1. Dataset Definition

For this project, we worked on the Airbnb New York dataset. Airbnb shares annual datasets for each city every year. Besides, all of these datasets are open-sourced. Our main goal for the classification is to predict price with categorical features.

Index	id	name	host_id	host_name	ighbourhood_gro	neighbourhood	latitude	longitude	room_type	price	imum_nic	ber_of_rev	last_review	ws_per_m	_host_listi	ailability_3
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.6475	-73.9724	Private room	149	1	9	2018-10-19	0.21	6	365
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.7536	-73.9838	Entire home/apt	225	1	45	2019-05-21	0.38	2	355
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.809	-73.9419	Private room	150	3	0	0	0	1	365
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.6851	-73.9598	Entire home/apt	89	1	270	2019-07-05	4.64	1	194
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.7985	-73.944	Entire home/apt	80	10	9	2018-11-19	0.1	1	0
5	5099	Large Cozy 1 BR Apartment In Midtown East	7322	Chris	Manhattan	Murray Hill	40.7477	-73.975	Entire home/apt	200	3	74	2019-06-22	0.59	1	129
6	5121	BlissArtsSpace!	7356	Garon	Brooklyn	Bedford-Stuyvesant	40.6869	-73.956	Private room	60	45	49	2017-10-05	0.4	1	0
7	5178	Large Furnished Room Near B'way	8967	Shunichi	Manhattan	Hell's Kitchen	40.7649	-73.9849	Private room	79	2	430	2019-06-24	3.47	1	220
8	5203	Cozy Clean Guest Room - Family Apt	7490	MaryEllen	Manhattan	Upper West Side	40.8018	-73.9672	Private room	79	2	118	2017-07-21	0.99	1	0
9	5238	Cute & Cozy Lower East Side 1 bdrm	7549	Ben	Manhattan	Chinatown	40.7134	-73.9904	Entire home/apt	150	1	160	2019-06-09	1.33	4	188
10	5295	Beautiful 1br on Upper West Side	7702	Lena	Manhattan	Upper West Side	40.8032	-73.9655	Entire home/apt	135	5	53	2019-06-22	0.43	1	6
11	5441	Central Manhattan/near Broadway	7989	Kate	Manhattan	Hell's Kitchen	40.7608	-73.9887	Private room	85	2	188	2019-06-23	1.5	1	39
12	5803	Lovely Room 1, Garden, Best Area, Legal rental	9744	Laurie	Brooklyn	South Slope	40.6683	-73.9878	Private room	89	4	167	2019-06-24	1.34	3	314
13	6021	Wonderful Guest Bedroom in Manhattan for SINGLES	11528	Claudio	Manhattan	Upper West Side	40.7983	-73.9611	Private room	85	2	113	2019-07-05	0.91	1	333
14	6090	West Village Nest - Superhost	11975	Alina	Manhattan	West Village	40.7353	-74.0053	Entire home/apt	120	90	27	2018-10-31	0.22	1	0
15	6848	Only 2 stops to Manhattan studio	15991	Allen & Irina	Brooklyn	Williamsburg	40.7084	-73.9535	Entire home/apt	140	2	148	2019-06-29	1.2	1	46
16	7097	Perfect for Your Parents + Garden	17571	Jane	Brooklyn	Fort Greene	40.6917	-73.9719	Entire home/apt	215	2	198	2019-06-28	1.72	1	321
17	7322	Chelsea Perfect	18946	Doti	Manhattan	Chelsea	40.7419	-73.995	Private room	140	1	260	2019-07-01	2.12	1	12
18	7726	Hip Historic Brownstone Apartment with Backyard	20950	Adam And Charity	Brooklyn	Crown Heights	40.6759	-73.9469	Entire home/apt	99	3	53	2019-06-22	4.44	1	21
19	7750	Huge 2 BR Upper East Cental Park	17985	Sing	Manhattan	East Harlem	40.7968	-73.9487	Entire home/apt	190	7	0	0	0	2	249
20	7801	Sweet and Spacious Brooklyn Loft	21207	Chaya	Brooklyn	Williamsburg	40.7184	-73.9572	Entire home/apt	299	3	9	2011-12-28	0.07	1	0
21	8024	CBG CtyBGd HelpsHaiti rm#1:1-4	22486	Lisel	Brooklyn	Park Slope	40.6807	-73.9771	Private room	130	2	130	2019-07-01	1.09	6	347
22	8025	CBG Helps Haiti Room#2.5	22486	Lisel	Brooklyn	Park Slope	40.6799	-73.978	Private room	80	1	39	2019-01-01	0.37	6	364
23	8110	CBG Helps Haiti Rm #2	22486	Lisel	Brooklyn	Park Slope	40.68	-73.9787	Private room	110	2	71	2019-07-02	0.61	6	304
24	8490	MAISON DES SIRENES1,bohemian apartment	25183	Nathalie	Brooklyn	Bedford-Stuyvesant	40.6837	-73.9403	Entire home/apt	120	2	88	2019-06-19	0.73	2	233
25	8505	Sunny Bedroom Across Prospect Park	25326	Gregory	Brooklyn	Windsor Terrace	40.656	-73.9752	Private room	60	1	19	2019-06-23	1.37	2	85
26	8700	Magnifique Suite au N de Manhattan - vue Cloitres	26394	Claude & Sophie	Manhattan	Inwood	40.8675	-73.9264	Private room	80	4	0	0	0	1	0
27	9357	Midtown Pied-a-terre	30193	Tommi	Manhattan	Hell's Kitchen	40.7672	-73.9853	Entire home/apt	150	10	58	2017-08-13	0.49	1	75
28	9518	SPACIOUS, LOVELY FURNISHED MANHATTAN BEDROOM	31374	Shon	Manhattan	Inwood	40.8648	-73.9211	Private room	44	3	108	2019-06-15	1.11	3	311
29	9657	Modern 1 BR / NYC / EAST VILLAGE	21904	Dana	Manhattan	East Village	40.7292	-73.9854	Entire home/apt	180	14	29	2019-04-19	0.24	1	67
30	9668	front room/double bed	32294	Ssameer Or Trip	Manhattan	Harlem	40.8225	-73.951	Private room	50	3	242	2019-06-01	2.04	3	355

So we have 16 columns/features, 10 of them are numerical (int64 and float), 6 of them are string(object) values, and 48895 rows on this dataset.

```
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
```

