Methodology Document

This document describes the method used to come to the inferences in this document. It is divided in two parts –

a. Data Modelling
b. Data Visualization

While the non-technical audience can skip the part related to Data Modelling and assume properties are clustered into 4 groups based on the data provided; the data visualization part will provide insight of how the recommendations were made and can be consumed by both – technical and non-technical audience.

a. Data Modelling

1. Processed the Data provided by Upgrad in Python to create clusters of properties with Airbnb.

2. To create the clusters, EDA was performed on the datasets to do the cleanup and then massaging of data to make it ready for modelling.

3. The data received initially had the following 16 columns:-

```
# Basic inspection of the dataframe
listings.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
```

4. As seen above the last_review column contains date but is identified as object column. Converting it to date column:-

```python
# Convert the last_review column to datetime
listings['last_review'] = pd.to_datetime(listings['last_review'])
```

```python
# Basic inspection of the dataframe
listings.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  datetime64[ns]
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
```

5. Based on the data dictionary, few columns were identified that may not have helped in modelling the data to create cluster and hence were dropped as below:-

```python
# Dropping all the columns that may not be helpful to group the data into different buckets
listings.drop(['name', 'host_id', 'host_name', 'latitude', 'longitude'],axis=1, inplace=True)
listings.head(2)
```

	id	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month
0	2539	Brooklyn	Kensington	Private room	149	1	9	2018-10-19	0.21
1	2595	Manhattan	Midtown	Entire home/apt	225	1	45	2019-05-21	0.38

6. Here is the updated dataframe:-

```
listings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   neighbourhood_group             48895 non-null  object
 2   neighbourhood                   48895 non-null  object
 3   room_type                       48895 non-null  object
 4   price                           48895 non-null  int64
 5   minimum_nights                  48895 non-null  int64
 6   number_of_reviews               48895 non-null  int64
 7   last_review                     38843 non-null  datetime64[ns]
 8   reviews_per_month               38843 non-null  float64
 9   calculated_host_listings_count  48895 non-null  int64
 10  availability_365                48895 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(6), object(3)
memory usage: 4.1+ MB
```

7. A quick peek into the missing data:-

```
# Checking the data for nulls in columns last_review and reviews_per_month
listings[(listings.last_review.isna()) | (listings.reviews_per_month.isna())].head()
```

	id	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month
2	3647	Manhattan	Harlem	Private room	150	3	0	NaT	NaN
19	7750	Manhattan	East Harlem	Entire home/apt	190	7	0	NaT	NaN
26	8700	Manhattan	Inwood	Private room	80	4	0	NaT	NaN
36	11452	Brooklyn	Bedford-Stuyvesant	Private room	35	60	0	NaT	NaN
38	11943	Brooklyn	Flatbush	Private room	150	1	0	NaT	NaN

8. Missing data belongs to same property:-

```
# Also all the missing (null) data for both the columns is for same record
listings[(listings.last_review.isna()) | (listings.reviews_per_month.isna())].count()
```

```
id                                10052
neighbourhood_group               10052
neighbourhood                     10052
room_type                         10052
price                             10052
minimum_nights                    10052
number_of_reviews                 10052
last_review                           0
reviews_per_month                     0
calculated_host_listings_count    10052
availability_365                  10052
dtype: int64
```

9. Imputing 0 to reviews_per_month column to make the model feel the property were never reviewed.

```
# Imputing null values of reviews_per_month to 0
listings.reviews_per_month.fillna(0, inplace=True)
```

```
listings.isna().sum()
```

```
id                                0
neighbourhood_group               0
neighbourhood                     0
room_type                         0
price                             0
minimum_nights                    0
number_of_reviews                 0
last_review                   10052
reviews_per_month                 0
calculated_host_listings_count    0
availability_365                  0
dtype: int64
```

10. Last_review date is important as it can be used to calculate the number of days passed since the property was last reviewed. However, this field is also having nulls. To calculate this days difference field, inserting a temporary column end_date with a value of 31st Dec 2019. The logic behind this date being 2019 is the most recent year any property was reviewed and hence calculating difference with this date will give all the days as positive.

```
# Inserting date column for end date. It will be used to calculate day difference last review. This date is set to
# 2019-12-31 as the most recent date in this column is from 2019.
listings['end_date'] = '2019-12-31'
listings['end_date'] = pd.to_datetime(listings['end_date'])
listings.head()
```

```
listings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   neighbourhood_group             48895 non-null  object
 2   neighbourhood                   48895 non-null  object
 3   room_type                       48895 non-null  object
 4   price                           48895 non-null  int64
 5   minimum_nights                  48895 non-null  int64
 6   number_of_reviews               48895 non-null  int64
 7   last_review                     38843 non-null  datetime64[ns]
 8   reviews_per_month               48895 non-null  float64
 9   calculated_host_listings_count  48895 non-null  int64
 10  availability_365                48895 non-null  int64
 11  end_date                        48895 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(1), int64(6), object(3)
memory usage: 4.5+ MB
```

