References Development Team Project: Project Report

Introduction:

The hotel industry has undergone significant changes due to the introduction of online platforms, with customer behaviours relying on online ratings and peers' opinions, pricing and distance to touristic attractions (Urvashi et al, 2021). Airbnb has rapidly grown to become one of the top travel accommodation solutions, with New York seeing significant growth in Airbnb usage, positively affecting the sector (Jiao et al, 2020). This report poses a business question about Airbnb's growth within New York city. Following data pre-processing and exploratory analysis, the dataset provides details on listings, price, geolocation and availability, property types and review history.

The analysis assesses which neighbourhoods show untapped potential for growth, based on current demand, pricing and availability trends; this enables targeting of areas where new listings could thrive, balancing supply and demand. Focusing on growth potential in underutilised areas could lead to increased revenue by optimising listings and attracting more hosts in these neighbourhoods.

Lesser-known neighbourhoods close to major landmarks may see increased demand as people seek authentic experiences or cheaper alternatives to established hotspots, which in turn can lead to an increase in revenue.

Methodology:

Data Pre-Processing:

The data is in good condition, with few features with missing or invalid values. ID, listing name, host ID, host name and the date of the last review have been removed as they are not well suited to use in a linear model. Last review date could have been used as a popularity metric, however the number of reviews per month is more suited. These two features have a spearman correlation of 0.8, as shown in Figure 1. Considering that both features have null values, and most machine learning algorithms prefer uncorrelated features, only reviews per month was kept, with null values set to 0. The availability feature was converted from days to a percentage for ease of interpretation.

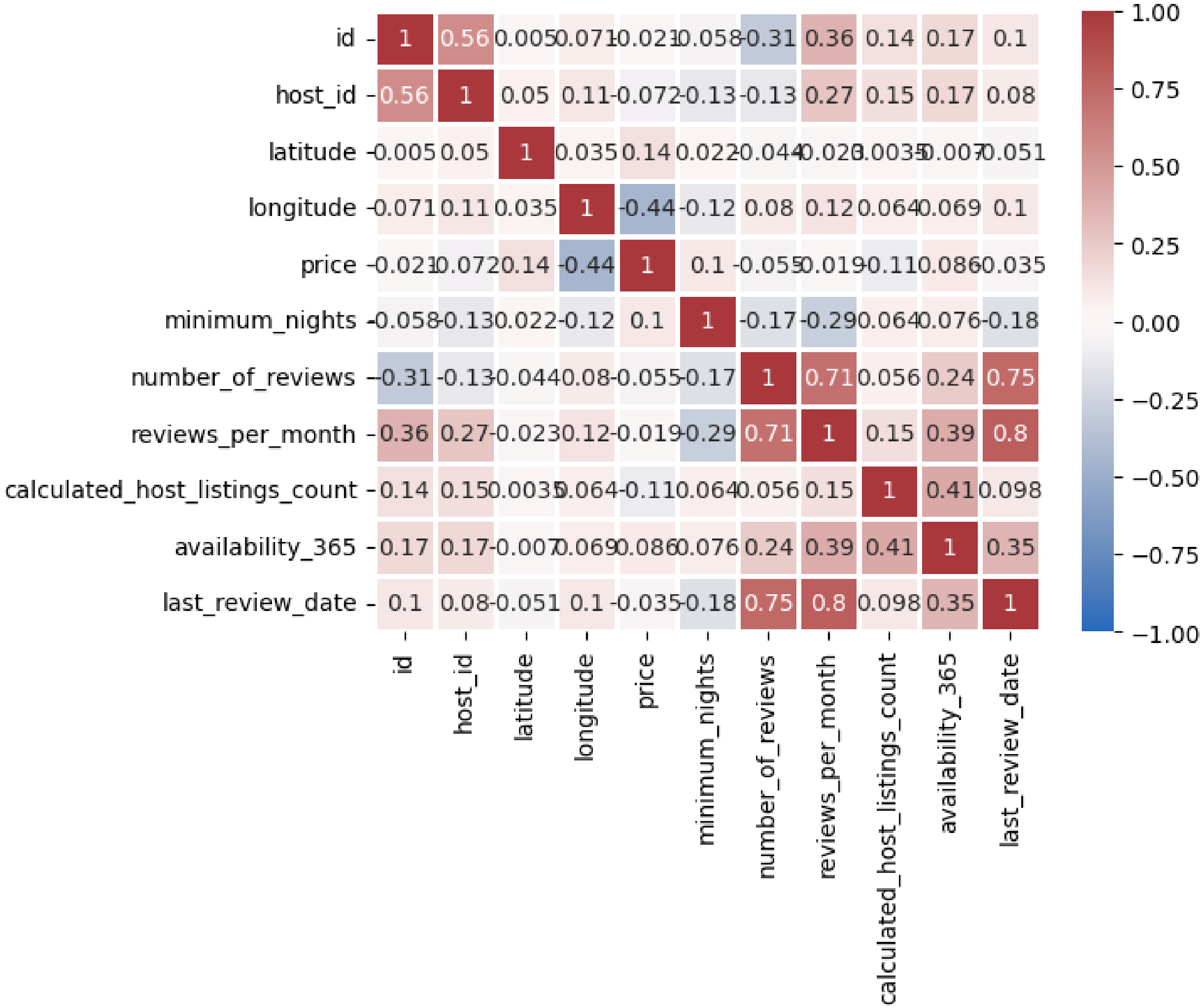


Figure 1: Correlation matrix of numeric features

The data set contains some significant outliers. The price field has a long-tailed distribution that will make difficult for a model to learn from. Price field and minimum nights fields contain values that are either errors or that are not representative of the short-term rental market. Thus, only data with a price in the range of $20-$300 and fewer than 30 minimum nights will be considered in the analysis. The remaining outliers in number of reviews, reviews per month and host listing count were then capped to reasonable values, the result is shown in Figure 2.

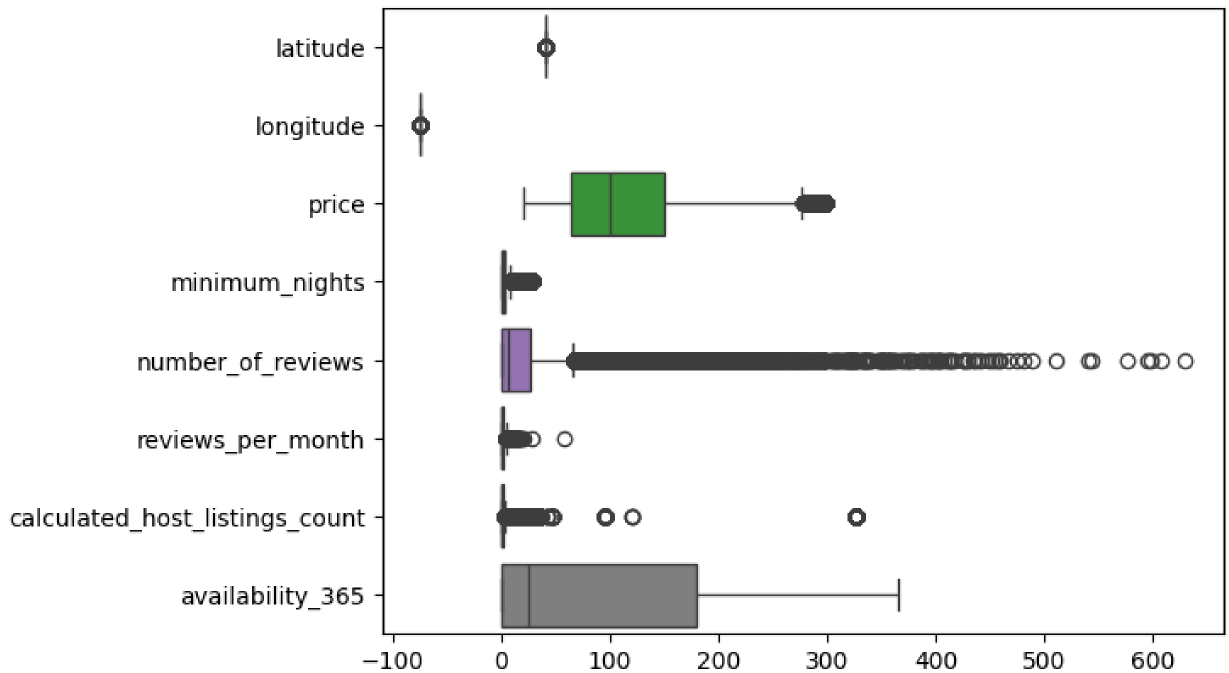


Figure 2: Distributions of selected features

The data was then augmented by adding the distances from each set of coordinates to a selection of 7 popular landmarks in NYC.

Data Analysis:

Initial analysis focused on the trends in price and availability with Staten Island and Bronx having moderate prices and high availability which might indicate untapped growth potential. However, the prices and availability can be explained by other features not included in the dataset such as neighbourhood safety, crime statistics, or neighbourhood amnesties such as proximity to public transport (Airbtics, 2023).

