Analyzing NYC Airbnb Open Data

Data Science 2024 Bootcamp →

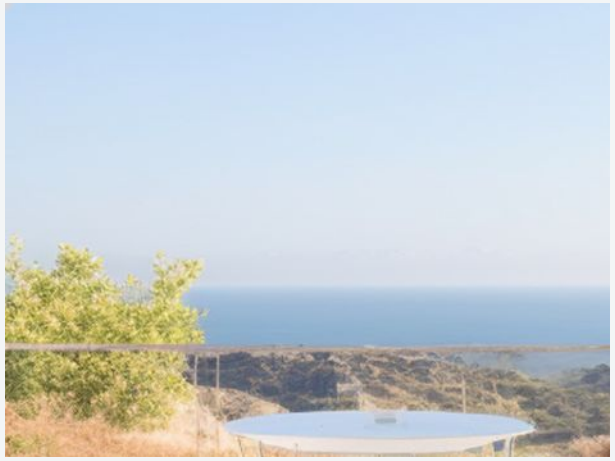
Shihui Feng, Dong Li, Sofia Celorio , Antong Lei, KaiYuan Ma, Tujie Guo

Table of Contents

01

Introduction

Overview of the Project



The main idea of this project is to use the **NYC Airbnb Open Data** to conduct a comprehensive **analysis** that will yield actionable insights for both hosts and potential renters.

By delving into Airbnb's extensive dataset, we hope to uncover patterns, trends, and factors **influencing rental prices** and **demand** in New York City's dynamic accommodation market.

02

Goal & Hypothesis

Current Goal

Goal

Predicting rental prices based on relevant features.

Learn how to use area (locations) to predict apartment pricings

Hypothesis

The size of the location will increase with season –> Affecting Price

Price of Location –> Depends on Availability

More Review –> Less Availability

1. **Holiday** season rents will be **more expensive** due to high demand
2. The **larger** the apts size, the more **pricey** it is
3. On average **Manhattan** apts price will be **more expensive** than other boroughs
4. The **central** the neighborhood, the **higher** the apts price
5. The neighborhood with **better public transportations** have a **higher price**
6. The more **reviews** the apts has, the **less availability** apt will be
7. The **cheaper** the apt is for the **location**, the **less availability** apt has
8. The more **listings**(calculated_host_listings_count) the host has, the **less availability** it would be (due to experience)
9. The **longer characters** the airbnb name has, the **more availability** there is
10. **Entire home/apt renting** has **less availability** than **private room**

03

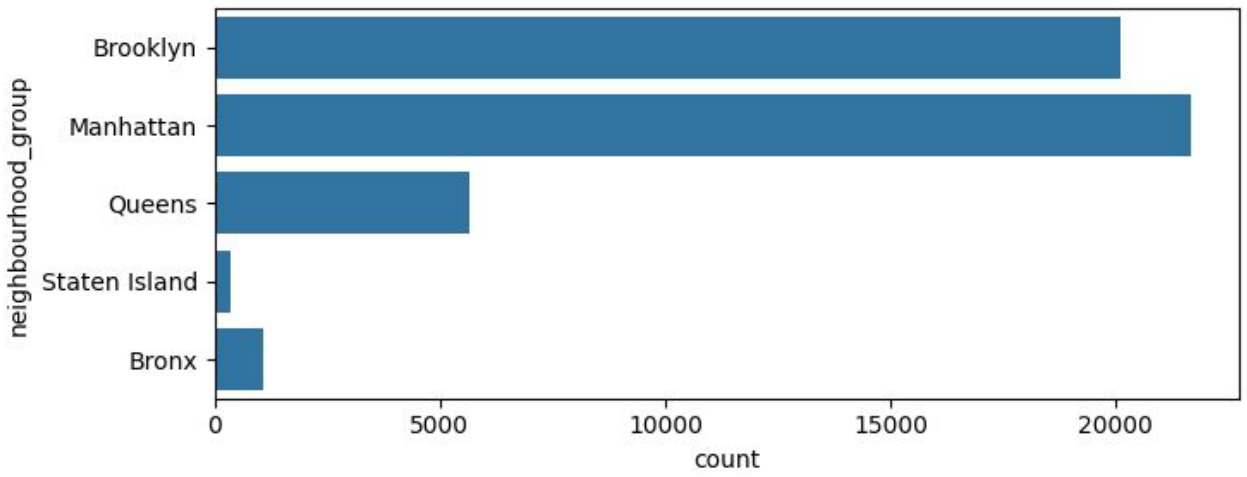
Our Process

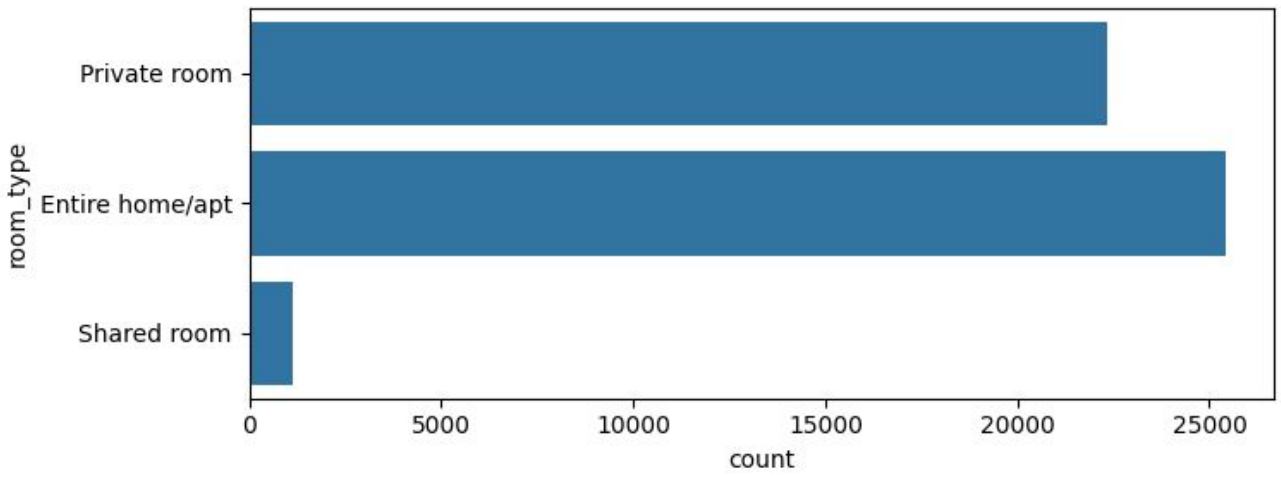
```
#data informtion
df.info()
```

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

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/AB_NYC_2019.csv')
df.head()
```

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





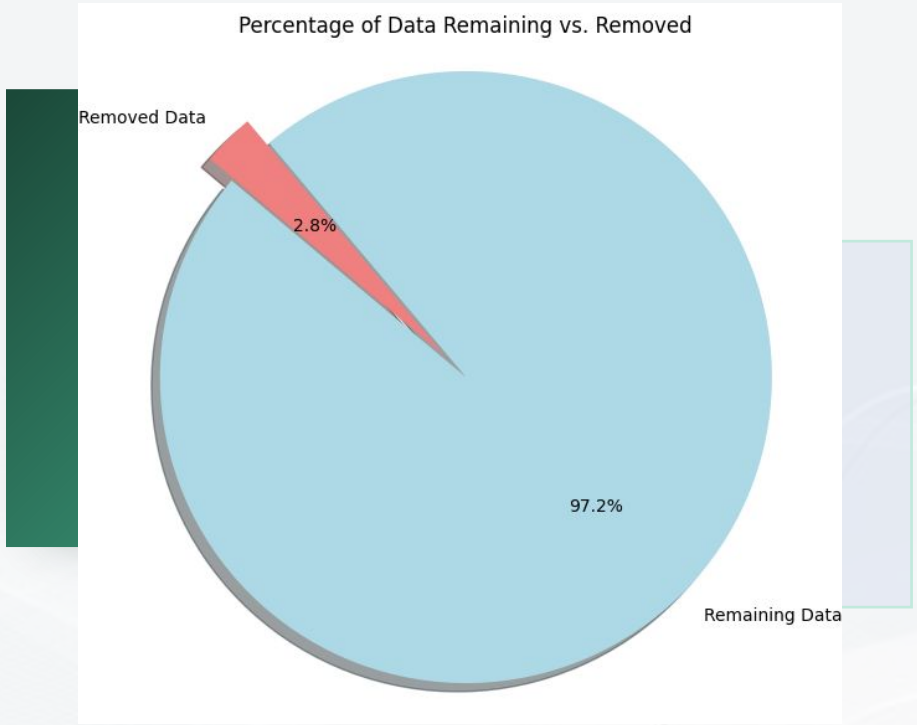
Overview of NYC Airbnb Dataset (Data Description)

- **Data Information**
  - Acquired the NYC Airbnb dataset containing 48,895 entries and 16 columns.
  - Checked the shape and info of the DataFrame to understand its structure and dimensions.
  - Data begin at **2019**
- **Delete Missing Data**:
  - Identified missing data by generating a boolean DataFrame.
  - Summarized missing data by column.
  - Found rows with missing data and specifically targeted columns ('name' and 'host_name').
    - Replaced missing values:
    - Filled missing values in 'name' and 'host_name' with 'unknown'.
    - Imputed missing values in 'reviews_per_month' with 0.
    - Forward-filled missing values in 'last_review' column.

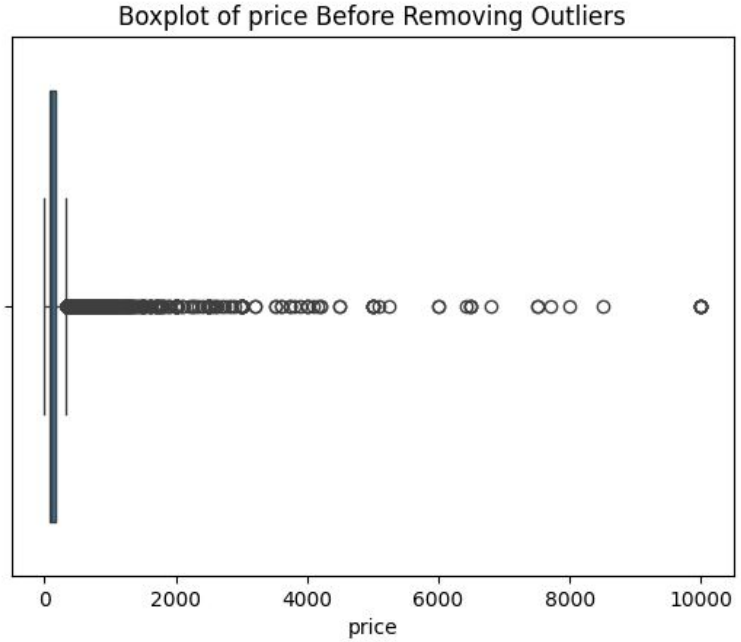
```
id                                0
name                             16
host_id                           0
host_name                        21
neighbourhood_group               0
neighbourhood                     0
latitude                          0
longitude                         0
room_type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

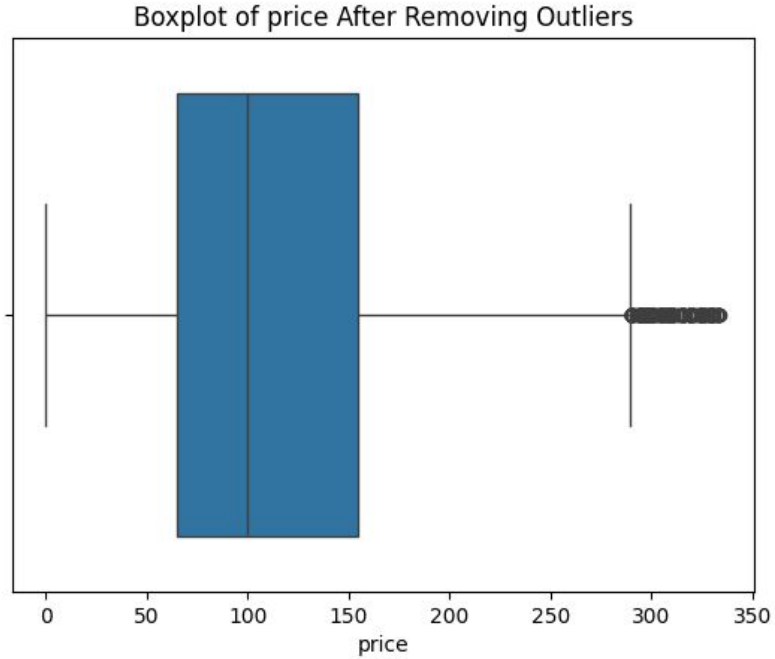
```
id                                0
name                              0
host_id                           0
host_name                         0
neighbourhood_group               0
neighbourhood                     0
latitude                          0
longitude                         0
room_type                         0
price                             0
minimum_nights                    0
number_of_reviews                 0
last_review                       0
reviews_per_month                 0
calculated_host_listings_count    0
availability_365                  0
dtype: int64
```

