New York City Airbnb (2019) Price Prediction Analysis

— Siqi Wei (sw3711), Mingzhou Jin(mj3096), Yisheng Chen (yc4180), Qingci Luo (ql2466)

Get the Data

Describes the listing activity and metrics in NYC, NY for 2019.

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

What should the price be if we want to open a private room type airbnb in Brooklyn Heights?

Get the Data

Pretty clean dataset

```
In [3]: data.shape

Out[3]: (48895, 16)

In [4]: data.columns

Out[4]: Index(['id', 'name', 'host_id', 'host_name', 'neighbourhood_group',
               'neighbourhood', 'latitude', 'longitude', 'room_type', 'price',
               'minimum_nights', 'number_of_reviews', 'last_review',
               'reviews_per_month', 'calculated_host_listings_count',
               'availability_365'],
              dtype='object')

In [5]: data.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
```

Get the Data

In [6]:
```
data.describe()
```

Out[6]:

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000	48895.0
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	7.1
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	32.9
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	1.0
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	1.0
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000	1.0
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000	2.0
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000	327.0

Data Cleaning

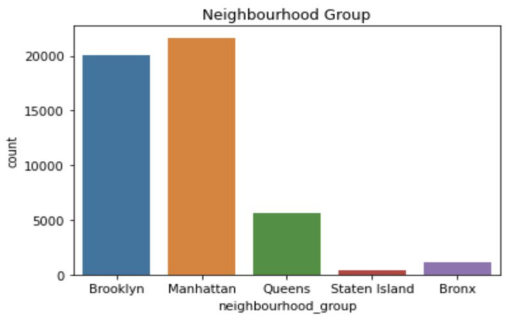
```
In [7]: # Removing the duplicates if any
        if data.duplicated().sum() != 0:
            data.drop_duplicates(inplace=True)
        else:
            print(None)

        None
```

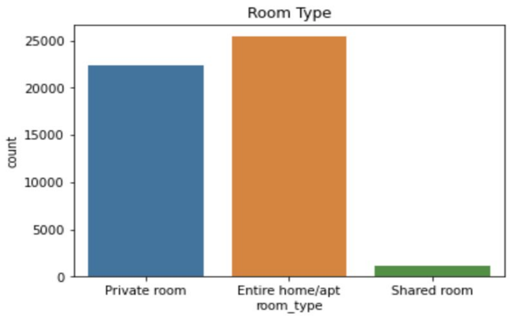
```
In [8]: # Check for the null values in each column
        data.isnull().sum()

Out[8]: 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
```

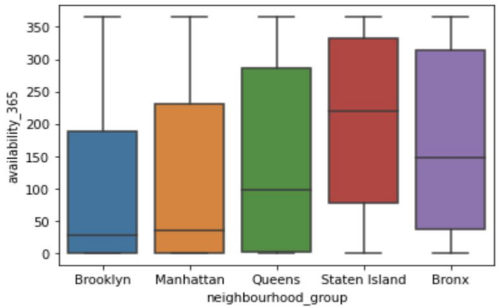
```
In [9]: # Drop unnecessary columns(removing all NaN values)
        df = data.drop(['id','name','host_name','last_review','reviews_per_month'], axis=1)
```

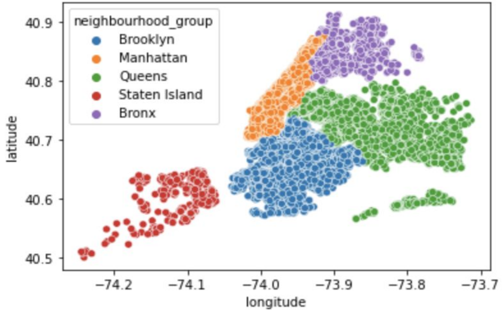
```
In [10]: df.isnull().sum()

Out[10]: host_id                           0
         neighbourhood_group               0
         neighbourhood                     0
         latitude                          0
         longitude                         0
         room_type                         0
         price                             0
         minimum_nights                    0
         number_of_reviews                 0
         calculated_host_listings_count    0
         availability_365                  0
         dtype: int64
```

