About Dataset

As of August 2019, This dataset has around 49,000 observations in it with 16 columns and it is a mix between categorical and numeric values.. The purpose of this task is to **predict the price of NYC Airbnb rentals** based on the data provided and any external dataset(s) with relevant information.

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Loading Dataset

In [2]:

```
df = pd.read_csv("AB_NYC_2019.csv")
```

In [3]:

```
df.head()
```

Out[3]:

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

print dataset info

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
id                                48895 non-null int64
name                              48879 non-null object
host_id                           48895 non-null int64
host_name                         48874 non-null object
neighbourhood_group               48895 non-null object
neighbourhood                     48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
last_review                       38843 non-null object
reviews_per_month                 38843 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

Check for the null values in each column

In [151]:

```
df.isnull().sum()
```

Out[151]:

```
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
```

Drop unnecessary columns

In [5]:

```
df.drop(['id','name','host_id','host_name','last_review'], axis=1, inplace=True)
```

Rreplace the 'reviews per month' by zero

In [6]:

```
df.fillna({'reviews_per_month':0}, inplace=True)
```

Remove the NaN values from the dataset

In [7]:

```
df.dropna(how='any',inplace=True)
```

Again check for null values

In [99]:

```
df.isnull().sum()
```

Out[99]:

```
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
```

Examining Changes

In [100]:

```
df.head()
```

Out[100]:

	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nigh
0	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
2	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
3	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	

Examine Continous Variables

In [101]:

```
df.describe()
```

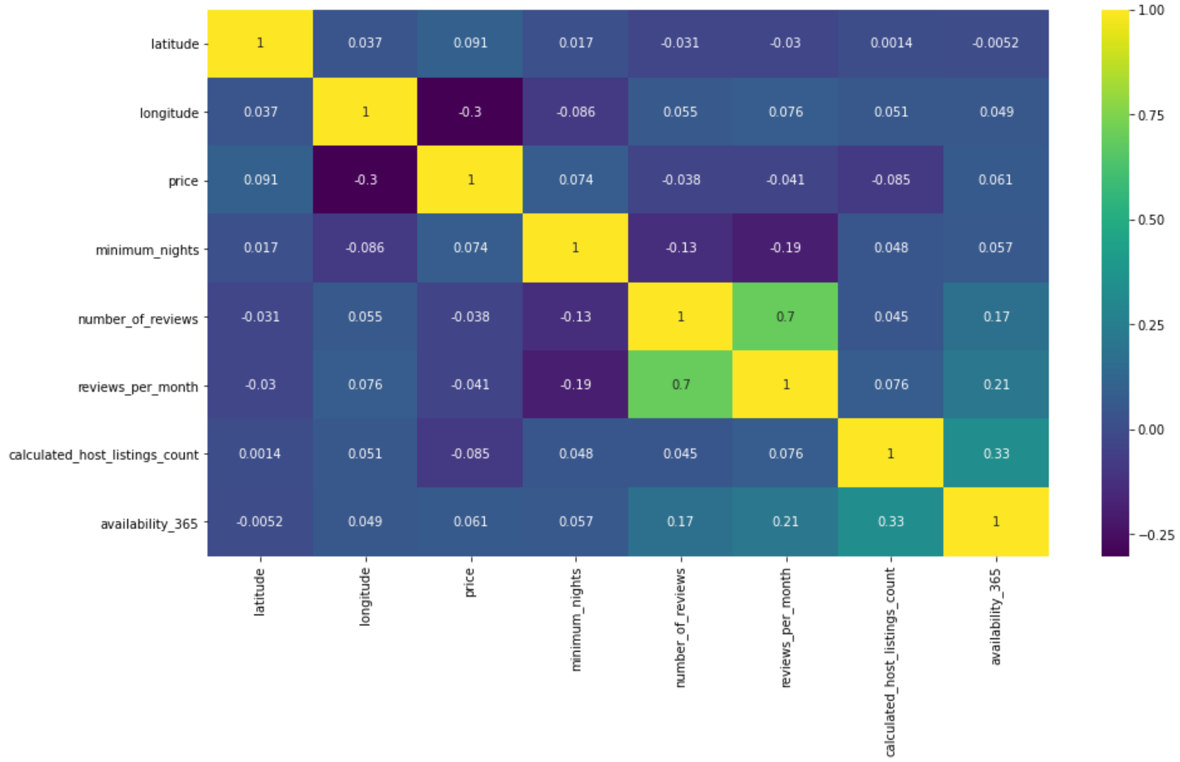
Out[101]:

	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_
count	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	48
mean	40.728949	-73.952170	152.720687	7.029962	23.274466	
std	0.054530	0.046157	240.154170	20.510550	44.550582	
min	40.499790	-74.244420	0.000000	1.000000	0.000000	
25%	40.690100	-73.983070	69.000000	1.000000	1.000000	
50%	40.723070	-73.955680	106.000000	3.000000	5.000000	
75%	40.763115	-73.936275	175.000000	5.000000	24.000000	
max	40.913060	-73.712990	10000.000000	1250.000000	629.000000	

Correlation between different variables

In [102]:

```
corr = df.corr(method='kendall')
plt.figure(figsize=(15,8))
sns.heatmap(corr, annot=True,cmap="viridis")
plt.show()
```



Data Visualization

In [103]:

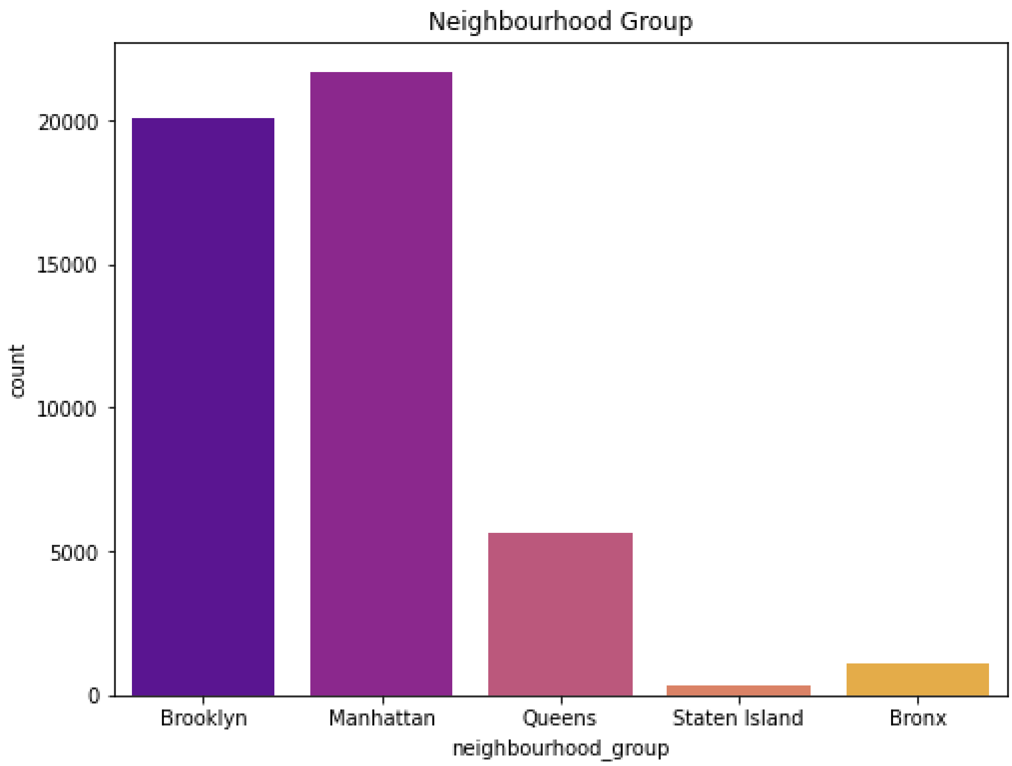
```
df["neighbourhood_group"].unique()
```

