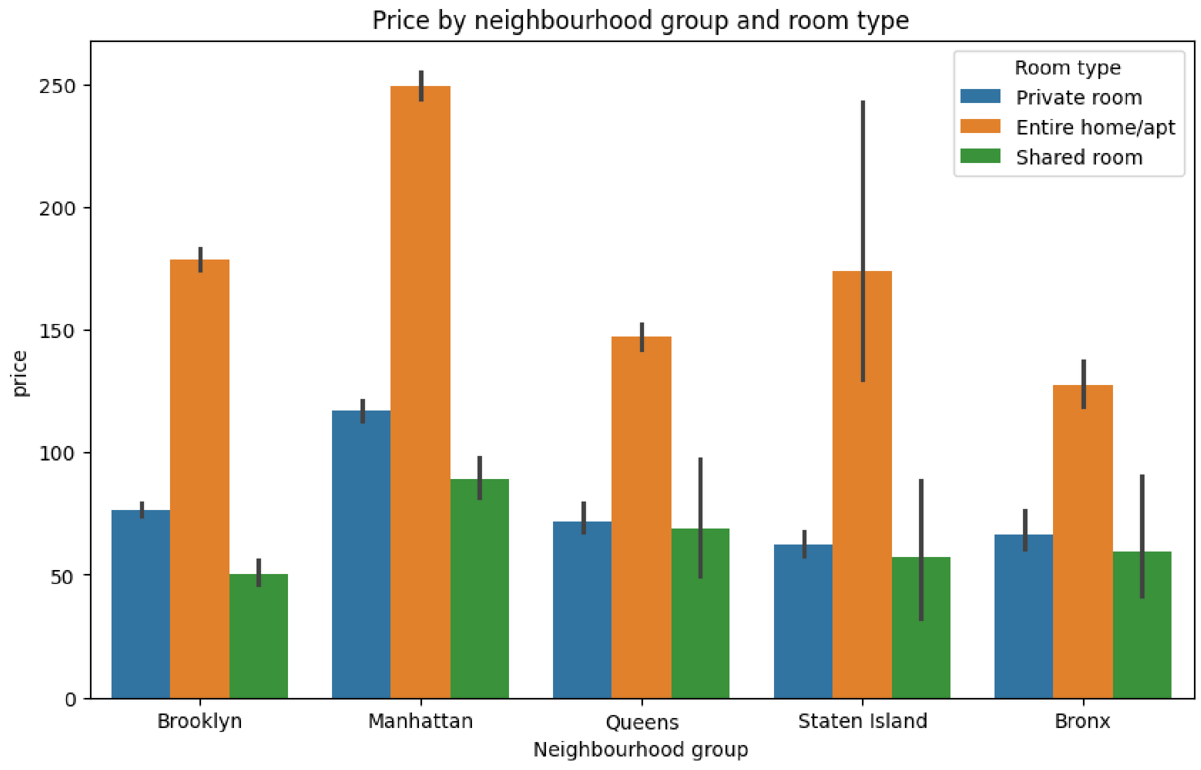


Figure 4: Rental prices by neighbourhood group and room type

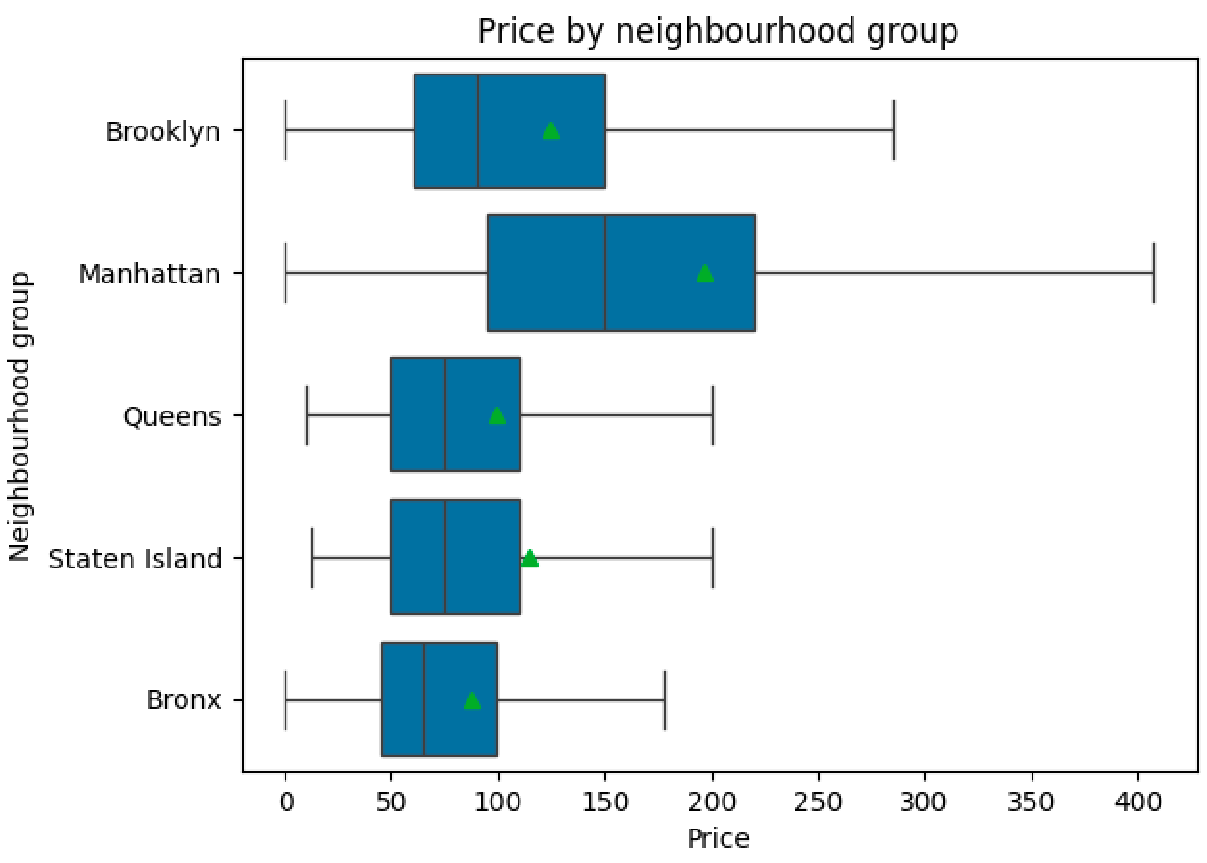


Figure 5: A box plot showing rental prices by neighbourhood group and without outliers

The figures above evidence the role that neighbourhood group and rental type have in influencing rental prices, with entire home/apartment having the highest prices across all types and Manhattan being the most expensive location out of all neighbourhood groups.

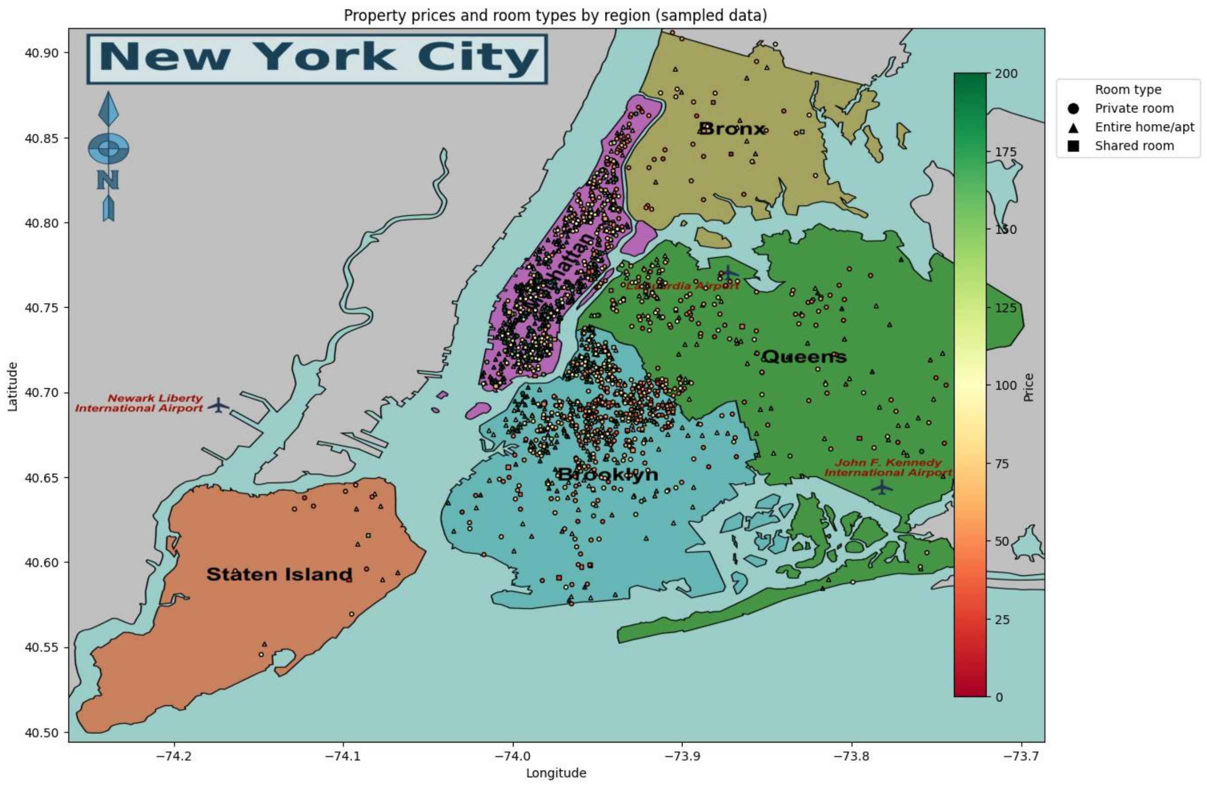


Figure 6: A map of New York City showing a sample of two thousand rentals. The high concentration of rentals in Manhattan (top) and Brooklyn (centre) is apparent, as is the presence of expensive options (shaded in green) (adapted from NYC Map 360º, N.D.)