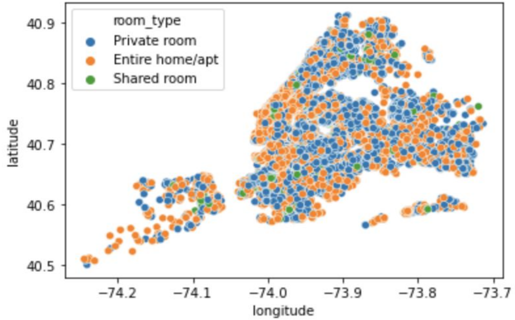
Data Visualization

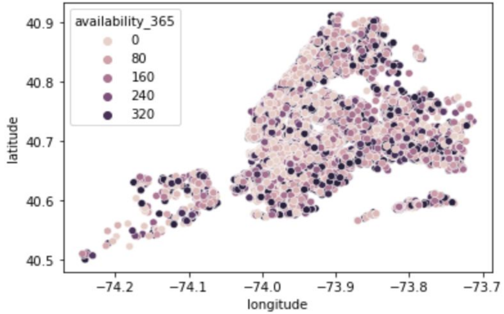






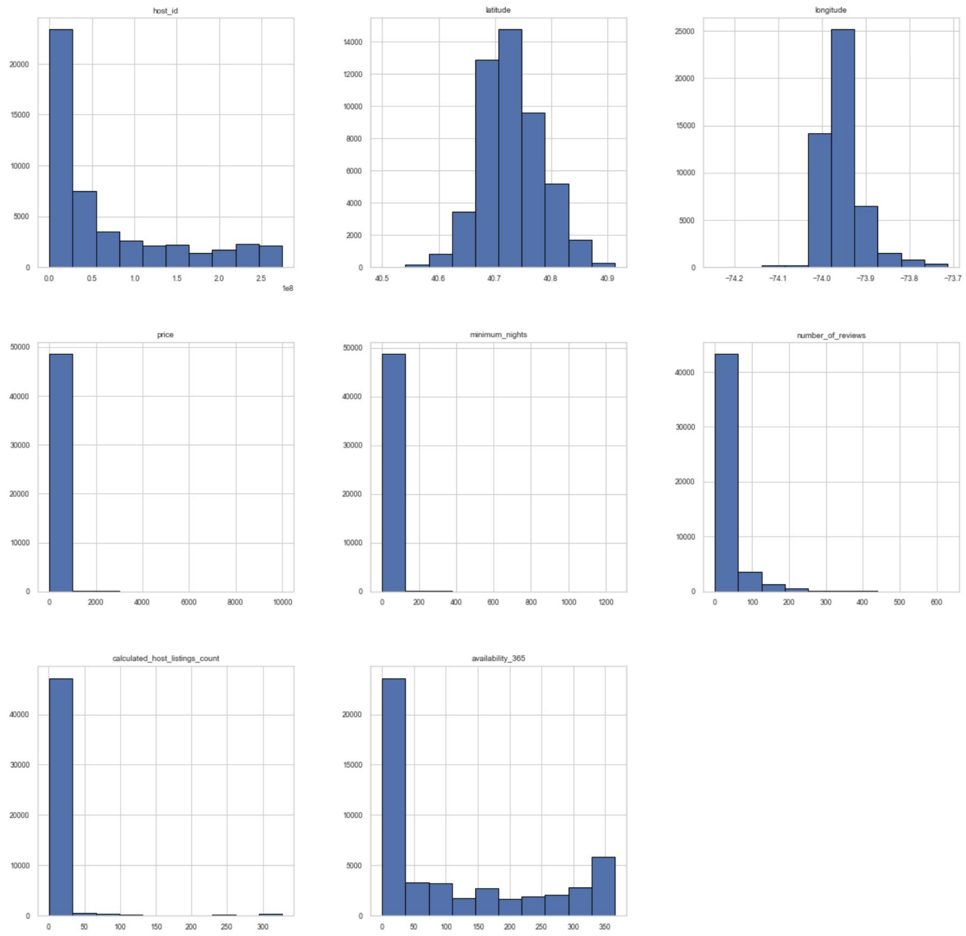






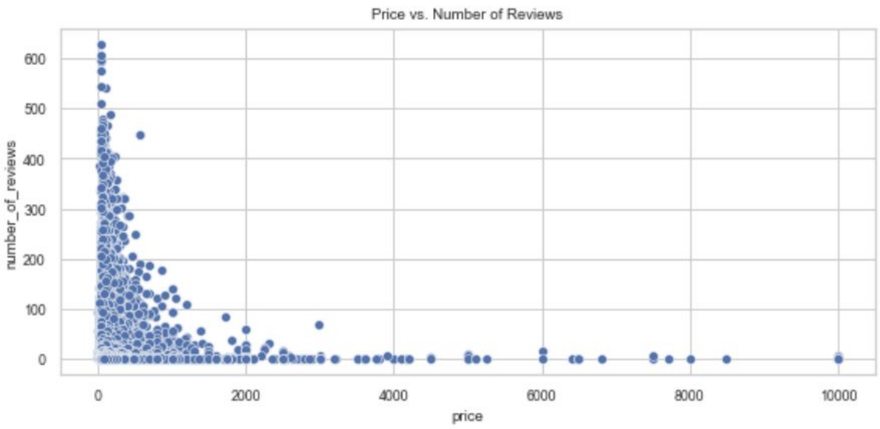
Data Visualization

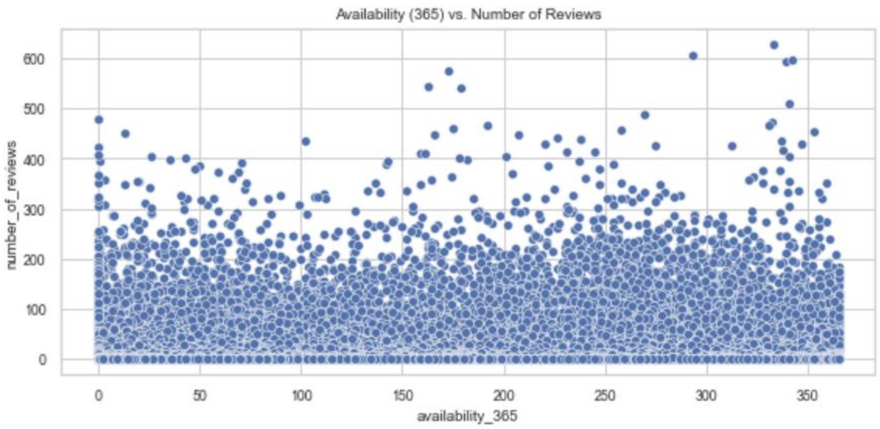
- *Latitude* and *longitude* have a normal distribution