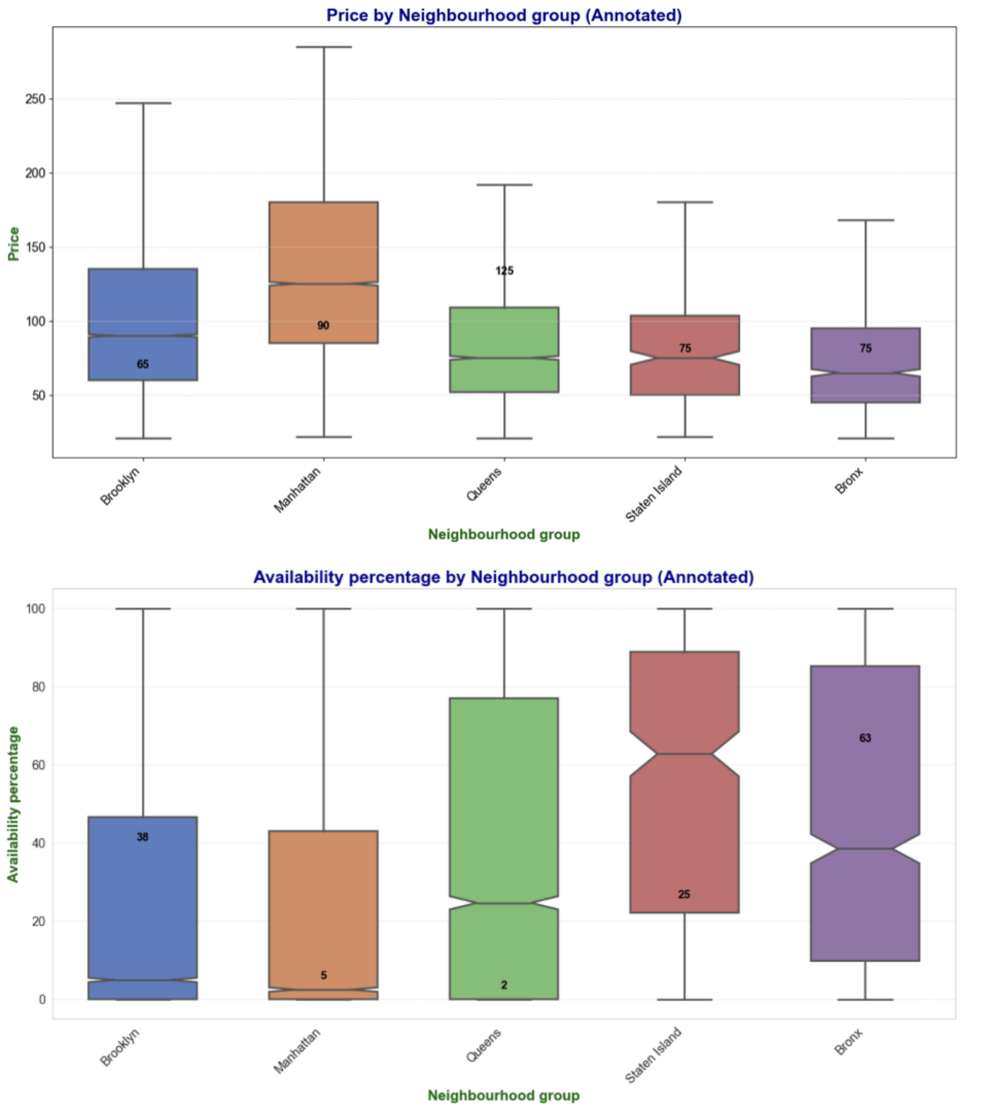


Figure 3: Price and availability by neighbourhood group

To check if prices differ significantly across neighbourhoods Kruskal-Wallis test was undertaken which indicates that there is strong evidence to reject the null-hypothesis. After removing outliers, the Kruskal-Wallis result of 4627.593557881785 with a p-value of 0.0 indicates that the differences between groups is pronounced.

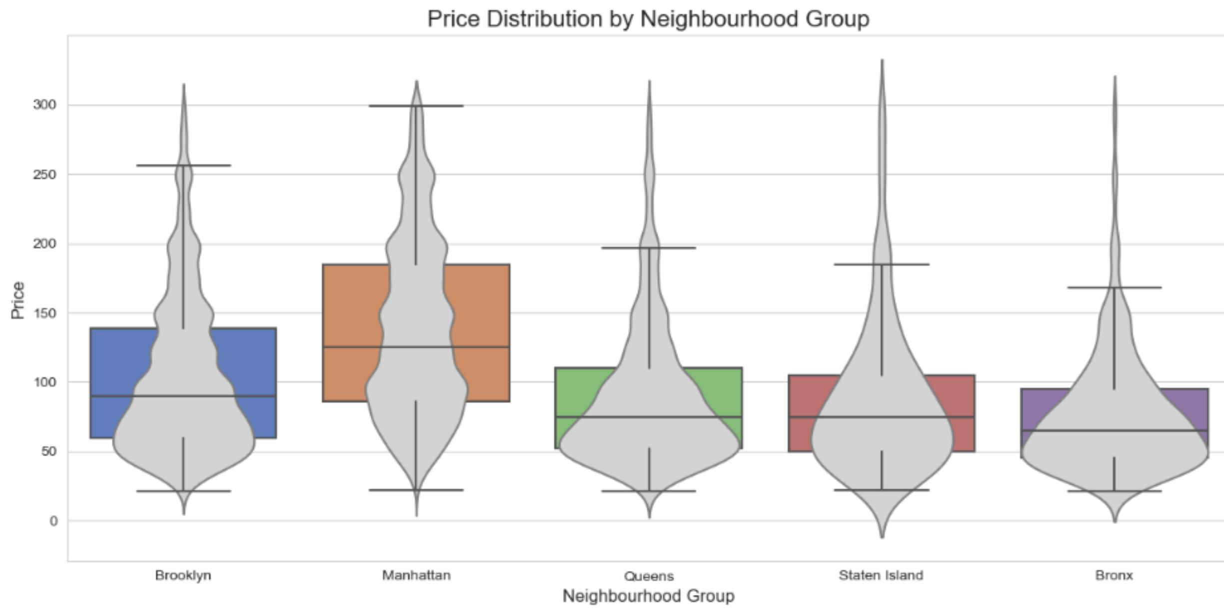


Figure 4: Price distribution by Neighbourhood group after outliers are removed.

Figure 3 highlights neighbourhoods based on demand and availability thresholds. The thresholds are dynamically calculated using percentiles to adapt to the data distribution and the interactive popups include details about the neighbourhood, such as average reviews, availability, price, and top listing. Potential growth can be found in neighbourhoods such as Little Italy, Woodlawn, and City Island.

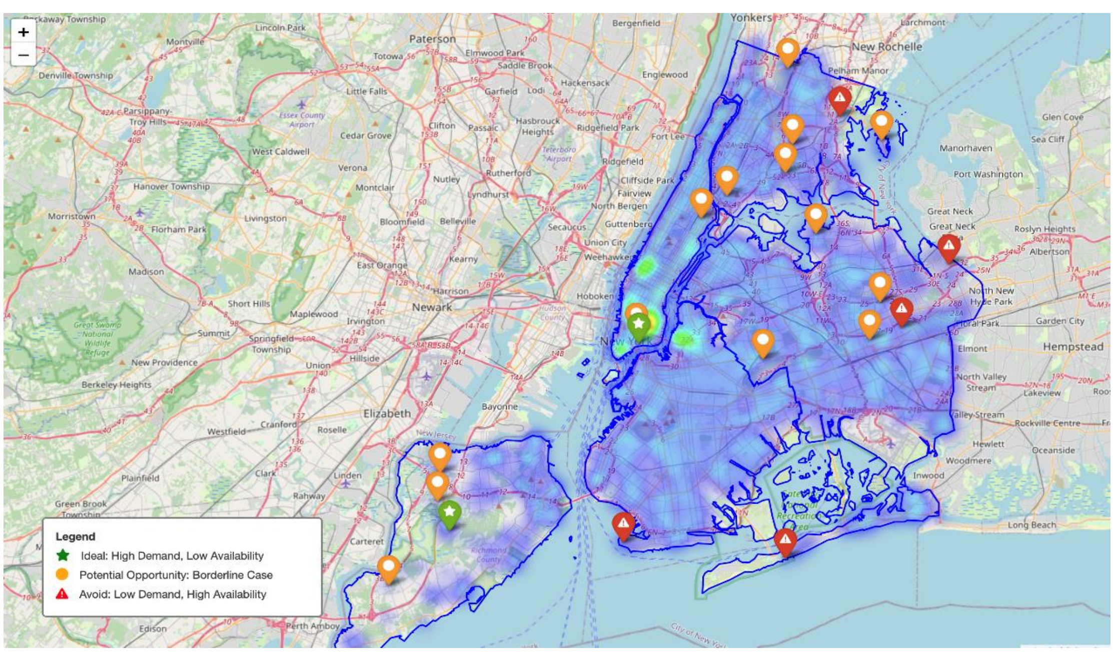


Figure 5: Demand and Availability map

Figure 4 groups neighbourhoods into clusters based on key metrics such as price, demand and availability (Scikit-learn Documentation, 2024).
Each cluster highlights neighbourhoods with similar characteristics, helping to identify patterns such as similar pricing or availability trends.