11. However, since there is missing data in last_review date field, the days difference field have null for such properties. It was noticed that 3200 is the highest number of days any property was last reviewed. Hence, imputing 3500 in the missing date to make the system believe these

properties were either reviewed long back or never reviewed. Here is a code snippet for this entire calculation.

```python
# Computing days since last review as difference of last_review date and 2019-12-31
listings['Days_since_last_review'] = listings.end_date - listings.last_review
listings['Days_since_last_review'] = listings['Days_since_last_review'] / np.timedelta64(1, 'D')
```

```python
listings['Days_since_last_review'].sort_values(ascending=False)
```

```
317      3200.0
163      3172.0
125      3026.0
143      3025.0
123      3002.0
          ...
48890       NaN
48891       NaN
48892       NaN
48893       NaN
48894       NaN
Name: Days_since_last_review, Length: 48895, dtype: float64
```

```python
# imputing 3500 to Days_since_last_review. This value is used as it is bit higher than the actual Days_since_last_review
# which was 3200. Hence, the missing records will have this field has highest (which can be considerd as not reviewed)
listings['Days_since_last_review'].fillna(3500, inplace=True)
listings['Days_since_last_review']
```

```
0         438.0
1         224.0
2        3500.0
3         238.0
4         407.0
          ...
48890    3500.0
```

12. Dropping the unwanted columns:-

```python
# Dropping unwanted columns:-
listings.drop(['last_review', 'end_date'],axis=1, inplace=True)
```

```python
# Check for dataframe shape and size
listings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   neighbourhood_group             48895 non-null  object 
 2   neighbourhood                   48895 non-null  object 
 3   room_type                       48895 non-null  object 
 4   price                           48895 non-null  int64  
 5   minimum_nights                  48895 non-null  int64  
 6   number_of_reviews               48895 non-null  int64  
 7   reviews_per_month               48895 non-null  float64
 8   calculated_host_listings_count  48895 non-null  int64  
 9   availability_365                48895 non-null  int64  
 10  Days_since_last_review          48895 non-null  float64
dtypes: float64(2), int64(6), object(3)
memory usage: 4.1+ MB
```

13. Preparing the data for Modelling:-
    i. Check to see if there is huge data imbalance for any variable. If so, clubbing multiple values as a common value.

ii. Converting the alphabetic values in various columns to categorical numeric values to be acceptable for modelling

iii. Creating Dummy variables

```
# Since, there are very vey properties in Queens and Bronx, it is better to club them as one entity for data modelling.
listings['neighbourhood_group'].value_counts(normalize=True)
```

```
Manhattan        0.443011
Brooklyn         0.411167
Queens           0.115881
Bronx            0.022313
Staten Island    0.007629
Name: neighbourhood_group, dtype: float64
```

```
listings.loc[((listings.neighbourhood_group == 'Bronx')|(listings.neighbourhood_group == 'Staten Island')|
              (listings.neighbourhood_group == 'Queens')),'neighbourhood_group'] = 'Others'
```

```
# Converting the data to numeric categorical data for ease of data modelling
listings['neighbourhood_group'] = listings['neighbourhood_group'].map({'Others': 0, 'Manhattan': 1, 'Brooklyn': 2})
```

```
listings['room_type'] = listings['room_type'].map({'Shared room': 0, 'Entire home/apt': 1, 'Private room': 2})
```

```
listings.head()
```

	id	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count
0	2539	2	Kensington	2	149	1	9	0.21	6
1	2595	1	Midtown	1	225	1	45	0.38	2
2	3647	1	Harlem	2	150	3	0	0.00	1
3	3831	2	Clinton Hill	1	89	1	270	4.64	1
4	5022	1	East Harlem	1	80	10	9	0.10	1

```
# dropping the unrequired columns
listings.drop(['neighbourhood'],axis=1, inplace=True)
```