Shifting the focus to availability over the next year (the “availability_365” column), the correlation analysis suggested that room type and neighbourhood group play a role in influencing this variable. Once more, a visual approach evidenced these relationships.

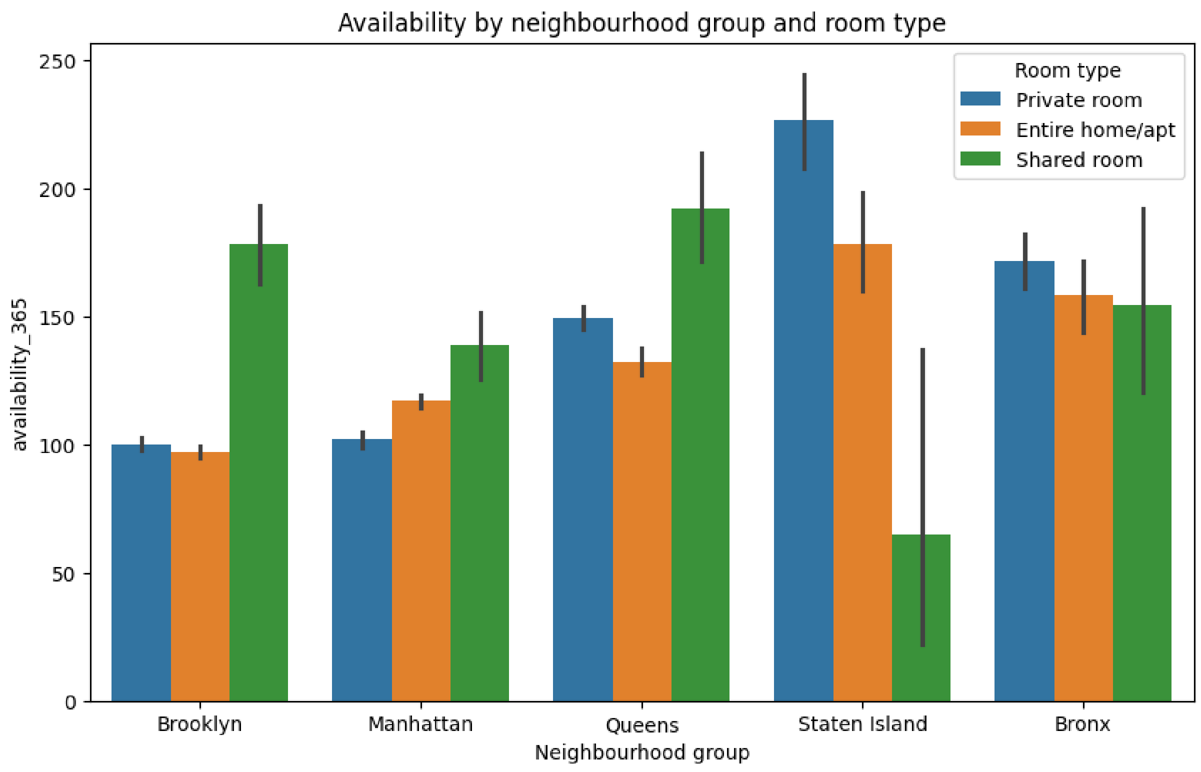


Figure 7: Rental availability by neighbourhood group and room type

General availability in Manhattan and Brooklyn over the next year is lower than in other locations. Interestingly, some outliers seem to indicate the role that other circumstances, such as price, play in availability.

Using a clustering approach, it's possible to group the data based on availability, room type and neighbourhood group. The following clusters have been defined for this task:

- **Cluster 0 (purple):** listings which are likely fully booked or have limited availability, possibly being the most popular;
- **Cluster 1 (blue):** listings with moderate availability, suggesting these are booked regularly but still have open days**;**

- **Cluster 2 (green):** listings with generally higher availability, such as those with low demand or used seasonally;
- **Cluster 3 (yellow):** listings with highest availability, representing those that are rarely booked or are consistently available, possibly due to premium prices.

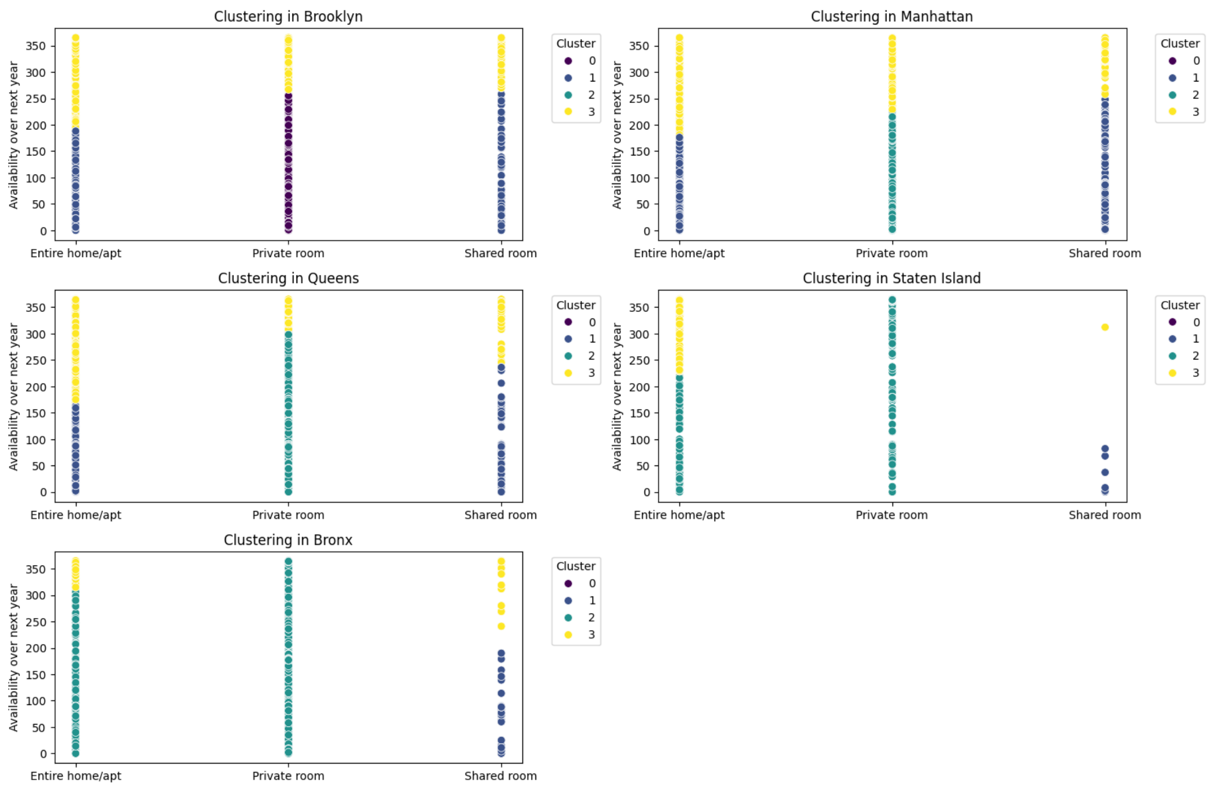


Figure 8: Clustering in various neighbourhood groups by availability over the next year and room type

This analysis suggests that Brooklyn and Queens tend to have the highest demand for shared rooms and private rooms, while Staten Island and the Bronx generally have lower demand, particularly for entire homes. Manhattan shows a mixed pattern, with higher availability for entire homes, possibly due to higher prices.

When evaluating the distribution of listings between hosts and property types, visualisation can once more be resorted to as a reliable and effective method to generate insights.