So in the last figure, we can see the dataset's missing value distribution. 16 on the name, 21 on the host_name, and 10052 for both last_reviews and reviews_per_month. If we decompose and analyze this dataset we can conclude that last_reviews and reviews_per_month values are NaN because nobody is accommodated in the listing house. We should replace these NaN values with zeros. On the other hand, we have some NaN values for name and host_name. But we are going to drop these columns because the name and host_name values do not affect pricing.

	id	name	host_id	host_name \
count	4.889500e+04	48879	4.889500e+04	48874
unique	NaN	47905	NaN	11452
top	NaN	Hillside Hotel	NaN	Michael
freq	NaN	18	NaN	417
mean	1.901714e+07	NaN	6.762001e+07	NaN
std	1.098311e+07	NaN	7.861097e+07	NaN
min	2.539000e+03	NaN	2.438000e+03	NaN
25%	9.471945e+06	NaN	7.822033e+06	NaN
50%	1.967728e+07	NaN	3.079382e+07	NaN
75%	2.915218e+07	NaN	1.074344e+08	NaN
max	3.648724e+07	NaN	2.743213e+08	NaN

	room_type	price	minimum_nights	number_of_reviews	\
count	48895	48895.000000	48895.000000	48895.000000	
unique	3	NaN	NaN	NaN	
top	Entire home/apt	NaN	NaN	NaN	
freq	25409	NaN	NaN	NaN	
mean	NaN	152.720687	7.029962	23.274466	
std	NaN	240.154170	20.510550	44.550582	
min	NaN	0.000000	1.000000	0.000000	
25%	NaN	69.000000	1.000000	1.000000	
50%	NaN	106.000000	3.000000	5.000000	
75%	NaN	175.000000	5.000000	24.000000	
max	NaN	10000.000000	1250.000000	629.000000	

	neighbourhood_group	neighbourhood	latitude	longitude	\
count	48895	48895	48895.000000	48895.000000	
unique	5	221	NaN	NaN	
top	Manhattan	Williamsburg	NaN	NaN	
freq	21661	3920	NaN	NaN	
mean	NaN	NaN	40.728949	-73.952170	
std	NaN	NaN	0.054530	0.046157	
min	NaN	NaN	40.499790	-74.244420	
25%	NaN	NaN	40.690100	-73.983070	
50%	NaN	NaN	40.723070	-73.955680	
75%	NaN	NaN	40.763115	-73.936275	
max	NaN	NaN	40.913060	-73.712990	

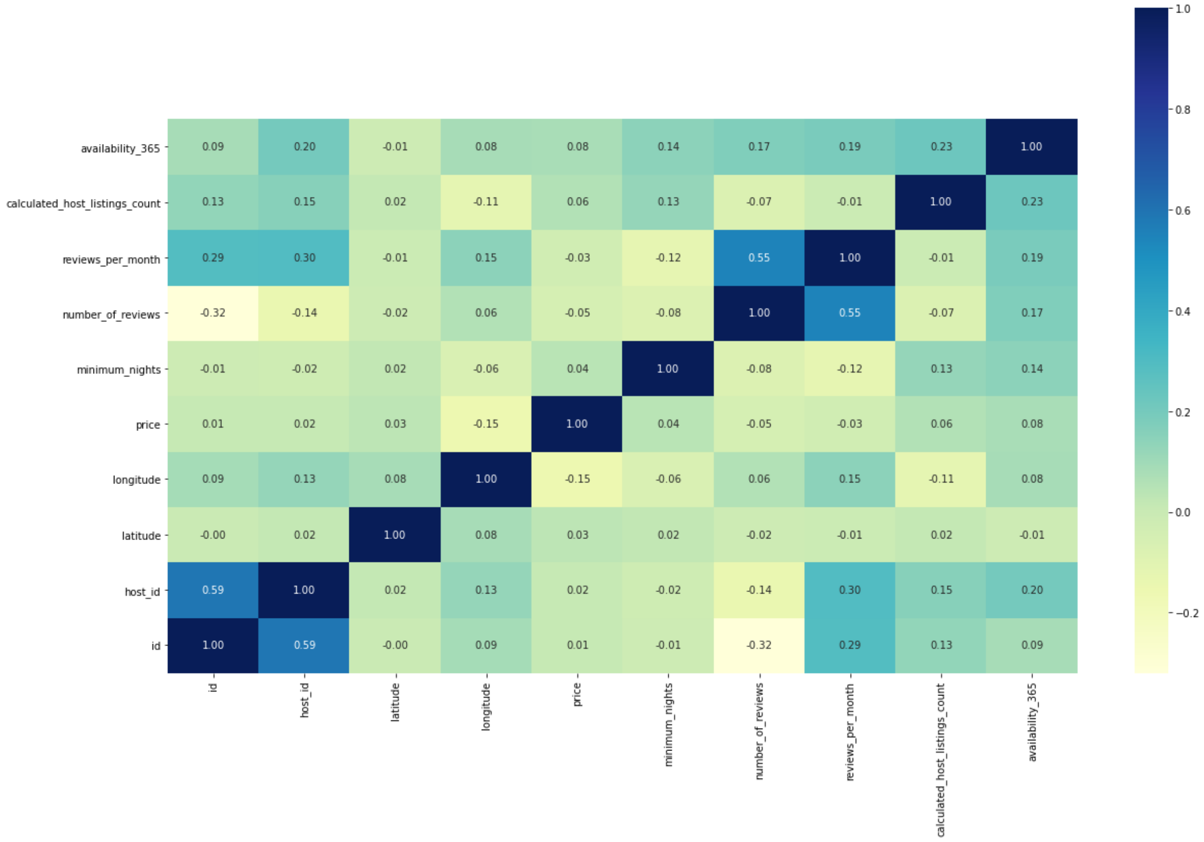
	last_review	reviews_per_month	calculated_host_listings_count	\
count	38843	38843.000000	48895.000000	
unique	1764	NaN	NaN	
top	2019-06-23	NaN	NaN	
freq	1413	NaN	NaN	
mean	NaN	1.373221	7.143982	
std	NaN	1.680442	32.952519	
min	NaN	0.010000	1.000000	
25%	NaN	0.190000	1.000000	
50%	NaN	0.720000	1.000000	
75%	NaN	2.020000	2.000000	
max	NaN	58.500000	327.000000	

On the following tables, dataset means, unique value, count, etc. described.