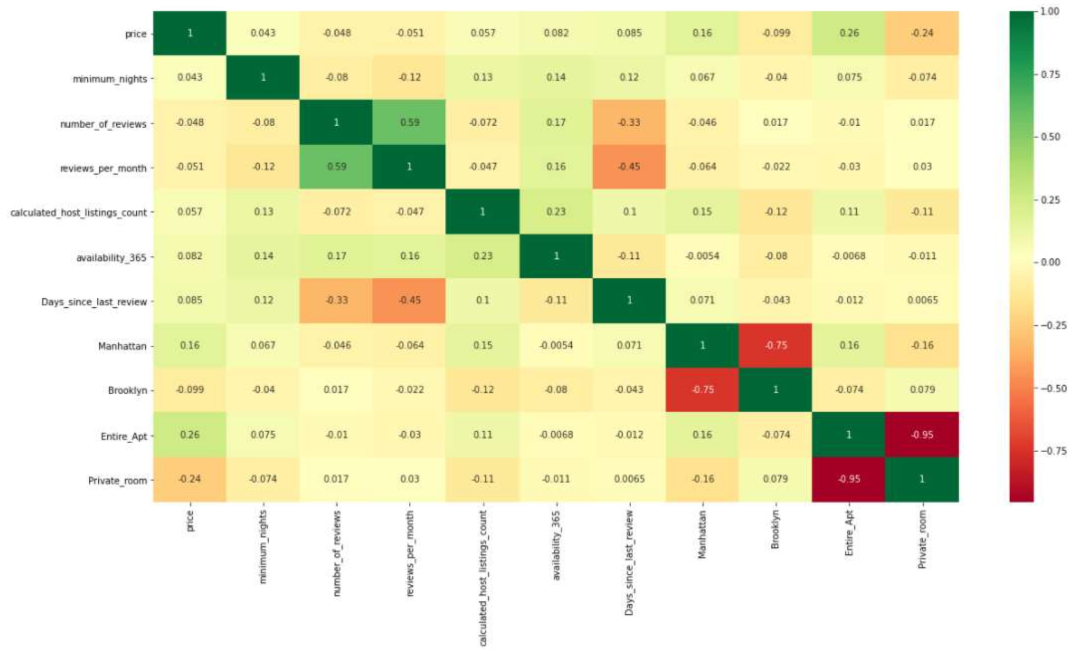
```
# Creating dummy variables for the remaining categorical variables
dummy_1 = pd.get_dummies(listings['neighbourhood_group'], drop_first=True)
dummy_1.rename(columns={1:'Manhattan', 2:'Brooklyn'}, inplace=True)
```

```
dummy_2 = pd.get_dummies(listings['room_type'], drop_first=True)
dummy_2.rename(columns={1:'Entire_Apt', 2:'Private_room'}, inplace=True)
```

```
listings = pd.concat([listings, dummy_1, dummy_2], axis=1)
listings.drop(['id', 'neighbourhood_group', 'room_type'], axis=1, inplace=True)
listings.head()
```

14. Now that the variables are ready, check for corelation among variables. Apart from some obvious variables (like variables for neighbourhood or room type); there isn't any strong corelation and hence continuing with all the variables:-

```python
# Now that the dataset is ready, checking the corelation between all the fields
plt.figure(figsize = (20,10))
sns.heatmap(listings.corr(), annot=True, cmap='RdYlGn')
plt.show()
```



15. Using standardization to scale all the variables. Standardisation in techinque to bring all the data into a standard normal distribution with mean 0 and standard deviation 1.

```python
vars_to_scale = ['price', 'minimum_nights', 'number_of_reviews', 'reviews_per_month', 'calculated_host_listings_count',
```

```python
scaler = StandardScaler()
```

```python
listings[vars_to_scale] = scaler.fit_transform(listings[vars_to_scale])
listings.head()
```

	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365	Days_since_last_review	Manhattan
0	-0.015493	-0.293996	-0.320414	-0.551511	-0.034716	1.916250	-0.506583	0
1	0.300974	-0.293996	0.487665	-0.445079	-0.156104	1.840275	-0.673429	1
2	-0.011329	-0.196484	-0.522433	-0.682986	-0.186451	1.916250	1.880712	1
3	-0.265335	-0.293996	5.538156	2.221978	-0.186451	0.617065	-0.662514	0
4	-0.302811	0.144807	-0.320414	-0.620379	-0.186451	-0.856865	-0.530752	1

```python
listings.describe()
```

