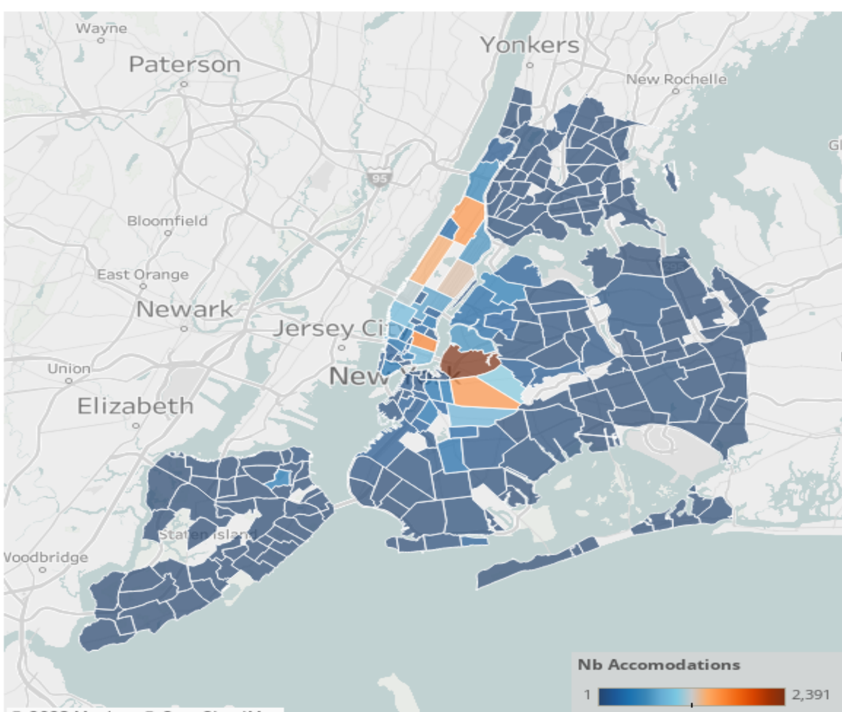
Open Ended Project

Made By:
Vaibhav Biyawala

DataSet:
New York City Airbnb

About Our DataSet:

The New York Airbnb dataset is a collection of data related to Airbnb listings in New York City. It provides information about various aspects of the listings, including the listing ID, host details, neighborhood, property type, room type, price, availability, and other relevant attributes. This dataset allows for exploration and analysis of the Airbnb market in New York City, including factors such as pricing trends, property types, and neighborhood preferences. It is commonly used for research, data analysis, and predicting rental prices in the Airbnb market in New York City.



Step 1:
Summarize Quality Assessment for AirBnb DataSet

Data Set: AirBnb.csv
Records: (48895, 16)
Total number of Features: 16
Features:

```
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')
```

Importing required libraries:

```
import pandas as pd
import seaborn as sb
from sklearn.impute import SimpleImputer
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from category_encoders import BinaryEncoder
```

Overview of Data Set:

```
ds.head()
```

index	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.9419
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399

To see datatypes of all features call ds.info():

```
ds.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
```

To see descriptive statistics of DataSet(Numerical Features):

```
ds.describe()
```

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000	48895.000000	48895.000000
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	7.143982	112.781327
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	32.952519	131.622289
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	1.000000	0.000000
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	1.000000	0.000000
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000	1.000000	45.000000
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000	2.000000	227.000000
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000	327.000000	365.000000

For Categorical Features

```
ds.iloc[:,[1,3,4,5,8,12]].describe()
```

	name	host_name	neighbourhood_group	neighbourhood	room_type	last_review
count	48879	48874	48895	48895	48895	38843
unique	47905	11452	5	221	3	1764
top	Hillside Hotel	Michael	Manhattan	Williamsburg	Entire home/apt	2019-06-23
freq	18	417	21661	3920	25409	1413

Finding Duplicates values:

```
a=ds.duplicated()
print("No. of duplicate values:",a.sum())
print(a)
```

```
No. of duplicate values: 0
0        False
1        False
2        False
3        False
4        False
         ...
48890    False
48891    False
48892    False
48893    False
48894    False
Length: 48895, dtype: bool
```

As this dataset has no duplicate values so we don't have to drop any values.