- Most the host has a *price* under $1000
- *calculated_host_listings_count* has lots of outliers
- *availability_365*: the most of the hosts are available for short term.



Data Visualization

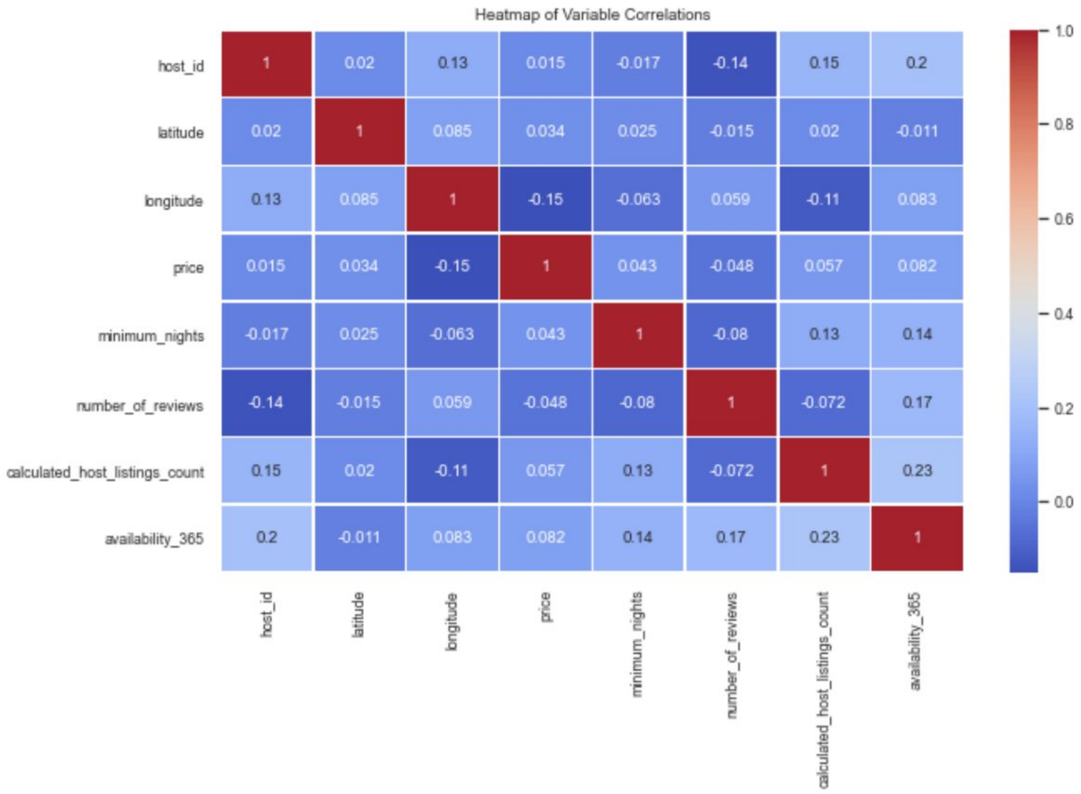
Draw scatter plots for price vs. number_of_reviews and availability_365 vs. number_of_reviews