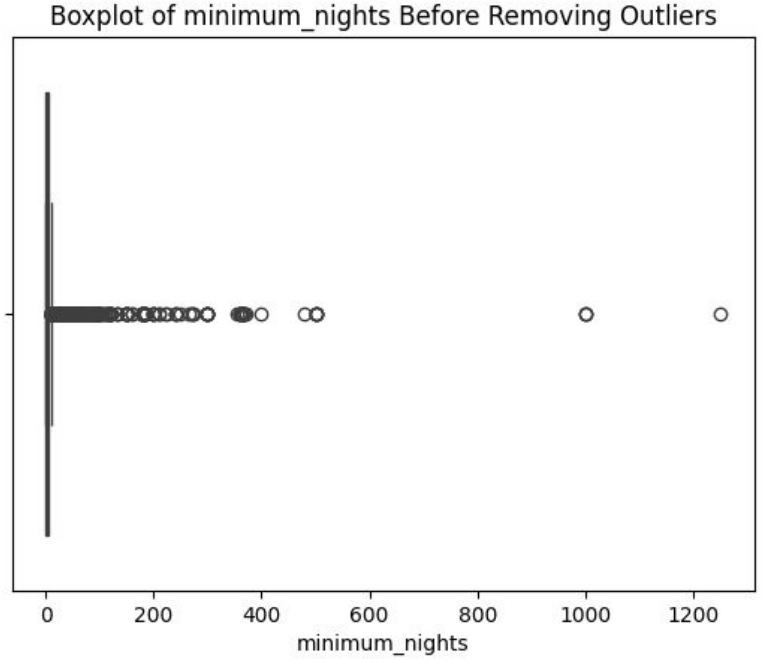
Distribution of features and identification of patterns

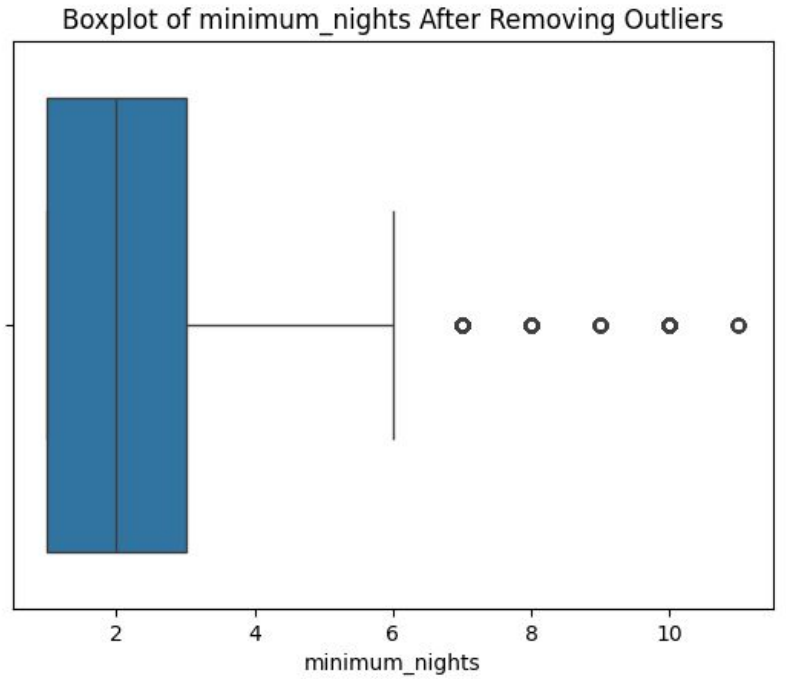


- **Removal**:
  - Detected outliers in the 'price' column using the Interquartile Range (IQR) method.
  - Calculated lower and upper bounds for outlier detection.
  - Removed outliers falling outside the defined bounds.
  - Compared the original data size (48,895 rows) with the size after outlier removal.
- **Visualization**:
  - Plotted a pie chart to illustrate the percentage of data remaining after outlier removal compared to the removed data.
  - Remaining Data- 97.2%
  - Removed Data - 2.8 %


Boxplot of price Before Removing Outliers
0
2000
4000
6000
8000
10000
price


Boxplot of price After Removing Outliers
0
50
100
150
200
250
300
350
price


Boxplot of minimum_nights Before Removing Outliers
0
200
400
600
800
1000
1200
minimum_nights


Boxplot of minimum_nights After Removing Outliers
2
4
6
8
10
minimum_nights

04

Data Visualization

Find the Median price for each neighborhood

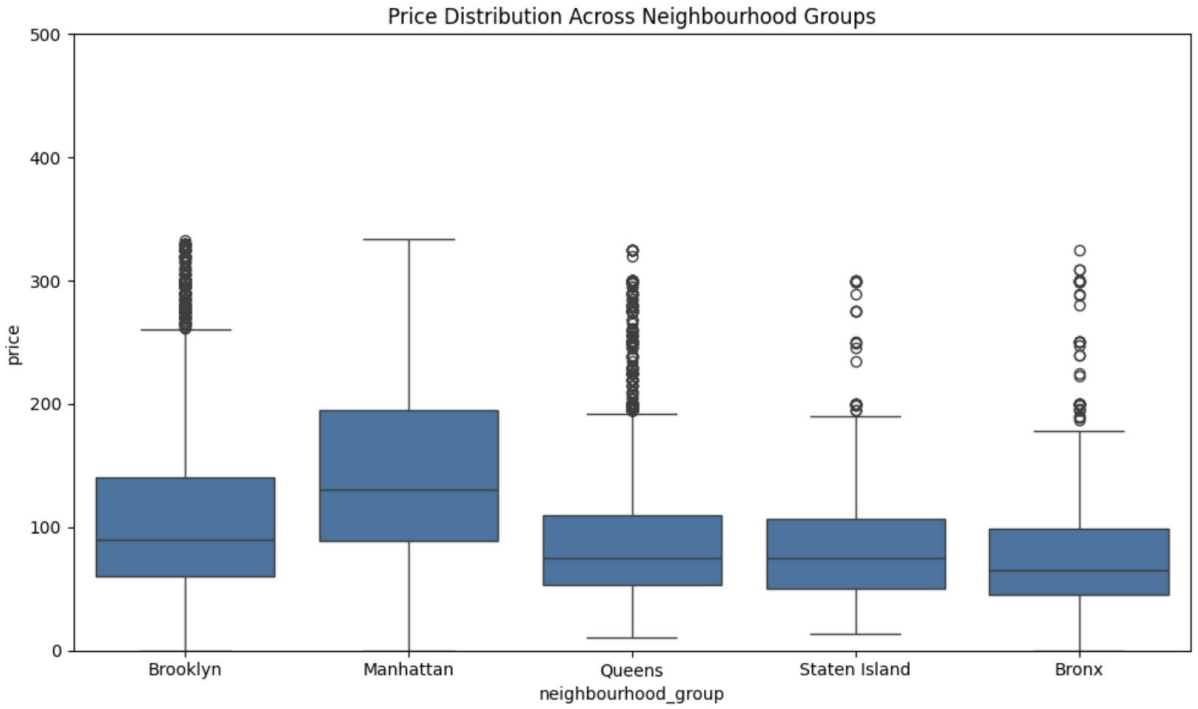
Median Price Analysis:

a. Get the median price per neighborhood group for every column.
b. Utilized **groupby** to calculate and display median prices per neighborhood group, both with and without outliers.

neighbourhood_group	mean	min	max
Bronx	87.496792	0	2500
Brooklyn	124.383207	0	10000
Manhattan	196.875814	0	10000
Queens	99.517649	10	10000
Staten Island	114.812332	13	5000

neighbourhood_group	mean	min	max
Bronx	77.365421	0	325
Brooklyn	105.699614	0	333
Manhattan	145.952835	0	334
Queens	88.904437	10	325
Staten Island	89.235616	13	300

Groupby Neighborhood box plot



- Price distribution across neighborhood groups using box plots.
- Displayed the number of listings in each neighborhood group using count plots.
- Plotted bar graphs showing the median price in each neighborhood group.
- correlation relationships between 'price', 'reviews_per_month', 'calculated_host_listings_count', 'availability_365', and 'minimum_nights'.

```
df_cleaned = df.dropna()
correlation_matrix_all_data = df_cleaned[['price', 'reviews_per_month', 'availability_365','minimum_nights','calculated_host_listings_count']].corr()
correlation_matrix_all_data
```

	price	reviews_per_month	availability_365	minimum_nights	calculated_host_listings_count
price	1.000000	-0.043289	0.027122	0.059901	0.088708
reviews_per_month	-0.043289	1.000000	0.254287	-0.231585	0.036455
availability_365	0.027122	0.254287	1.000000	-0.097462	0.129584
minimum_nights	0.059901	-0.231585	-0.097462	1.000000	-0.031173
calculated_host_listings_count	0.088708	0.036455	0.129584	-0.031173	1.000000

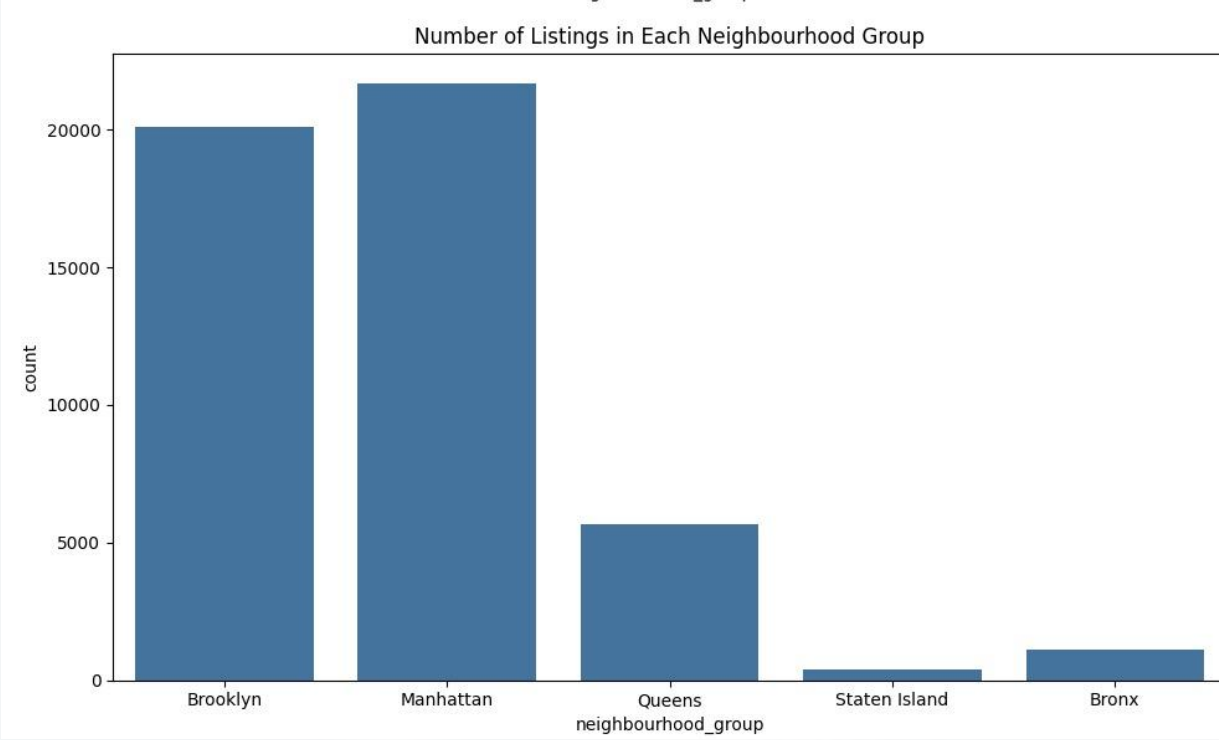
Median Price + Housing Numbers

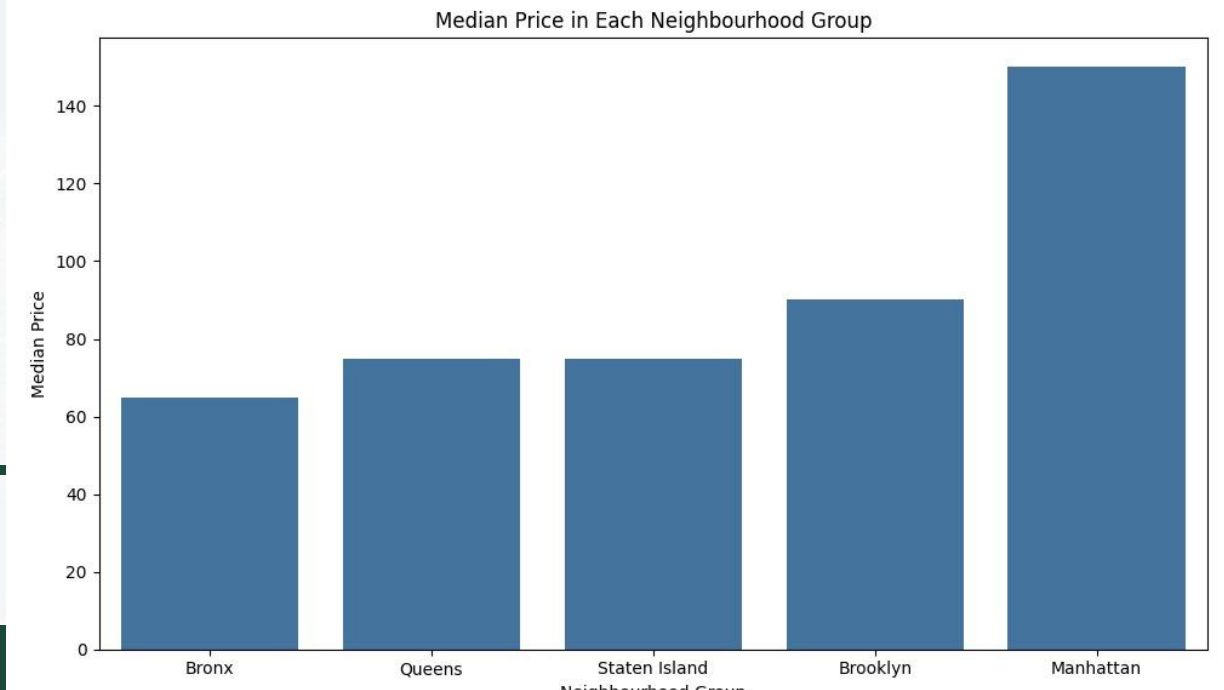
From the Number of Housing :