	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365	Days_since_last_review	
count	4.889500e+04	4.889500e+04	4.889500e+04	4.889500e+04	4.889500e+04	4.889500e+04	4.889500e+04	488
mean	-5.581769e-16	-7.665145e-16	-2.315687e-14	4.282961e-15	3.545441e-14	1.387977e-14	-7.759922e-15	
std	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	
min	-6.359342e-01	-2.939962e-01	-5.224332e-01	-6.829856e-01	-1.864514e-01	-8.568646e-01	-8.285794e-01	
25%	-3.486158e-01	-2.939962e-01	-4.999866e-01	-6.579428e-01	-1.864514e-01	-8.568646e-01	-6.936996e-01	
50%	-1.945465e-01	-1.964844e-01	-4.102001e-01	-4.513398e-01	-1.864514e-01	-5.149737e-01	-5.681756e-01	
75%	9.277183e-02	-9.897262e-02	1.628580e-02	3.062045e-01	-1.561044e-01	8.677849e-01	2.364255e-01	
max	4.100441e+01	6.060212e+01	1.359649e+01	3.594209e+01	9.706672e+00	1.916250e+00	1.880712e+00	

16. Creating the cluster using k-means algorithm with initial cluster size of 4:-

- K means clustering with initial cluster size of 4

```python
kmeans = KMeans(n_clusters = 4)
```

```python
kmeans.fit(listings)
```

```
: KMeans(n_clusters=4)
```

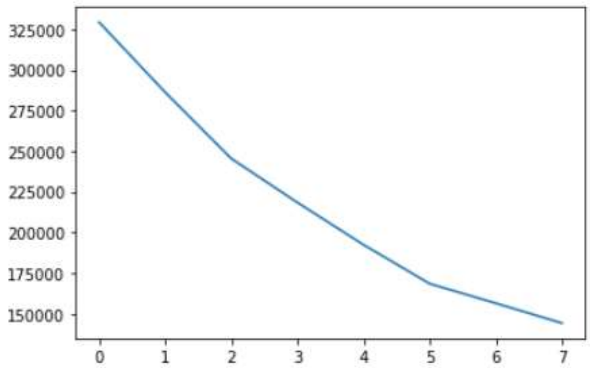
```python
kmeans.labels_
```

```
: array([3, 3, 1, ..., 1, 1, 1])
```

17. Used elbow curve and silhouette score techniques to determine the best fit cluster size is 4:-

```python
#### elbow curve/ SSD to determine best fit value for number of cluster
ssd = []
for i in range (2,10):
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(listings)
    ssd.append(kmeans.inertia_)

plt.plot(ssd)
plt.show()
```




```python
for i in range (2,6):
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(listings)
    labels = kmeans.labels_
    sil_scr = silhouette_score(listings, labels)
    print ('for cluster size {0} silhoutte score is {1}'.format(i, sil_scr))
```

```
for cluster size 2 silhoutte score is 0.23274574489071598
for cluster size 3 silhoutte score is 0.266077535146209
for cluster size 4 silhoutte score is 0.2768605579226678
for cluster size 5 silhoutte score is 0.2611014076318599
```

18. Finalizing the model and hence the clusters:-

```python
kmeans = KMeans(n_clusters = 4, random_state=100)
```

```python
kmeans.fit(listings)
```