Finding Missing values:

```
b=ds.isna()
print("No of Missing values \n",b.sum())
```

```
No of Missing values
 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
```

Handling Missing values(Using Simple Imputer):

An imputer is a technique used to fill in missing values in a dataset. It replaces the missing values with estimated or imputed values based on the available data. Imputation strategies can include mean, median, mode, regression, K-nearest neighbors, or multiple imputation techniques. The goal is to handle missing values and ensure the dataset is complete for further analysis or modeling.

Code:

```
imputer=SimpleImputer(missing_values=np.nan, strategy='most_frequent')
ds=imputer.fit_transform(ds)
ds=pd.DataFrame(ds)
b=ds.isna()
print("No of missing values after replacing it with most frequent values are: \n",b.sum()
```

Output:

```
No of missing values after replacing it with most frequent values are:
 0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
dtype: int64
```

Step 2:
Mention all the data preprocessing tasks done by you as per analysis done in Step 1

In step 1 we did following tasks:
- Overview of Data Set: The dataset contains information about a specific dataset, but further details are not provided.
- Datatypes of all features: Identify the data types (numeric, categorical, etc.) of each feature in the dataset.
- Describing Data Set: Perform descriptive statistical analysis to understand the distribution and central tendency of numerical features.
- Finding and Handling Duplicate Values: Detect and handle duplicate values in the dataset to avoid bias and maintain accuracy.
- Finding and Handling Missing Values: Identify and address missing values by removing or filling them using appropriate techniques.

These steps aim to provide an understanding of the dataset, handle duplicates and missing values, and perform statistical analysis to summarize the data effectively.

Encoding:

Encoding is the process of transforming categorical or non-numeric data into a numeric representation that can be easily understood by machine learning algorithms.

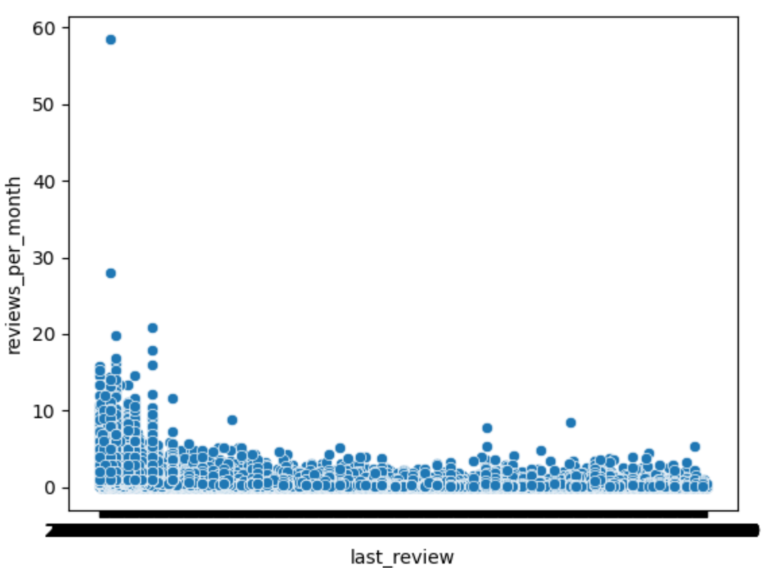
One Hot Encoding:

One-hot encoding is used for categorical variables with no inherent order or hierarchy. It creates binary columns for each unique category and assigns a value of 1 if the category is present and 0 otherwise. This method avoids assigning any ordinal relationship between the categories.