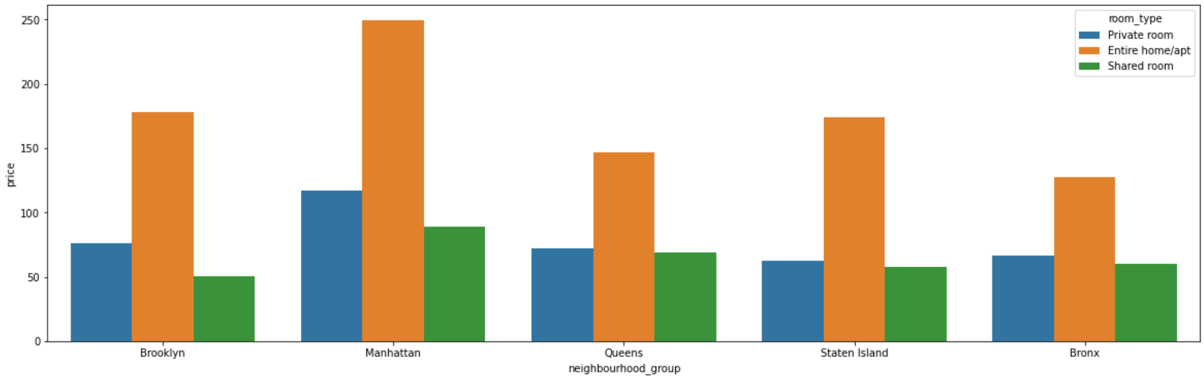
	availability_365
count	48895.000000
unique	NaN
top	NaN
freq	NaN
mean	112.781327
std	131.622289
min	0.000000
25%	0.000000
50%	45.000000
75%	227.000000
max	365.000000

2. Data Visualization

2.1 Heatmap:

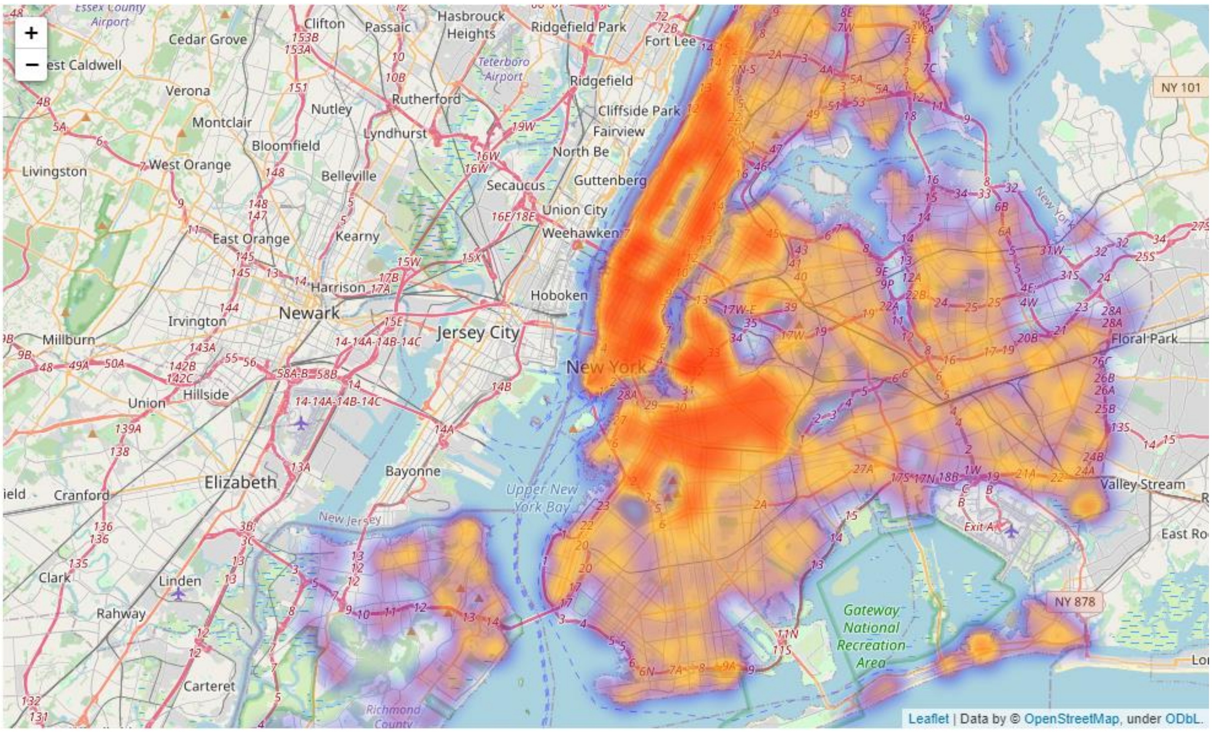


2.2 Box Chart Neighbourhood/Price with Room_type Hue:



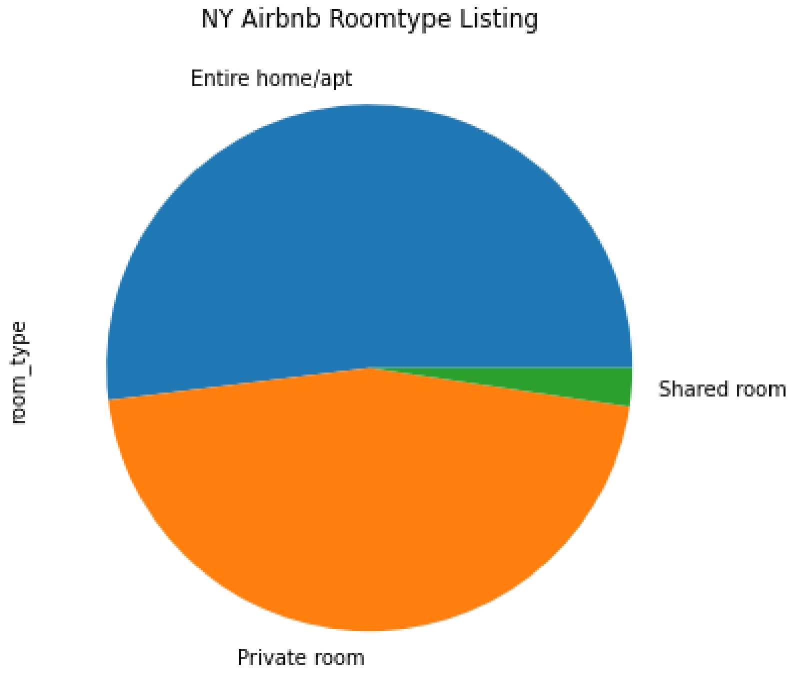
This box chart explains general information about the Airbnb dataset. Looking at this chart can tell us, room_type and neighbourhood_group are important variables for our model.