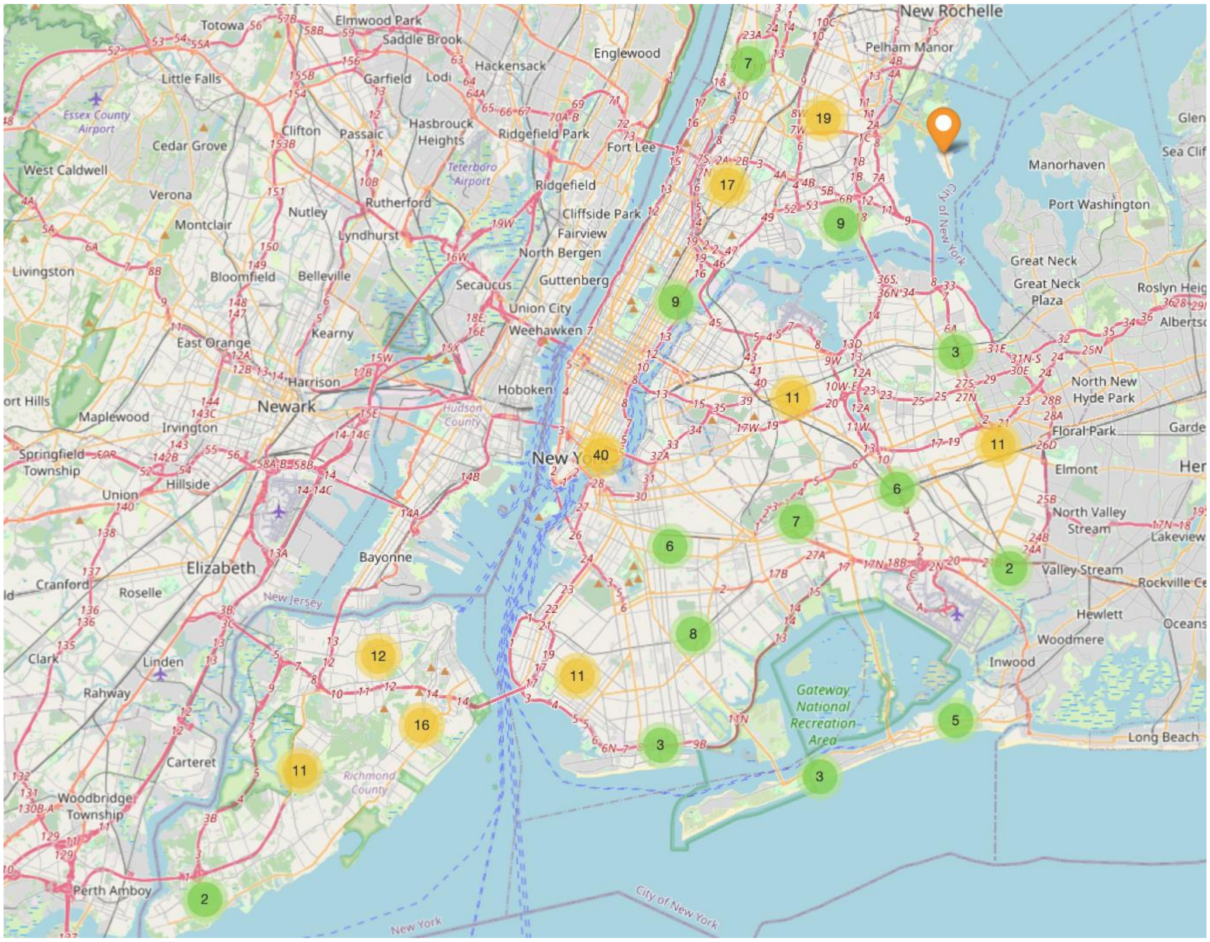


Figure 6: Clustered map

Multiple regression shows 53% of the variance is explained by the model features. It is usual to have moderate r-squares for housing models due to variability in pricing. The model is not overfitting and generalises well to unknown data.

Positive price gaps indicate that the predicted values are higher than the actual price which might suggest that the neighbourhoods are undervalued. Further research looking into each neighbourhood would be useful. Additional data such as demographics, safety, public transport and other amenities can help explain the difference in pricing.

With the current data we can check if neighbourhood has a large price gap but low demand (e.g., low number_of_reviews or reviews_per_month), as it may indicate a lack of awareness rather than untapped potential. Conversely, high demand and a price gap might confirm real untapped potential.

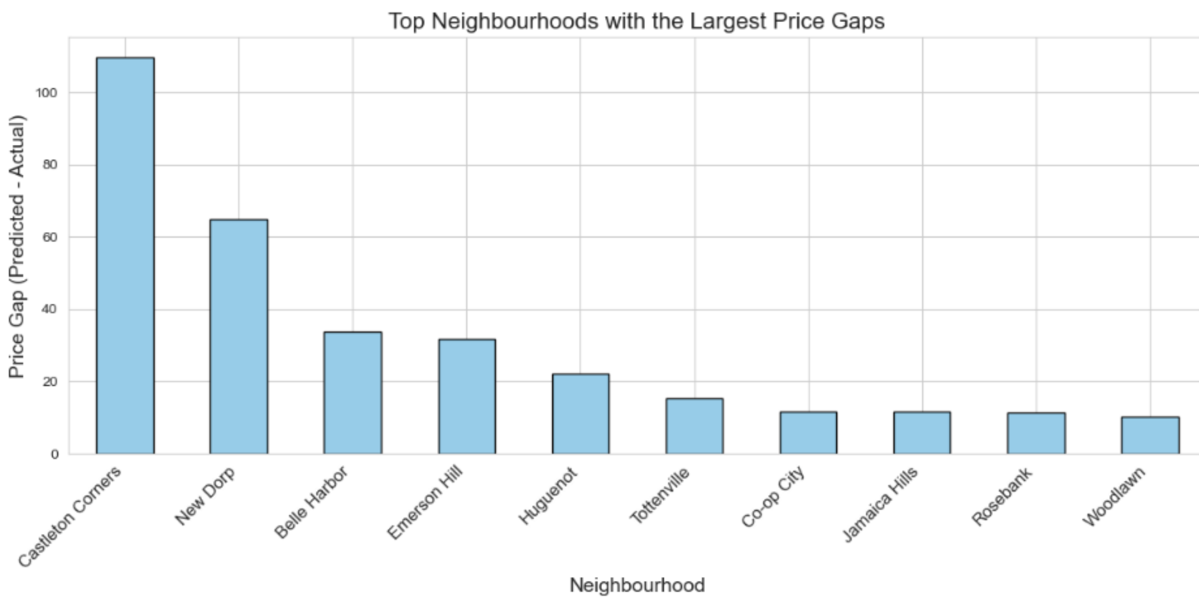


Figure 7: Top neighbourhoods with the largest Price Gaps

Random Forest Regression was used to predict the prices based on all available features with a raining R-squared: 0.5300 Test R-squared: 0.5377. The R-squared is reasonable for the given dataset and the results are similar with the Multiple regression, which might show untapped potential. The actual vs predicted prices are fitting well in the middle with the minimum and maximum showing significant differences.

```
Statistics for Actual vs Predicted Prices:
                      min        25%         50%         75%         max  \
Actual Price    22.000000  65.000000  100.000000  150.000000  299.000000
Predicted Price 66.358711  66.418048  117.520517  151.454927  188.923513

                      mean
Actual Price    114.726884
Predicted Price 115.295819
```

Figure 8: Actual vs predicted prices based on Random Forest

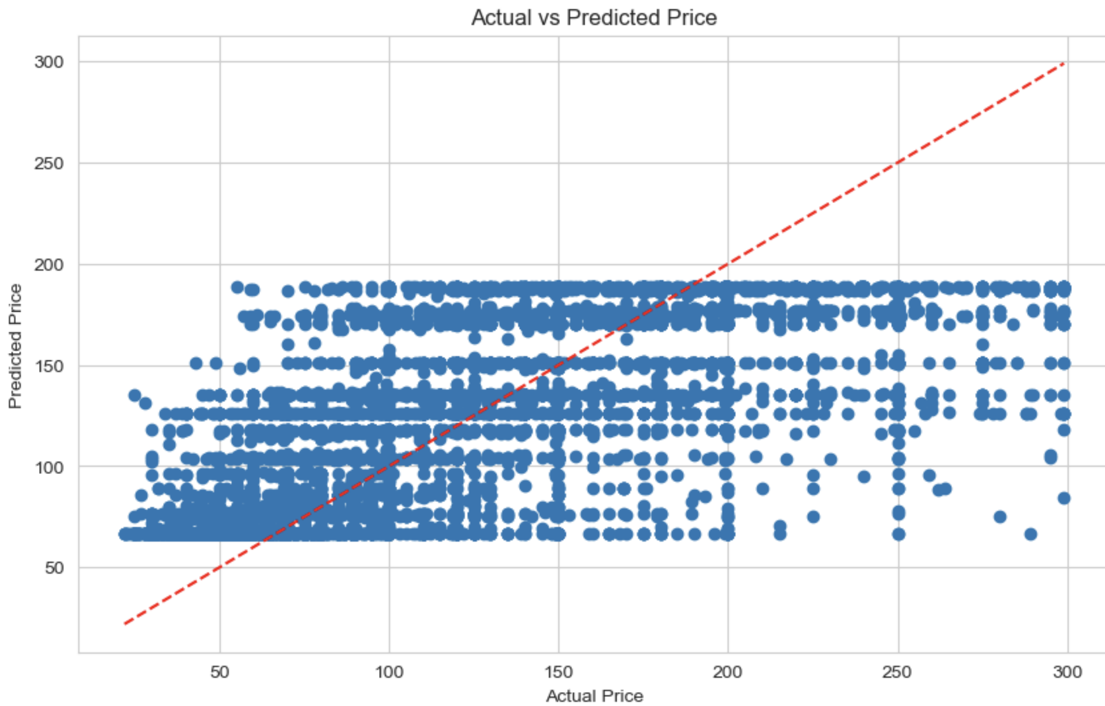


Figure 9: Actual vs predicted prices based on Random Forest

The statistics for each neighbourhood highlight that the model struggles to capture the low and high prices, this is expected as those regions were removed as outliers. However, the mean prices are well matched between the actual and predicted prices, indicating that the model is missing specific nuances but capturing the overall trend.