Code:

```
encoded_df = pd.get_dummies(ds['last_review'])
df_encoded = pd.concat([ds, encoded_df], axis=1)
sb.scatterplot(x='last_review', y='reviews_per_month', data=ds)
```

Output:



Label Encoding:

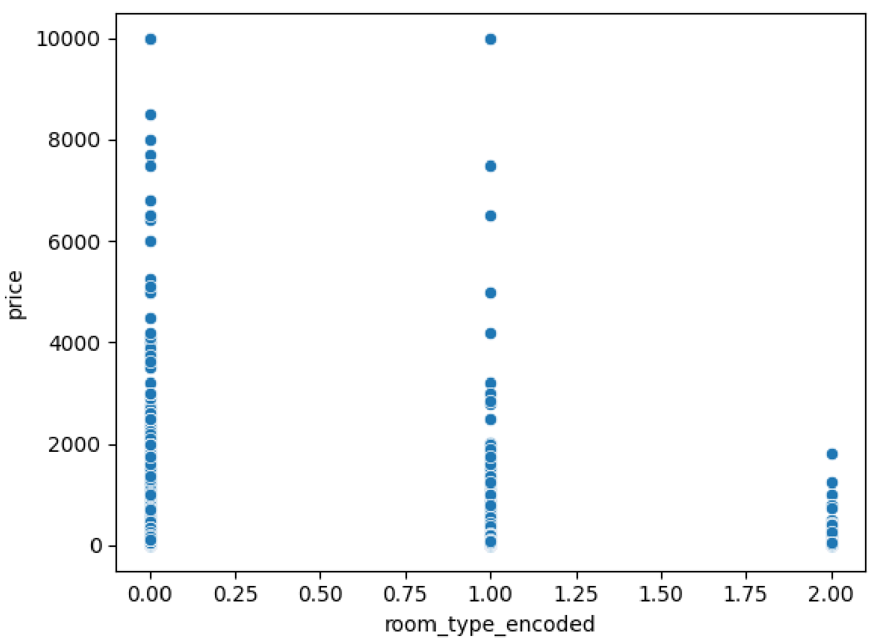
Label encoding is suitable for categorical variables with an inherent order or ordinal relationship. It assigns numeric labels to each category, converting them into ordinal values.

Code:

```
le = LabelEncoder()
ds['room_type_encoded'] = le.fit_transform(ds['room_type']).astype(int)
c = ds['room_type'].tail()
d = ds['room_type_encoded'].tail()
print("Encoded data:",d)
sb.scatterplot(x='room_type_encoded', y='price', data=ds)
```

Output:

```
Encoded data:
48890    1
48891    1
48892    0
48893    2
48894    1
```

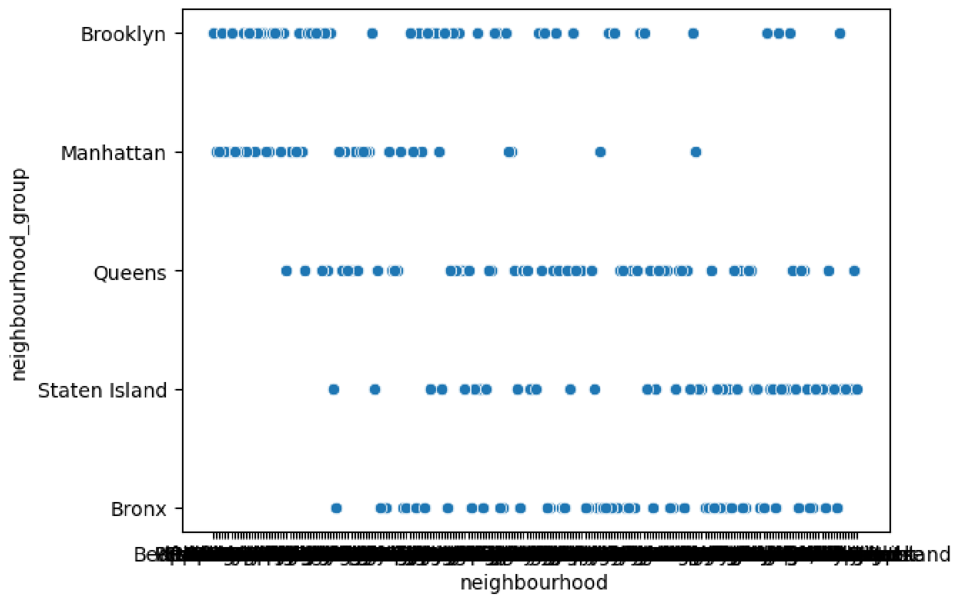


Binary Encoding:

Binary encoding combines aspects of one-hot encoding and label encoding. It represents each category as a binary code, reducing the number of columns compared to one-hot encoding while preserving the ordinal relationship between categories.