2.3 Heatmap of the New York

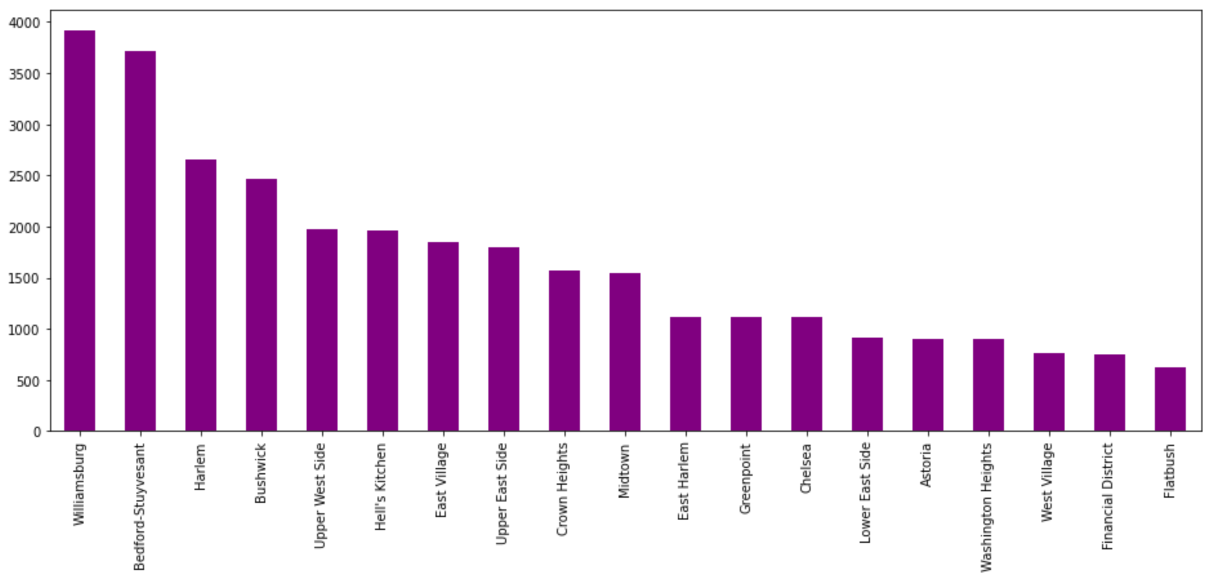


2.4 Room_type Pie Chart

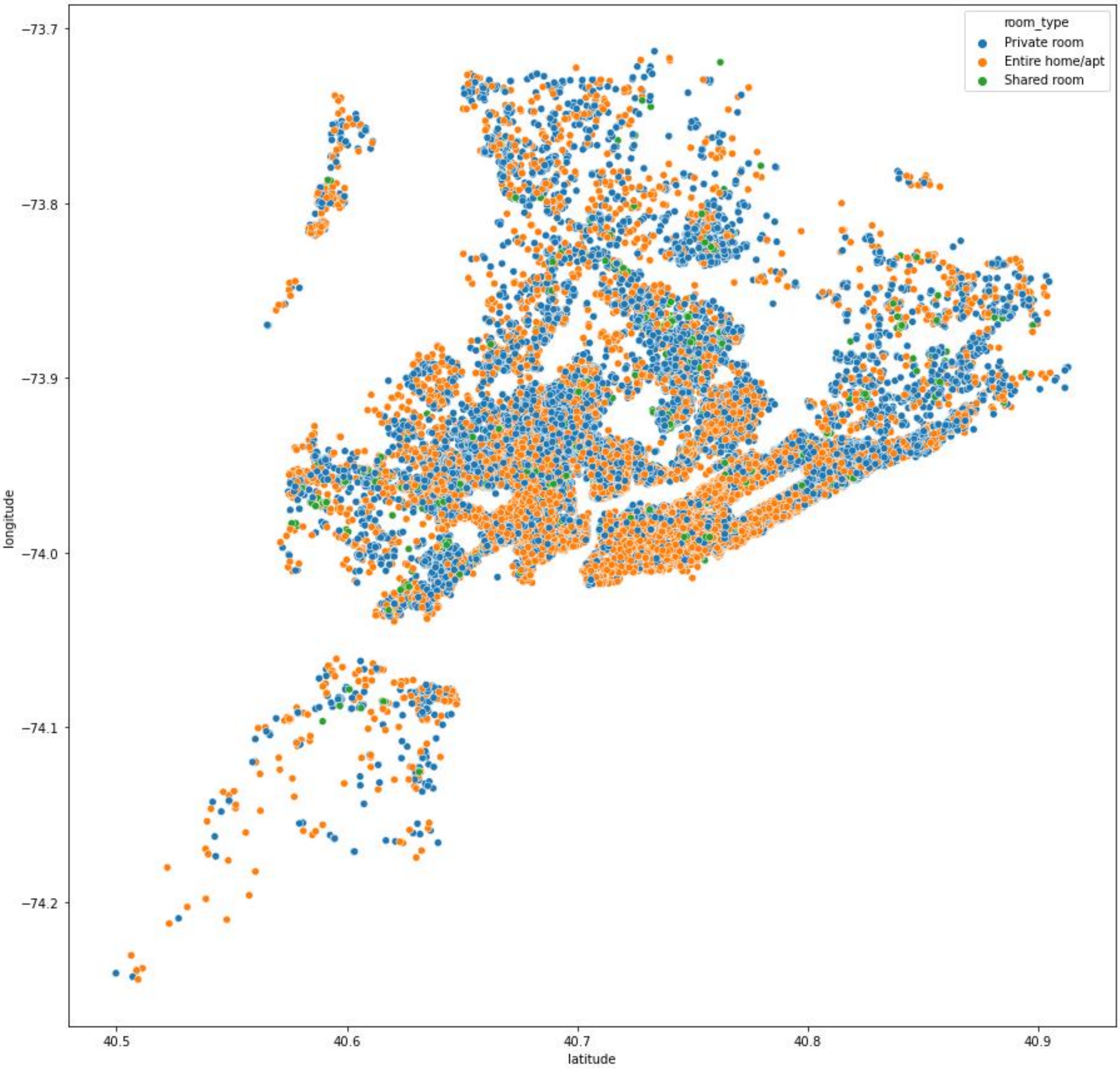
The pie chart looks so simple but in the aspect of data distribution, it provides clear distribution information for one of the most important features.