- Manhattan & Brooklyn have the most housing availability and listing at NYC
- Queens, Bronx & Staten Island are less

From the Median Price:

- Manhattan has the Highest Median Price
- The other parts (Bronx, Queens, Staten Island & Brooklyn) are mostly the same



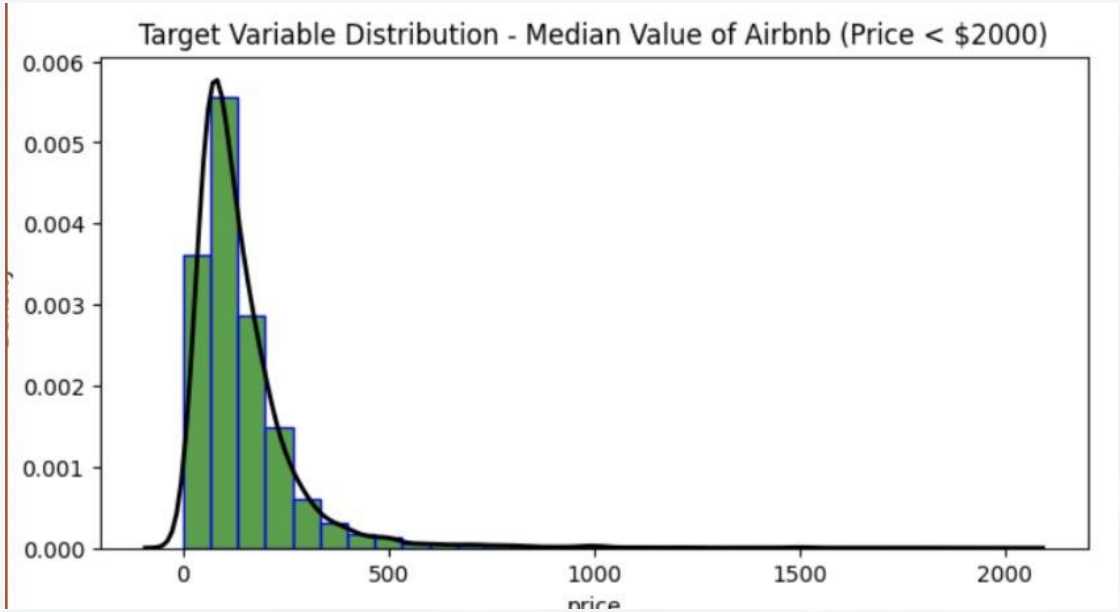


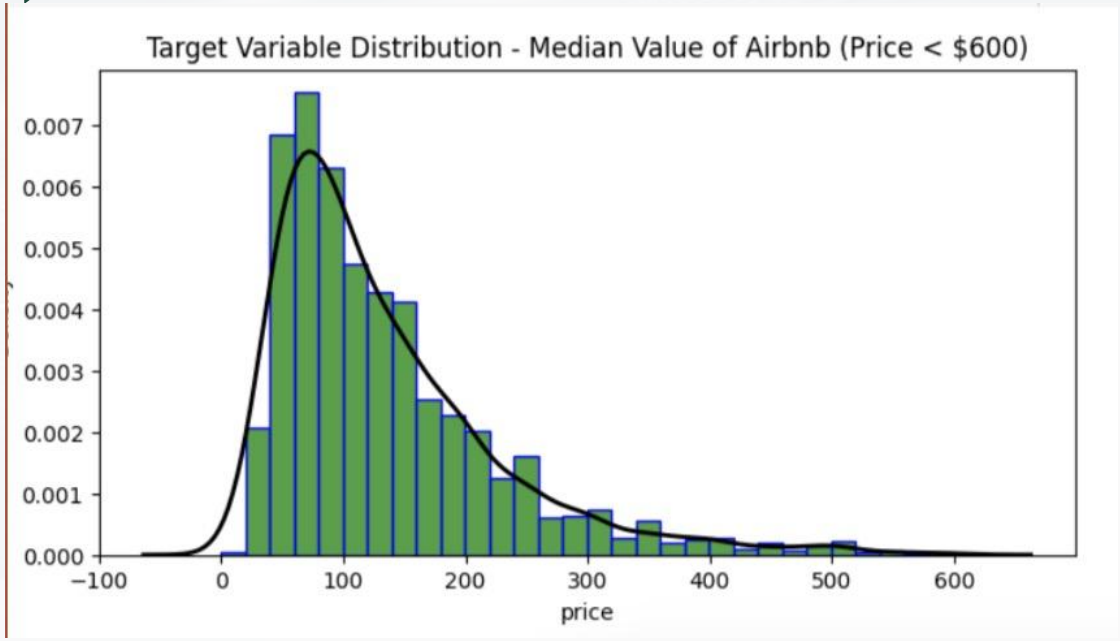
Median Value of Airbnb

Target Variable Distribution:

- Visualized the distribution of the target variable 'price' using histograms with Kernel Density Estimation (KDE).
- Plotted histograms and KDE plots for the entire dataset and subsets filtered for prices less than $600

The Median Value of NYC will be at the range (0<x<200)