Code:

```
binary_encoder = BinaryEncoder(cols=['neighbourhood'])
ds_encoded = binary_encoder.fit_transform(ds)
sb.scatterplot(x='neighbourhood', y='neighbourhood_group', data=ds)
```



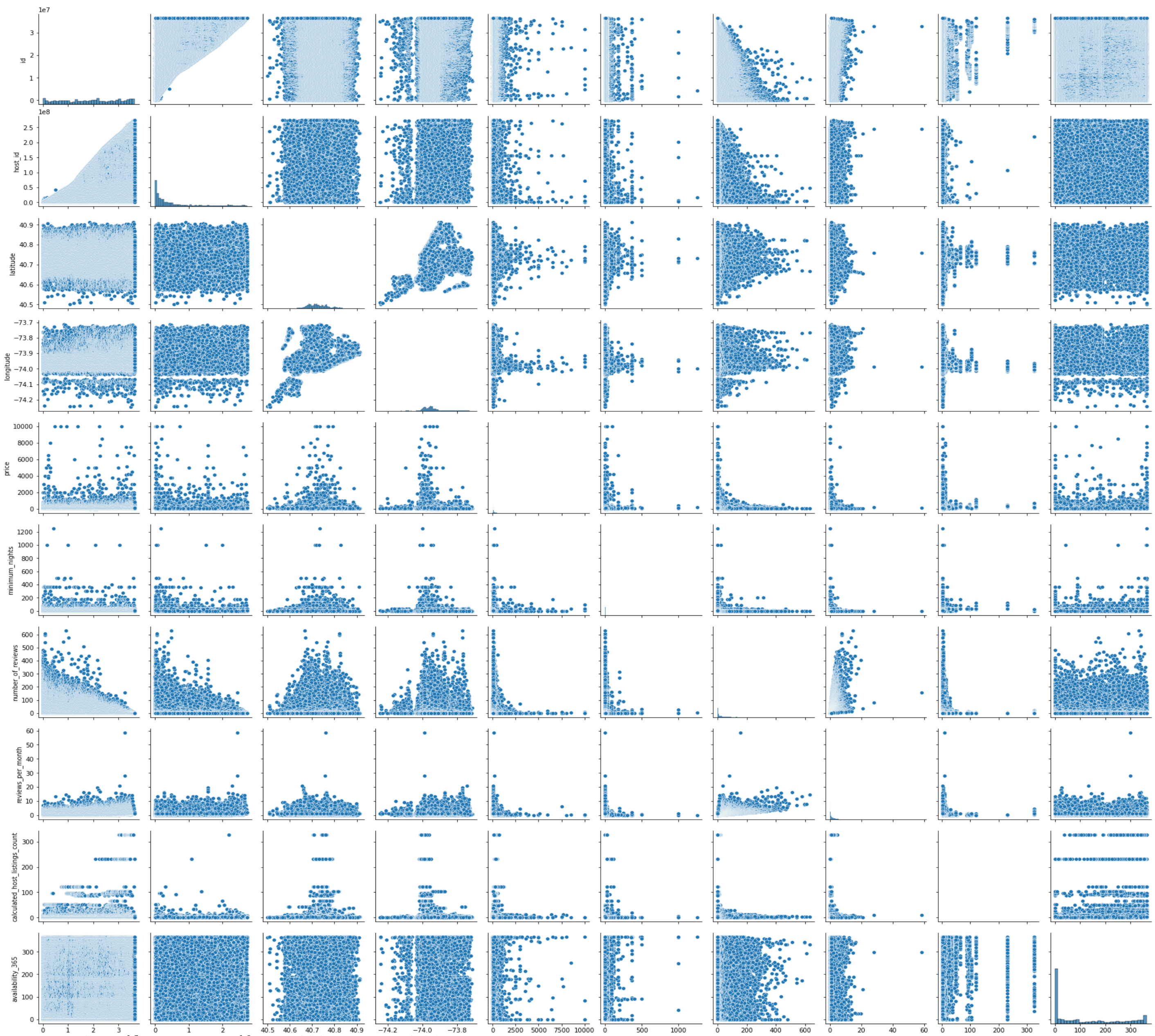
Step 3
Summarize insights obtained from your data set after performing EDA