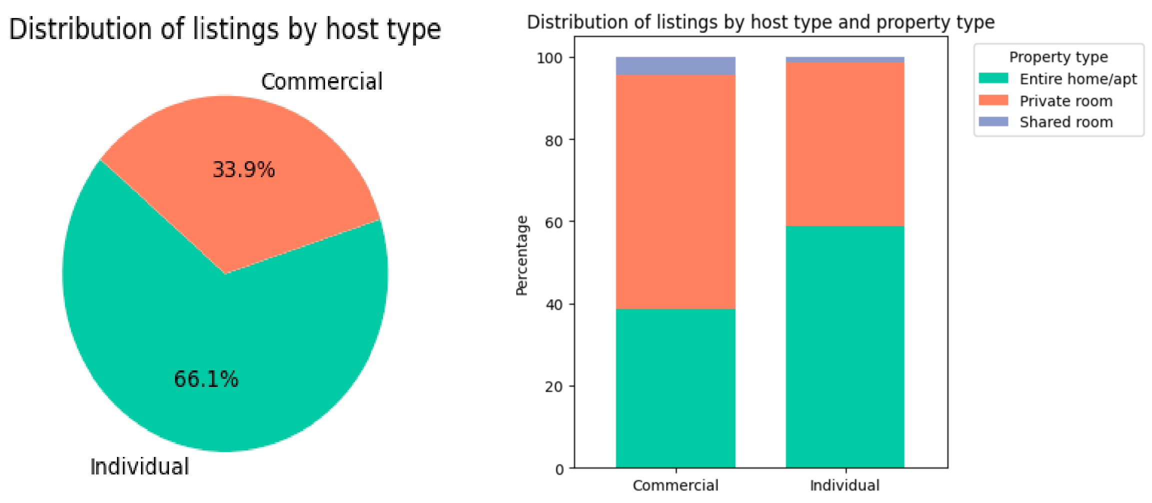


Figure 9: Distribution of listings by host and property types

As shown in the figure, individual hosts manage most listings (66.1%), while commercial hosts account for about one-third. Among property types, entire homes/apartments and private rooms dominate, with shared rooms being rare.

5. Conclusion and recommendations

The analysis shows location and room type drive rental prices, with Manhattan commanding the highest rates and Brooklyn and Queens sustaining demand for private and shared rooms. Listings in Manhattan could benefit from competitive pricing to improve occupancy rates. Budget-focused marketing in Brooklyn and

Queens and dynamic pricing across neighbourhoods can optimize performance in New York City's rental market.

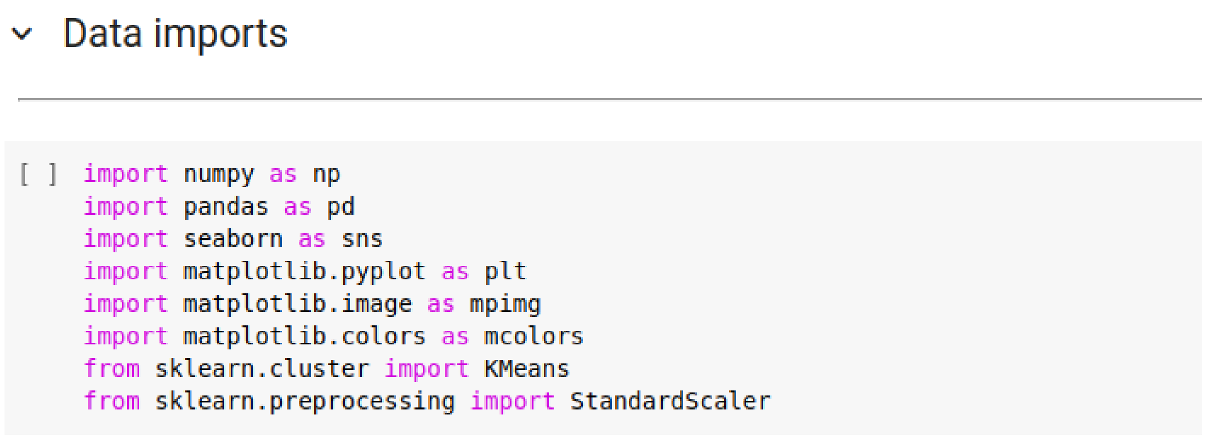
References

Schober, P, Boer, C. & Schwarte, L.A. (2018) Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & Analgesia* 126(5): 1763-1768. DOI: https://doi.org/10.1213/ANE.0000000000002864.

Kornbrot, D. (2014) 'Point Biserial Correlation', in: N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri & J.L. Teugels (eds). *Wiley StatsRef: Statistics Reference Online.* Hoboken, New Jersey: John Wiley & Sons, Inc. 1-2.

NYC Map 360º. (N.D.) New York City Boroughs & Neighborhoods Map. Available from: https://nycmap360.com/nyc-boroughs-map (Accessed 27 November 2024).

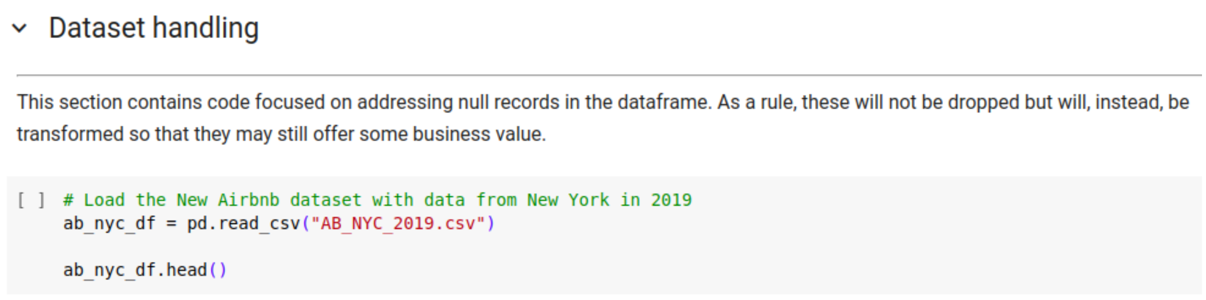
Appendix



Data imports

```
[ ] import numpy as np
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg
    import matplotlib.colors as mcolors
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import StandardScaler
```

Figure 10: Importing necessary packages

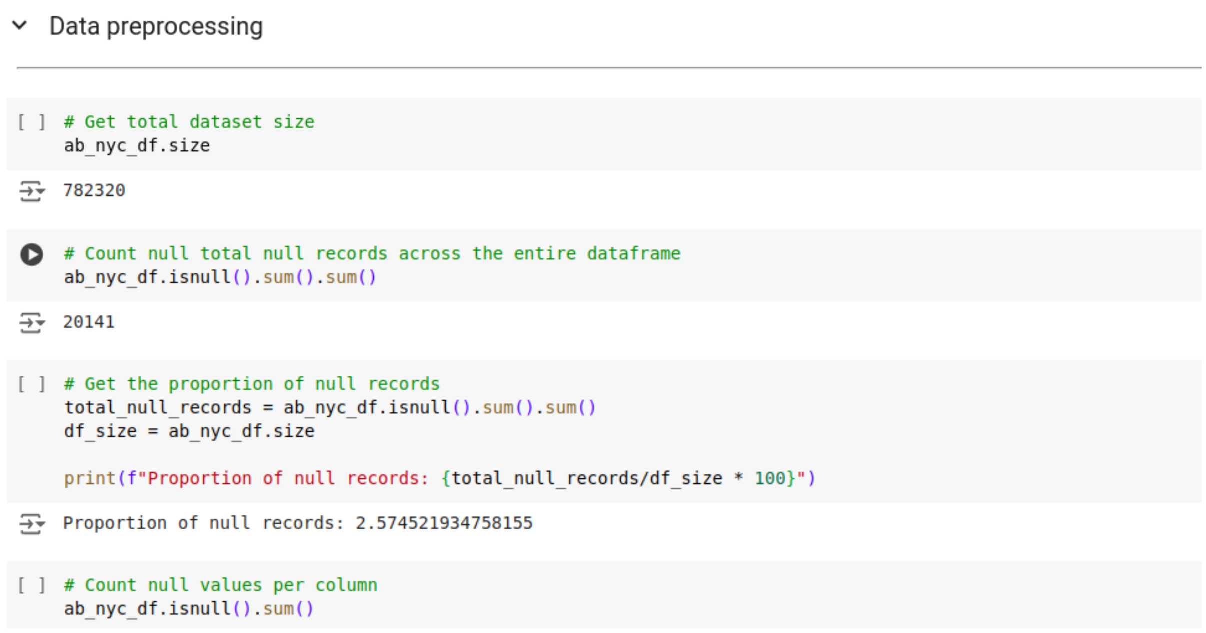


Dataset handling

This section contains code focused on addressing null records in the dataframe. As a rule, these will not be dropped but will, instead, be transformed so that they may still offer some business value.

```
[ ] # Load the New Airbnb dataset with data from New York in 2019
    ab_nyc_df = pd.read_csv("AB_NYC_2019.csv")

    ab_nyc_df.head()
```