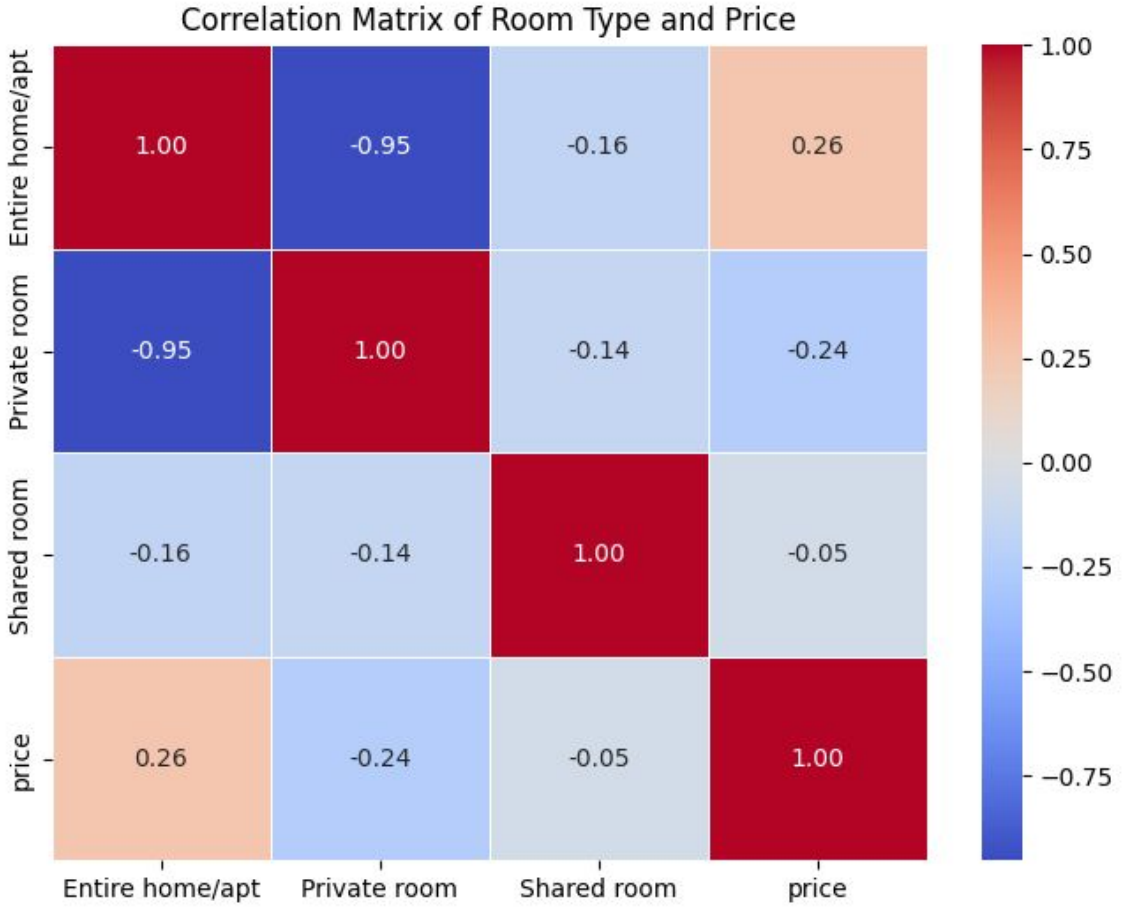


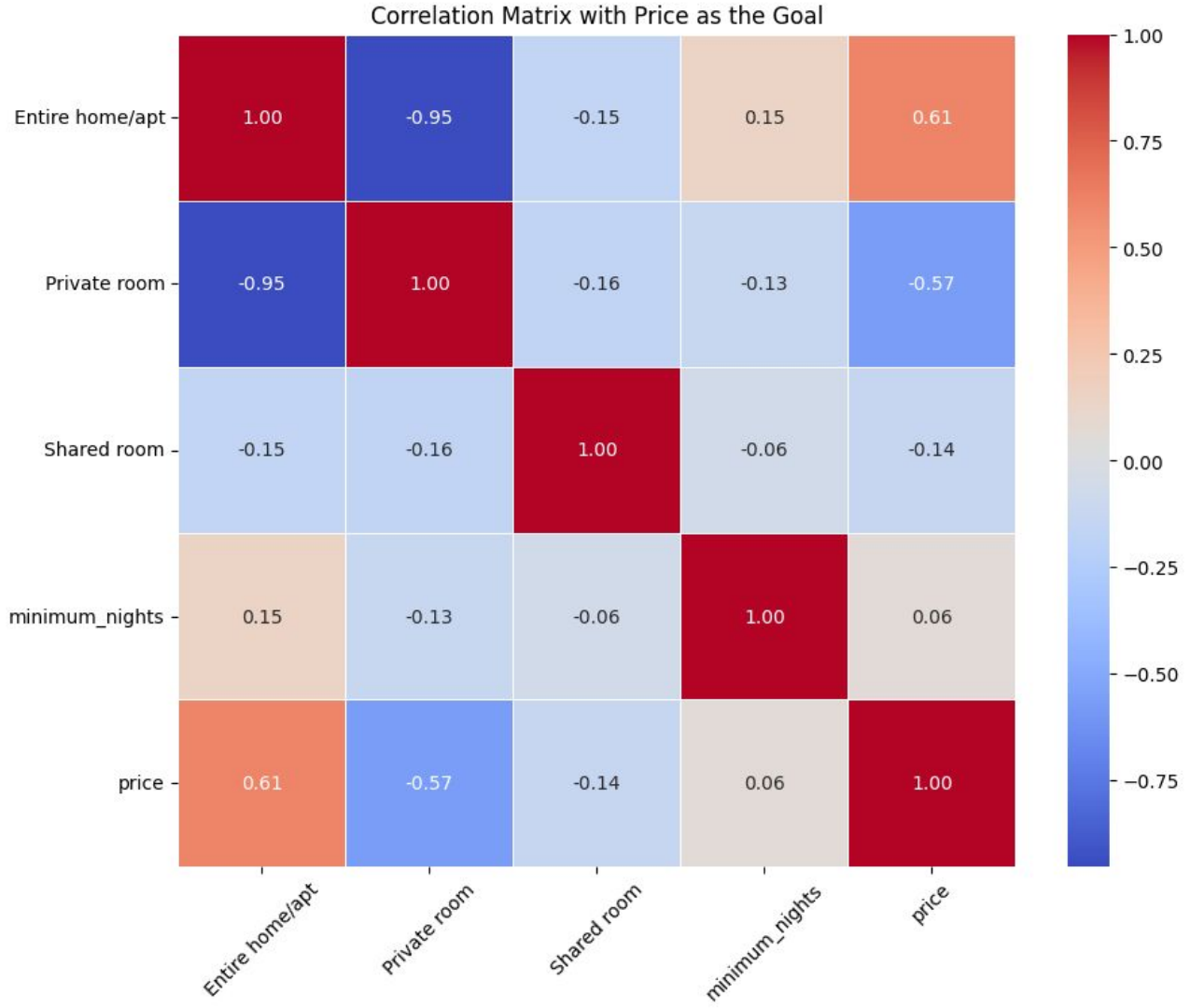
Data Transformation

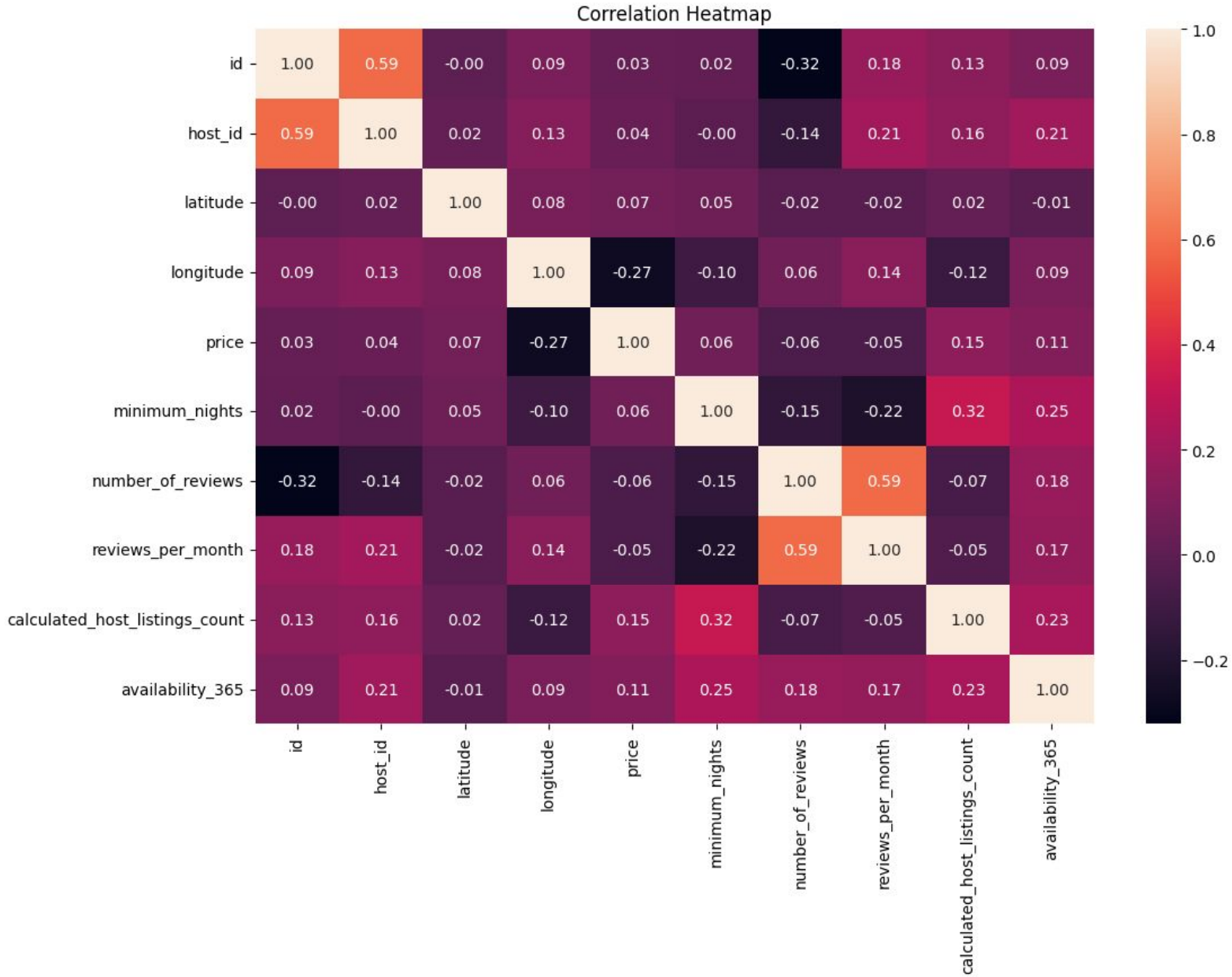
- **Data Transformation**:
  i. Created the list named room_type containing the possible values for the 'room_type' column: **'Entire home/apt', 'Private room', and 'Shared room**'.
- **Encoding to Dummy Variables**:
  i. Room type and neighborhood group are both strings, and they both have to be dummy variables before you can see how they relate to price.
  ii. Used pd.get_dummies() function to encode the 'room_type' column into dummy variables.
  iii. Specified the columns parameter as ['room_type'] to indicate the column to be encoded.
  iv. Set the dummy_na parameter to False to avoid creating dummy variables for any potential missing values.
  v. Assigned the resulting DataFrame to df_encoded.
- **Output Display**:
  i. Printed the DataFrame df_encoded to show the encoded representation of the 'room_type' column using dummy variables.


Correlation Matrix of Neighbourhood Group and Price
Bronx
Brooklyn
Manhattan
Queens
Staten Island
price
1.00 -0.13 -0.13 -0.05 -0.01 -0.04
-0.13 1.00 -0.75 -0.30 -0.07 -0.10
-0.13 -0.75 1.00 -0.32 -0.08 0.16
-0.05 -0.30 -0.32 1.00 -0.03 -0.08
-0.01 -0.07 -0.08 -0.03 1.00 -0.01
-0.04 -0.10 0.16 -0.08 -0.01 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Correlations






Correlation Heatmap
id 1.00 0.59 -0.00 0.09 0.03 0.02 -0.32 0.18 0.13 0.09
host_id 0.59 1.00 0.02 0.13 0.04 -0.00 -0.14 0.21 0.16 0.21
latitude -0.00 0.02 1.00 0.08 0.07 0.05 -0.02 -0.02 0.02 -0.01
longitude 0.09 0.13 0.08 1.00 -0.27 -0.10 0.06 0.14 -0.12 0.09
price 0.03 0.04 0.07 -0.27 1.00 0.06 -0.06 -0.05 0.15 0.11
minimum_nights 0.02 -0.00 0.05 -0.10 0.06 1.00 -0.15 -0.22 0.32 0.25
number_of_reviews -0.32 -0.14 -0.02 0.06 -0.06 -0.15 1.00 0.59 -0.07 0.18
reviews_per_month 0.18 0.21 -0.02 0.14 -0.05 -0.22 0.59 1.00 -0.05 0.17
calculated_host_listings_count 0.13 0.16 0.02 -0.12 0.15 0.32 -0.07 -0.05 1.00 0.23
availability_365 0.09 0.21 -0.01 0.09 0.11 0.25 0.18 0.17 0.23 1.00
id
host_id
latitude
longitude
price
minimum_nights
number_of_reviews
reviews_per_month
calculated_host_listings_count
availability_365
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

05

Model Selection

Features Used:

- neighbourhood_group, neighbourhood, room_type: Categorical features that categorize the listings. These are key determinants of pricing dynamics in the rental market.
- latitude, longitude: Location coordinates that capture the geographical nuances of pricing.
- minimum_nights, number_of_reviews, reviews_per_month, availability_365, calculated_host_listings_count: Numeric features that describe the listing's popularity, demand, and operational metrics which can influence price.

Linear Regression only for neighbourhood_group and room_type

```
y=df_encoded2['price']
x=df_encoded2.drop(columns=['price'])

df_encoded.iloc[:,15:18].shape
df_encoded2.iloc[:,14:19].shape
X=pd.concat([df_encoded.iloc[:,15:18],df_encoded2.iloc[:,14:19]],axis=1)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("Training Set Dimensions:", x_train.shape)
print("Validation Set Dimensions:", x_test.shape)
```

```
reg=LinearRegression()
reg.fit(x_train, y_train)
y_pred=reg.predict(x_train)
mse=mean_squared_error(y_pred,y_train)
print('mean_squared_error:',mse)
```

mean_squared_error: 7177.255675589521 ——-------- > **need to update and change our model**