PairPlot:

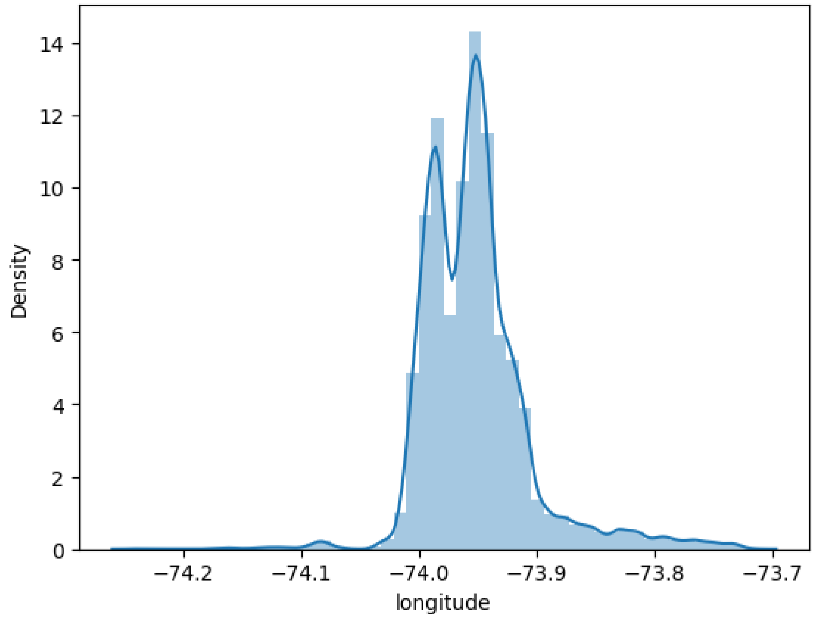
A pair plot is a type of plot in the seaborn library in Python that shows the pairwise relationships between variables in a data set. It is a matrix of scatterplots where each variable is plotted against every other variable.

Code:

```
sb.pairplot(ds)
```

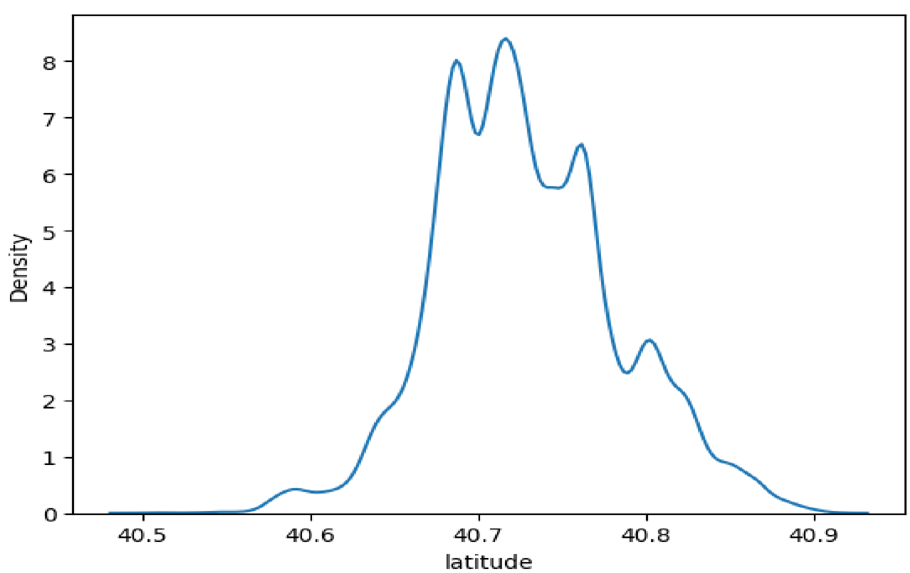


Distplot:
sb.distplot(ds['longitude'])



Kdeplot:
In the seaborn library is used to plot the Kernel Density Estimate (KDE) of a univariate variable. The KDE plot provides a smooth estimate of the underlying probability density function of the data.