Data Visualization

Correlation analysis helps us to see features relations



Linear Regression

To predict the response variable, we construct a linear regression model including all predictors...

Before constructing the model:

- Remove all the irrelevant features(id,name,host name…)
- Convert ‘neighbourhood_group’ and ‘room_type’ to dummy variables.

Result:

```
Training RMSE:  214.08653454183846
Testing RMSE:  252.03564007924012
train_score 0.10793175416491141
test_score 0.08395212651231476
```

Overfitting?

Regularization (Lasso Regression)

shrinks the coefficients of less important variables → Simpler model → Reduce overfitting

Method: K-fold cross validation with optimal alpha

Before regularization:

- Standardize features by using `StandardScaler()s`
  Center values around 0 with scaling to unit variance

After regularization:

```
array([ 8.68575114, 24.85160432,  3.15859772, 13.22863466,  4.86789376,
       26.46619172,  0.21817066, 13.02992975, 13.92480814,  0.        ,
       11.96525697, 52.60058581,  0.        ,  5.34126988])
```

→ Remove predictors 'neighbourhood_group_Queens' and 'room_type_Private room'

Potential Problems

Model	RMSE for train	RMSE for test
Linear regression	214.08653454183846	252.03564007924012
Lasso regression	214.0871254248083	252.03843462132136

Why worse?

Remove small coefficients which could be important to the model

The original dataset contains too much irrelevant information

Something is Missing

- Plot the frequency distribution of the Price
- Right-skewed!

```python
# Plot the distribution of the response variable
df['price'].hist(bins=50)
plt.title('Histogram of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

Transformation Needed

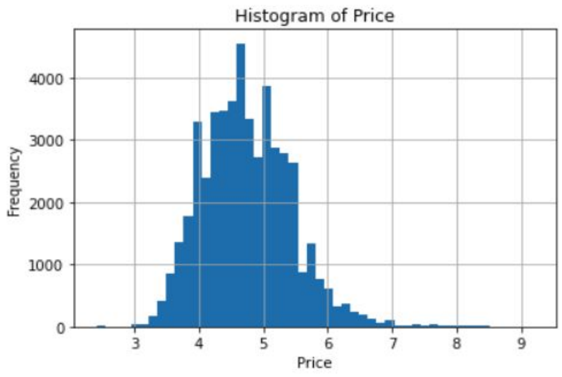
Take the logarithm of the Price, making it

- More symmetric
- Reduce the impact of outliers

```
[50] df["price"] = np.log1p(df["price"])
     df = df[df['price'] != 0]
     print(df['price'].describe())
```

```
count    48884.000000
mean         4.737951
std          0.691782
min          2.397895
25%          4.248495
50%          4.672829
75%          5.170484
max          9.210440
Name: price, dtype: float64
```

```
# Plot the histogram of the response variable
df['price'].hist(bins=50)
plt.title('Histogram of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



Decision Tree Model

- Build the model with default hyperparameter
- Fit the model and evaluate the performance
- Test RMSE >> Train RMSE
- OVERFITTING!

```python
# Fit a decision tree model with default hyperparameters
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train1, y_train1)

# Predict on the test set
y_pred_test = dt.predict(X_test1)

# Make predictions on the training set
y_pred_train = dt.predict(X_train1)

# Evaluate the model
from sklearn.metrics import mean_squared_error
mse_test = mean_squared_error(y_test1, y_pred_test)
rmse_test = np.sqrt(mse_test)

mse_train = mean_squared_error(y_train1, y_pred_train)
rmse_train = np.sqrt(mse_train)

print('Training RMSE:', rmse_train)
print('Test RMSE:', rmse_test)
```

```
Training RMSE: 0.0012367286622135619
Test RMSE: 0.6163014783981189
```

Hyperparameter Tuning

- Use this approach to address overfitting
- Find the optimal hyperparameters to fit