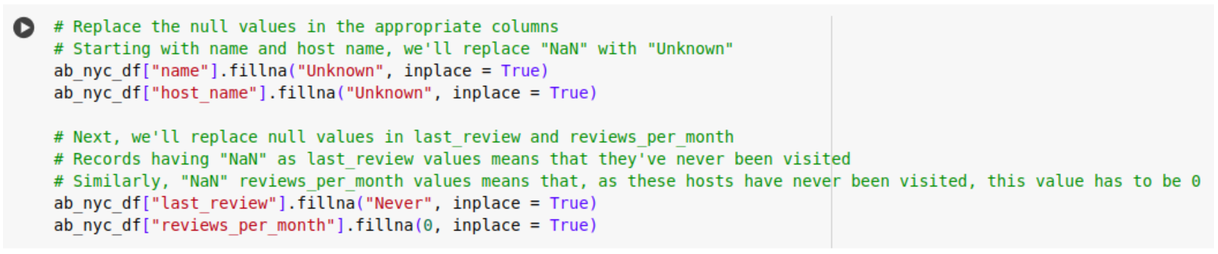
Figure 11: Reading the 2019 New York City Airbnb rentals dataset



Data preprocessing

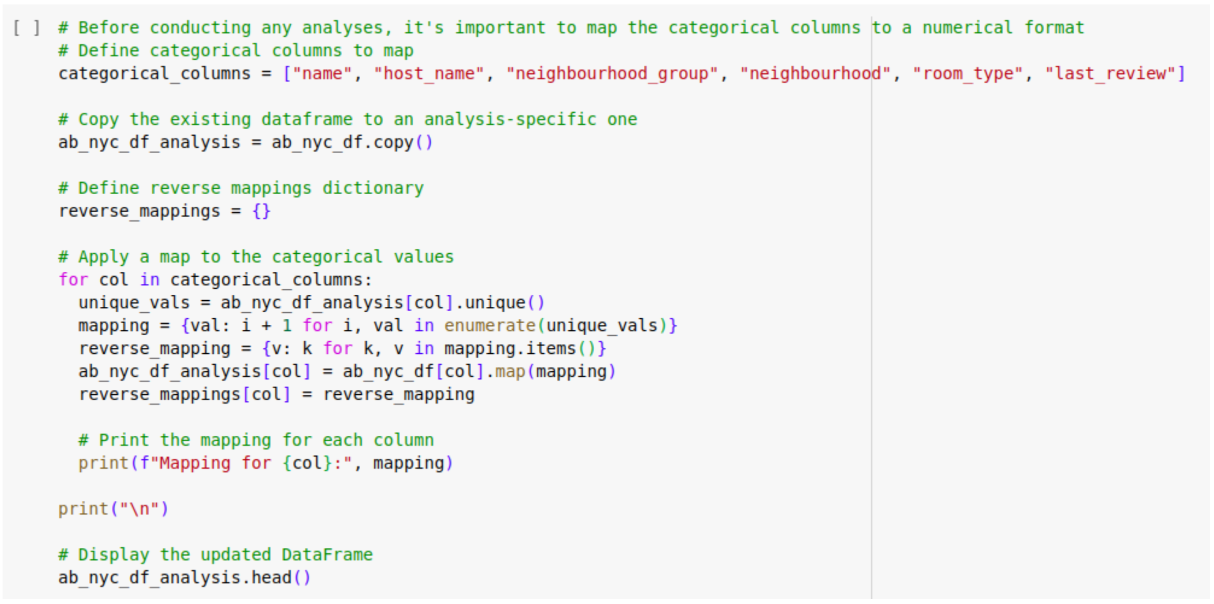
```
[ ] # Get total dataset size
    ab_nyc_df.size

    782320

    # Count null total null records across the entire dataframe
    ab_nyc_df.isnull().sum().sum()

    20141

[ ] # Get the proportion of null records
    total_null_records = ab_nyc_df.isnull().sum().sum()
    df_size = ab_nyc_df.size

    print(f"Proportion of null records: {total_null_records/df_size * 100}")

    Proportion of null records: 2.574521934758155

[ ] # Count null values per column
    ab_nyc_df.isnull().sum()
```

Figure 12: Checking the number and proportion of null records in the dataset



```python
# Replace the null values in the appropriate columns
# Starting with name and host name, we'll replace "NaN" with "Unknown"
ab_nyc_df["name"].fillna("Unknown", inplace = True)
ab_nyc_df["host_name"].fillna("Unknown", inplace = True)

# Next, we'll replace null values in last_review and reviews_per_month
# Records having "NaN" as last_review values means that they've never been visited
# Similarly, "NaN" reviews_per_month values means that, as these hosts have never been visited, this value has to be 0
ab_nyc_df["last_review"].fillna("Never", inplace = True)
ab_nyc_df["reviews_per_month"].fillna(0, inplace = True)
```

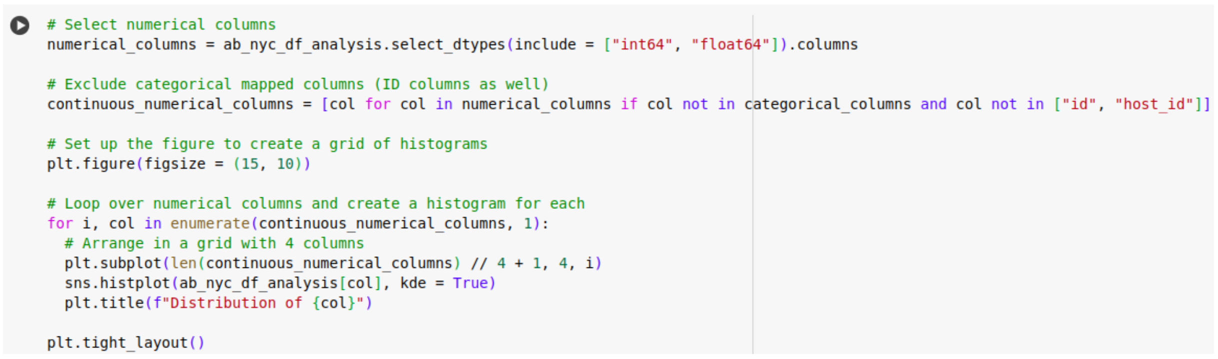
Figure 13: Replacing null data with appropriate values



```python
# Before conducting any analyses, it's important to map the categorical columns to a numerical format
# Define categorical columns to map
categorical_columns = ["name", "host_name", "neighbourhood_group", "neighbourhood", "room_type", "last_review"]

# Copy the existing dataframe to an analysis-specific one
ab_nyc_df_analysis = ab_nyc_df.copy()

# Define reverse mappings dictionary
reverse_mappings = {}

# Apply a map to the categorical values
for col in categorical_columns:
  unique_vals = ab_nyc_df_analysis[col].unique()
  mapping = {val: i + 1 for i, val in enumerate(unique_vals)}
  reverse_mapping = {v: k for k, v in mapping.items()}
  ab_nyc_df_analysis[col] = ab_nyc_df[col].map(mapping)
  reverse_mappings[col] = reverse_mapping

  # Print the mapping for each column
  print(f"Mapping for {col}:", mapping)

print("\n")

# Display the updated DataFrame
ab_nyc_df_analysis.head()
```