Code:
sb.kdeplot(ds['latitude'])

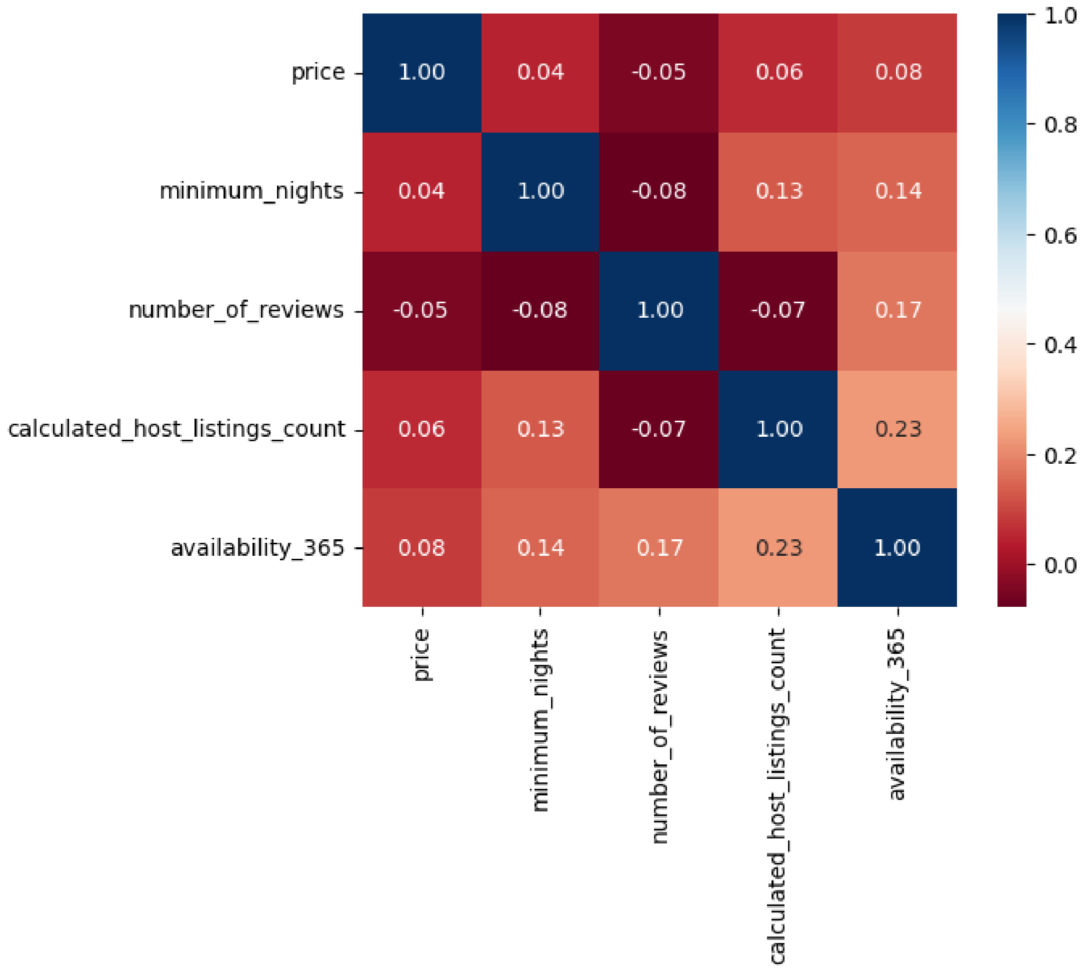


Heat Map:

A heatmap is a graphical representation of data where values are encoded as colors on a two-dimensional grid. It is commonly used to visualize the relationships and patterns in a matrix or a table of data. Heatmaps are particularly useful for displaying large datasets and identifying areas of high or low concentration.