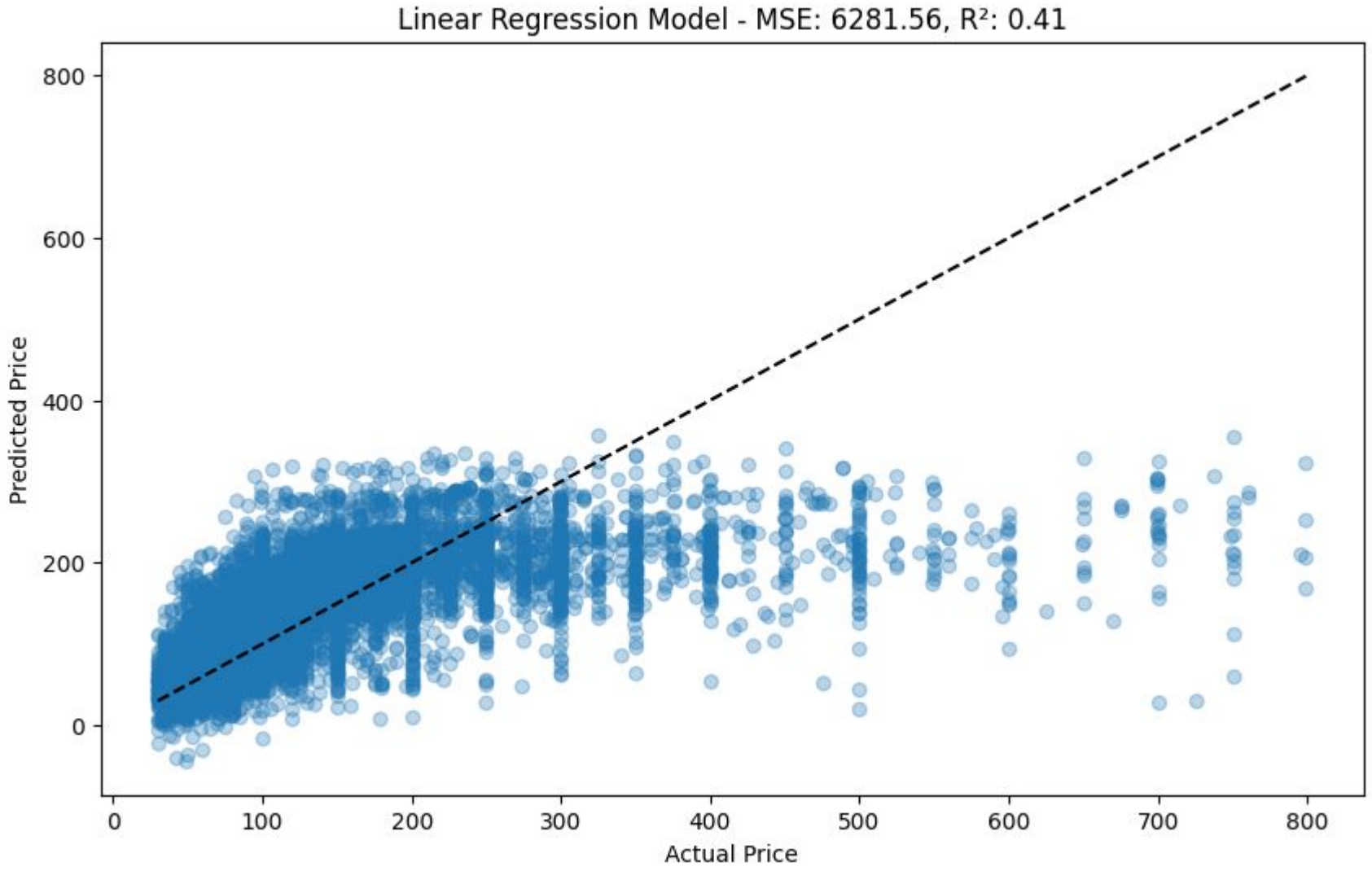
Linear Regression for all features

```
data = df
# Prepare transformers for preprocessing the data
categorical_features = ['neighbourhood_group', 'neighbourhood', 'room_type']
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

numerical_features = ['latitude', 'longitude', 'minimum_nights', 'number_of_reviews', 'reviews_per_month', 'calculated_host_listings_count', 'availability_
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=0)),
    ('scaler', StandardScaler())
])

# Create the preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Apply preprocessing to the data
X = data.drop('price', axis=1)  # Features
y = data['price']  # Target variable

X_transformed = preprocessor.fit_transform(X)

# Check the shape of the transformed data
X_transformed.shape
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, test_size=0.2, random_state=42)

# Initialize and train the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predict the rental prices on the testing set
y_pred = linear_model.predict(X_test)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

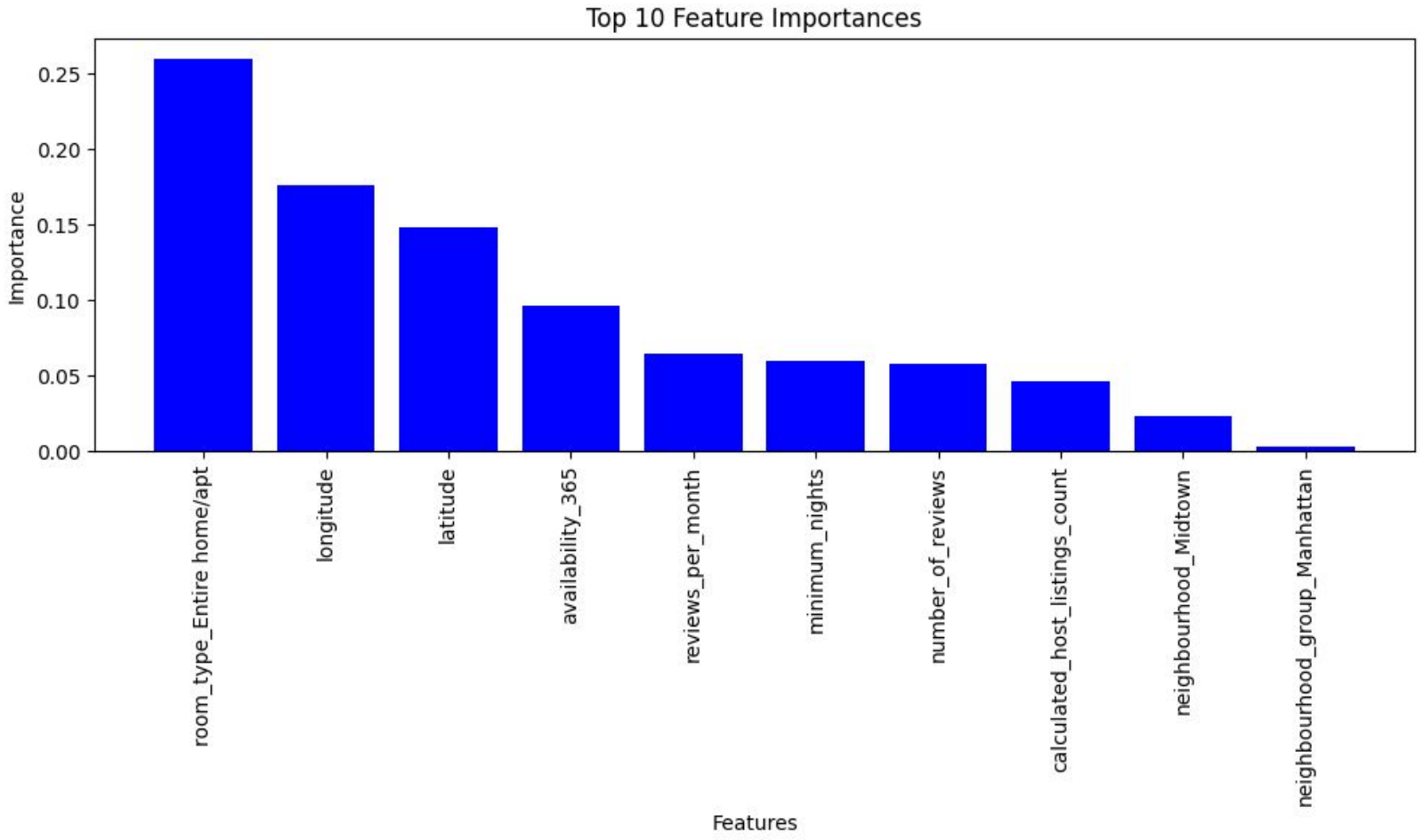
MSE: 6281.556536958027
R^2: 0.406597060546814

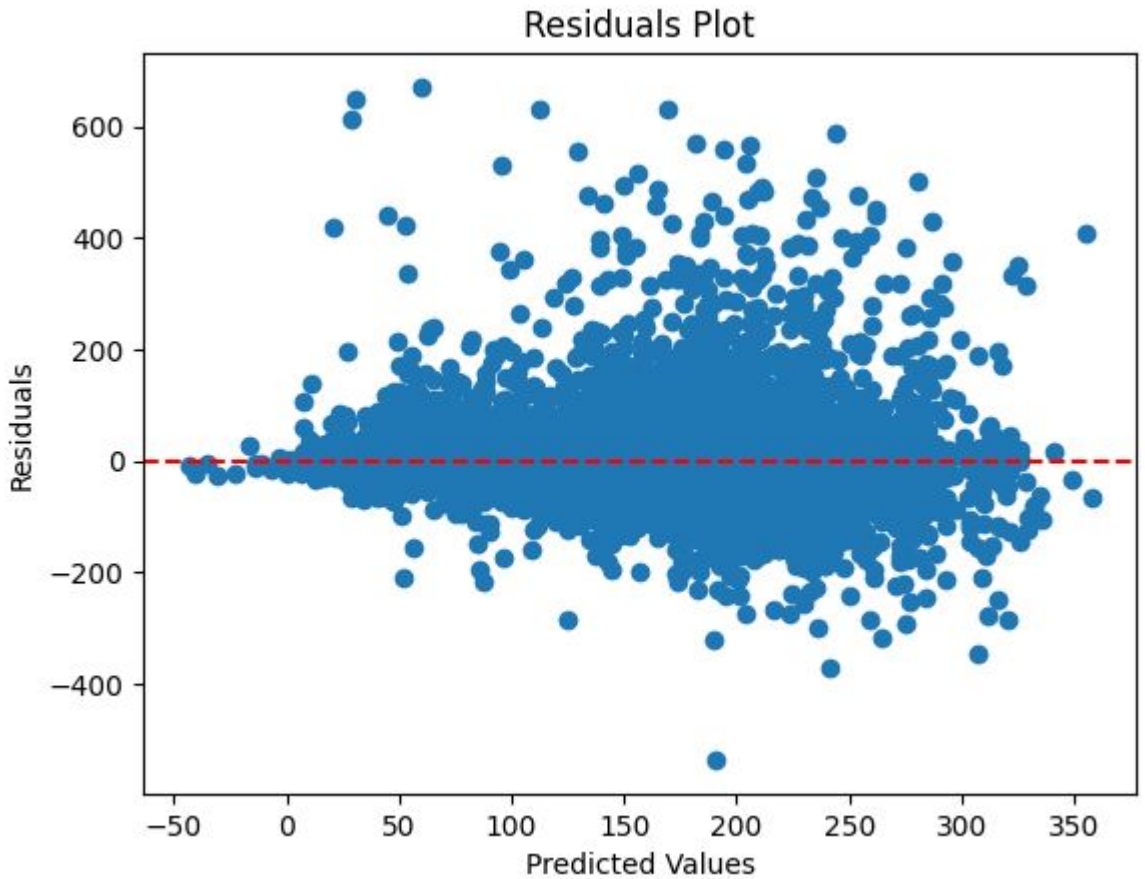
Linear Regression Accuracy

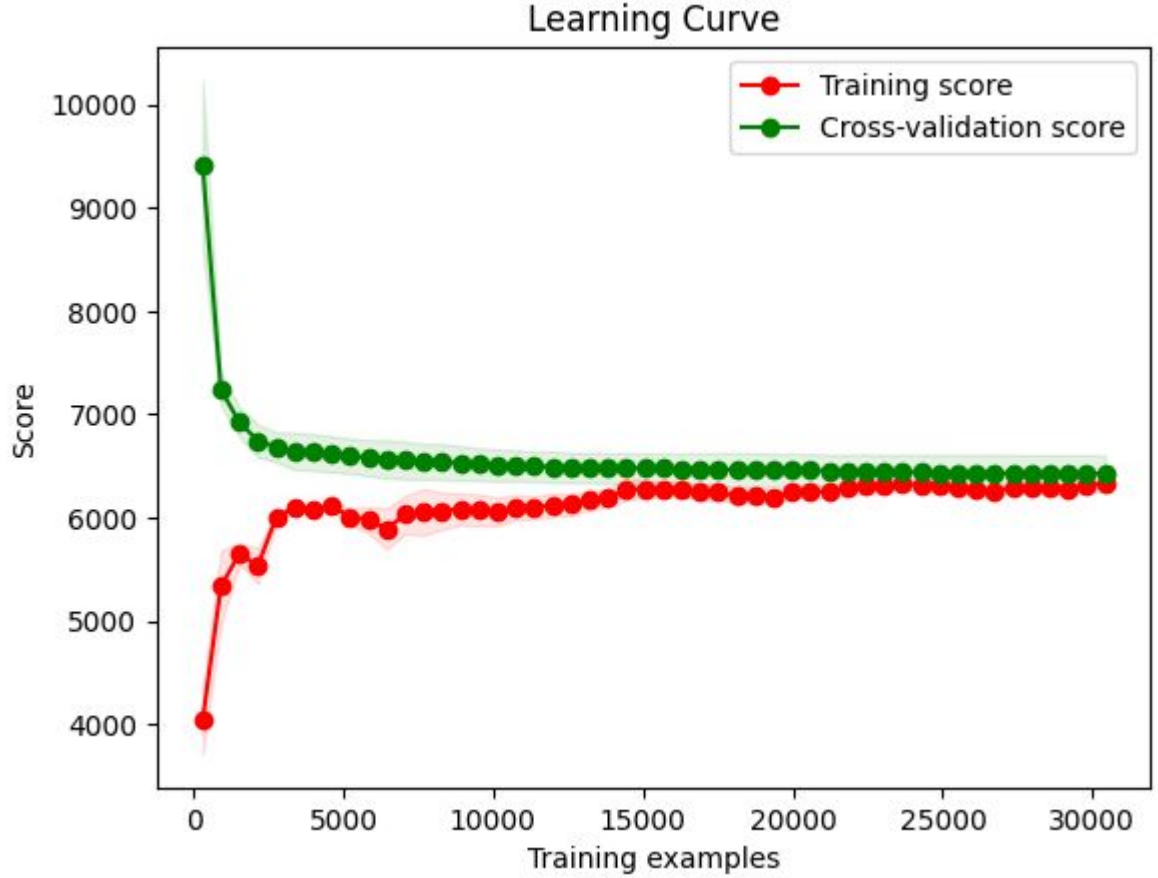


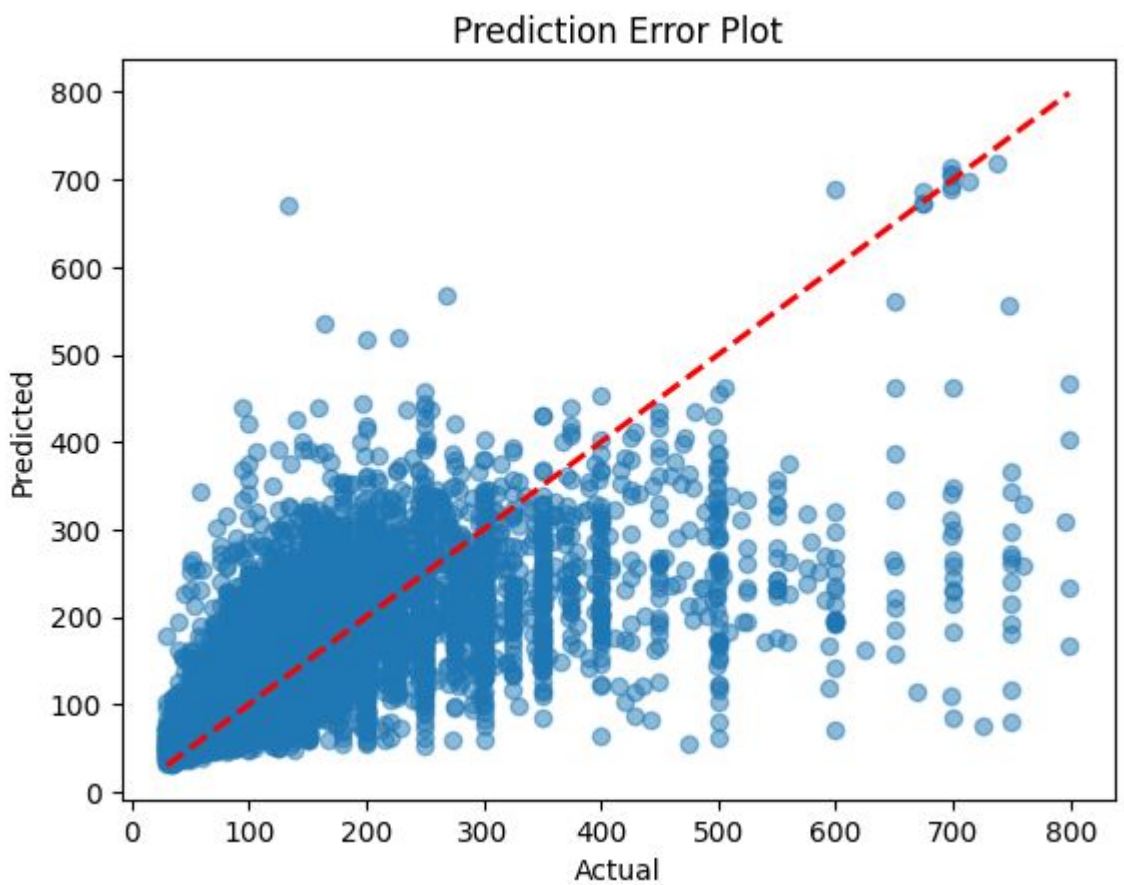