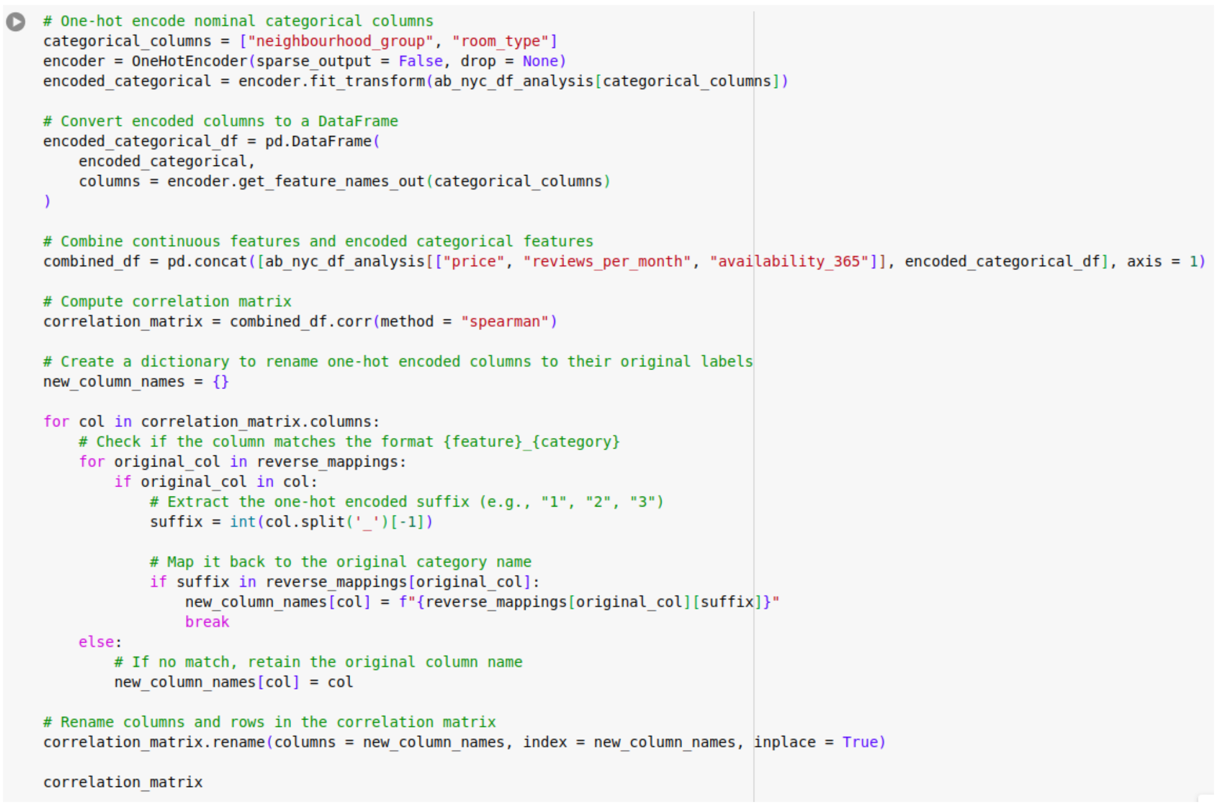
Figure 14: Mapping categorical columns to a numerical format

Distribution analysis



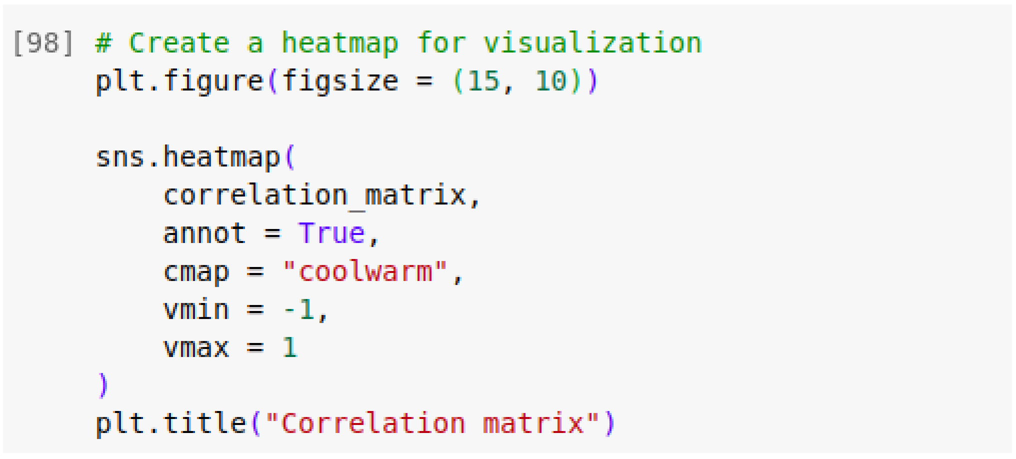
```python
# Select numerical columns
numerical_columns = ab_nyc_df_analysis.select_dtypes(include = ["int64", "float64"]).columns

# Exclude categorical mapped columns (ID columns as well)
continuous_numerical_columns = [col for col in numerical_columns if col not in categorical_columns and col not in ["id", "host_id"]]

# Set up the figure to create a grid of histograms
plt.figure(figsize = (15, 10))

# Loop over numerical columns and create a histogram for each
for i, col in enumerate(continuous_numerical_columns, 1):
  # Arrange in a grid with 4 columns
  plt.subplot(len(continuous_numerical_columns) // 4 + 1, 4, i)
  sns.histplot(ab_nyc_df_analysis[col], kde = True)
  plt.title(f"Distribution of {col}")

plt.tight_layout()
```

Figure 15: Creating a plot for data distribution analysis



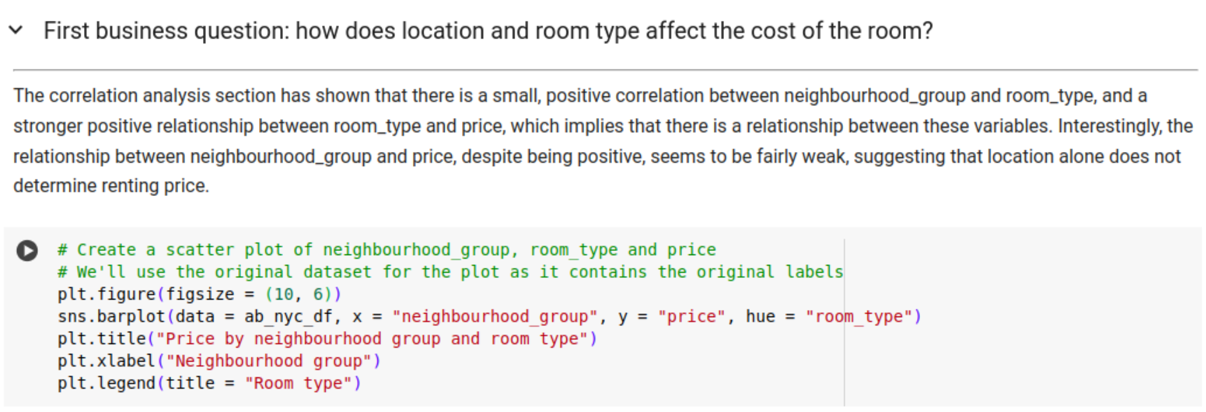
```python
# One-hot encode nominal categorical columns
categorical_columns = ["neighbourhood_group", "room_type"]
encoder = OneHotEncoder(sparse_output = False, drop = None)
encoded_categorical = encoder.fit_transform(ab_nyc_df_analysis[categorical_columns])

# Convert encoded columns to a DataFrame
encoded_categorical_df = pd.DataFrame(
    encoded_categorical,
    columns = encoder.get_feature_names_out(categorical_columns)
)

# Combine continuous features and encoded categorical features
combined_df = pd.concat([ab_nyc_df_analysis[["price", "reviews_per_month", "availability_365"]], encoded_categorical_df], axis = 1)

# Compute correlation matrix
correlation_matrix = combined_df.corr(method = "spearman")

# Create a dictionary to rename one-hot encoded columns to their original labels
new_column_names = {}

for col in correlation_matrix.columns:
    # Check if the column matches the format {feature}_{category}
    for original_col in reverse_mappings:
        if original_col in col:
            # Extract the one-hot encoded suffix (e.g., "1", "2", "3")
            suffix = int(col.split('_')[-1])

            # Map it back to the original category name
            if suffix in reverse_mappings[original_col]:
                new_column_names[col] = f"{reverse_mappings[original_col][suffix]}"
                break
    else:
        # If no match, retain the original column name
        new_column_names[col] = col

# Rename columns and rows in the correlation matrix
correlation_matrix.rename(columns = new_column_names, index = new_column_names, inplace = True)

correlation_matrix
```

Figure 16: Creating a correlation matrix



```python
[98] # Create a heatmap for visualization
     plt.figure(figsize = (15, 10))

     sns.heatmap(
         correlation_matrix,
         annot = True,
         cmap = "coolwarm",
         vmin = -1,
         vmax = 1
     )
     plt.title("Correlation matrix")
```

Figure 17: Plotting the correlation matrix

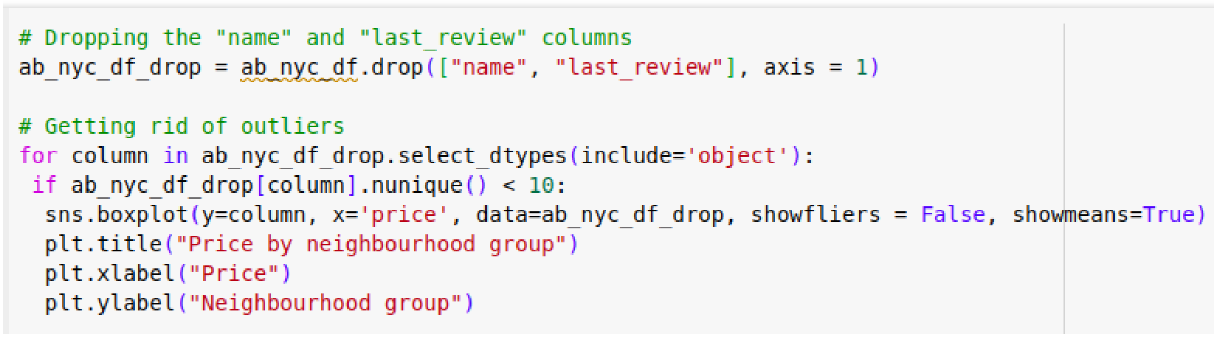


First business question: how does location and room type affect the cost of the room?