Out[103]:

```
array(['Brooklyn', 'Manhattan', 'Queens', 'Staten Island', 'Bronx'],
      dtype=object)
```

In [104]:

```
plt.figure(figsize=(8,6))
sns.countplot(df['neighbourhood_group'], palette="plasma")
plt.title('Neighbourhood Group')
plt.show()
```



In [105]:

```
df["neighbourhood"].nunique()
```

Out[105]:

```
221
```

In [106]:

```
from wordcloud import WordCloud
```

In [107]:

```
plt.figure(figsize=(25,15))
wordcloud = WordCloud(
                          background_color='white',
                          width=1920,
                          height=1080
                         ).generate(" ".join(df.neighbourhood))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

In [108]:

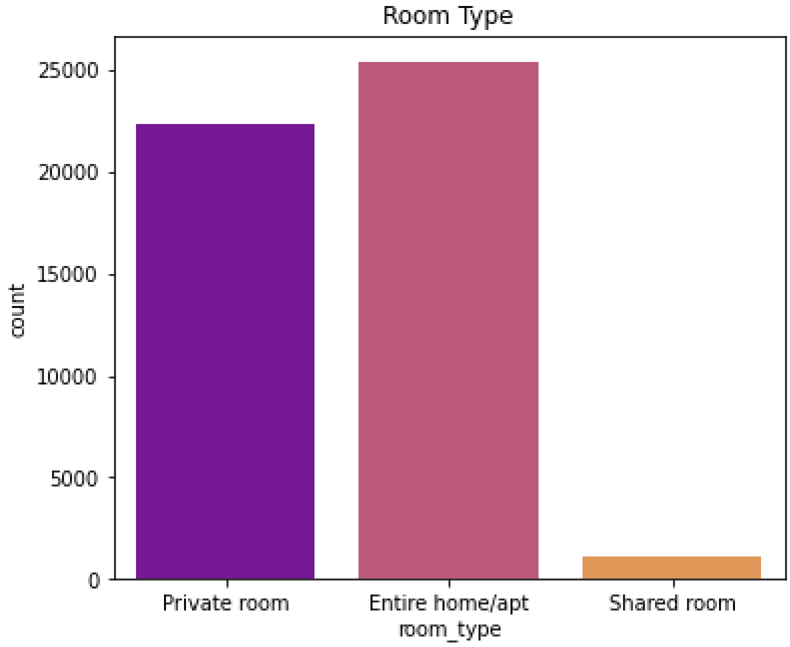
```
df["room_type"].unique()
```

Out[108]:

```
array(['Private room', 'Entire home/apt', 'Shared room'], dtype=object)
```

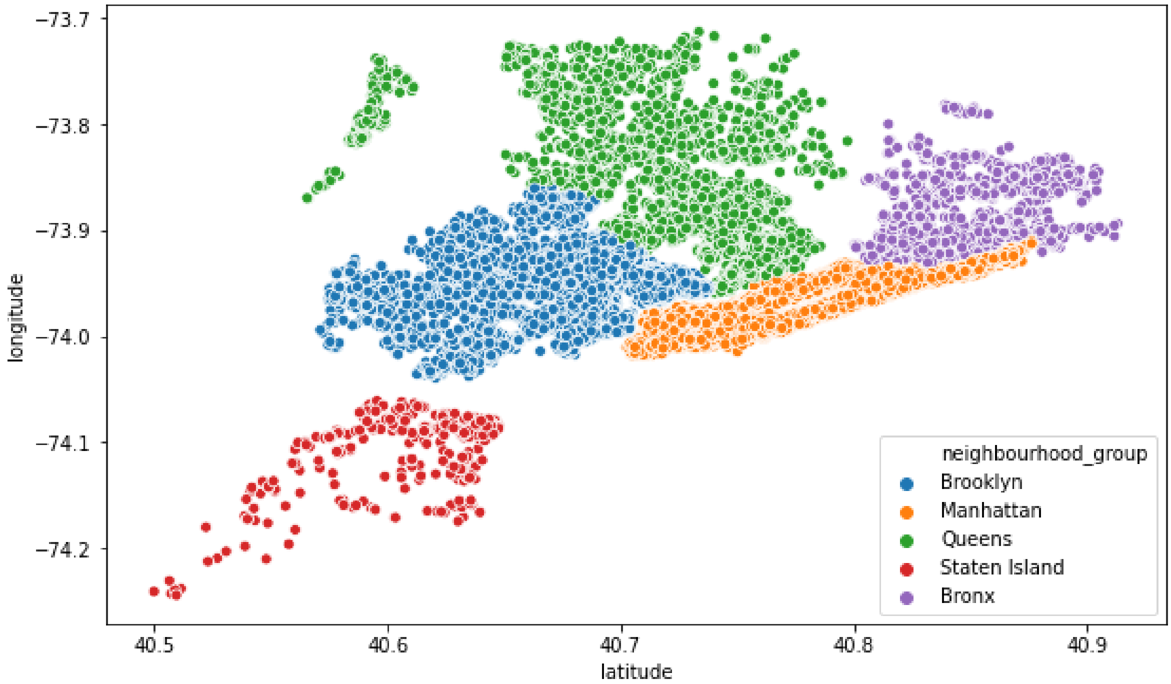
In [109]:

```
plt.figure(figsize=(6,5))
sns.countplot(df['room_type'], palette="plasma")
plt.title('Room Type')
plt.show()
```



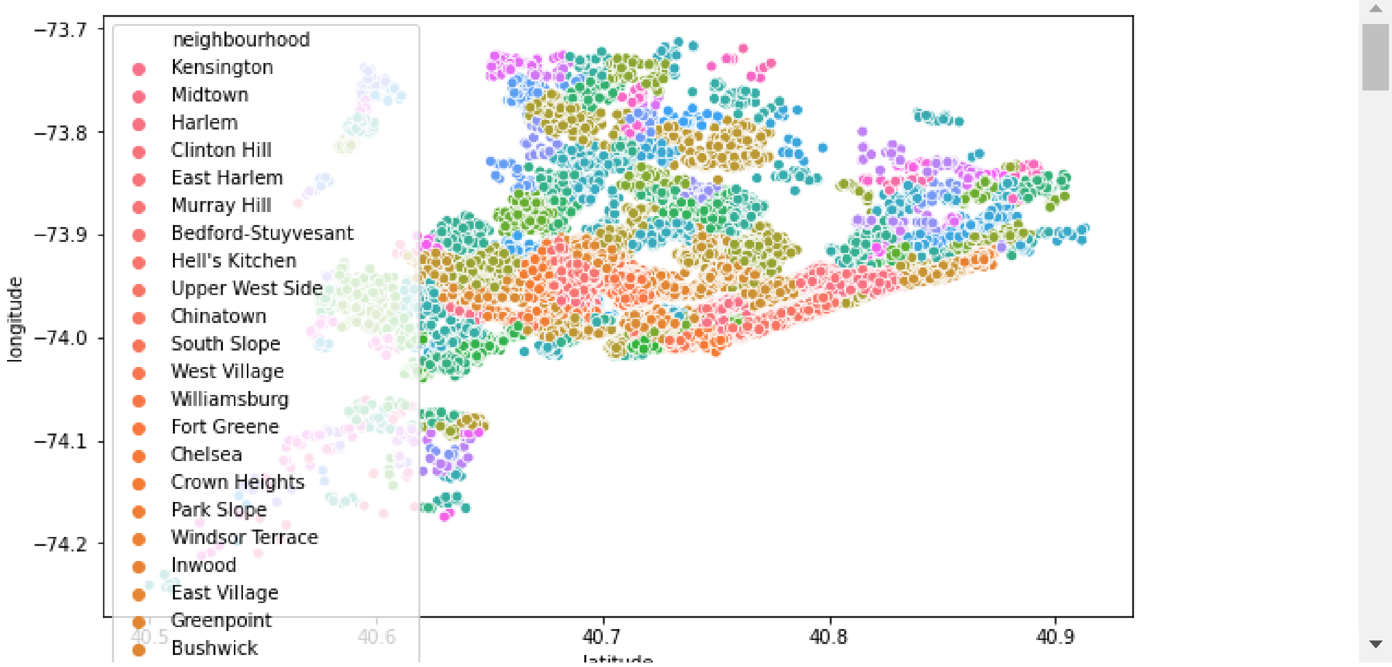
In [110]:

```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df,x="latitude",y="longitude",hue="neighbourhood_group")
plt.show()
```



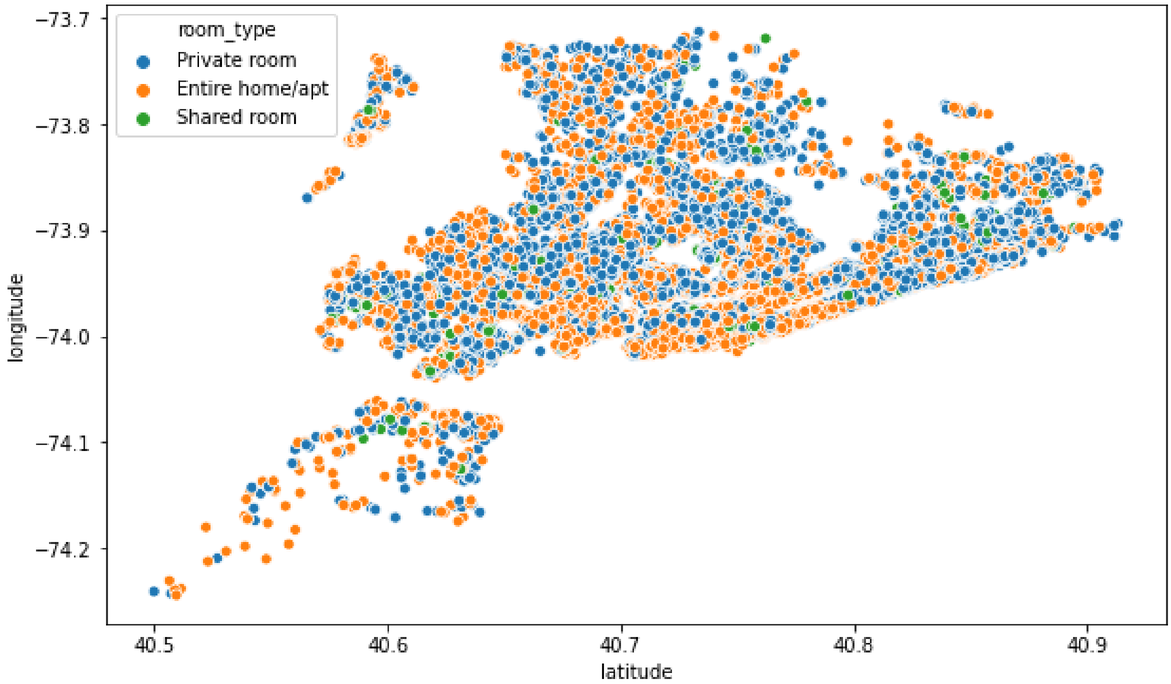
In [111]:

```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df,x="latitude",y="longitude",hue="neighbourhood")
plt.show()
```



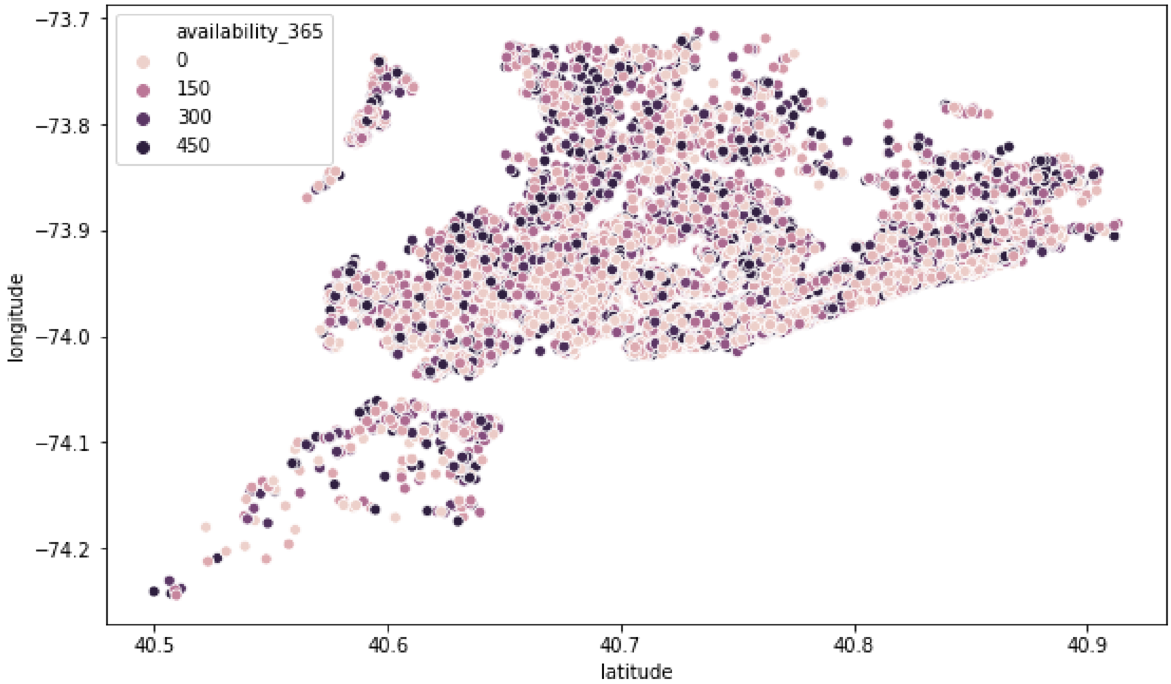
In [112]:

```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df,x="latitude",y="longitude",hue="room_type")
plt.show()
```



In [113]:

```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df,x="latitude",y="longitude",hue="availability_365")
plt.show()
```



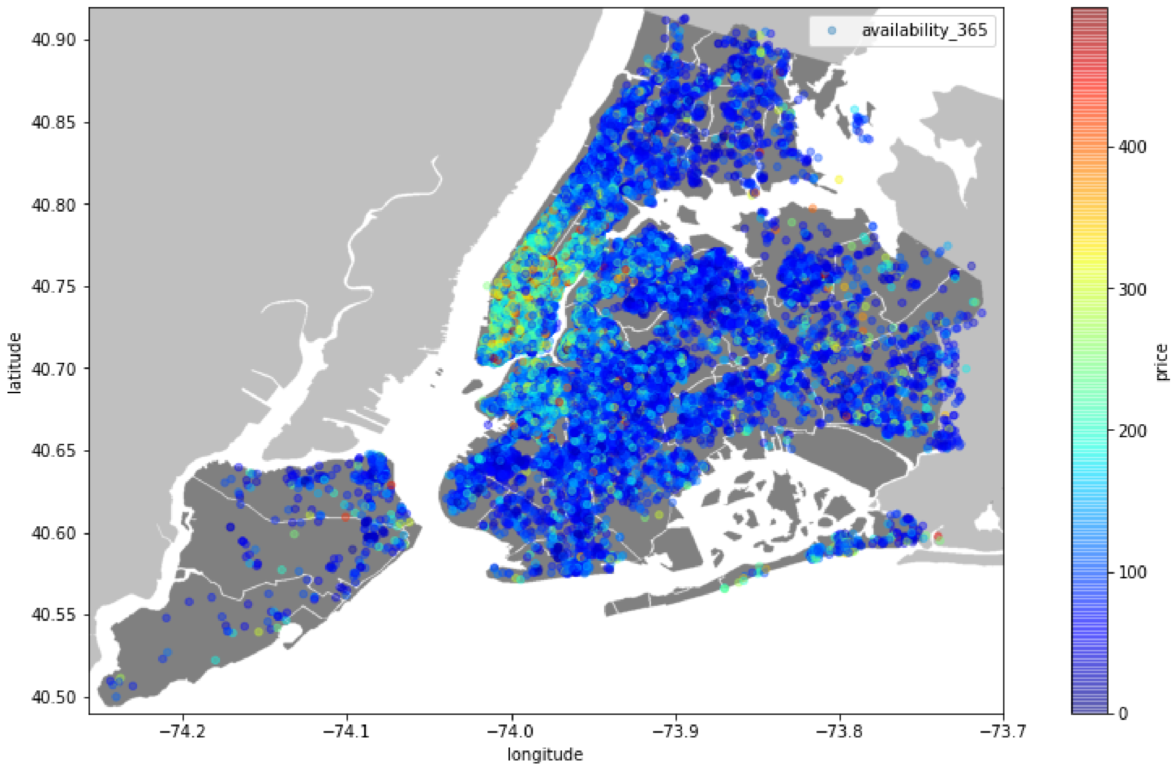
Scatterplot in url image

In [114]:

```
#creating a sub-dataframe with no extreme values / less than 500
sub_df = df[df.price < 500]
```

In [115]:

```
import urllib
#initializing the figure size
plt.figure(figsize=(15,8))
#loading the png NYC image found on Google and saving to my local folder along with the pro
i=urllib.request.urlopen('https://upload.wikimedia.org/wikipedia/commons/e/ec/Neighbourhood
nyc_img=plt.imread(i)
#scaling the image based on the latitude and longitude max and mins for proper output
plt.imshow(nyc_img,zorder=0,extent=[-74.258, -73.7, 40.49,40.92])
ax=plt.gca()
#using scatterplot again
sub_df.plot(kind='scatter', x='longitude', y='latitude', label='availability_365', c='price
           cmap=plt.get_cmap('jet'), colorbar=True, alpha=0.4, zorder=5)
plt.legend()
plt.show()
```

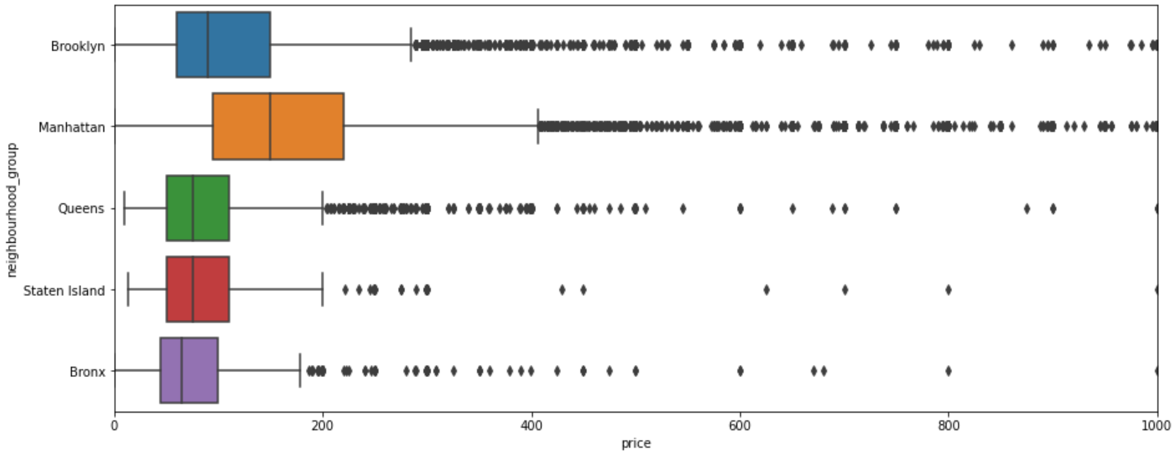


Handling outliers

Relation between neighbporhood_group and price

In [116]:

```
plt.figure(figsize=(15,6))
sns.boxplot(data=df,x="price",y="neighbourhood_group")
plt.xlim(0,1000)
plt.show()
```



In [8]:

```
df.drop(df[(df['neighbourhood_group'] == "Manhattan") & (df['price'] > 450)].index,axis=0,i
```

In [9]:

```
df.drop(df[(df['neighbourhood_group'] == "Brooklyn") & (df['price'] > 300)].index,axis=0,ir
```

In [10]:

```
df.drop(df[(df['neighbourhood_group'] == "Queens") & (df['price'] > 200)].index,axis=0,inpl
```

In [11]:

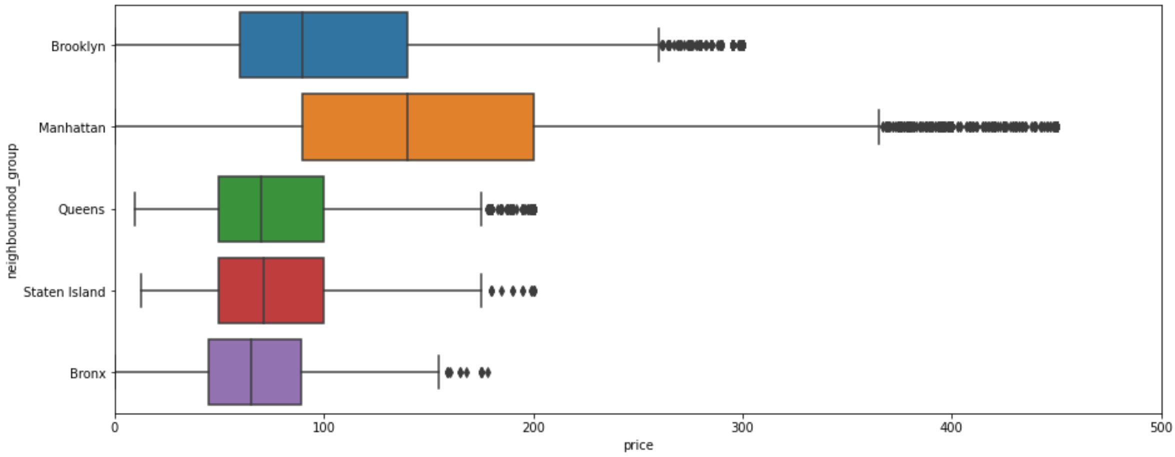
```
df.drop(df[(df['neighbourhood_group'] == "Staten Island") & (df['price'] > 220)].index,axis
```

In [12]:

```
df.drop(df[(df['neighbourhood_group'] == "Bronx") & (df['price'] > 180)].index,axis=0,inpla
```

In [122]:

```
plt.figure(figsize=(15,6))
sns.boxplot(data=df,x="price",y="neighbourhood_group")
plt.xlim(0,500)
plt.show()
```



Now, look somewhat cleaned with a few amoumt of outlier in the dataset.

Separate categorical and numerical data

In [13]:

```
df_cat = df.select_dtypes(object)
df_cat.head()
```

Out[13]:

	neighbourhood_group	neighbourhood	room_type
0	Brooklyn	Kensington	Private room
1	Manhattan	Midtown	Entire home/apt
2	Manhattan	Harlem	Private room
3	Brooklyn	Clinton Hill	Entire home/apt
4	Manhattan	East Harlem	Entire home/apt

In [14]:

```
df_num = df.select_dtypes(["float64","int64"])
df_num.head()
```

Out[14]:

	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculat
0	40.64749	-73.97237	149	1	9	0.21	
1	40.75362	-73.98377	225	1	45	0.38	
2	40.80902	-73.94190	150	3	0	0.00	
3	40.68514	-73.95976	89	1	270	4.64	
4	40.79851	-73.94399	80	10	9	0.10	

Applying label encoding to categorical columns

In [15]:

```
from sklearn.preprocessing import LabelEncoder
```

In [16]:

```
for col in df_cat:
    le = LabelEncoder()
    df_cat[col] = le.fit_transform(df_cat[col])
```

In [17]:

```
# Check for change
df_cat.head()
```

Out[17]:

	neighbourhood_group	neighbourhood	room_type
0	1	107	1
1	2	126	0
2	2	93	1
3	1	41	0
4	2	61	0

Skewness

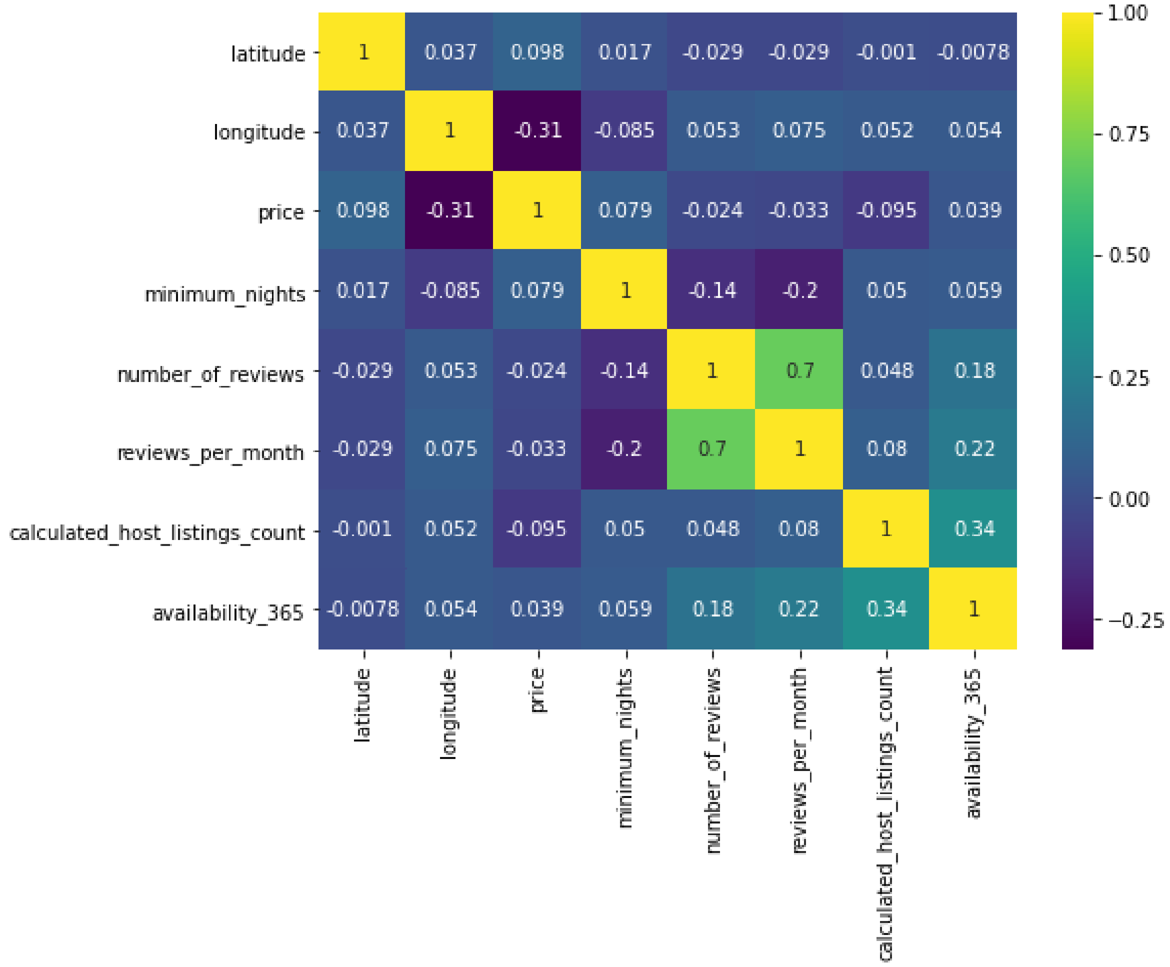
In [18]:

```
from scipy.stats import skew
```

Get Correlation between different numerical variables

In [167]:

```
corr = df_num.corr(method='kendall')
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True,cmap="viridis")
plt.show()
```

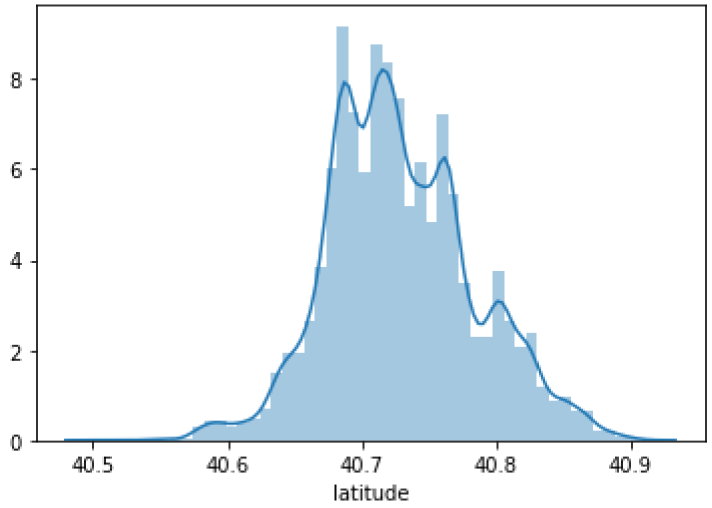


In [190]:

```
for col in df_num:
    print("Column: ",col)
    print("Skewness: ",skew(df_num[col]))

    plt.figure()
    sns.distplot(df_num[col])
    plt.show()
    print("---------------------------------------------------")
```

```
Column:  latitude
Skewness:  0.2490138817984112
```




```
---------------------------------------------------
```

In [20]:

```
skew_columns = df_num.columns[[3,4,5,6,7]]
skew_columns
```

Out[20]:

```
Index(['minimum_nights', 'number_of_reviews', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365'],
      dtype='object')
```

In [21]:

```
skewness = {}
for col in skew_columns:
    skewness[col] = skew(df_num[col])
print("Skewness before skew")
pd.DataFrame(data=list(skewness.values()),index=list(skewness.keys()),columns=["Skewness"])
```

```
Skewness before skew
```

Out[21]:

	Skewness
minimum_nights	22.569695
number_of_reviews	3.650308
reviews_per_month	3.318658
calculated_host_listings_count	7.976658
availability_365	0.802924

Reducing skewness all numerical columns excluding latitude,longitude and price columns

In [22]:

```
for col in skew_columns:
    df_num[col] = np.sqrt(df_num[col])
```

In [23]:

```
skewness = {}
for col in skew_columns:
    skewness[col] = skew(df_num[col])
print("Skewness after skewed")
pd.DataFrame(data=list(skewness.values()),index=list(skewness.keys()),columns=["Skewness"])
```

```
Skewness after skewed
```

Out[23]:

	Skewness
minimum_nights	3.877927
number_of_reviews	1.476768
reviews_per_month	0.819086
calculated_host_listings_count	5.535107
availability_365	0.309342

Joining both the dataframe

In [24]:

```
df_new = pd.concat([df_num,df_cat],axis=1)
df_new.head()
```

Out[24]:

	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculat
0	40.64749	-73.97237	149	1.000000	3.000000	0.458258	
1	40.75362	-73.98377	225	1.000000	6.708204	0.616441	
2	40.80902	-73.94190	150	1.732051	0.000000	0.000000	
3	40.68514	-73.95976	89	1.000000	16.431677	2.154066	
4	40.79851	-73.94399	80	3.162278	3.000000	0.316228	

Defining the independent variables and dependent variables

In [25]:

```
from sklearn.model_selection import train_test_split
```

In [26]:

```
X = df_new.drop('price',axis=1)
y = df_new['price']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

Standard scaling of training data

In [27]:

```
from sklearn.preprocessing import StandardScaler
```

In [28]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Creating Common model function

In [29]:

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [65]:

```
a,b,c,d,e = [],[],[],[],[]
def create_model(model):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    a.append(mean_absolute_error(y_test,y_pred))
    b.append(mean_squared_error(y_test,y_pred))
    c.append(np.sqrt(mean_squared_error(y_test,y_pred)))
    d.append(r2_score(y_test,y_pred))
    e.append(1 - (1-r2_score(y_test,y_pred)) * (len(X_test)-1)/(len(X_test)-len(X.columns)-
```

In [31]:

```
def select_model(model):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test,y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test,y_pred)
    print("mse: {},\nrmse: {},\nr2: {}".format(mse,rmse,r2))
    return model
```

Baseline model - Linear Regression

In [32]:

```
from sklearn.linear_model import LinearRegression
```

In [33]:

```
ln = LinearRegression()
select_model(ln)
```

```
mse: 3379.2571717292126,
rmse: 58.13137854660951,
r2: 0.42915292159866814
```

Out[33]:

```
LinearRegression()
```

In [66]:

```
create_model(ln)
pd.DataFrame([a,b,c,d,e],index = ['MAE','MSE','RMSE','R2','Adjusted R2'], columns = ['Linea
```

Out[66]:

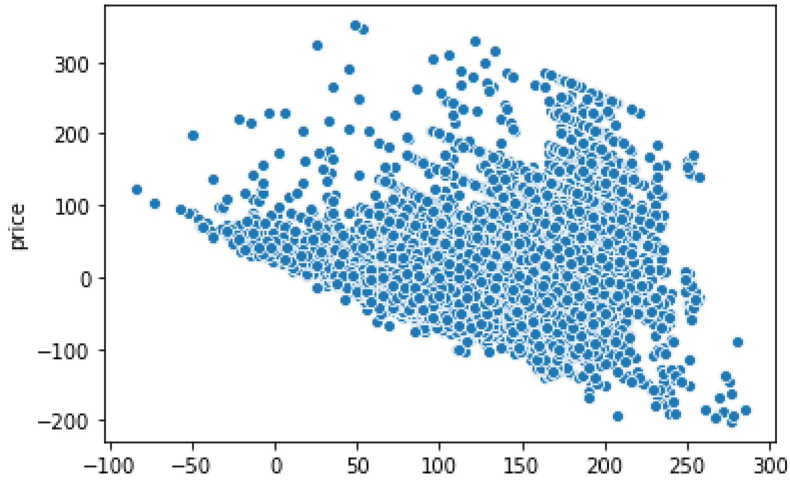
	Linear Reg
MAE	41.710448
MSE	3379.257172
RMSE	58.131379
R2	0.429153
Adjusted R2	0.428745

In [35]:

```
ln.fit(X_train,y_train)
y_pred = ln.predict(X_test)
residuals = y_test - y_pred
```

In [36]:

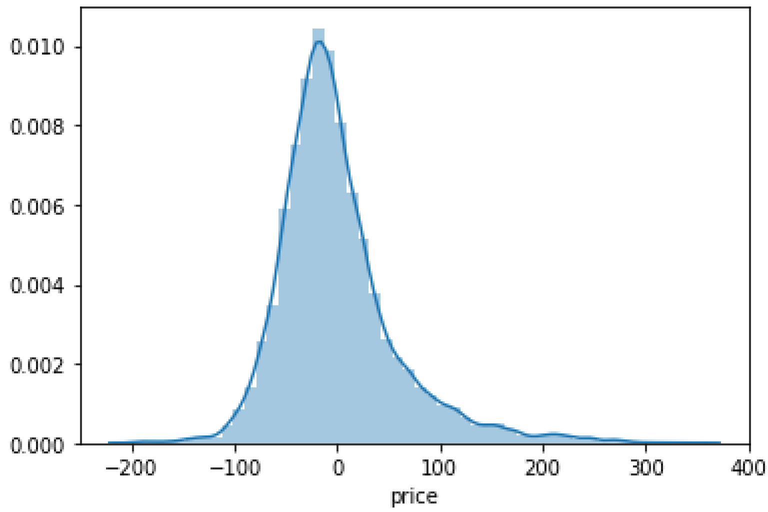
```
plt.figure()
sns.scatterplot(y_pred,residuals)
plt.show()
```



This shows clearly no linear relationship

In [37]:

```
plt.figure()
sns.distplot(residuals)
plt.show()
```



This shows positive distribution. Hence we can not appling polynomial regression

Gradient Descent

In [38]:

```
from sklearn import linear_model
```

In [39]:

```
gdm = linear_model.SGDRegressor(max_iter=1000, tol=1e-3)
gdm.fit(X_train,y_train)
```