Code:

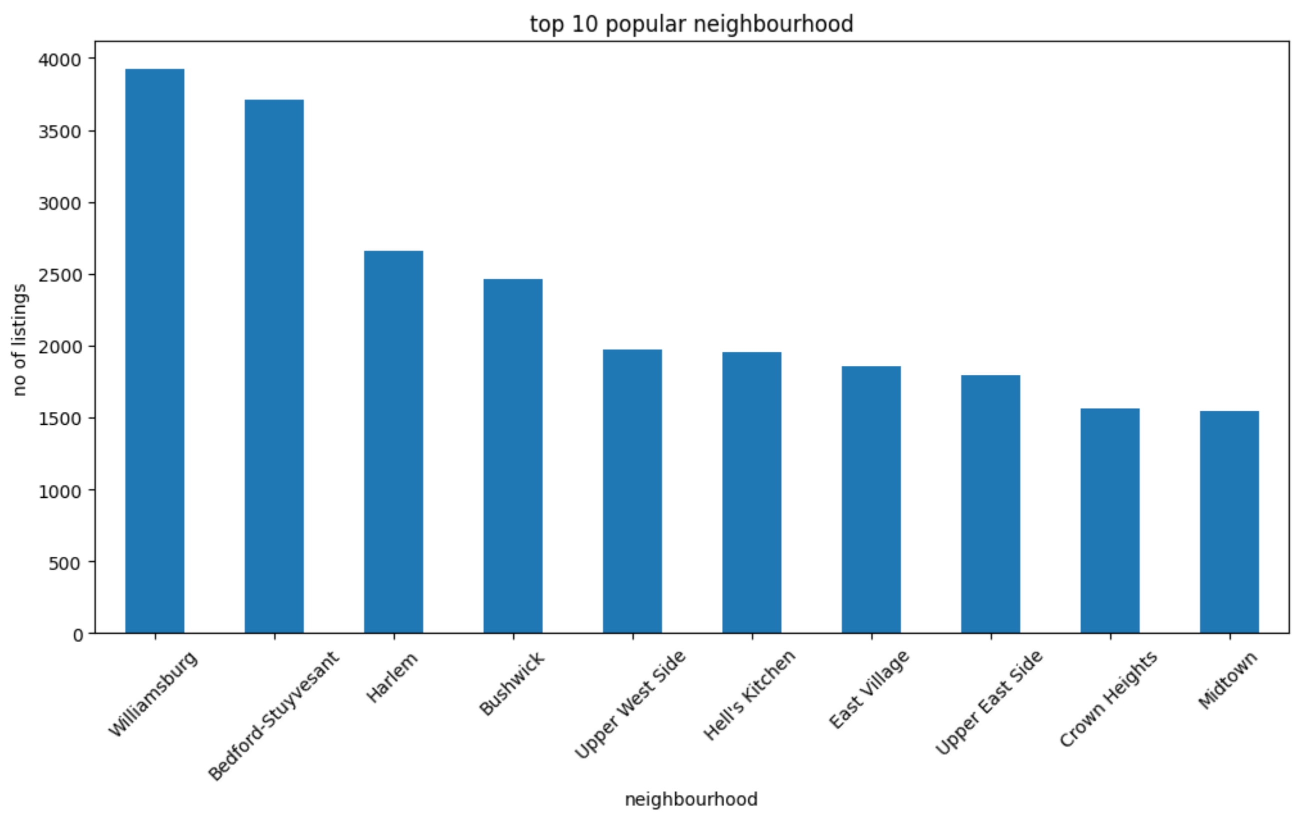
```
corr = ds[['price', 'minimum_nights', 'number_of_reviews', 'calculated_host_listings_count',
'availability_365']].corr()
sb.heatmap(corr, cmap='RdBu', fmt='.2f', square=True, linecolor='white', annot=True);
```



```
popular_neighbourhoods = ds['neighbourhood'].value_counts().head(10)
print(popular_neighbourhoods)
plt.figure(figsize = (12,6))
popular_neighbourhoods.plot(kind = 'bar')
plt.xlabel('neighbourhood')
plt.ylabel('no of listings')
plt.title('top 10 popular neighbourhood')
plt.show
```