The correlation analysis section has shown that there is a small, positive correlation between neighbourhood_group and room_type, and a stronger positive relationship between room_type and price, which implies that there is a relationship between these variables. Interestingly, the relationship between neighbourhood_group and price, despite being positive, seems to be fairly weak, suggesting that location alone does not determine renting price.

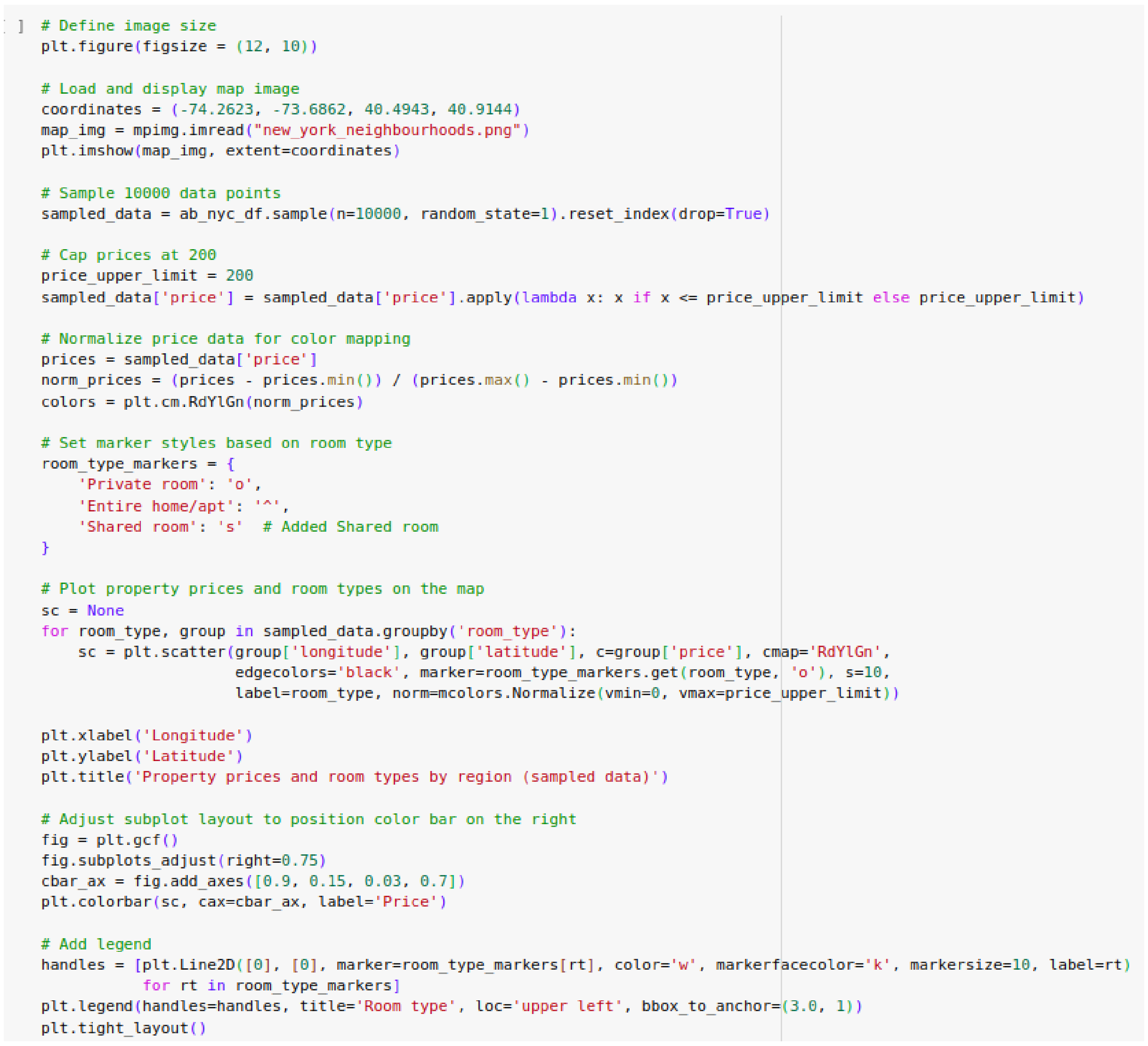
```python
# Create a scatter plot of neighbourhood_group, room_type and price
# We'll use the original dataset for the plot as it contains the original labels
plt.figure(figsize = (10, 6))
sns.barplot(data = ab_nyc_df, x = "neighbourhood_group", y = "price", hue = "room_type")
plt.title("Price by neighbourhood group and room type")
plt.xlabel("Neighbourhood group")
plt.legend(title = "Room type")
```

Figure 18: Creating a bar plot for the first question



```python
# Dropping the "name" and "last_review" columns
ab_nyc_df_drop = ab_nyc_df.drop(["name", "last_review"], axis = 1)

# Getting rid of outliers
for column in ab_nyc_df_drop.select_dtypes(include='object'):
 if ab_nyc_df_drop[column].nunique() < 10:
  sns.boxplot(y=column, x='price', data=ab_nyc_df_drop, showfliers = False, showmeans=True)
  plt.title("Price by neighbourhood group")
  plt.xlabel("Price")
  plt.ylabel("Neighbourhood group")
```

Figure 19: Creating a box plot for the first question

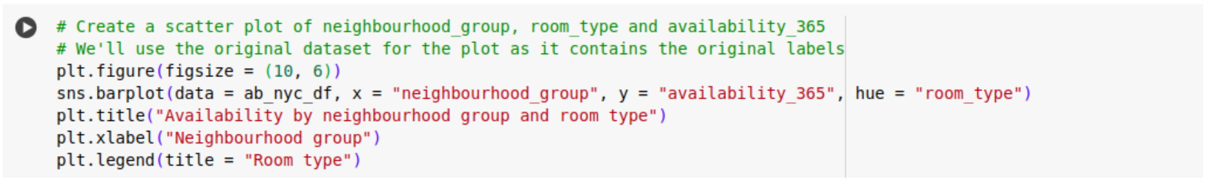


```python
# Define image size
plt.figure(figsize = (12, 10))

# Load and display map image
coordinates = (-74.2623, -73.6862, 40.4943, 40.9144)
map_img = mpimg.imread("new_york_neighbourhoods.png")
plt.imshow(map_img, extent=coordinates)

# Sample 10000 data points
sampled_data = ab_nyc_df.sample(n=10000, random_state=1).reset_index(drop=True)

# Cap prices at 200
price_upper_limit = 200
sampled_data['price'] = sampled_data['price'].apply(lambda x: x if x <= price_upper_limit else price_upper_limit)

# Normalize price data for color mapping
prices = sampled_data['price']
norm_prices = (prices - prices.min()) / (prices.max() - prices.min())
colors = plt.cm.RdYlGn(norm_prices)

# Set marker styles based on room type
room_type_markers = {
    'Private room': 'o',
    'Entire home/apt': '^',
    'Shared room': 's'  # Added Shared room
}

# Plot property prices and room types on the map
sc = None
for room_type, group in sampled_data.groupby('room_type'):
    sc = plt.scatter(group['longitude'], group['latitude'], c=group['price'], cmap='RdYlGn',
                     edgecolors='black', marker=room_type_markers.get(room_type, 'o'), s=10,
                     label=room_type, norm=mcolors.Normalize(vmin=0, vmax=price_upper_limit))

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Property prices and room types by region (sampled data)')

# Adjust subplot layout to position color bar on the right
fig = plt.gcf()
fig.subplots_adjust(right=0.75)
cbar_ax = fig.add_axes([0.9, 0.15, 0.03, 0.7])
plt.colorbar(sc, cax=cbar_ax, label='Price')

# Add legend
handles = [plt.Line2D([0], [0], marker=room_type_markers[rt], color='w', markerfacecolor='k', markersize=10, label=rt)
           for rt in room_type_markers]
plt.legend(handles=handles, title='Room type', loc='upper left', bbox_to_anchor=(3.0, 1))
plt.tight_layout()
```

Figure 20: Creating a scatter plot for the New York City neighbourhoods' rental samples