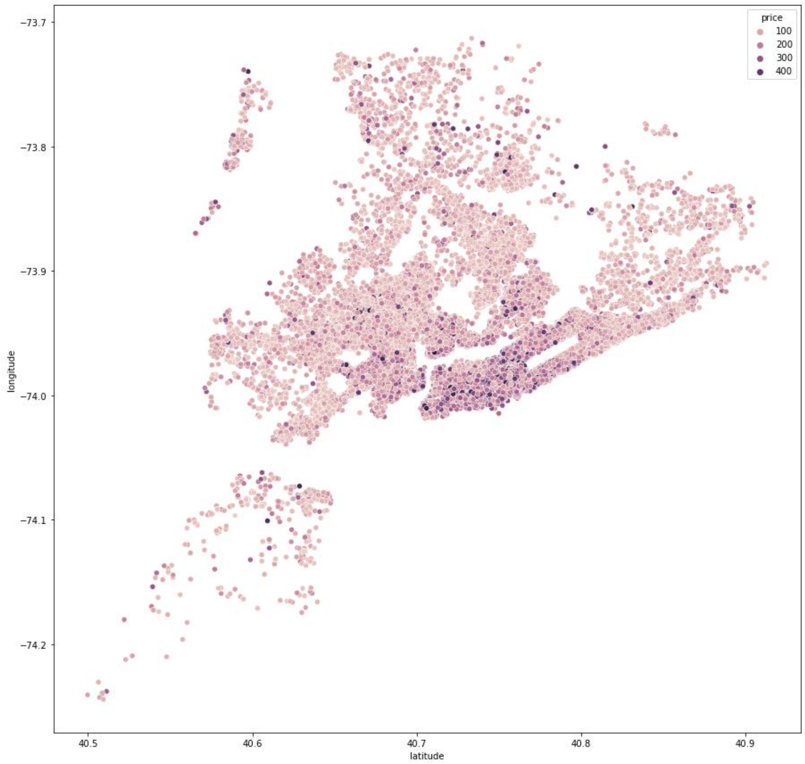
2.5 Popular Neigbourhood Groups Bar Chart



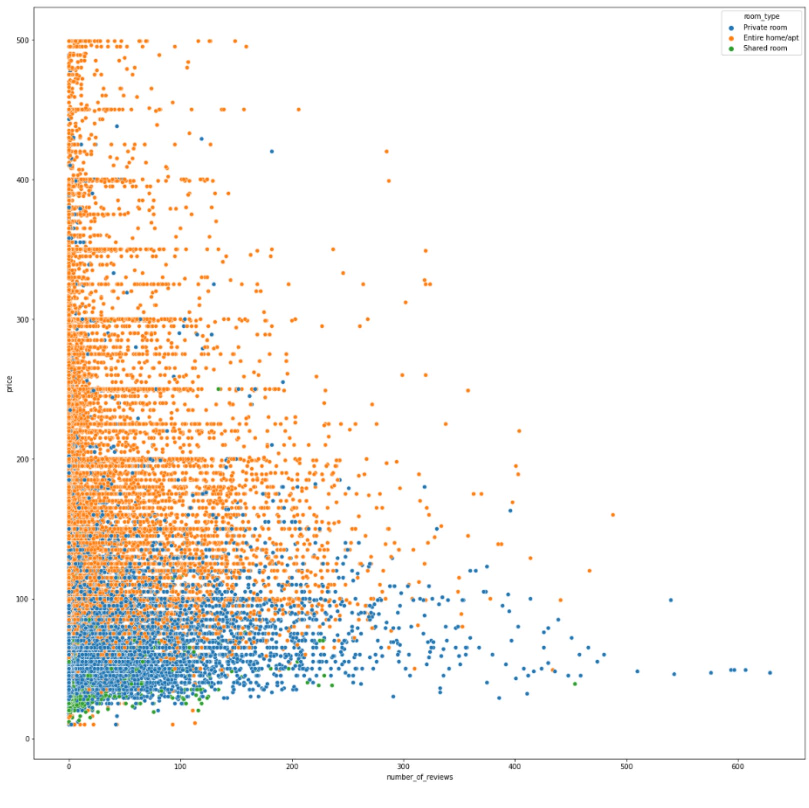
2.6 Latitude/Longitude Scatter Hue Room Type



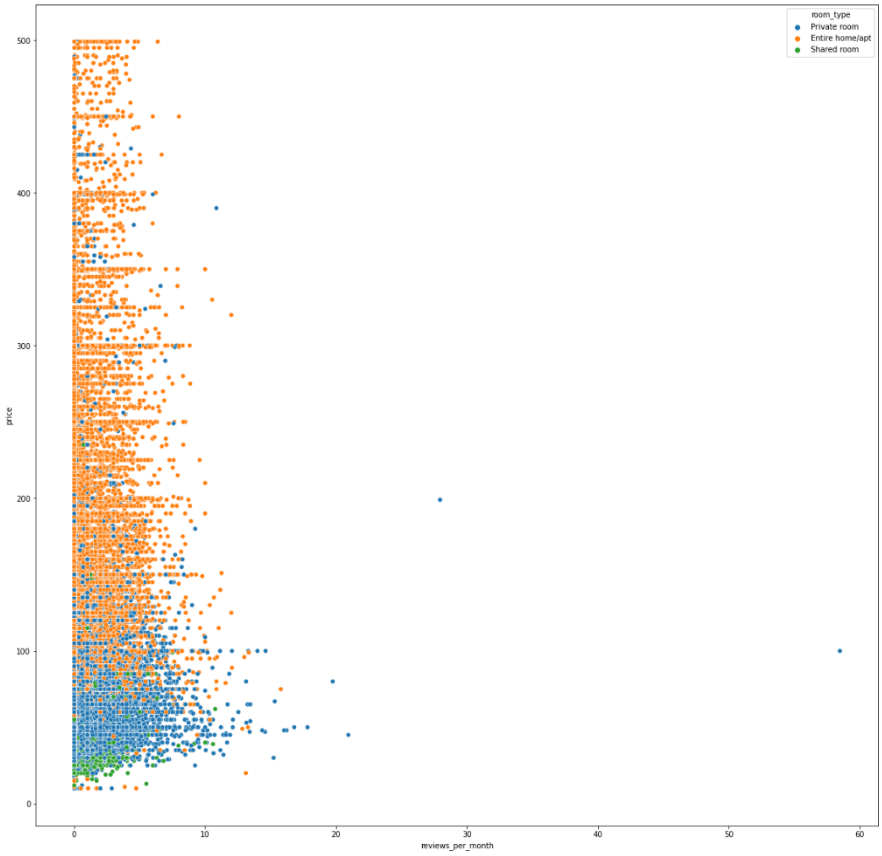
2.7 Latitude/Longitude Scatter Hue Price