To assess how important the geographical position is to the pricing, a Random Forst Regression based only on the proximity to different landmarks has been undertaken with the model explaining around 28% (Training R-squared: 0.2811) of the variance, with distance_to_Empire_State_Building having the most impact to the price out of all the other landmarks.

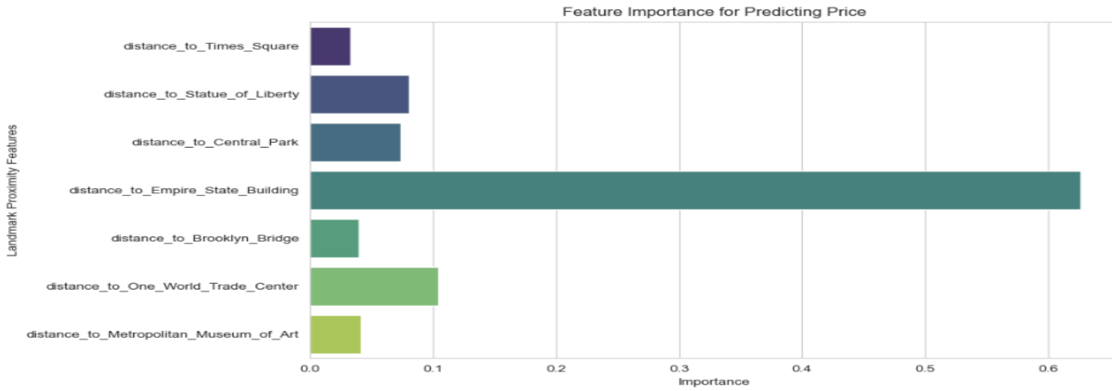


Figure 10: Predicted price feature importance in Random Forest

```
Statistics by Neighbourhood:
                  min_actual  q1_actual  median_actual  q3_actual  max_actual  \
neighbourhood
Allerton                  33       38.0           49.0      66.00          80
Arrochar                  33       33.5           34.0      34.50          35
Arverne                   35       90.5          137.0     150.00         250
Astoria                   27       65.0           90.0     125.00         250
Bath Beach                45       48.0           74.0      99.25         100
...                      ...        ...            ...        ...         ...
Williamsburg              27       70.0          100.0     160.00         298
Windsor Terrace           40       70.0          120.0     143.50         250
Woodhaven                 30       45.0           50.0      70.50         170
Woodlawn                  70       70.0           70.0      70.00          70
Woodside                  30       40.0           82.5     120.00         195

                  mean_actual  min_predicted  q1_predicted  median_predicted  \
neighbourhood
Allerton            52.875000      66.358711     66.358711         66.358711
Arrochar            34.000000      66.358711     66.358711         66.358711
Arverne            131.000000      66.358711     66.358711        126.080605
Astoria            100.636943      74.713935     75.199961         92.428104
Bath Beach          73.250000      66.358711     66.358711         66.358711
...                       ...            ...           ...               ...
Williamsburg       119.707355      66.418048     79.751267         85.743738
Windsor Terrace    114.400000      66.358711     66.358711        126.080605
Woodhaven           63.222222      66.358711     66.358711         66.358711
Woodlawn            70.000000      66.358711     66.358711         66.358711
Woodside            85.214286      66.358711     66.418048         74.617364

                  q3_predicted  max_predicted  mean_predicted
neighbourhood
Allerton             66.358711     126.080605       73.823948
Arrochar             66.358711      66.358711       66.358711
Arverne             126.080605     126.080605      104.363552
Astoria             135.232961     169.509422      109.029216
Bath Beach           81.289185     126.080605       81.289185
...                        ...            ...             ...
Williamsburg        174.177130     174.652751      119.259046
Windsor Terrace     126.288344     131.095657      104.380769
Woodhaven            66.358711     126.080605       77.418321
Woodlawn             66.358711      66.358711       66.358711
Woodside            135.057327     135.057327       92.787856

[196 rows x 12 columns]
```

Figure 11: Actual vs predicted prices by neighbourhood based on Random Forest

Recommendations

There is significant potential for growth in Staten Island and The Bronx due to moderate pricing and high availability. Airbnb should focus on these boroughs, encouraging new listings in high demand areas near major landmarks (Investopedia, 2023). Offering incentives such as reduced fees or promotional support may attract more hosts to these underused neighbourhoods. They could also promote private rooms in affordable areas and entire homes in premium areas, optimising room types based on area-specific trends to maximise revenue. Encouragement of dynamic pricing strategies would help with fluctuation in demand, availability and location-specific trends.

Conclusion

The analysis highlights boroughs such as Staten Island and The Bronx as areas with untapped growth potential, where affordable pricing and high demand present significant opportunities. By targeting these neighbourhoods and adopting data-driven strategies, Airbnb can expand its market, increase host participation, and optimise revenue (AirDNA, 2023). These insights offer a strategic pathway for growth while providing actionable guidance for both Airbnb and its hosts.

Appendix A: Project Code

Data Preprocessing

```
print(raw_data.shape)
print(10052/raw_data.shape[0])
pd.concat([raw_data.isnull().sum(), raw_data.dtypes], axis=1,
            keys=['Null Count', 'Data Types'])
```

```
(48895, 16)
0.20558339298496778
```

	Null Count	Data Types
id	0	int64
name	16	object
host_id	0	int64
host_name	21	object
neighbourhood_group	0	object
neighbourhood	0	object
latitude	0	float64
longitude	0	float64
room_type	0	object
price	0	int64
minimum_nights	0	int64
number_of_reviews	0	int64
last_review	10052	object
reviews_per_month	10052	float64
calculated_host_listings_count	0	int64
availability_365	0	int64

```python
raw_data[raw_data['number_of_reviews'] == 0].isnull().sum()
```

Python

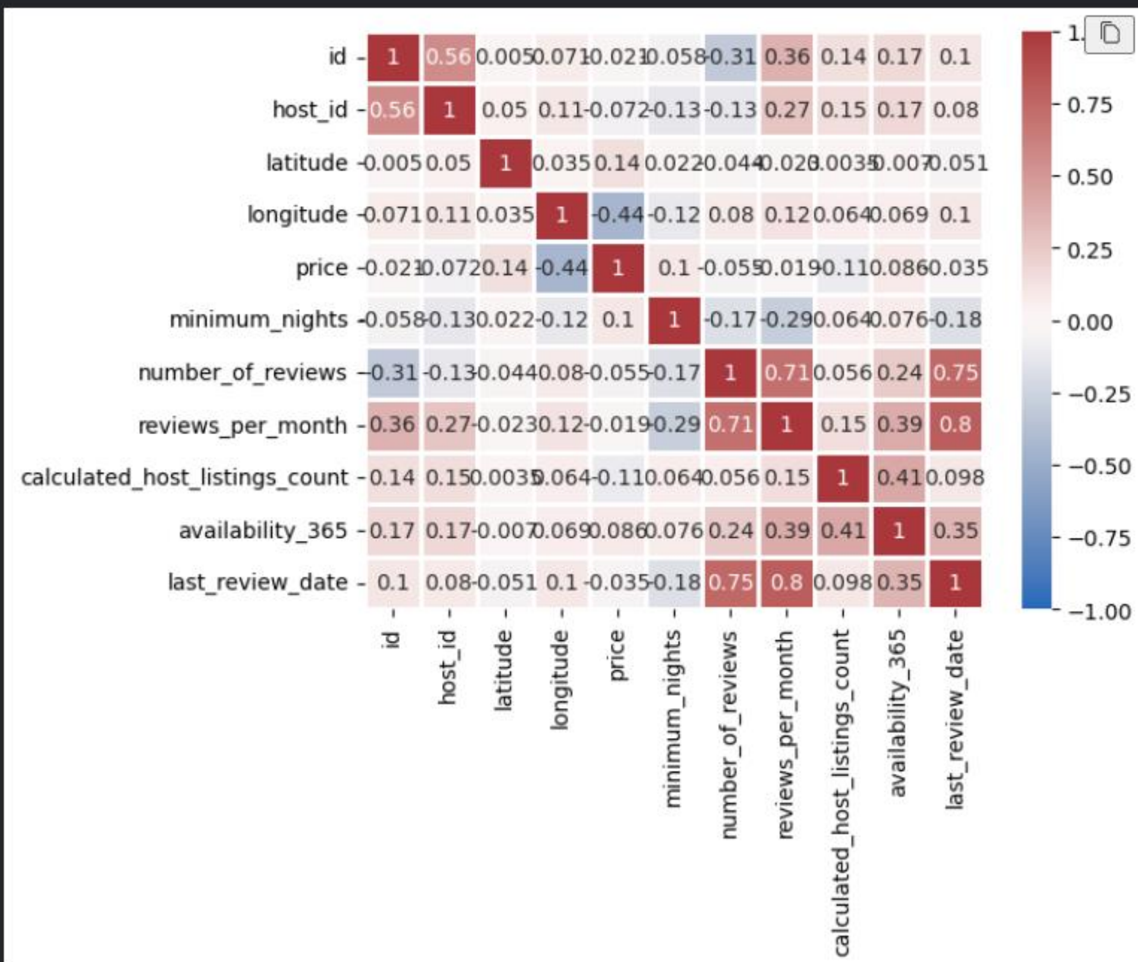
```
id                                    0
name                                 10
host_id                               0
host_name                             5
neighbourhood_group                   0
neighbourhood                         0
latitude                              0
longitude                             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
last_review_date                  10052
dtype: int64
```