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Split data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(df.drop('

# Define a Decision Tree model
dt = DecisionTreeRegressor(random_state=42)

# Define hyperparameters to search over
params = {'max_depth': [2,5,10],
          'min_samples_split': [2,5,10],
          'min_samples_leaf': [10,20,30,40,50,60]}

# Define performance metric
scorer = 'neg_mean_squared_error'

# Define Grid Search with cross-validation
grid_search = GridSearchCV(dt, params, scoring=scorer, cv=5

# Fit Grid Search on training data
grid_search.fit(X_train, y_train)
```

```python
# Print best hyperparameters and corresponding score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f'Best parameters: {best_params}')
print(f'Best score: {best_score}')

# Train final model on entire training set using best hyperparameters
dt_best = DecisionTreeRegressor(random_state=42, **best_params)
dt_best.fit(X_train, y_train)

# Evaluate final model on test set
y_pred = dt_best.predict(X_val)
mse = mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse)
print(f'Test RMSE: {rmse}')
```

```
Best parameters: {'max_depth': 10, 'min_samples_leaf': 60, 'min_samples_split': 2}
Best score: -0.20890856936720342
Test RMSE: 0.46239772061940654
```

Random Forest

Further improve RMSE?

Yes!

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
# Initialize a random forest regressor
rf = RandomForestRegressor(random_state = 42,
                                n_estimators = 100,
                                min_samples_split = 10,
                                min_samples_leaf = 1,
                                bootstrap = True,
                                max_depth = 100,
                                max_features = 'sqrt')

# Fit the model on the training data
rf.fit(X_train1, y_train1)

# Predict on the test set
y_pred = rf.predict(X_test1)

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test1, y_pred))
print('Test RMSE:', rmse)
```

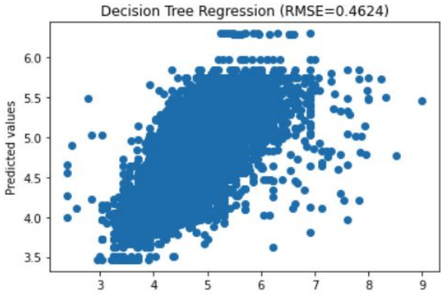
```
Test RMSE: 0.43718390199288315
```

Actual vs. Predicted Plot

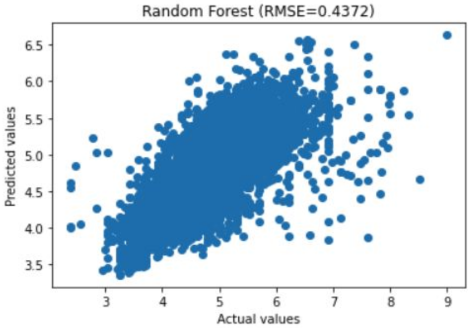
Fall clost the 45 degree line is the ideal scenario

Not very ideal but definitely not bad

```python
import matplotlib.pyplot as plt
plt.scatter(y_val, y_pred)
plt.xlabel("Actual values")
plt.ylabel("Predicted values")
plt.title(f"Decision Tree Regression (RMSE={rmse1:.4f})")
plt.show()
```




```python
plt.scatter(y_val, y_pred)
plt.xlabel("Actual values")
plt.ylabel("Predicted values")
plt.title(f"Random Forest (RMSE={rmse:.4f})")
plt.show()
```



Pricing for our Airbnb

Based on our prediction, we should charge $153.2/night for our newly listed housing locating in Brooklyn Heights with private room

```python
input_features = {'latitude': 40.693841,
                  'longitude': -73.995136,
                  'neighbourhood_group_Bronx': 0,
                  'neighbourhood_group_Brooklyn': 1,
                  'neighbourhood_group_Manhattan': 0,
                  'neighbourhood_group_Queens': 0,
                  'neighbourhood_group_Staten Island': 0,
                  'room_type_Entire home/apt': 0,
                  'room_type_Private_room': 1,
                  'room_type_Shared room': 0,
                  'minimum_nights': 1,
                  'number_of_reviews': 0,
                  'calculated_host_listings_count': 1,
                  'availability_365': 365}
input_df = pd.DataFrame(input_features, index=[0])

# make predictions using the trained random forest model
predicted_price = rf.predict(input_df)

print('The predicted price is: $', np.round(np.expm1(predicted_price[0]), 2))
```

```
The predicted price is: $ 153.2
```


THANKS