2.8 Price/Number of Reviews Scatter Hue: Room_type

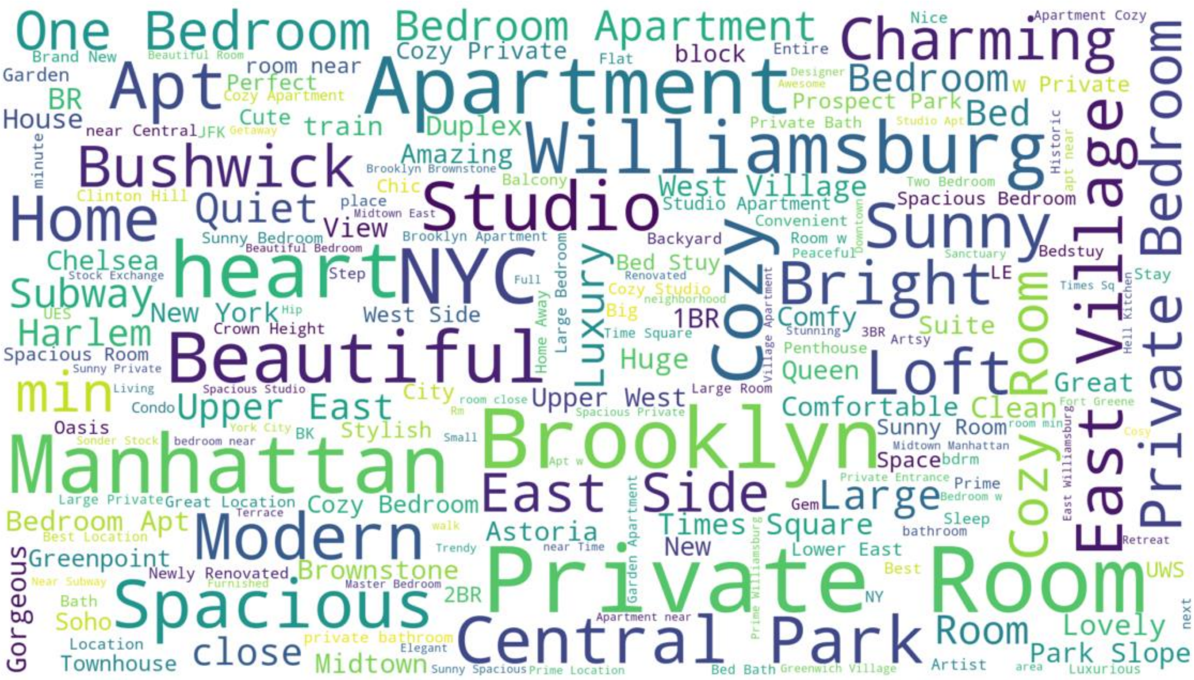


2.9 Price/Reviews Per Month Scatter Hue: RoomType

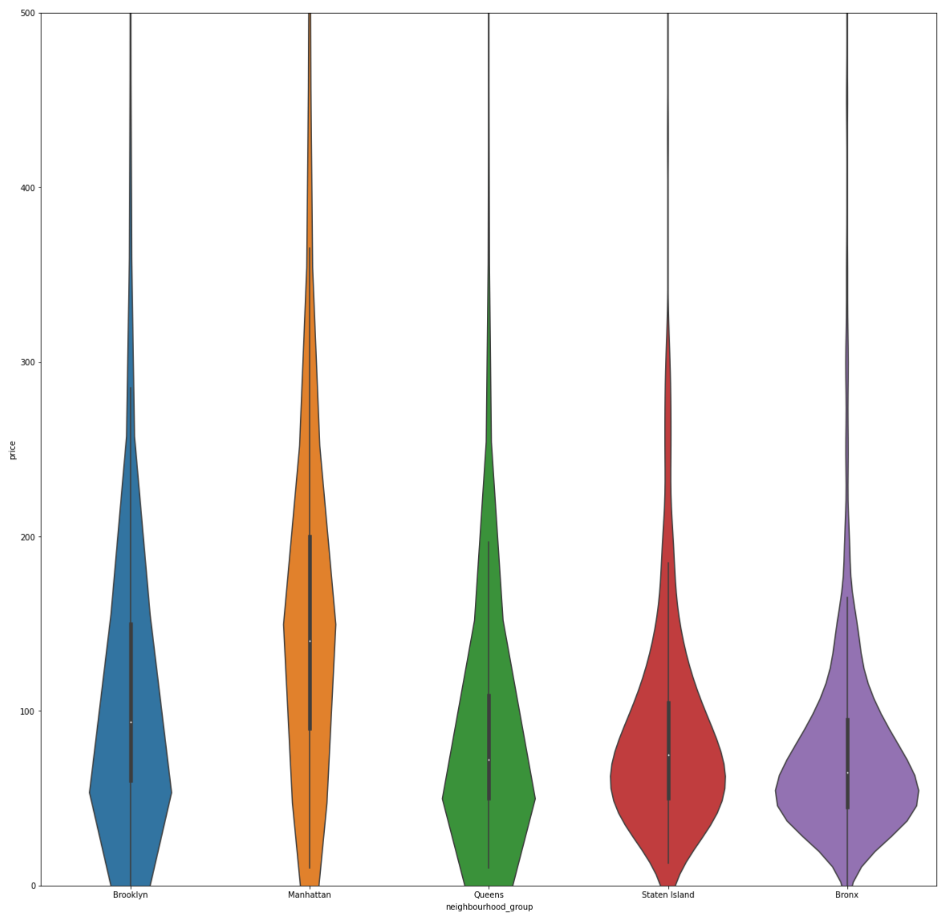


In 2.9 and 2.8 we have "reviews per month/price" and "the number of reviews/price" scatter plots. We can easily say, there is no correlation between these two parameters and the price parameter on this dataset.