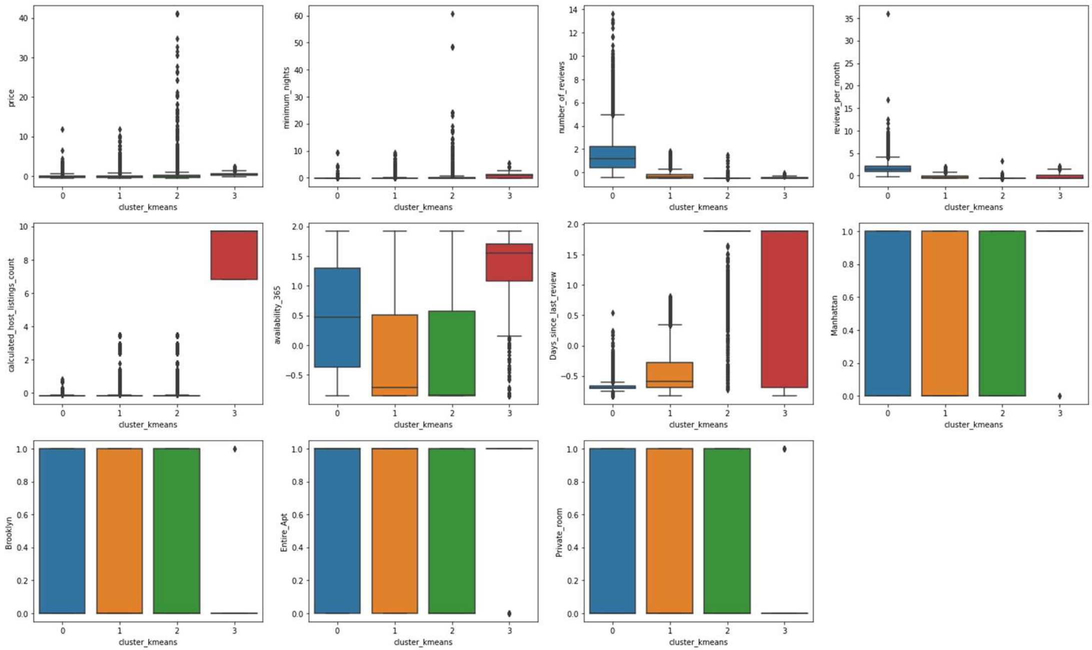
```
KMeans(n_clusters=4, random_state=100)
```

```python
kmeans.labels_
```

```
array([1, 1, 2, ..., 2, 2, 2])
```

```python
listings['cluster_kmeans'] = kmeans.labels_
listings.head()
```

19. Initial plotting of the new variable for clusters against various variables:-



20. Inserted the column into the original dataset and exporting it to excel:-

```
air_bnb['property_category'] = kmeans.labels_
air_bnb.head()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_(
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	

```
air_bnb.to_excel("air_bnb.xlsx")
```

b. Visualization

Before moving to the visualizations, a short description of various clusters created from the above process:-

1. Cluster 0 seems to be one with best reviews. These are the properties with low prices, less restriction based on minimum nights and individual owners (owners having less property listings). They are spread across localities and room type.

2. Cluster 1 seems to be one lagging behind in reviews but is the biggest portfolio for Airbnb.

3. Cluster 2 and 3 seems to be poorly performing properties. These generally are costlier properties with some times restriction of higher minimum nights and are run by groups having multiple properties.

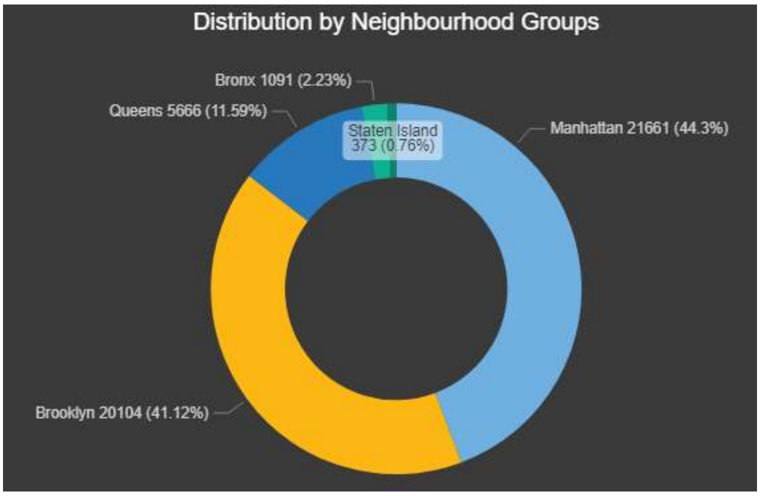
4. The above details in clusters were realised after analysing the data using Power BI tool and based on these visualizations, the recommendations were made.

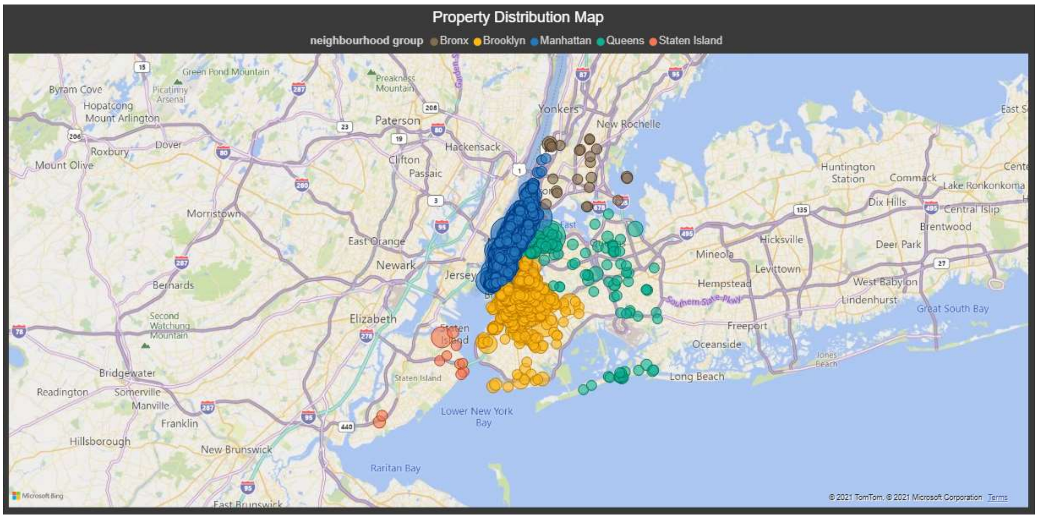
5. Once the data was imported to Power BI, Bin was created for the price column as below:-