Random Forest

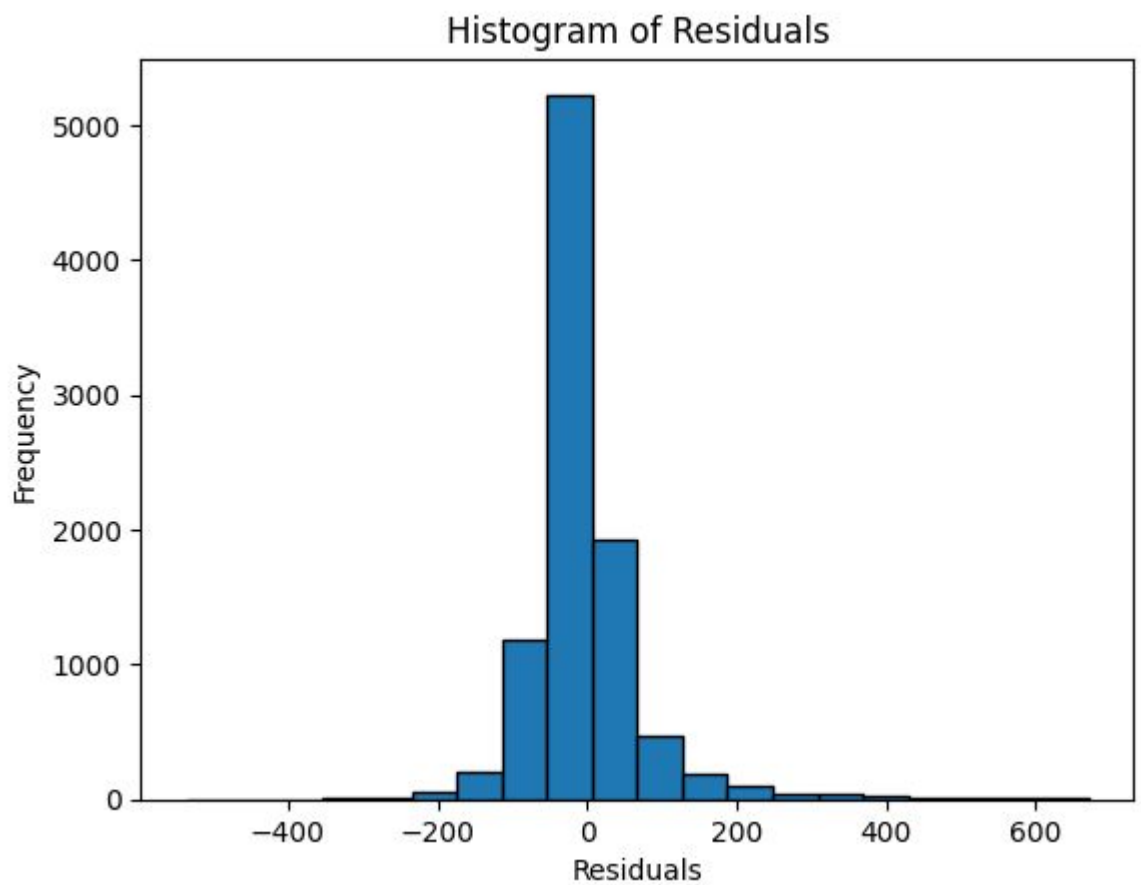
- Random Forest: R^2 = 0.47, MSE = 5,576




Residuals Plot
Residuals
Predicted Values
600
400
200
0
−200
−400
−50
0
50
100
150
200
250
300
350

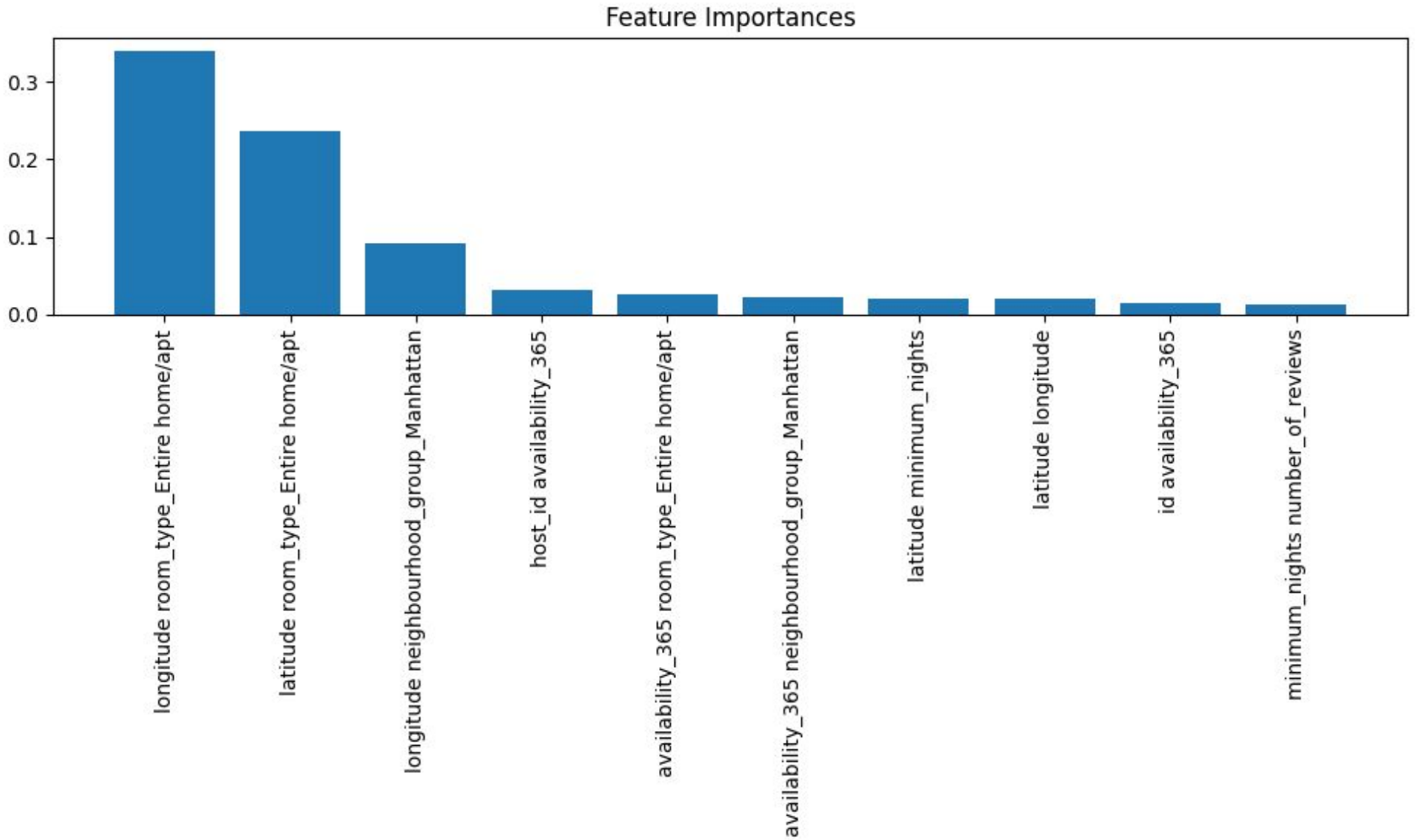

Learning Curve
Training score
Cross-validation score
Score
Training examples
10000
9000
8000
7000
6000
5000
4000
0
5000
10000
15000
20000
25000
30000

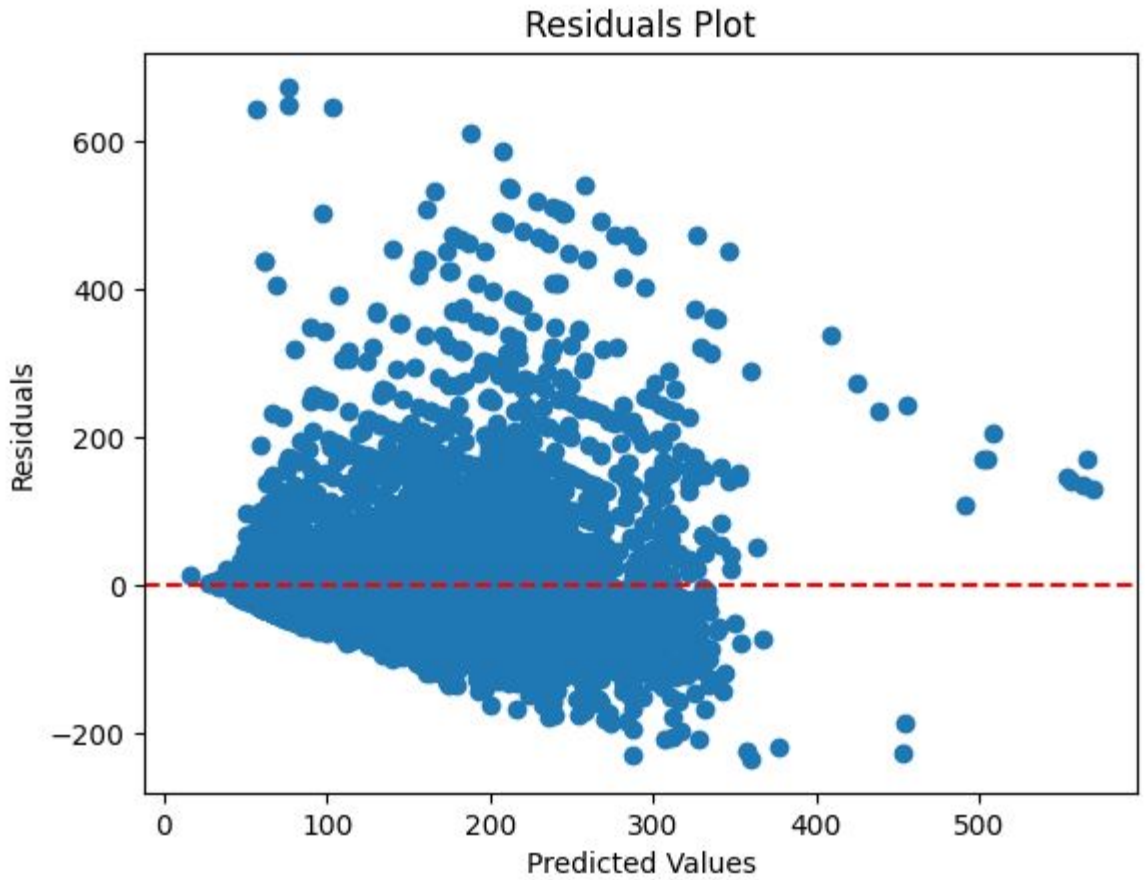

Prediction Error Plot
Predicted
Actual
0
100
200
300
400
500
600
700
800

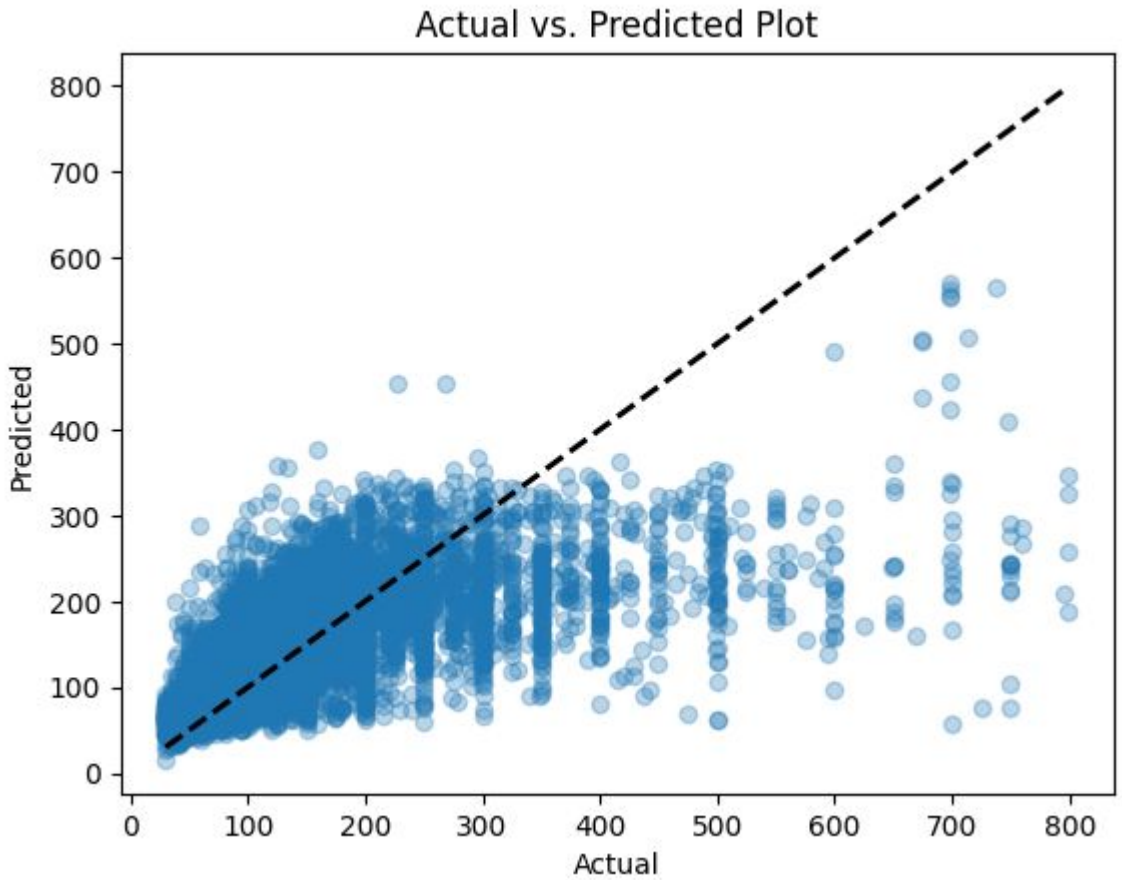

Histogram of Residuals
Frequency
Residuals
0
1000
2000
3000
4000
5000
−400
−200
0
200
400
600

Gradient Boosting Regressor

- R^2 = 0.46, MSE = 5,707



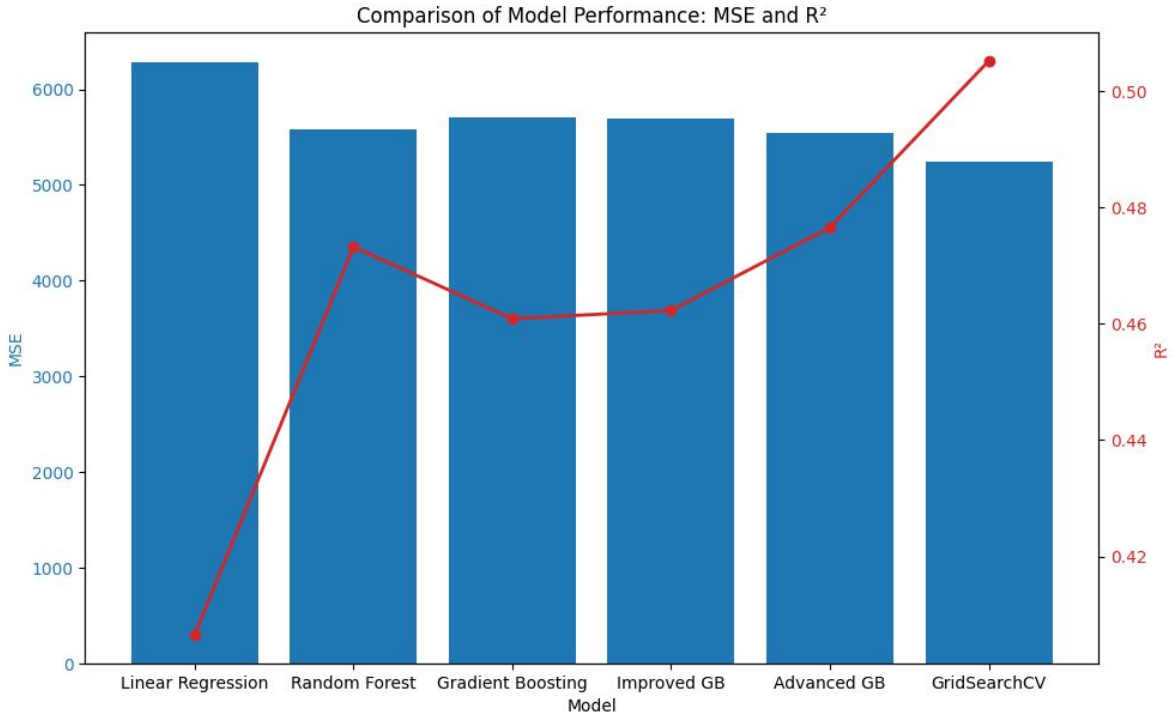

Residuals Plot
Residuals
Predicted Values
600
400
200
0
−200
0
100
200
300
400
500


Actual vs. Predicted Plot
Predicted
Actual
800
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
800

GridSearchCV

- R^2 = 0.51, MSE = 5,238

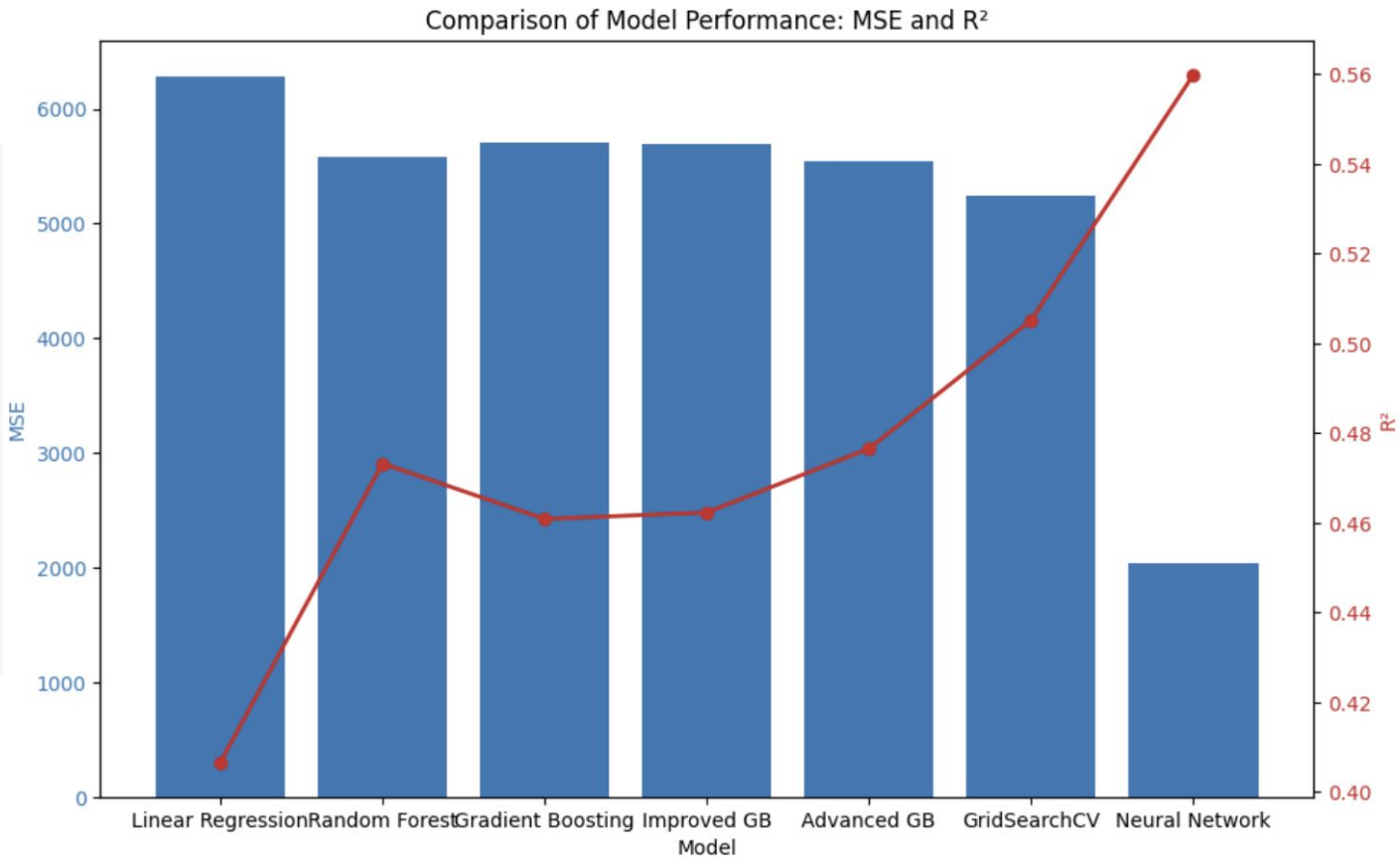
Best parameters: {'regressor__learning_rate': 0.1, 'regressor__max_depth': 5, 'regressor__max_features': 'sqrt', 'regressor__n_estimators': 200, 'regressor__subsample': 0.9}
Improved MSE: 5238.639461032996,
Improved R^2: 0.5051188289681703



Deep Learning

- Deep Learning Model: R^2 = 0.559, MSE = 2040