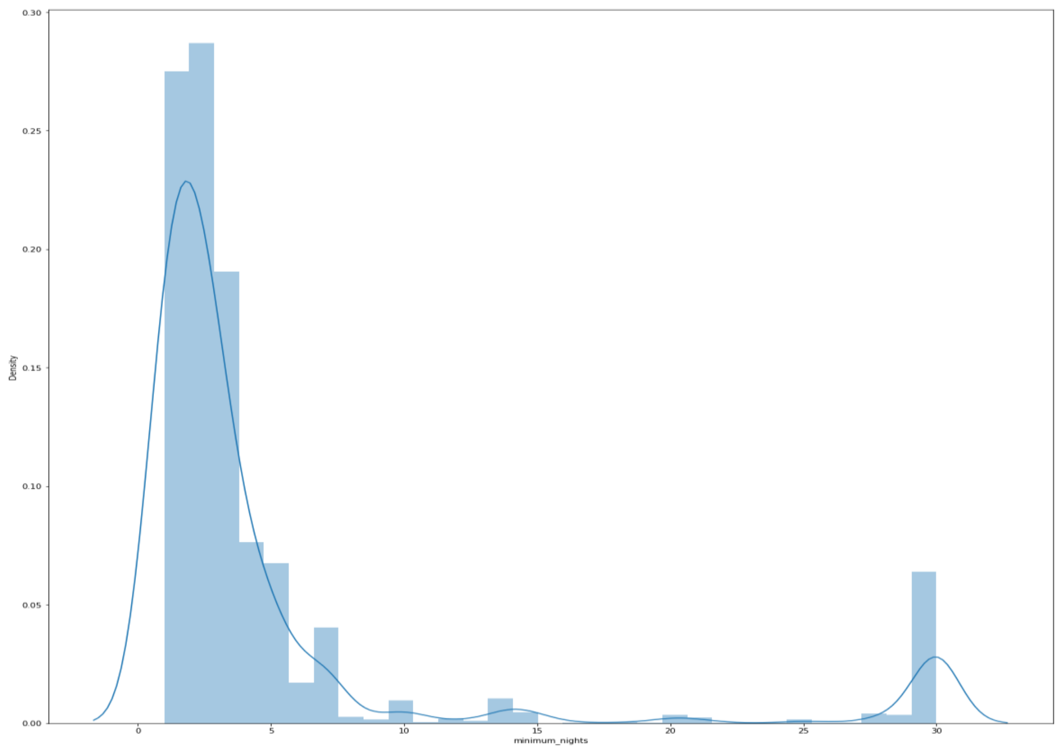
2.10 Dataset Most Used Word Graph



2.11 Neighbourhood Group/Price Violin (Price < 500)



2.12 Distribution Plot For Minimum Nights



3. Model Evaluation

In this part, we are going to build our model. At the start, we dropped our unnecessary columns.

```
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
```

At the start, we dropped our unnecessary columns. As we mentioned before, we have NaN values on our dataset. For the last_review and reviews_per_month, we changed the NaN values with 0 because according to our dataset NaN values on this constraint mean "never been reviewed before". For the name and host_name, we are just going to drop those columns. We used a backward elimination technique for the dataset. First, we visualized our data without NaN values, after that we analyzed the correlations in the graphs. For example, in 2.3 Heatmap you can see that the Northern East of New York is very popular but if we look at our 2.7 Scatter plot, we can see that the Southern West of New York is more expensive than Northern East. On the other hand, if we look at 2.9 and 2.8 we can easily say that reviews per month and the number of reviews do not influence the price. After the feature selection part we have this dataset;

Index	›ourhood_	oom_type	imum_nig	_host_listi	ailability_3	latitude	longitude	price
0	0	0	1	6	365	40.6475	-73.9724	149
1	1	1	1	2	355	40.7536	-73.9838	225
3	0	1	1	1	194	40.6851	-73.9598	89
4	1	1	10	1	0	40.7985	-73.944	80
5	1	1	3	1	129	40.7477	-73.975	200
7	1	0	2	1	220	40.7649	-73.9849	79
8	1	0	2	1	0	40.8018	-73.9672	79
9	1	1	1	4	188	40.7134	-73.9904	150
10	1	1	5	1	6	40.8032	-73.9655	135
11	1	0	2	1	39	40.7608	-73.9887	85
12	0	0	4	3	314	40.6683	-73.9878	89
13	1	0	2	1	333	40.7983	-73.9611	85
15	0	1	2	1	46	40.7084	-73.9535	140
16	0	1	2	1	321	40.6917	-73.9719	215
17	1	0	1	1	12	40.7419	-73.995	140
18	0	1	3	1	21	40.6759	-73.9469	99

As you can see we have numeric values on room_type and neigbourhood_group we factorized it because in linear regression we are not able to use string values. So we convert these values to the float.

```
df_reg['room_type'] = df_reg['room_type'].factorize()[0]
df_reg['neighbourhood_group'] = df_reg['neighbourhood_group'].factorize()[0]
```

So we run tried to find our best constraint ranges, first, we tried linear regression without any data cleaning. We saw that some of the listings have unusable values inside our constraints. So we decided to not use Airbnb listings that have more than $500 pricing, minimum_nights more than 40, and availability_365 more than 366.

```
df_reg = df_reg[(df_reg["minimum_nights"] < 40) & (df_reg["price"] < 500) &(df_reg["availability_365"] < 366)]
```

Again we run several linear regression processes and we conclude that the test size should be "0.26" and the random state should be "42".

For the scaling, we used a standard scaler from sklearn.preprocessing library.

```
X= df_reg.loc[:, df_reg.columns != 'price']
Y = df_reg["price"]
X_train, X_test, y_train, y_test =  train_test_split(X,Y,test_size =0.26, random_state= 42)
```

A technique used to normalize the range of independent variables or data characteristics is feature scaling. It is also known as data normalization in data processing and is normally done during the preprocessing stage of the data.