Output:

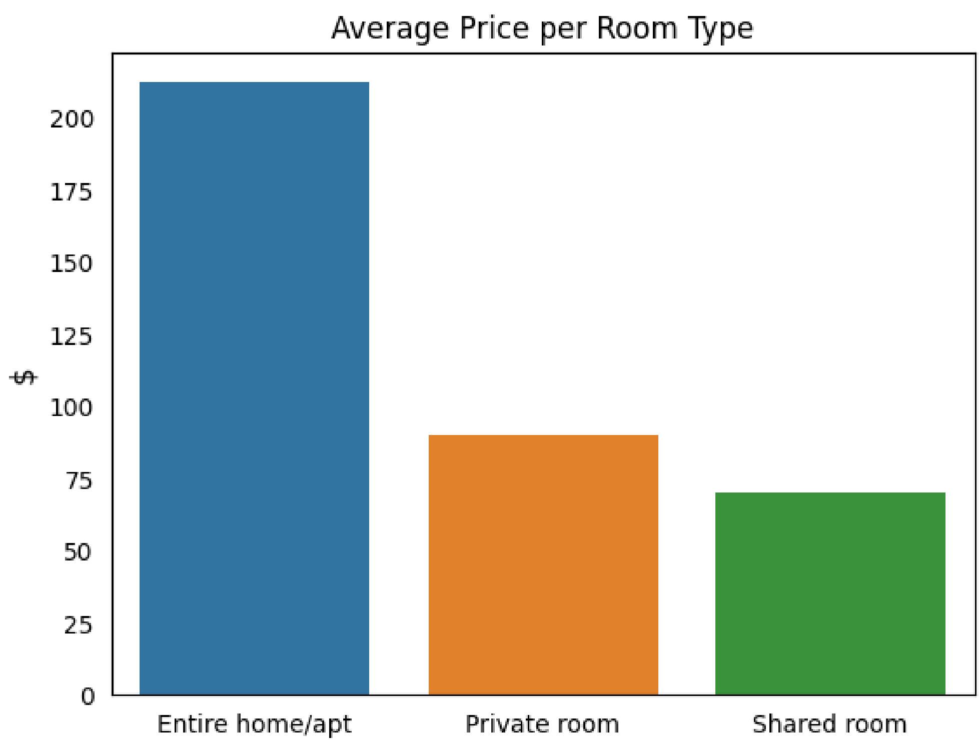
```
Williamsburg          3920
Bedford-Stuyvesant    3714
Harlem                2658
Bushwick              2465
Upper West Side       1971
Hell's Kitchen        1958
East Village          1853
Upper East Side       1798
Crown Heights         1564
Midtown               1545
```



```
type_price = round(df.groupby('room_type').price.mean(), 2).sort_values(ascending=False)
print(type_price)
print('\n')ax = sb.barplot(x=type_price.index, y=type_price.values)
ax.set_title('Average Price per Room Type')
ax.tick_params(bottom=False, top=False, left=False, right=False)
ax.set_ylabel('$', fontsize=12)
ax.set_xlabel('')
```

Output:

```
room_type
Entire home/apt    211.79
Private room        89.78
Shared room         70.13
```



Conclusion:

- The number of Airbnb listings in NewYork City affects the availability, options, and pricing of accommodations. More listings provide a wider range of choices, while fewer listings may result in limited availability and higher prices. Additionally, the choice of neighborhood influences the overall experience, convenience, and amenities available during a stay. It's important to consider both factors when selecting an Airbnb for a satisfying and tailored experience.

- The average price per room type for Airbnb listings in New York City varies based on factors such as location and the type of accommodation. Private rooms tend to be more affordable compared to entire homes or apartments.