It is clear that having no reviews fully explains null entries for `last_review` and `reviews_per_month`, it also might be related to some of the null name values.

```python
# Using Spearman correlation, we have not yet removed outliers or tested for normality.
sns.heatmap(numeric_data.corr(method='spearman'), annot=True, cmap='vlag',
            linewidths=0.8, vmin=-1, vmax=1)
```

Python

<Axes: >



```python
data = raw_data.drop(['id', 'name', 'host_id', 'host_name', 'last_review',
                      'last_review_date'], axis=1)
data['reviews_per_month'] = data['reviews_per_month'].fillna(0)
data.isnull().sum()
```

```
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
```

```python
data_cleaned['availability_percentage'] = (data_cleaned['availability_365'] / 365) * 100
```

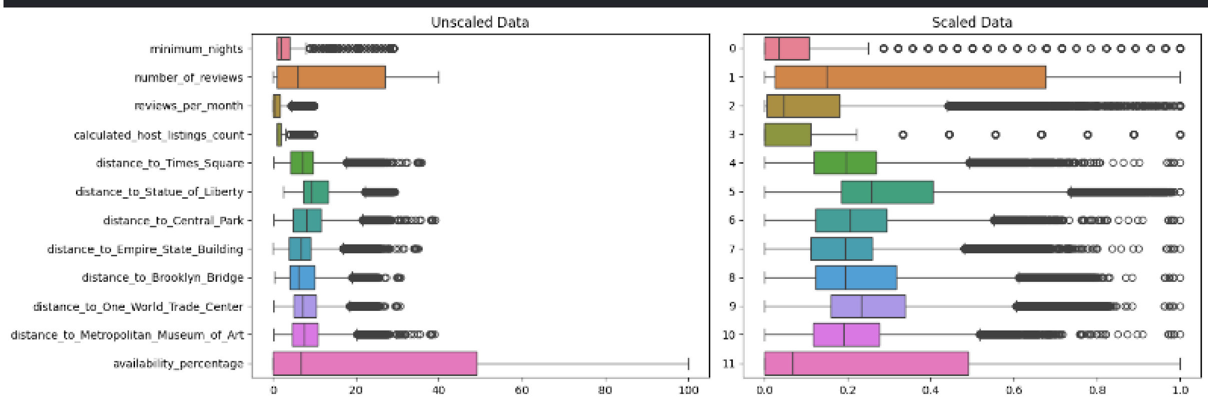
```python
data_cleaned = data_cleaned.drop(columns=['availability_365'])
```

```python
data_preprocessed = data_cleaned.copy()
data_preprocessed['number_of_reviews'] = data_preprocessed[
    'number_of_reviews'].clip(upper=40)
data_preprocessed['reviews_per_month'] = data_preprocessed[
    'reviews_per_month'].clip(upper=10)
data_preprocessed['calculated_host_listings_count'] = data_preprocessed[
    'calculated_host_listings_count'].clip(upper=10)

numeric_data = data_preprocessed.select_dtypes(include=[np.number, np.datetime64]
                ).drop(['price'], axis=1) # This column is in an acceptable state and can be

fig, axes = plt.subplots(1, 2, figsize=(15, 5))
sns.boxplot(data=numeric_data, orient='h', ax=axes[0])
axes[0].set_title('Unscaled Data')
scaled_data = minmax_scale(numeric_data)
sns.boxplot(data=scaled_data, orient='h', ax=axes[1])
axes[1].set_title('Scaled Data')
plt.tight_layout()
```

Python



Data Augmentation

```python
# Landmark coordinates
landmarks = {
    "Times Square": (40.7580, -73.9855),
    "Statue of Liberty": (40.6892, -74.0445),
    "Central Park": (40.7851, -73.9683),
    "Empire State Building": (40.7488, -73.9854),
    "Brooklyn Bridge": (40.7061, -73.9969),
    "One World Trade Center": (40.7128, -74.0131),
    "Metropolitan Museum of Art": (40.7794, -73.9632),
}

# Haversine formula
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Earth's radius in km
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    return R * c

# Add distances to landmarks
for landmark, coords in landmarks.items():
    data_cleaned[f"distance_to_{landmark.replace(' ', '_')}"] = data_cleaned.apply(
        lambda row: haversine(row['latitude'], row['longitude'], coords[0], coords[1]), axis
    )
```

Python

```python
data_preprocessed.head()
```

Python

	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_pe
0	Brooklyn	Kensington	Private room	149	1	9	
1	Manhattan	Midtown	Entire home/apt	225	1	40	
2	Manhattan	Harlem	Private room	150	3	0	
3	Brooklyn	Clinton Hill	Entire home/apt	89	1	40	
4	Manhattan	East Harlem	Entire	80	10	9	

Data Processing

Data Analysis

Price and availability by neighbourhood group

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

def plot_boxplots_with_annotations(data, column, group_by):

    plt.figure(figsize=(14, 8))

    boxplot = sns.boxplot(
        data=data,
        x=group_by,
        y=column,
        palette='muted',
        linewidth=2.0,
        showfliers=False,
        notch=True,
        width=0.6
    )

    sns.set_style('whitegrid')
    plt.grid(color='lightgrey', linestyle='--', linewidth=0.5, axis='y')

    medians = data.groupby(group_by)[column].median()

    for tick, median in enumerate(medians):
        boxplot.text(
            tick, median + (0.05 * median),
            f'{median:.0f}',
            horizontalalignment='center',
            fontsize=12,
            color='black',
            weight='bold'
        )


    plt.title(
        f'{column.replace("_", " ").capitalize()} by {group_by.replace("_", " ").capita
        fontsize=18,
        fontweight='bold',
        color='darkblue'
    )
    plt.xlabel(group_by.replace("_", " ").capitalize(), fontsize=15, fontweight='bold',
    plt.ylabel(column.replace("_", " ").capitalize(), fontsize=15, fontweight='bold', c


    plt.xticks(fontsize=13, rotation=45, ha='right')
    plt.yticks(fontsize=13)

    plt.tight_layout()

    plt.show()

plot_boxplots_with_annotations(data_cleaned, 'price', 'neighbourhood_group')
plot_boxplots_with_annotations(data_cleaned, 'availability_percentage', 'neighbourhood_
```

Price distribution by neighbourhood group

```python
groups = [data[data['neighbourhood_group'] == group]['price'] for group in data['neighb

# Check normality and equal variance assumptions
normality = all(shapiro(group)[1] > 0.05 for group in groups)
variance = levene(*groups)[1] > 0.05

# Perform the appropriate test
if normality and variance:
    result = f_oneway(*groups)
    print('ANOVA result:', result)
else:
    result = kruskal(*groups)
    print('Kruskal-Wallis result:', result)
```

```
Kruskal-Wallis result: KruskalResult(statistic=4627.593557881785, pvalue=0.0)
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

def plot_boxplots_with_annotations(data, column, group_by):

    plt.figure(figsize=(14, 8))

    boxplot = sns.boxplot(
        data=data,
        x=group_by,
        y=column,
        palette='muted',
        linewidth=2.0,
        showfliers=False,
        notch=True,
        width=0.6
    )

    sns.set_style('whitegrid')
    plt.grid(color='lightgrey', linestyle='--', linewidth=0.5, axis='y')

    medians = data.groupby(group_by)[column].median()

    for tick, median in enumerate(medians):
        boxplot.text(
            tick, median + (0.05 * median),
            f'{median:.0f}',
            horizontalalignment='center',
            fontsize=12,
            color='black',
            weight='bold'
        )


    plt.title(
        f'{column.replace("_", " ").capitalize()} by {group_by.replace("_", " ").capita
        fontsize=18,
        fontweight='bold',
        color='darkblue'
    )
    plt.xlabel(group_by.replace("_", " ").capitalize(), fontsize=15, fontweight='bold',
    plt.ylabel(column.replace("_", " ").capitalize(), fontsize=15, fontweight='bold', c


    plt.xticks(fontsize=13, rotation=45, ha='right')
    plt.yticks(fontsize=13)

    plt.tight_layout()

    plt.show()

plot_boxplots_with_annotations(data_cleaned, 'price', 'neighbourhood_group')
plot_boxplots_with_annotations(data_cleaned, 'availability_percentage', 'neighbourhood_
```

Demand and availability map