```
[ ] # Building the neural network model
    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train_prepared.shape[1],)),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1)  # Output layer: predicting price
    ])
    X_train_prepared_dense = X_train_prepared.toarray()
    X_test_prepared_dense = X_test_prepared.toarray()
    # Compiling the model
    model.compile(optimizer='adam', loss='mse', metrics=[RootMeanSquaredError()])

    # Training the model
    # Training the model on the dense version of the training data
    history = model.fit(X_train_prepared_dense, y_train, epochs=10, validation_split=0.2, verbose=1)

    # Predicting on the dense version of the test set
    y_pred = model.predict(X_test_prepared_dense)

    # Evaluating the model on the dense version of the test set
    test_mse = model.evaluate(X_test_prepared_dense, y_test, verbose=0)[0]
    test_rmse = model.evaluate(X_test_prepared_dense, y_test, verbose=0)[1]
    test_r2 = r2_score(y_test, y_pred.flatten())  # Ensure y_pred is the correct shape


    test_mse, test_rmse, test_r2
```



Deep Learning

- Deep Learning Model: $R^2 = 0.564$, MSE = 2019

```
# Convert last_review to datetime and handle missing values
data['last_review'] = pd.to_datetime(data['last_review'])
data['days_since_last_review'] = (pd.to_datetime('today') - data['last_review']).dt.days

# Month or year of the last review
data['last_review_month'] = data['last_review'].dt.month
data['last_review_year'] = data['last_review'].dt.year

# Number of words in the name
data['name_length'] = data['name'].apply(lambda x: len(str(x).split()))

# Merging average price by neighbourhood
avg_price_neighbourhood = data.groupby('neighbourhood')['price'].mean().rename('avg_price_neighbourhood')
avg_price_neighbourhood_df = avg_price_neighbourhood.reset_index()
avg_price_neighbourhood_df.columns = ['neighbourhood', 'avg_price_neighbourhood']  # Ensure this matches the number of columns
data = data.merge(avg_price_neighbourhood_df, on='neighbourhood', how='left', suffixes=('', '_mean'))
print(data.shape)  # Check if the merge was correct

# Calculating average minimum nights by room type
avg_min_nights_room = data.groupby('room_type')['minimum_nights'].mean()

# Convert the Series to a DataFrame for easier merging
avg_min_nights_room_df = avg_min_nights_room.reset_index()
avg_min_nights_room_df = avg_min_nights_room_df.iloc[:,[0,1]]
# Verify the correct operation by printing the DataFrame
print(avg_min_nights_room_df.head())

# There should only be two columns here, let's rename them correctly
avg_min_nights_room_df.columns = ['room_type', 'avg_min_nights_room']

# Merging this DataFrame with the main DataFrame
data = data.merge(avg_min_nights_room_df, on='room_type', how='left')
```

```
model = Sequential([
        Dense(256, activation='relu', input_shape=(X_train_prepared.shape[1],)),    # Increased from 128 to 256
        Dense(128, activation='relu'),    # Added an extra layer
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1)    # Output layer: predicting price
])

X_train_prepared_dense = X_train_prepared.toarray()
X_test_prepared_dense = X_test_prepared.toarray()


# Compiling the model
model.compile(optimizer='adam', loss='mse', metrics=[RootMeanSquaredError()])

# Training the model
# Training the model on the dense version of the training data
history = model.fit(X_train_prepared_dense, y_train, epochs=20, batch_size=64, validation_split=0.2, verbose=1)


# Predicting on the dense version of the test set
y_pred = model.predict(X_test_prepared_dense)

# Evaluating the model on the dense version of the test set
test_mse = model.evaluate(X_test_prepared_dense, y_test, verbose=0)[0]
test_rmse = model.evaluate(X_test_prepared_dense, y_test, verbose=0)[1]
test_r2 = r2_score(y_test, y_pred.flatten())    # Ensure y_pred is the correct shape