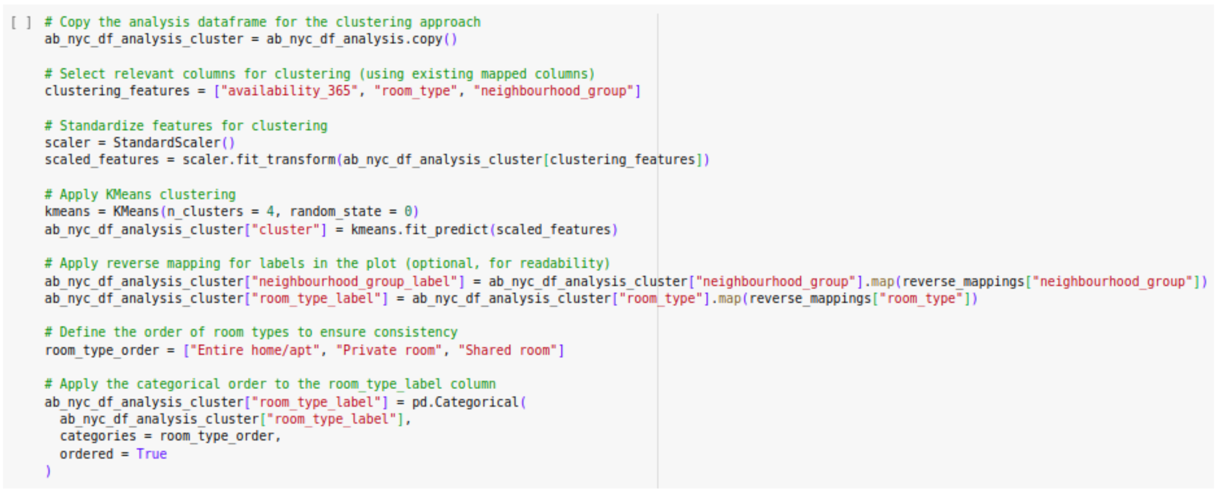
Second business question: how does availability over the next year correlate to room type and location?

The correlation analysis section has shown that there is a small, positive correlation between availability_365 and room_type, and a stronger positive relationship between room_type and price, as seen previously. Interestingly, the relationship between availability_365 and neighbourhood_group, despite being positive, seems to be fairly weak, suggesting that availability varies significantly between locations, as expected.

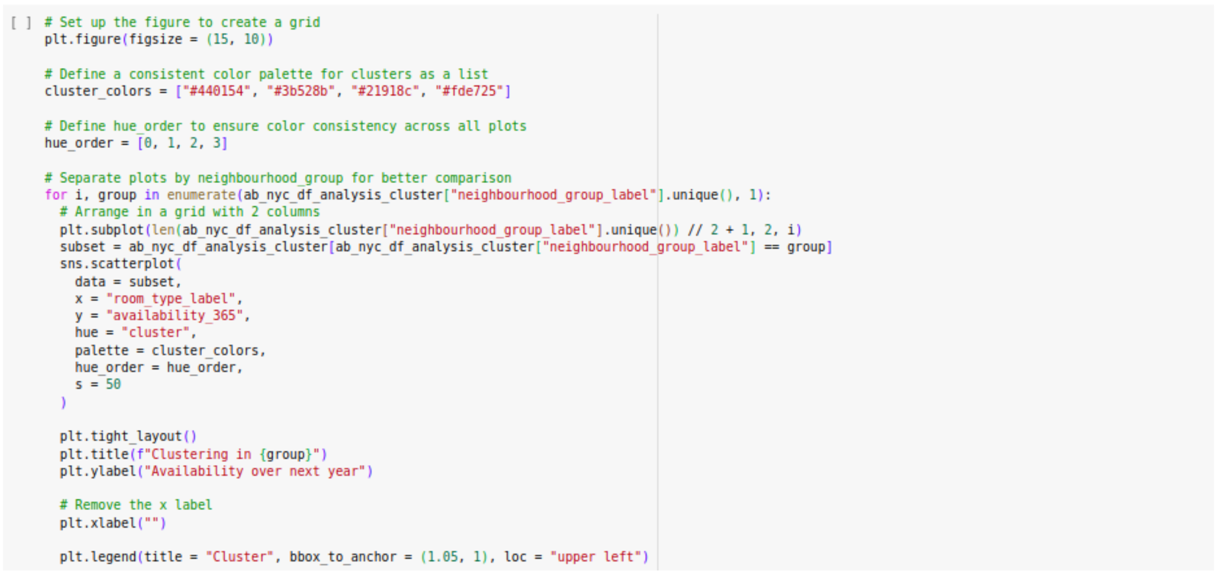


```python
# Create a scatter plot of neighbourhood_group, room_type and availability_365
# We'll use the original dataset for the plot as it contains the original labels
plt.figure(figsize = (10, 6))
sns.barplot(data = ab_nyc_df, x = "neighbourhood_group", y = "availability_365", hue = "room_type")
plt.title("Availability by neighbourhood group and room type")
plt.xlabel("Neighbourhood group")
plt.legend(title = "Room type")
```

Figure 21: Creating a bar plot for the second question

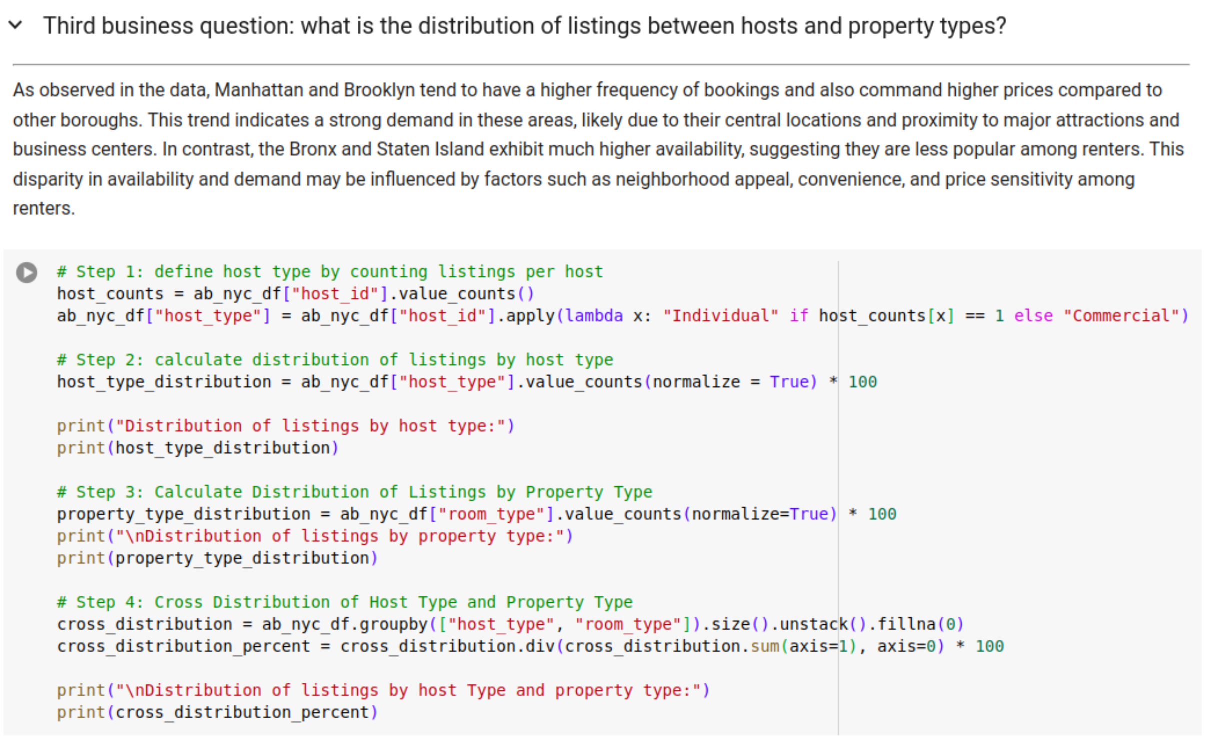


```
[ ] # Copy the analysis dataframe for the clustering approach
    ab_nyc_df_analysis_cluster = ab_nyc_df_analysis.copy()

    # Select relevant columns for clustering (using existing mapped columns)
    clustering_features = ["availability_365", "room_type", "neighbourhood_group"]

    # Standardize features for clustering
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(ab_nyc_df_analysis_cluster[clustering_features])

    # Apply KMeans clustering
    kmeans = KMeans(n_clusters = 4, random_state = 0)
    ab_nyc_df_analysis_cluster["cluster"] = kmeans.fit_predict(scaled_features)

    # Apply reverse mapping for labels in the plot (optional, for readability)
    ab_nyc_df_analysis_cluster["neighbourhood_group_label"] = ab_nyc_df_analysis_cluster["neighbourhood_group"].map(reverse_mappings["neighbourhood_group"])
    ab_nyc_df_analysis_cluster["room_type_label"] = ab_nyc_df_analysis_cluster["room_type"].map(reverse_mappings["room_type"])

    # Define the order of room types to ensure consistency
    room_type_order = ["Entire home/apt", "Private room", "Shared room"]

    # Apply the categorical order to the room_type_label column
    ab_nyc_df_analysis_cluster["room_type_label"] = pd.Categorical(
      ab_nyc_df_analysis_cluster["room_type_label"],
      categories = room_type_order,
      ordered = True
    )
```



```
[ ] # Set up the figure to create a grid
    plt.figure(figsize = (15, 10))

    # Define a consistent color palette for clusters as a list
    cluster_colors = ["#440154", "#3b528b", "#21918c", "#fde725"]

    # Define hue_order to ensure color consistency across all plots
    hue_order = [0, 1, 2, 3]

    # Separate plots by neighbourhood_group for better comparison
    for i, group in enumerate(ab_nyc_df_analysis_cluster["neighbourhood_group_label"].unique(), 1):
      # Arrange in a grid with 2 columns
      plt.subplot(len(ab_nyc_df_analysis_cluster["neighbourhood_group_label"].unique()) // 2 + 1, 2, i)
      subset = ab_nyc_df_analysis_cluster[ab_nyc_df_analysis_cluster["neighbourhood_group_label"] == group]
      sns.scatterplot(
        data = subset,
        x = "room_type_label",
        y = "availability_365",
        hue = "cluster",
        palette = cluster_colors,
        hue_order = hue_order,
        s = 50
      )

      plt.tight_layout()
      plt.title(f"Clustering in {group}")
      plt.ylabel("Availability over next year")

      # Remove the x label
      plt.xlabel("")

      plt.legend(title = "Cluster", bbox_to_anchor = (1.05, 1), loc = "upper left")
```