```python
availability_price_analysis = data.groupby(['neighbourhood_group', 'neighbourhood']).ag
    avg_price=('price', 'mean'),
    avg_availability=('availability_percentage', 'mean'),
    avg_reviews=('reviews_per_month', 'mean'),
    listings_count=('neighbourhood', 'count')  # Count of listings in the neighbourhood
).reset_index()

# Refine thresholds dynamically using percentiles
availability_threshold = availability_price_analysis['avg_availability'].quantile(0.25)
review_threshold = availability_price_analysis['avg_reviews'].quantile(0.75)  # High de

# Define tolerances for borderline cases using median-based flexibility
availability_tolerance = availability_price_analysis['avg_availability'].quantile(0.50)
review_tolerance = availability_price_analysis['avg_reviews'].quantile(0.50)  # Median
```

```python
# Generate data for price heatmap
price_heat_data = [
    [row['latitude'], row['longitude'], row['price']]
    for _, row in data.iterrows()
]
```

```
# Create high-demand, low-availability neighborhoods (Green Markers)
high_demand_low_availability = availability_price_analysis[
    (availability_price_analysis['avg_reviews'] >= review_threshold) &
    (availability_price_analysis['avg_availability'] <= availability_threshold)
]

# Create borderline neighborhoods (Orange Markers), excluding green markers
borderline_neighbourhoods = availability_price_analysis[
    ~availability_price_analysis['neighbourhood'].isin(high_demand_low_availability['ne
        ((availability_price_analysis['avg_reviews'] >= review_tolerance) &
         (availability_price_analysis['avg_availability'] <= availability_threshold)) |
        ((availability_price_analysis['avg_reviews'] >= review_threshold) &
         (availability_price_analysis['avg_availability'] <= availability_tolerance))
    )
]

# Create neighborhoods to avoid (Red Markers)
avoid_neighbourhoods = availability_price_analysis[
    (availability_price_analysis['avg_reviews'] <= review_threshold/5) &
    (availability_price_analysis['avg_availability'] >= availability_threshold/5)
]

# Function to generate popup content
def create_popup(row, category):
    # Example placeholder trend (replace with real calculations as needed)
    price_trend = "Price trend: Upward"

    # Top listings approximation using 'number_of_reviews'
    top_rated = data[data['neighbourhood'] == row['neighbourhood']].sort_values('number
    top_listings = "<br>".join([
        f"- {listing['room_type']} (${listing['price']}, {listing['number_of_reviews']}
        for _, listing in top_rated.iterrows()
    ])

    return folium.Popup(
        f"<b>{category}</b><br>"
        f"<b>Neighborhood:</b> {row['neighbourhood']}<br>"
        f"<b>Average Reviews:</b> {row['avg_reviews']:.2f}<br>"
        f"<b>Average Availability:</b> {row['avg_availability']:.2f}%<br>"
        f"<b>Average Price:</b> ${row['avg_price']:.2f}<br>"
        f"<b>{price_trend}</b><br>"
        f"<b>Top Listings:</b><br>{top_listings}",
        max_width=300
    )


# Create the flexible map
flexible_map_with_enhanced_markers = folium.Map(location=[40.7128, -74.0060], zoom_star

# Add price heatmap
HeatMap(price_heat_data, min_opacity=0.3, radius=10, blur=15).add_to(flexible_map_with_
```

```
# Add neighbourhood boundaries
folium.GeoJson(
    geojson_path,
    name="Neighbourhood Boundaries",
    style_function=lambda x: {
        'fillColor': 'none',
        'color': 'blue',
        'weight': 2
    }
).add_to(flexible_map_with_enhanced_markers)

# Add custom markers for high-demand, low-availability neighborhoods (Green)
for _, row in high_demand_low_availability.iterrows():
    latitude = data[data['neighbourhood'] == row['neighbourhood']]['latitude'].mean()
    longitude = data[data['neighbourhood'] == row['neighbourhood']]['longitude'].mean()
    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, "Ideal: High Demand, Low Availability"),
            icon=folium.Icon(color='green', icon='star', prefix='fa')  # Star icon
        ).add_to(flexible_map_with_enhanced_markers)

# Add custom markers for borderline neighborhoods (Orange)
for _, row in borderline_neighbourhoods.iterrows():
    latitude = data[data['neighbourhood'] == row['neighbourhood']]['latitude'].mean()
    longitude = data[data['neighbourhood'] == row['neighbourhood']]['longitude'].mean()
    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, "Potential Opportunity: Borderline Case"),
            icon=folium.Icon(color='orange', icon='circle', prefix='fa')  # Circle icon
        ).add_to(flexible_map_with_enhanced_markers)

# Add custom markers for areas to avoid (Red)
for _, row in avoid_neighbourhoods.iterrows():
    latitude = data[data['neighbourhood'] == row['neighbourhood']]['latitude'].mean()
    longitude = data[data['neighbourhood'] == row['neighbourhood']]['longitude'].mean()
    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, "Avoid: Low Demand, High Availability"),
            icon=folium.Icon(color='red', icon='exclamation-triangle', prefix='fa')  #
        ).add_to(flexible_map_with_enhanced_markers)

# Define our map legend
legend_html = """
<div style="
position: fixed;
bottom: 50px; left: 50px; width: 350px; height: 130px;
background-color: white;
border: 2px solid grey;
border-radius: 5px;
z-index:9999;
font-size:14px;
padding: 10px 15px;
line-height: 1.8;">
<b>Legend</b><br>
<i class="fa fa-star" style="color: green; font-size: 16px; margin-right: 10px;"></i>
Ideal: High Demand, Low Availability<br>
<i class="fa fa-circle" style="color: orange; font-size: 16px; margin-right: 10px;"></i
Potential Opportunity: Borderline Case<br>
<i class="fa fa-exclamation-triangle" style="color: red; font-size: 16px; margin-right:
Avoid: Low Demand, High Availability
</div>
"""

flexible_map_with_enhanced_markers.get_root().html.add_child(folium.Element(legend_html

# Save the map
flexible_map_with_enhanced_markers.save("../visualisations/map.html")
print("Enhanced map saved as '../visualisations/map.html'.")
```

Neighbourhood clustering map

```
from sklearn.cluster import KMeans
from folium.plugins import MarkerCluster, HeatMap

# Refine thresholds dynamically using percentiles
availability_threshold = availability_price_analysis['avg_availability'].quantile(0.25)
review_threshold = availability_price_analysis['avg_reviews'].quantile(0.75)  # High der

# Define tolerances for borderline cases using median-based flexibility
availability_tolerance = availability_price_analysis['avg_availability'].quantile(0.50)
review_tolerance = availability_price_analysis['avg_reviews'].quantile(0.50)  # Median i

# Assign a marker type for each neighborhood
def get_marker_type(row):
    if row['avg_reviews'] >= review_threshold and row['avg_availability'] <= availabilit
        return 'green'  # High-demand, low-availability
    elif row['avg_reviews'] <= review_threshold / 5 and row['avg_availability'] >= avai
        return 'red'  # Avoidable neighborhoods
    else:
        return 'orange'  # Borderline cases

# Add marker type column
availability_price_analysis['marker_type'] = availability_price_analysis.apply(get_marke

# Create high-demand, low-availability neighborhoods (Green)
high_demand_low_availability = availability_price_analysis[availability_price_analysis[

# Create borderline neighborhoods (Orange)
borderline_neighbourhoods = availability_price_analysis[availability_price_analysis['ma

# Create neighborhoods to avoid (Red)
avoid_neighbourhoods = availability_price_analysis[availability_price_analysis['marker_t

# Merge latitude and longitude into availability_price_analysis if missing
availability_price_analysis = availability_price_analysis.merge(
    data[['neighbourhood', 'latitude', 'longitude']],
    on='neighbourhood',
    how='left'
)

# Verify latitude and longitude presence
if 'latitude' not in availability_price_analysis.columns or 'longitude' not in availabil
    raise KeyError("The 'latitude' and 'longitude' columns are missing after the merge."