test_mse, test_rmse, test_r2
```

```
Epoch 1/20
451/451 [==============================] - 7s 10ms/step - loss: 3833.9290 - root_mean_squared_error: 61.9187 - val_loss: 2146.4778 - val_root_mean_squared_error: 46.3301
Epoch 2/20
451/451 [==============================] - 6s 13ms/step - loss: 2117.7290 - root_mean_squared_error: 46.0188 - val_loss: 2147.8247 - val_root_mean_squared_error: 46.3446
Epoch 3/20
451/451 [==============================] - 4s 9ms/step - loss: 2084.2769 - root_mean_squared_error: 45.6539 - val_loss: 2113.6938 - val_root_mean_squared_error: 45.9749
Epoch 4/20
451/451 [==============================] - 2s 5ms/step - loss: 2069.9995 - root_mean_squared_error: 45.4972 - val_loss: 2118.3433 - val_root_mean_squared_error: 46.0255
Epoch 5/20
451/451 [==============================] - 2s 5ms/step - loss: 2052.1877 - root_mean_squared_error: 45.3011 - val_loss: 2124.6560 - val_root_mean_squared_error: 46.0940
Epoch 6/20
451/451 [==============================] - 2s 5ms/step - loss: 2027.9143 - root_mean_squared_error: 45.0324 - val_loss: 2098.8997 - val_root_mean_squared_error: 45.8138
Epoch 7/20
451/451 [==============================] - 3s 7ms/step - loss: 2012.4153 - root_mean_squared_error: 44.8600 - val_loss: 2098.0454 - val_root_mean_squared_error: 45.8044
Epoch 8/20
451/451 [==============================] - 3s 6ms/step - loss: 2007.7950 - root_mean_squared_error: 44.8084 - val_loss: 2101.7666 - val_root_mean_squared_error: 45.8450
Epoch 9/20
451/451 [==============================] - 2s 5ms/step - loss: 1992.8605 - root_mean_squared_error: 44.6415 - val_loss: 2151.7554 - val_root_mean_squared_error: 46.3870
Epoch 10/20
451/451 [==============================] - 2s 4ms/step - loss: 1990.3840 - root_mean_squared_error: 44.6137 - val_loss: 2080.8035 - val_root_mean_squared_error: 45.6158
Epoch 11/20
451/451 [==============================] - 2s 5ms/step - loss: 1976.3214 - root_mean_squared_error: 44.4558 - val_loss: 2087.6970 - val_root_mean_squared_error: 45.6913
Epoch 12/20
451/451 [==============================] - 2s 5ms/step - loss: 1968.0792 - root_mean_squared_error: 44.3630 - val_loss: 2118.3318 - val_root_mean_squared_error: 46.0253
Epoch 13/20
451/451 [==============================] - 3s 8ms/step - loss: 1957.6709 - root_mean_squared_error: 44.2456 - val_loss: 2101.7839 - val_root_mean_squared_error: 45.8452
Epoch 14/20
451/451 [==============================] - 3s 6ms/step - loss: 1953.4633 - root_mean_squared_error: 44.1980 - val_loss: 2091.8340 - val_root_mean_squared_error: 45.7366
Epoch 15/20
451/451 [==============================] - 2s 5ms/step - loss: 1947.6848 - root_mean_squared_error: 44.1326 - val_loss: 2090.6985 - val_root_mean_squared_error: 45.7242
Epoch 16/20
451/451 [==============================] - 2s 4ms/step - loss: 1937.3048 - root_mean_squared_error: 44.0148 - val_loss: 2085.3247 - val_root_mean_squared_error: 45.6654
Epoch 17/20
451/451 [==============================] - 2s 5ms/step - loss: 1929.9629 - root_mean_squared_error: 43.9312 - val_loss: 2105.7102 - val_root_mean_squared_error: 45.8880
Epoch 18/20
451/451 [==============================] - 2s 5ms/step - loss: 1927.5537 - root_mean_squared_error: 43.9039 - val_loss: 2084.2461 - val_root_mean_squared_error: 45.6535
Epoch 19/20
451/451 [==============================] - 4s 8ms/step - loss: 1920.5946 - root_mean_squared_error: 43.8246 - val_loss: 2092.7415 - val_root_mean_squared_error: 45.7465
Epoch 20/20
451/451 [==============================] - 2s 4ms/step - loss: 1915.2917 - root_mean_squared_error: 43.7640 - val_loss: 2088.3142 - val_root_mean_squared_error: 45.6981
282/282 [==============================] - 1s 2ms/step
(2019.616943359375, 44.940147399902344, 0.5641847163568055)
```

06

Conclusion & Limitation

Results & Discussion

Overview of Model Performance

- Linear Regression: $R^2 = 0.40$, MSE = 6,281
- Random Forest: $R^2 = 0.47$, MSE = 5,576
- Gradient Boosting: Initial $R^2 = 0.46$, MSE = 5,707
- Improved Gradient Boosting with Hyperparameter Tuning: $R^2 = 0.46$, MSE = 5,692
- Advanced Gradient Boosting with Feature Engineering: $R^2 = 0.48$, MSE = 5,541
- GridSearchCV: $R^2 = 0.51$, MSE = 5,238
- Deep Learning Model: $R^2 = 0.564$, MSE = 2019

Overfitting vs. Underfitting:

- **Overfitting**: If any model had a considerably higher performance on training data compared to test data, it might be overfitting.
- **Underfitting**: Conversely, if models are too simple to capture the data complexity, they might underfit, leading to low R^2 values across both training and testing phases.
- **Model Assumptions:** For instance, linear regression assumes a linear relationship between features and the target variable. If this assumption doesn't hold (which is common in real-world data), the model will likely underperform.

Discussion on Low R^2 Value

1. Complexity of the Dataset: The dataset might have a high level of inherent variability that simple or even complex models struggle to capture fully. Features in the dataset may not provide enough information to predict outcomes accurately.
2. Quality and Quantity of Data: If the data has issues like missing values, insufficient records, or noisy data, models may not learn effectively.
3. Feature Relevance: The features used may not have a strong predictive relationship with the target variable. This can be a limitation if the dataset lacks fields that are significant predictors of the outcome.

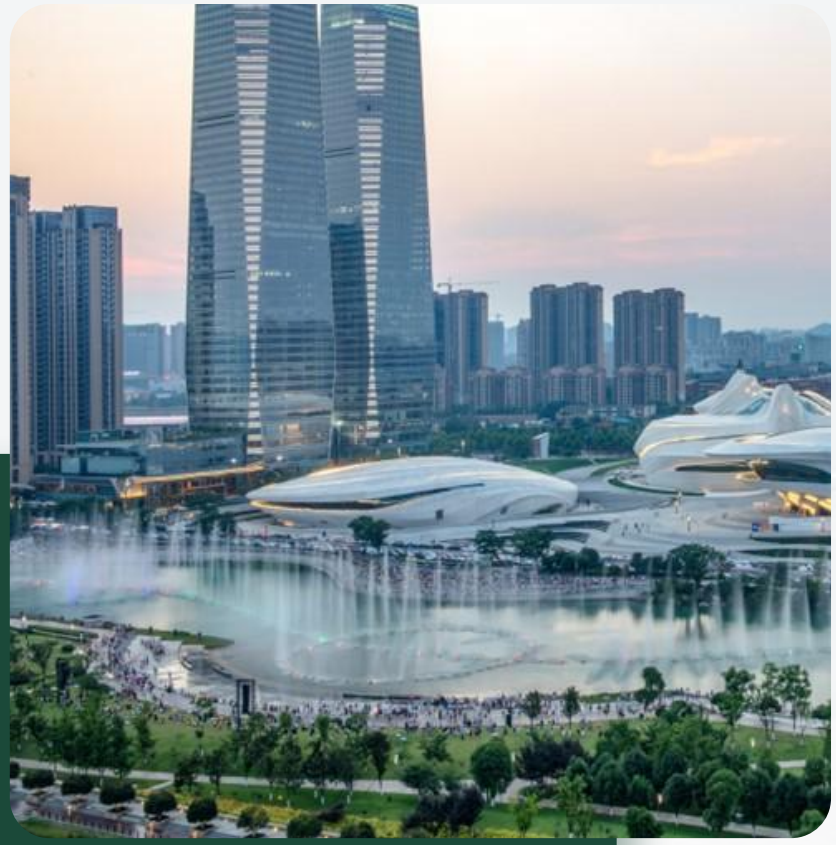
Limitation of this Project

Model Limitations:

Linear regression:

- Sensitive to outliers.
- Assumes a linear relationship between features and targets, which may not always hold.

Potential Areas of Improvement

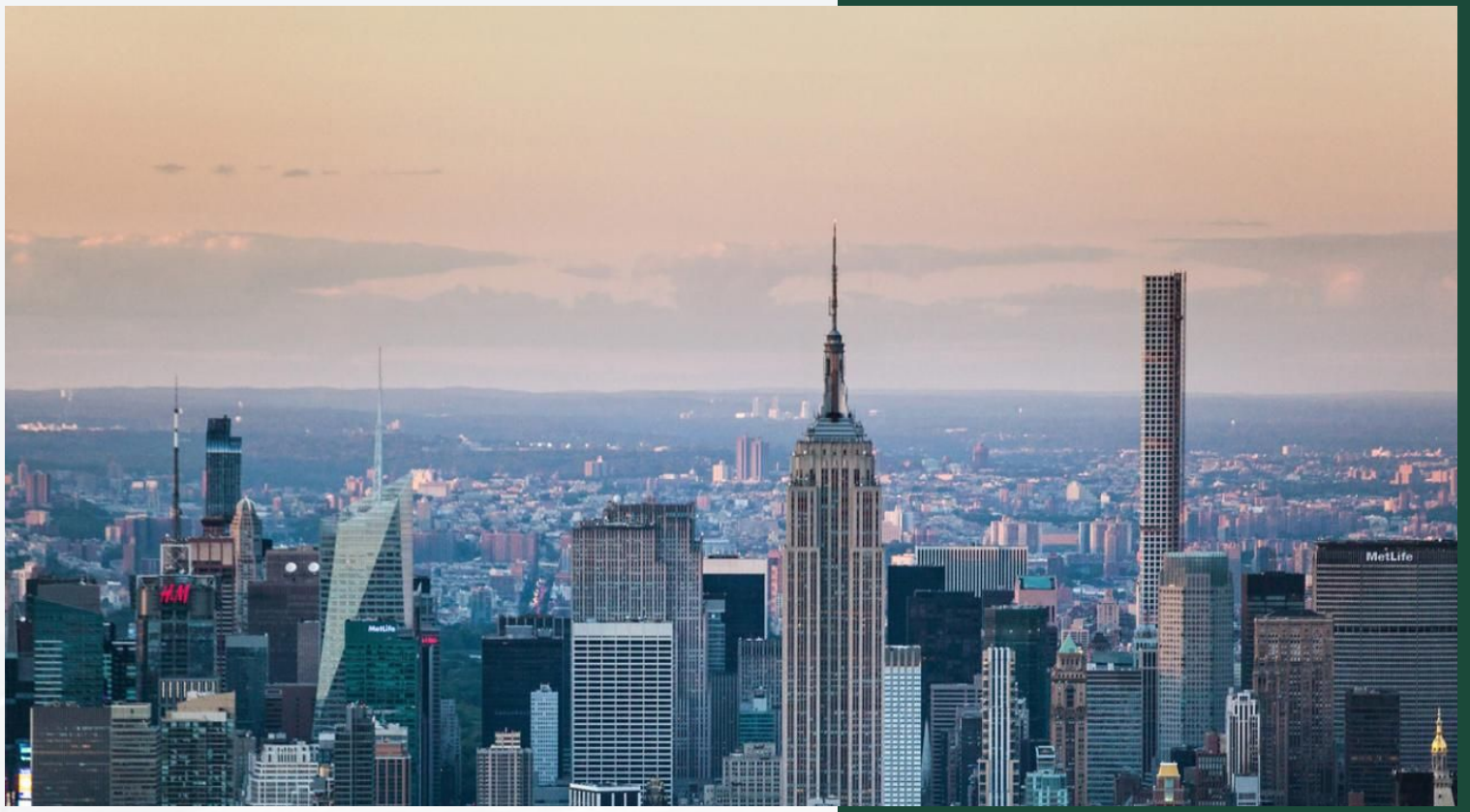


- **Improve the Feature Engineering**
  - Creating more meaningful features from existing dataset
  - Obtaining additional data from other resources
- **Increased Data Collection:**
  - collect more comprehensive or higher-quality data that could improve the model's learning capability.
- **Experimentation with Non-linear Models:**
  - Since traditional models have limitations, experimenting with non-linear models or neural networks might yield better results.

Next Step : Future

- We will consider to improve our accurate rate at the summer
- Edit base on peer & professor‘s recommendation

Thanks for listening!



Data Science 2024 Bootcamp →

Shihui Feng, Dong Li, Sofia Celorio , Antong Lei, KaiYuan Ma, Tujie Guo

Compare & Analysis

Compare and analyze:

- PERFORMANCE: The gradient boosted regression tree outperforms linear regression on all metrics, showing lower MAE and RMSE, and higher R^2 values.
- Model Complexity: Gradient boosted regression trees are more complex than linear regression, requiring more computational resources and time for training and tuning.
- Interpretability: Linear regression models are usually easier to interpret because they provide direct coefficients to describe the relationship between features and targets. Gradient boosted regression trees, on the other hand, are more difficult to interpret because they are integrated models based on a large number of decision trees.

Challenges Faced during Project

Discuss the challenges faced during the project.

Improvement Strategies

- Feature Engineering: Potential for creating more meaningful features from the existing data, or obtaining additional data that could help improve model accuracy.
- Advanced Modeling Techniques: Consider whether more sophisticated models or ensemble methods might capture the complexity of the data better.
- Increased Data Collection: Suggest ways to collect more comprehensive or higher-quality data that could enhance the model’s learning capability.
- Experimentation with Non-linear Models: Since traditional models have limitations, experimenting with non-linear models or neural networks might yield better results.

Works Cited

Aydin, Rebecca. “How 3 Guys Turned Renting Air Mattresses in Their Apartment into a $31 Billion Company, Airbnb.” *Business Insider*, www.businessinsider.com/how-airbnb-was-founded-a-visual-history-2016-2. Accessed 30 Mar. 2024.

Hypothesis Testing

Description of Hypothesis Testing Methodology

01

Describe the methodology used for hypothesis testing.

Table of Contents

Company Background

- Online platform that allows people to **rent out** their **homes** or **spare rooms** to guests looking for **short-term lodging**.
- Founded in **2008**, has rapidly transformed the hospitality industry by offering an alternative to traditional hotels.
- The platform connects **hosts** and **guests**, providing a variety of accommodation options such as apartments, houses, and unique stays.

Importance of Airbnb in NYC:

- Economic impact
- Flexibility and affordability
- Accommodation diversity
- Neighborhood revitalization
  - Encouraging tourism beyond traditional tourist areas, benefiting local businesses and communities.
- Cultural exchange

Closing Remarks + Implications

01

Conclude with closing remarks and discuss the implications of the findings.