Out[39]:

```
SGDRegressor()
```

In [40]:

```
y_pred = gdm.predict(X_test)
```

In [41]:

```
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,y_pred)

print("\nmse: {},\nrmse: {},\nr2: {}".format(mse,rmse,r2))
```

```
mse: 3384.7489838004108,
rmse: 58.17859558119645,
r2: 0.428225207395026
```

No major improvement on MSE,RMSE and R2 score

Regularization

In [42]:

```
print(ln.score(X_train,y_train))
print(ln.score(X_test,y_test))
print("Difference:",(ln.score(X_train,y_train)-ln.score(X_test,y_test)))
```

```
0.4400615227304835
0.42915292159866814
Difference: 0.010908601131815354
```

In [43]:

```
from sklearn.linear_model import Lasso # Lambda*sum(abs(coef))
from sklearn.linear_model import Ridge # Lambda*sum(square(coef))
```

Finding right lambda

In [44]:

```
# Ridge
for i in range(1,200,10):
    l2 = Ridge(i)
    l2.fit(X_train,y_train)
    print(i,":",l2.score(X_test,y_test))
```

```
1 : 0.4291535557745375
11 : 0.4291598457492338
21 : 0.42916604172866013
31 : 0.42917214393729963
41 : 0.4291781525989914
51 : 0.4291840679369321
61 : 0.42918989017367903
71 : 0.42919561953115215
81 : 0.42920125623063665
91 : 0.42920680049278515
101 : 0.4292122525376201
111 : 0.4292176125845366
121 : 0.42922288085230387
131 : 0.42922805755906845
141 : 0.4292331429223557
151 : 0.4292381371590728
161 : 0.42924304048551065
171 : 0.42924785311734637
181 : 0.42925257526964533
191 : 0.4292572071568638
```

In [45]:

```
# Lasso
for i in range(1,200,20):
    l1 = Lasso(i)
    l1.fit(X_train,y_train)
    print(i,":",l1.score(X_test,y_test))
```

```
1 : 0.42759960859531043
21 : 0.26183292366876854
41 : 0.05253819655597536
61 : -9.433527526581109e-05
81 : -9.433527526581109e-05
101 : -9.433527526581109e-05
121 : -9.433527526581109e-05
141 : -9.433527526581109e-05
161 : -9.433527526581109e-05
181 : -9.433527526581109e-05
```

Ridge model

In [46]:

```
# Ridge - 1
# Lasso - 1
```

In [47]:

```
l2 = Ridge(alpha=1)
l2.fit(X_train,y_train)
print(l2.score(X_test,y_test))
```

```
0.4291535557745375
```

In [48]:

```
c = -1
for col in X:
    c = c + 1
    print(f"Coef of {col}:",l2.coef_[c])
```

```
Coef of latitude: 7.028919288372428
Coef of longitude: -20.7536824197843
Coef of minimum_nights: -9.012002992924492
Coef of number_of_reviews: -6.301814766722203
Coef of reviews_per_month: -1.3574347254768941
Coef of calculated_host_listings_count: 5.218134034936023
Coef of availability_365: 11.660911325615176
Coef of neighbourhood_group: 4.9172551446257655
Coef of neighbourhood: 3.546112850165196
Coef of room_type: -41.85533864464062
```

Lasso model

In [49]:

```
l1 = Lasso(alpha=1)
l1.fit(X_train,y_train)
print(l1.score(X_test,y_test))
```

```
0.42759960859531043
```

In [50]:

```
c = -1
for col in X:
    c = c + 1
    print(f"Coef of {col}:",l1.coef_[c])
```

```
Coef of latitude: 6.245796490754709
Coef of longitude: -19.72970820934962
Coef of minimum_nights: -6.5431850875561075
Coef of number_of_reviews: -5.514870634257459
Coef of reviews_per_month: -0.0
Coef of calculated_host_listings_count: 4.479161112259227
Coef of availability_365: 9.658108123466958
Coef of neighbourhood_group: 4.2699500882557615
Coef of neighbourhood: 2.902160881346436
Coef of room_type: -40.748477379046214
```

Droping "review per month" column

In [51]:

```
X.drop('reviews_per_month',axis=1,inplace=True)
```

In [52]:

```
X = df_new.drop('price',axis=1)
y = df_new['price']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

In [53]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Decision Tree model

In [54]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [55]:

```
dt1 = DecisionTreeRegressor()
select_model(dt1)
```

```
mse: 4990.8313330475785,
rmse: 70.64581610433542,
r2: 0.15691486605437566
```

Out[55]:

```
DecisionTreeRegressor()
```

Purning Techniques

In [56]:

```
dt1.get_depth()
```

Out[56]:

```
39
```

In [57]:

```
dt2 = DecisionTreeRegressor(max_depth=7,random_state=1)
select_model(dt2)
```

```
mse: 2799.51376133571,
rmse: 52.91043149829446,
r2: 0.5270871169639194
```

Out[57]:

```
DecisionTreeRegressor(max_depth=7, random_state=1)
```

In [58]:

```
dt3 = DecisionTreeRegressor(min_samples_split=55,random_state=1)
select_model(dt3)
```

```
mse: 3108.207314928029,
rmse: 55.75129877346382,
r2: 0.47494050478425875
```

Out[58]:

```
DecisionTreeRegressor(min_samples_split=55, random_state=1)
```

In [59]:

```
dt4 = DecisionTreeRegressor(min_samples_leaf=75,random_state=1)
select_model(dt4)
```

```
mse: 2743.866543724713,
rmse: 52.38192955327928,
r2: 0.5364874230016352
```

Out[59]:

```
DecisionTreeRegressor(min_samples_leaf=75, random_state=1)
```

In [60]:

```
dt5 = DecisionTreeRegressor(max_features=4,random_state=1)
select_model(dt5)
```

```
mse: 5001.184097728247,
rmse: 70.71905045833299,
r2: 0.15516600671310454
```

Out[60]:

```
DecisionTreeRegressor(max_features=4, random_state=1)
```

In [61]:

```
dt6 = DecisionTreeRegressor(max_leaf_nodes=75,random_state=1)
select_model(dt6)
```

```
mse: 2765.3575226931202,
rmse: 52.58666677678973,
r2: 0.532857020835522
```

Out[61]:

```
DecisionTreeRegressor(max_leaf_nodes=75, random_state=1)
```

In [62]:

```
dt6.feature_importances_
```

Out[62]:

```
array([0.09960489, 0.18398076, 0.02423603, 0.01230719, 0.01176581,
       0.01577025, 0.03057665, 0.        , 0.00818132, 0.6135771 ])
```

In [63]:

```
X.columns
```

Out[63]:

```
Index(['latitude', 'longitude', 'minimum_nights', 'number_of_reviews',
       'reviews_per_month', 'calculated_host_listings_count',
       'availability_365', 'neighbourhood_group', 'neighbourhood',
       'room_type'],
      dtype='object')
```

In [67]:

```
create_model(dt6)
```

In [255]:

```
pd.DataFrame([a,b,c,d,e],index = ['MAE','MSE','RMSE','R2','Adjusted R2'], columns = ['Linea
```

Out[255]:

	Linear Reg	Decision Tree
MAE	41.710448	36.576407
MSE	3379.257172	2765.357523
RMSE	58.131379	52.586667
R2	0.429153	0.532857
Adjusted R2	0.428745	0.532523

Random Forest model

In [68]:

```
from sklearn.ensemble import RandomForestRegressor
```

Finding best value for max_depth

In [256]:

```
for i in range(10,15):
    rf1 = RandomForestRegressor(n_estimators=100,max_depth=i,random_state=1)
    print("Max_depth:",i)
    select_model(rf1)
    print("-------------------------------------------")
```

```
Max_depth: 10
mse: 2529.8351203913576,
rmse: 50.29746634166931,
r2: 0.5726430650516317
-------------------------------------------
Max_depth: 11
mse: 2511.3438523417526,
rmse: 50.113310131558386,
r2: 0.575766735670832
-------------------------------------------
Max_depth: 12
mse: 2506.5581791103823,
rmse: 50.06553883771134,
r2: 0.5765751641045027
-------------------------------------------
Max_depth: 13
mse: 2503.8955677507556,
rmse: 50.03894051387135,
r2: 0.577024950503797
-------------------------------------------
Max_depth: 14
mse: 2509.3249543517168,
rmse: 50.093162750536294,
r2: 0.576107781634674
-------------------------------------------
```

Applying in model

In [86]:

```
# max_depth - 13
rf2 = RandomForestRegressor(n_estimators=100,max_depth=13,random_state=1)
select_model(rf2)
```

```
mse: 2503.8955677507556,
rmse: 50.03894051387135,
r2: 0.577024950503797
```

Out[86]:

```
RandomForestRegressor(max_depth=13, random_state=1)
```

Compare with previous model performance

In [258]:

```
create_model(rf2)
pd.DataFrame([a,b,c,d,e],index = ['MAE','MSE','RMSE','R2','Adjusted R2'], columns = ['Linea
```

Out[258]:

	Linear Reg	Decision Tree	Random Forest
MAE	41.710448	36.576407	34.499214
MSE	3379.257172	2765.357523	2503.895568
RMSE	58.131379	52.586667	50.038941
R2	0.429153	0.532857	0.577025
Adjusted R2	0.428745	0.532523	0.576723

Look like random forest improve the result

Gradient Boosting Regression model

In [70]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

In [267]:

```
for i in range(5,10):
    gb1 = GradientBoostingRegressor(n_estimators=100,max_depth=i,random_state=1)
    print("Max_depth:",i)
    select_model(gb1)
    print("----------------------------------------")
```

```
Max_depth: 5
mse: 2509.021479183322,
rmse: 50.09013355126259,
r2: 0.5761590467217752
----------------------------------------
Max_depth: 6
mse: 2488.6682026751646,
rmse: 49.88655332527158,
r2: 0.5795972604593317
----------------------------------------
Max_depth: 7
mse: 2487.6826512219636,
rmse: 49.876674420233385,
r2: 0.5797637465061413
----------------------------------------
Max_depth: 8
mse: 2501.35428382721,
rmse: 50.01354100468402,
r2: 0.5774542414483514
----------------------------------------
Max_depth: 9
mse: 2505.3663044460855,
rmse: 50.053634278102976,
r2: 0.5767765036697854
----------------------------------------
```

In [71]:

```
# max_depth - 7
gb2 = GradientBoostingRegressor(n_estimators=100,max_depth=7,random_state=1)
select_model(gb2)
```

```
mse: 2487.6826512219636,
rmse: 49.876674420233385,
r2: 0.5797637465061413
```

Out[71]:

```
GradientBoostingRegressor(max_depth=7, random_state=1)
```

In [72]:

```
create_model(gb2)
pd.DataFrame([a,b,c,d,e],index = ['MAE','MSE','RMSE','R2','Adjusted R2'], columns = ['Linea
```

Out[72]:

	Linear Reg	Decision Tree	Random Forest	Gradient Boosting Reg
MAE	41.710448	36.576407	34.499214	34.471202
MSE	3379.257172	2765.357523	2503.895568	2487.682651
RMSE	58.131379	52.586667	50.038941	49.876674
R2	0.429153	0.532857	0.577025	0.579764
Adjusted R2	0.428745	0.532523	0.576723	0.579463

Looking some improvement from our previous models

SVR model

In [73]:

```
from sklearn.svm import SVR
```

In [80]:

```
radial_svr = SVR(kernel="rbf")
select_model(radial_svr)
```

```
mse: 2966.6335228725097,
rmse: 54.46681120528822,
r2: 0.4988561115185446
```

Out[80]:

```
SVR()
```

In [81]:

```
poly_svr = SVR(kernel="poly")
select_model(poly_svr)
```

```
mse: 3765.752123033254,
rmse: 61.365724333973716,
r2: 0.36386356877445614
```

Out[81]:

```
SVR(kernel='poly')
```

Summarizing our findings

In [82]:

```
# Here we go with radial Bais kernel
create_model(radial_svr)
pd.DataFrame([a,b,c,d,e],index = ['MAE','MSE','RMSE','R2','Adjusted R2'], columns = ['Linea
```

Out[82]:

	Linear Reg	Decision Tree	Random Forest	Gradient Boosting Reg	SVR
MAE	41.710448	36.576407	34.499214	34.471202	35.575171
MSE	3379.257172	2765.357523	2503.895568	2487.682651	2966.633523
RMSE	58.131379	52.586667	50.038941	49.876674	54.466811
R2	0.429153	0.532857	0.577025	0.579764	0.498856
Adjusted R2	0.428745	0.532523	0.576723	0.579463	0.498498

Our SVR model cannot get much improvement in validation scores

Cross Validation

In [83]:

```
from sklearn.model_selection import cross_val_score
```

In [85]:

```
def cross_validate(model):
    cross_val = cross_val_score(model,X,y,cv=5)
    print(cross_val)
    print("mean:",cross_val.mean())
```

In [118]:

```
model_list = {"Linear Reg":ln,"Decision Tree":dt6,"Random Forest":rf2,"Gradient Boosting Re
```

In [119]:

```
for k,v in model_list.items():
    print("Model:",k)
    result = cross_validate(v)
    print("---------------------------------------")
```

```
Model: Linear Reg
[0.37709104 0.43207145 0.41145609 0.44482342 0.45133381]
mean: 0.42335516307252163
---------------------------------------
Model: Decision Tree
[0.44346121 0.50017864 0.51849012 0.54735706 0.51959894]
mean: 0.5058171940968844
---------------------------------------
Model: Random Forest
[0.48666555 0.54024003 0.56024257 0.58845271 0.56331639]
mean: 0.5477834519312543
---------------------------------------
Model: Gradient Boosting Reg
[0.47990711 0.54596261 0.56729809 0.59742529 0.5714802 ]
mean: 0.5524146598922576
---------------------------------------
Model: SVR
[-0.14826539 -0.0135213   0.01809621  0.01271365 -0.0442245 ]
mean: -0.035040263627554441
---------------------------------------
```

Conclusion

As we see in our summary there is linear improve in result.First start with baseline model Linear Regression,Decision Tree with some improvement then check Random Forest which gives more improvement in its scores again check from Gradient Boosting Regression it gives some amount of improveness in scores and then last model used is Support vector Regression it shows not much improvement in evaluation scores.Overall, we discovered a very good number of interesting relationships between features and models, explained each step of the process. This data science is very much mimicked on a higher level on Airbnb Data/Machine Learning team for better business decisions, control over the platform, marketing initiatives, implementation of new features and much more. Therefore, I hope this project helps everyone!

In [ ]:

```
```