`StandardScaler` changes each **feature** column $f_{:,i}$ to

$$f'_{:,i} = \frac{f_{:,i} - mean(f_{:,i})}{std(f_{:,i})}.$$

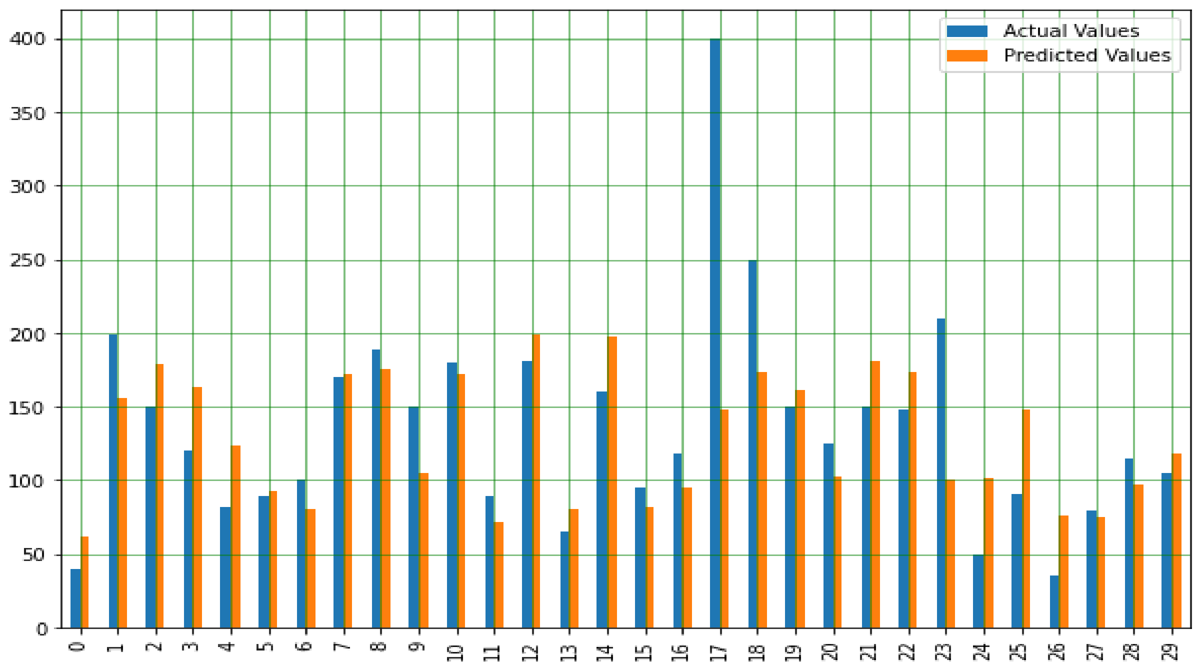
For the scaling, we used a standard scaler from sklearn.preprocessing library.

By fitting a linear equation to observed data, linear regression attempts to model the relationship between two variables. An explanatory variable is considered to be one variable, and a dependent variable is considered to be the other. For example, using a linear regression model, a modeler will want to relate the weights of individuals to their heights.

```python
df_reg = df[["neighbourhood_group","room_type","minimum_nights","calculated_host_listings_count",
            "availability_365","latitude","longitude","price"]]
df_reg['room_type'] = df_reg['room_type'].factorize()[0]
df_reg['neighbourhood_group'] = df_reg['neighbourhood_group'].factorize()[0]
df_reg = df_reg[(df_reg["minimum_nights"] < 40) & (df_reg["price"] < 500) &(df_reg["availability_365"] < 366)]
X= df_reg.loc[:, df_reg.columns != 'price']
Y = df_reg["price"]
X_train, X_test, y_train, y_test =  train_test_split(X,Y,test_size =0.26, random_state= 42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
reg=LinearRegression()
reg.fit(X_train,y_train)
y_pred=reg.predict(X_test)
```

So we used linear regression to test and optimize our model. Here is the results;

	Actual Values	Predicted Values
0	40	62.079760
1	199	156.167713
2	150	179.097398
3	120	163.180621
4	82	124.005661
5	90	92.243860
6	100	80.697174
7	170	172.436697
8	189	175.004367
9	150	104.889349
10	180	172.121118
11	90	71.780055
12	181	199.123684
13	65	80.314651
14	160	197.881337
15	95	81.999164
16	118	95.416942
17	400	147.763984
18	250	173.595073
19	150	161.232344
20	125	102.924442
21	150	180.941915
22	148	173.839407
23	210	100.877836
24	50	101.596776
25	91	147.894535
26	35	76.041142
27	80	75.303773



For the optimization and evaluation of our regression model's performance, we used the R Squared and Root Mean Square Error. R-squared (R2) is a statistical measure that represents the proportion of the variance in a regression model for a dependent variable explained by an independent variable or variables. Root Mean Square Error (RMSE) is the residuals' standard deviation (prediction errors). Residuals are a measure of how far data points are from the regression line; RMSE is a measure of how these residuals can be spread out. This tells you, in other words, how clustered the data is along the best fit line. To check experimental outcomes, Root Mean Square Error is widely used in regression analysis.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

```
print("R2 score: ",r2_score(y_test,y_pred))
print("RMSE: ",np.sqrt(mean_squared_error(y_test,y_pred)))
```

Here are our model's results;

```
R2 score:  0.39429240961294165
RMSE:  63.99475080993863
```

Our R2 score is almost "0.4" which means our model is doing pretty fine on the other hand we have a "63" RMSE. We tried different variables to improve our RMSE score and in the end, we conclude that our model final modem is pretty suitable for the work with.

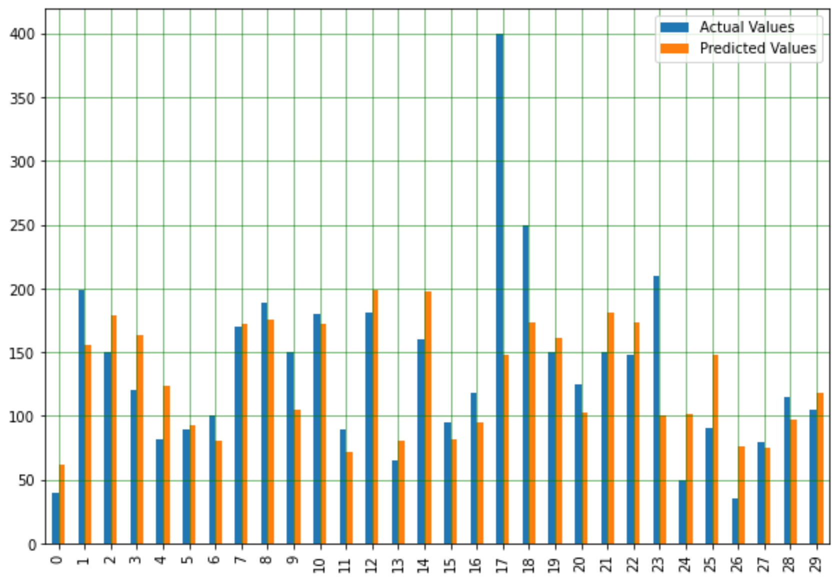
In the end, we applied k-NN with all features classification method to our model.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.26, random_state = 42)
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.fit_transform(x_test)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 21)
knn.fit(x_train, y_train)

y_pred = knn.predict(x_test)
```

Results;

	Actual Values	Predicted Values
0	60	55
1	168	95
2	50	45
3	73	50
4	175	45
5	339	269
6	65	100
7	80	85
8	159	130
9	125	180
10	50	34
11	60	52

11	60	52
12	65	140
13	87	120
14	135	200
15	250	200
16	200	45
17	45	130
18	160	189
19	68	150
20	125	100
21	139	100
22	115	140
23	130	54



```
R2 score:  0.0945834544275449
RMSE:  79.72733656781034
```