# Prepare data for K-means clustering
features = availability_price_analysis[['avg_price', 'avg_reviews', 'avg_availability']]
kmeans = KMeans(n_clusters=3, random_state=42)  # Adjust the number of clusters as neede
availability_price_analysis['cluster'] = kmeans.fit_predict(features)

# Cluster color assignment based on dominant marker type in the cluster
cluster_marker_summary = availability_price_analysis.groupby('cluster')['marker_type'].v
```

Multiple refression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

X = data[['neighbourhood_group', 'neighbourhood', 'room_type', 'minimum_nights', 'numbe
          'reviews_per_month', 'calculated_host_listings_count', 'distance_to_Times_Squ
          'distance_to_Statue_of_Liberty', 'distance_to_Central_Park', 'distance_to_Emp
          'distance_to_Brooklyn_Bridge', 'distance_to_One_World_Trade_Center', 'distanc
y = data['price']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['minimum_nights', 'number_of_reviews', 'reviews_per_
                                   'calculated_host_listings_count', 'distance_to_Times
                                   'distance_to_Statue_of_Liberty', 'distance_to_Centra
                                   'distance_to_Empire_State_Building', 'distance_to_Br
                                   'distance_to_One_World_Trade_Center', 'distance_to_M

        ('cat', OneHotEncoder(handle_unknown='ignore'), ['neighbourhood_group', 'neighb
    ])

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

model.fit(X_train, y_train)

print(f"Training R-squared: {model.score(X_train, y_train)}")
print(f"Test R-squared: {model.score(X_test, y_test)}")
```

```
Training R-squared: 0.5365612165151818
Test R-squared: 0.527005711297084
```

```
# Assign dominant color to each cluster
def assign_cluster_color(row):
    if row['green'] >= row['orange'] and row['green'] >= row['red']:
        return 'green'
    elif row['red'] >= row['green'] and row['red'] >= row['orange']:
        return 'red'
    else:
        return 'orange'

cluster_marker_summary['cluster_color'] = cluster_marker_summary.apply(assign_cluster_c

# Define the popup creation function
def create_popup(row, category):
    # Example placeholder trend (replace with real calculations as needed)
    price_trend = "Price trend: Upward"

    # Top listings approximation using 'number_of_reviews'
    top_rated = data[data['neighbourhood'] == row['neighbourhood']].sort_values('number
    top_listings = "<br>".join([
        f"- {listing['room_type']} (${listing['price']}, {listing['number_of_reviews']}
        for _, listing in top_rated.iterrows()
    ])

    return folium.Popup(
        f"<b>{category}</b><br>"
        f"<b>Neighborhood:</b> {row['neighbourhood']}<br>"
        f"<b>Average Reviews:</b> {row['avg_reviews']:.2f}<br>"
        f"<b>Average Availability:</b> {row['avg_availability']:.2f}%<br>"
        f"<b>Average Price:</b> ${row['avg_price']:.2f}<br>"
        f"<b>{price_trend}</b><br>"
        f"<b>Top Listings:</b><br>{top_listings}",
        max_width=300
    )

# Create the clustered map
clustered_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)
marker_cluster = MarkerCluster().add_to(clustered_map)

# Add price heatmap
HeatMap(price_heat_data, min_opacity=0.3, radius=10, blur=15).add_to(clustered_map)

# Add borough boundaries
folium.GeoJson(
    geojson_path,
    name="Neighbourhood Boundaries",
    style_function=lambda x: {
        'fillColor': 'none',
        'color': 'blue',
        'weight': 2
    }
).add_to(clustered_map)

# Add individual markers for each neighborhood
for _, row in availability_price_analysis.iterrows():
    latitude = row['latitude']
    longitude = row['longitude']
    marker_type = row['marker_type']

    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, f"Neighborhood Type: {marker_type.capitalize()}"),
            icon=folium.Icon(color=marker_type, icon='info-sign')
        ).add_to(marker_cluster)
```

```python
# Add cluster markers with dominant colors
for cluster_id, row in cluster_marker_summary.iterrows():
    cluster_data = availability_price_analysis[availability_price_analysis['cluster'] =
    latitude = cluster_data['latitude'].mean()
    longitude = cluster_data['longitude'].mean()
    cluster_color = row['cluster_color']

    folium.Marker(
        location=[latitude, longitude],
        popup=(
            f"<b>Cluster {cluster_id}</b><br>"
            f"Dominant Color: {cluster_color}<br>"
            f"Total Green: {row['green']}<br>"
            f"Total Orange: {row['orange']}<br>"
            f"Total Red: {row['red']}"
        ),
        icon=folium.Icon(color=cluster_color, icon='info-sign')
    ).add_to(clustered_map)

# Define a custom legend
legend_html = """
<div style="
position: fixed;
bottom: 50px; left: 50px; width: 350px; height: 130px;
background-color: white;
border: 2px solid grey;
border-radius: 5px;
z-index:9999;
font-size:14px;
padding: 10px 15px;
line-height: 1.8;">
<b>Legend</b><br>
<i class="fa fa-star" style="color: green; font-size: 16px; margin-right: 10px;"></i>
Ideal: High Demand, Low Availability<br>
<i class="fa fa-circle" style="color: orange; font-size: 16px; margin-right: 10px;"></i
Potential Opportunity: Borderline Case<br>
<i class="fa fa-exclamation-triangle" style="color: red; font-size: 16px; margin-right:
Avoid: Low Demand, High Availability
</div>
"""

clustered_map.get_root().html.add_child(folium.Element(legend_html))

# Save the clustered map
clustered_map.save("../visualisations/clustered_map_with_dominant_colors.html")
print("Clustered map with dominant colors saved as '../visualisations/clustered_map_wit
```

```python
from folium.plugins import MarkerCluster
import geopandas as gpd

# Load GeoJSON data for neighborhoods if centroids are needed
geo_data = gpd.read_file(geojson_path)
geo_data['centroid'] = geo_data['geometry'].centroid

# Create a clustered map
clustered_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)
marker_cluster = MarkerCluster().add_to(clustered_map)

# Add markers to the cluster for high-demand neighborhoods (Green)
for _, row in high_demand_low_availability.iterrows():
    centroid = geo_data[geo_data['BoroName'] == row['neighbourhood']]['centroid']
    latitude = centroid.y.values[0] if not centroid.empty else data[data['neighbourhood
    longitude = centroid.x.values[0] if not centroid.empty else data[data['neighbourhoo

    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, "Ideal: High Demand, Low Availability"),
            icon=folium.Icon(color='green', icon='star', prefix='fa')
        ).add_to(marker_cluster)

# Add markers to the cluster for borderline neighborhoods (Orange)
for _, row in borderline_neighbourhoods.iterrows():
    centroid = geo_data[geo_data['BoroName'] == row['neighbourhood']]['centroid']
    latitude = centroid.y.values[0] if not centroid.empty else data[data['neighbourhood
    longitude = centroid.x.values[0] if not centroid.empty else data[data['neighbourhoo

    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, "Potential Opportunity: Borderline Case"),
            icon=folium.Icon(color='orange', icon='circle', prefix='fa')
        ).add_to(marker_cluster)

# Add markers to the cluster for areas to avoid (Red)
for _, row in avoid_neighbourhoods.iterrows():
    centroid = geo_data[geo_data['BoroName'] == row['neighbourhood']]['centroid']
    latitude = centroid.y.values[0] if not centroid.empty else data[data['neighbourhood
    longitude = centroid.x.values[0] if not centroid.empty else data[data['neighbourhoo

    if pd.notnull(latitude) and pd.notnull(longitude):
        folium.Marker(
            location=[latitude, longitude],
            popup=create_popup(row, "Avoid: Low Demand, High Availability"),
            icon=folium.Icon(color='red', icon='exclamation-triangle', prefix='fa')
        ).add_to(marker_cluster)

# Save the clustered map
clustered_map.save("../visualisations/clustered_map.html")
print("Clustered map saved as '../visualisations/clustered_map.html'.")
```

```
data['predicted_price'] = model.predict(X)
neighbourhood_analysis = data.groupby('neighbourhood')[['price', 'predicted_price']].me
neighbourhood_analysis['price_gap'] = neighbourhood_analysis['predicted_price'] - neigh
neighbourhood_analysis.sort_values('price_gap', ascending=False, inplace=True)

print(neighbourhood_analysis.head())
```

```
                        price  predicted_price   price_gap
neighbourhood
Castleton Corners  171.333333       281.031329  109.697996
New Dorp            57.000000       121.839629   64.839629
Belle Harbor       146.000000       179.660903   33.660903
Emerson Hill        68.200000       100.024010   31.824010
Huguenot           118.333333       140.459281   22.125948
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Select features for clustering
features = availability_price_analysis[['avg_price', 'avg_reviews', 'avg_availability']]

# Handle missing values (if any)
features = features.dropna()

# Scale the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Convert back to a DataFrame for clarity
scaled_features_df = pd.DataFrame(scaled_features, columns=['avg_price', 'avg_reviews',
```

```
import matplotlib.pyplot as plt

# Calculate inertia for different k values
inertia = []
k_values = range(1, 10)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.show()
```

```
data['predicted_price'] = model.predict(X)
neighbourhood_analysis = data.groupby('neighbourhood')[['price', 'predicted_price']].me
neighbourhood_analysis['price_gap'] = neighbourhood_analysis['predicted_price'] - neigh
neighbourhood_analysis.sort_values('price_gap', ascending=False, inplace=True)

print(neighbourhood_analysis.head())
```

```
                        price  predicted_price   price_gap
neighbourhood
Castleton Corners  171.333333       281.031329  109.697996
New Dorp            57.000000       121.839629   64.839629
Belle Harbor       146.000000       179.660903   33.660903
Emerson Hill        68.200000       100.024010   31.824010
Huguenot           118.333333       140.459281   22.125948
```

```
import matplotlib.pyplot as plt


top_neighbourhoods = neighbourhood_analysis.head(10)

plt.figure(figsize=(12, 6))
top_neighbourhoods['price_gap'].plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Top Neighbourhoods with the Largest Price Gaps', fontsize=16)
plt.xlabel('Neighbourhood', fontsize=14)
plt.ylabel('Price Gap (Predicted - Actual)', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.tight_layout()
plt.show()
```

Random forest regression:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Prepare data (assume `data` is already loaded as a DataFrame)
data['sum_of_reviews'] = data.groupby('neighbourhood')['number_of_reviews'].transform('