Figure 22: Clustering and plotting the data by availability over the next year, neighbourhood group and room type

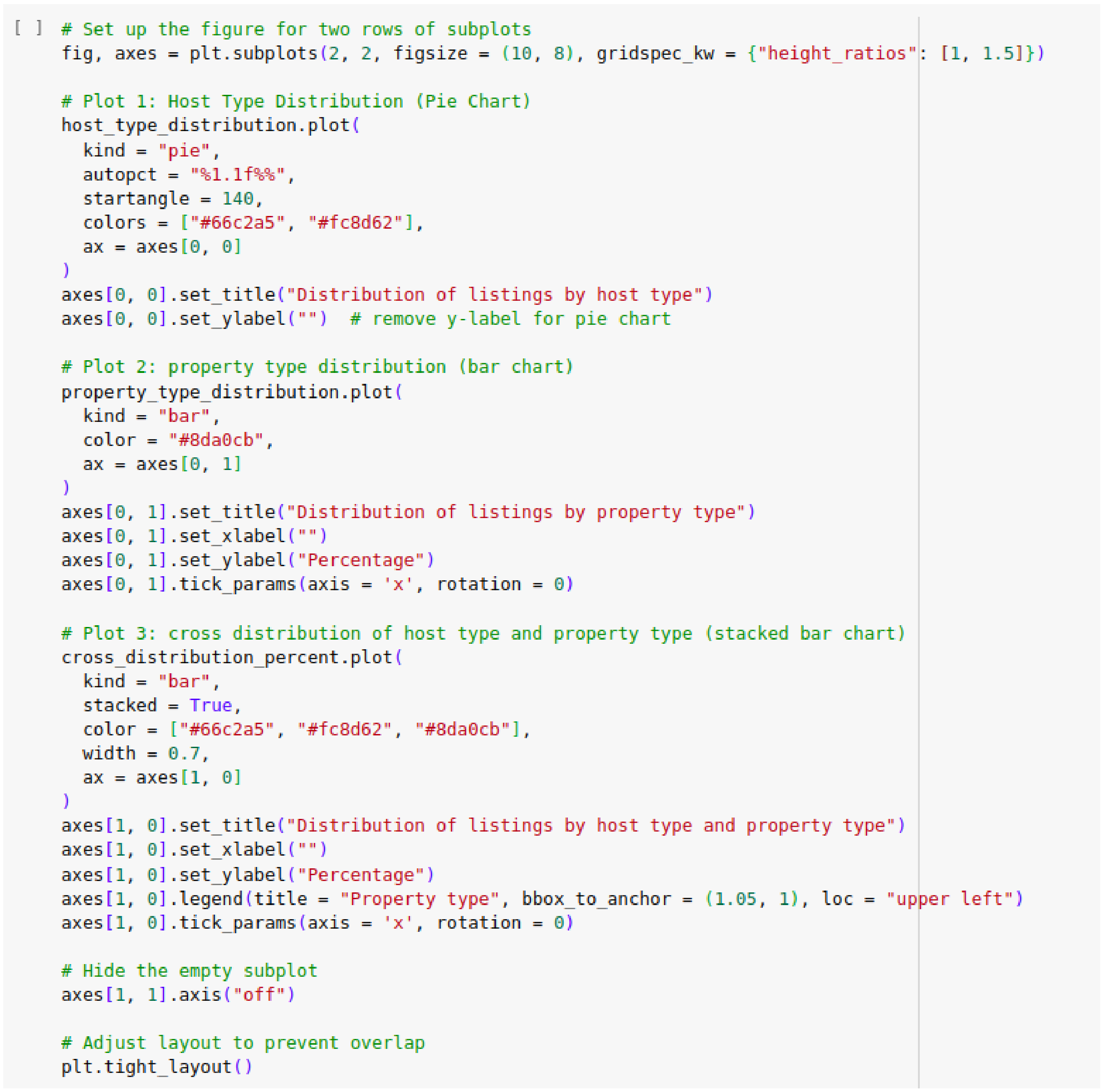


Third business question: what is the distribution of listings between hosts and property types?

As observed in the data, Manhattan and Brooklyn tend to have a higher frequency of bookings and also command higher prices compared to other boroughs. This trend indicates a strong demand in these areas, likely due to their central locations and proximity to major attractions and business centers. In contrast, the Bronx and Staten Island exhibit much higher availability, suggesting they are less popular among renters. This disparity in availability and demand may be influenced by factors such as neighborhood appeal, convenience, and price sensitivity among renters.

```python
# Step 1: define host type by counting listings per host
host_counts = ab_nyc_df["host_id"].value_counts()
ab_nyc_df["host_type"] = ab_nyc_df["host_id"].apply(lambda x: "Individual" if host_counts[x] == 1 else "Commercial")

# Step 2: calculate distribution of listings by host type
host_type_distribution = ab_nyc_df["host_type"].value_counts(normalize = True) * 100

print("Distribution of listings by host type:")
print(host_type_distribution)

# Step 3: Calculate Distribution of Listings by Property Type
property_type_distribution = ab_nyc_df["room_type"].value_counts(normalize=True) * 100
print("\nDistribution of listings by property type:")
print(property_type_distribution)

# Step 4: Cross Distribution of Host Type and Property Type
cross_distribution = ab_nyc_df.groupby(["host_type", "room_type"]).size().unstack().fillna(0)
cross_distribution_percent = cross_distribution.div(cross_distribution.sum(axis=1), axis=0) * 100

print("\nDistribution of listings by host Type and property type:")
print(cross_distribution_percent)
```

Figure 23: Separating individual from commercial hosts



```
[ ] # Set up the figure for two rows of subplots
    fig, axes = plt.subplots(2, 2, figsize = (10, 8), gridspec_kw = {"height_ratios": [1, 1.5]})

    # Plot 1: Host Type Distribution (Pie Chart)
    host_type_distribution.plot(
      kind = "pie",
      autopct = "%1.1f%%",
      startangle = 140,
      colors = ["#66c2a5", "#fc8d62"],
      ax = axes[0, 0]
    )
    axes[0, 0].set_title("Distribution of listings by host type")
    axes[0, 0].set_ylabel("")  # remove y-label for pie chart

    # Plot 2: property type distribution (bar chart)
    property_type_distribution.plot(
      kind = "bar",
      color = "#8da0cb",
      ax = axes[0, 1]
    )
    axes[0, 1].set_title("Distribution of listings by property type")
    axes[0, 1].set_xlabel("")
    axes[0, 1].set_ylabel("Percentage")
    axes[0, 1].tick_params(axis = 'x', rotation = 0)

    # Plot 3: cross distribution of host type and property type (stacked bar chart)
    cross_distribution_percent.plot(
      kind = "bar",
      stacked = True,
      color = ["#66c2a5", "#fc8d62", "#8da0cb"],
      width = 0.7,
      ax = axes[1, 0]
    )
    axes[1, 0].set_title("Distribution of listings by host type and property type")
    axes[1, 0].set_xlabel("")
    axes[1, 0].set_ylabel("Percentage")
    axes[1, 0].legend(title = "Property type", bbox_to_anchor = (1.05, 1), loc = "upper left")
    axes[1, 0].tick_params(axis = 'x', rotation = 0)

    # Hide the empty subplot
    axes[1, 1].axis("off")

    # Adjust layout to prevent overlap
    plt.tight_layout()
```

Figure 24: Creating several plots to showcase the distribution of rental types per host type