X = data[['neighbourhood_group', 'neighbourhood', 'room_type', 'sum_of_reviews',
          'distance_to_Times_Square', 'distance_to_Statue_of_Liberty', 'distance_to_Cen
          'distance_to_Empire_State_Building', 'distance_to_Brooklyn_Bridge',
          'distance_to_One_World_Trade_Center', 'distance_to_Metropolitan_Museum_of_Art

y = data['price']  # Changed target variable to 'price'

# Preprocessing for numerical and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['sum_of_reviews', 'distance_to_Times_Square',
                                   'distance_to_Statue_of_Liberty', 'distance_to_Centra
                                   'distance_to_Empire_State_Building', 'distance_to_Br
                                   'distance_to_One_World_Trade_Center', 'distance_to_M
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['neighbourhood_group', 'neighb
    ])

# Define pipeline with Random Forest, limit max depth to 5
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42, max_depth=5)
])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Fit the model
model.fit(X_train, y_train)

# Evaluate the model
print(f"Training R-squared: {model.score(X_train, y_train):.4f}")
print(f"Test R-squared: {model.score(X_test, y_test):.4f}")
```

```
Training R-squared: 0.5300
Test R-squared: 0.5377
```

```
import numpy as np
import pandas as pd

# Predict prices using the trained model
y_pred = model.predict(X_test)

# Create a DataFrame to compare actual vs predicted prices
comparison_df = pd.DataFrame({
    'Actual Price': y_test,
    'Predicted Price': y_pred
})

# Calculate the min, max, average, and quartiles for both actual and predicted prices
statistics = comparison_df.describe().T[['min', '25%', '50%', '75%', 'max', 'mean']]

# Display the statistics for actual vs predicted prices
print("Statistics for Actual vs Predicted Prices:")
print(statistics)
```

```
Statistics for Actual vs Predicted Prices:
                       min        25%         50%         75%         max  \
Actual Price     22.000000  65.000000  100.000000  150.000000  299.000000
Predicted Price  66.358711  66.418048  117.520517  151.454927  188.923513

                       mean
Actual Price     114.726884
Predicted Price  115.295819
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

def plot_predicted_vs_actual(model, X_test, y_test):
    y_pred = model.predict(X_test)

    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', l
    plt.xlabel('Actual Price')
    plt.ylabel('Predicted Price')
    plt.title('Actual vs Predicted Price')
    plt.show()



# Predicted vs Actual Plot
plot_predicted_vs_actual(model, X_test, y_test)
```

Random Forest regression 2

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Use only proximity to landmarks as features
landmark_features = [
    'distance_to_Times_Square',
    'distance_to_Statue_of_Liberty',
    'distance_to_Central_Park',
    'distance_to_Empire_State_Building',
    'distance_to_Brooklyn_Bridge',
    'distance_to_One_World_Trade_Center',
    'distance_to_Metropolitan_Museum_of_Art'
]

X = data[landmark_features]
y = data['price']

# Preprocessing for numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), landmark_features)
    ])

# Define pipeline with Random Forest, limit max depth to 7
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42, max_depth=7)
])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Fit the model
model.fit(X_train, y_train)

# Evaluate the model
print(f"Training R-squared: {model.score(X_train, y_train):.4f}")
print(f"Test R-squared: {model.score(X_test, y_test):.4f}")
```

```
Training R-squared: 0.2811
Test R-squared: 0.2548
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Get the feature importances from the Random Forest model
feature_importances = model.named_steps['regressor'].feature_importances_

# Plot Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances, y=landmark_features, palette='viridis')
plt.title('Feature Importance for Predicting Price')
plt.xlabel('Importance')
plt.ylabel('Landmark Proximity Features')
plt.show()

# Make predictions
y_pred = model.predict(X_test)

# Scatter Plot of Actual vs Predicted Prices
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, edgecolors='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2, label
plt.title('Actual vs Predicted Prices')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.legend()
plt.show()

# Residual Plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True, color='purple', bins=30)
plt.title('Residuals Distribution (Actual - Predicted Prices)')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.axvline(0, color='r', linestyle='--', label='Perfect Prediction')
plt.legend()
plt.show()
```

```python
# Add the neighbourhood column to the X_test DataFrame for grouping purposes
X_test_with_neighbourhood = X_test.copy()
X_test_with_neighbourhood['neighbourhood'] = data.loc[X_test.index, 'neighbourhood']

# Predict prices using the trained model
y_pred = model.predict(X_test)

# Create a DataFrame to compare actual vs predicted prices, including the neighbourhoo
comparison_df = pd.DataFrame({
    'Actual Price': y_test,
    'Predicted Price': y_pred,
    'neighbourhood': X_test_with_neighbourhood['neighbourhood']
})

# Group by neighbourhood and calculate the statistics for actual vs predicted prices
statistics_by_neighbourhood = comparison_df.groupby('neighbourhood').agg(
    min_actual=('Actual Price', 'min'),
    q1_actual=('Actual Price', lambda x: np.percentile(x, 25)),
    median_actual=('Actual Price', 'median'),
    q3_actual=('Actual Price', lambda x: np.percentile(x, 75)),
    max_actual=('Actual Price', 'max'),
    mean_actual=('Actual Price', 'mean'),

    min_predicted=('Predicted Price', 'min'),
    q1_predicted=('Predicted Price', lambda x: np.percentile(x, 25)),
    median_predicted=('Predicted Price', 'median'),
    q3_predicted=('Predicted Price', lambda x: np.percentile(x, 75)),
    max_predicted=('Predicted Price', 'max'),
    mean_predicted=('Predicted Price', 'mean')
)

# Display the grouped statistics
print("Statistics by Neighbourhood:")
print(statistics_by_neighbourhood)
```

```
Statistics by Neighbourhood:
                 min_actual  q1_actual  median_actual  q3_actual  max_actual  \
neighbourhood
Allerton                 33       38.0           49.0      66.00          80
Arrochar                 33       33.5           34.0      34.50          35
Arverne                  35       90.5          137.0     150.00         250
Astoria                  27       65.0           90.0     125.00         250
Bath Beach               45       48.0           74.0      99.25         100
...                     ...        ...            ...        ...         ...
Williamsburg             27       70.0          100.0     160.00         298
Windsor Terrace          40       70.0          120.0     143.50         250
Woodhaven                30       45.0           50.0      70.50         170
Woodlawn                 70       70.0           70.0      70.00          70
Woodside                 30       40.0           82.5     120.00         195

                 mean_actual  min_predicted  q1_predicted  median_predicted  \
neighbourhood
Allerton           52.875000      66.358711     66.358711         66.358711
Arrochar           34.000000      66.358711     66.358711         66.358711
Arverne           131.000000      66.358711     66.358711        126.080605
Astoria           100.636943      74.713935     75.199961         92.428104
Bath Beach         73.250000      66.358711     66.358711         66.358711
...                      ...            ...           ...               ...
Williamsburg      119.707355      66.418048     79.751267         85.743738
Windsor Terrace   114.400000      66.358711     66.358711        126.080605
Woodhaven          63.222222      66.358711     66.358711         66.358711
Woodlawn           70.000000      66.358711     66.358711         66.358711
Woodside           85.214286      66.358711     66.418048         74.617364

                 q3_predicted  max_predicted  mean_predicted
neighbourhood
Allerton            66.358711     126.080605       73.823948
Arrochar            66.358711      66.358711       66.358711
Arverne            126.080605     126.080605      104.363552
Astoria            135.232961     169.509422      109.029216
Bath Beach          81.289185     126.080605       81.289185
...                       ...            ...             ...
Williamsburg       174.177130     174.652751      119.259046
Windsor Terrace    126.288344     131.095657      104.380769
```

References:

Urvashi, SH. & Gupta, D. (2021) Analyzing the applications of internet of things in hotel industry. Journal of Physics 6(1969): 26-27. Available at: [Accessed 1 December 2024].

Jiao, J., & Bai, S. (2020) Cities reshaped by Airbnb: A case study in New York City, Chicago, and Los Angeles. Environment and Planning A: Economy and Space, 52(1): 10-13. Available at: https://journals.sagepub.com/doi/full/10.1177/0308518X19853275 [Accessed 1 December 2024].

Dgomonov (2019) New York City Airbnb Open Data. Available at: https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data/data [Accessed 10 November 2024].

Airbtics (2023) 'Best Airbnb Markets in New York'. Available at: https://airbtics.com/best-airbnb-markets-in-new-york/ [Accessed 1 December 2024].

Investopedia (2023) 'Early Impacts of New York City's Airbnb Regulations'. Available at: https://www.investopedia.com/new-york-city-enforces-de-facto-ban-of-airbnbs-7967344 [Accessed: 2 December 2024].

Scikit-learn Documentation (2024) 'Clustering Methods'. Available at: https://scikit-learn.org/stable/modules/clustering.html [Accessed: 29 November 2024].

GeoNames.org. (2024) GeoNames geographical database. Available at: https://www.geonames.org/ [Accessed 2 December 2024].

AirDNA (2023) 'Airbnb Data on 34,472 Vacation Rentals in New York, NY'. Available at: https://www.airdna.co/vacation-rental-data/app/us/new-york/new-york/overview [Accessed: 1 December 2024].