```
Price Slab = if(Sheet1[price]<PERCENTILE.EXC(Sheet1[price],.01), "Economy",
if(Sheet1[price]<PERCENTILE.EXC(Sheet1[price],.25), "Low-Range",
if(Sheet1[price]<PERCENTILE.EXC(Sheet1[price],.75), "Mid-Range",
if(Sheet1[price]<PERCENTILE.EXC(Sheet1[price],.99), "High-Range", "Extremely
Costly"))))
```

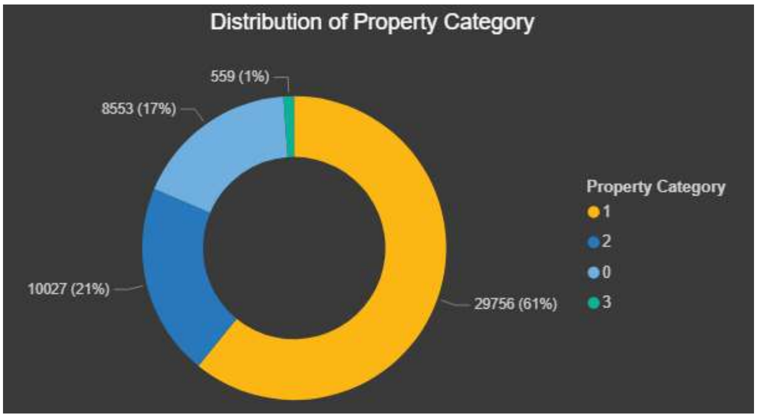
Here are some graphs to showcase the above findings:-

1. Here is the Portfolio distribution:-

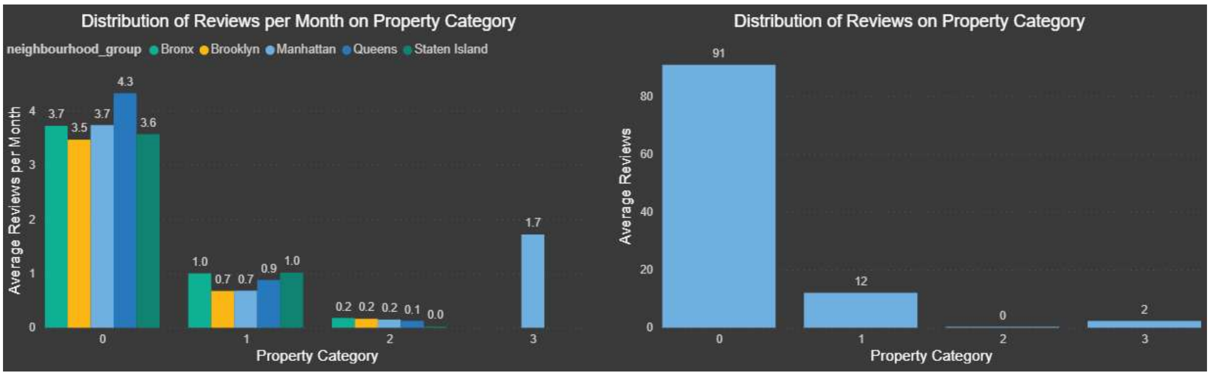




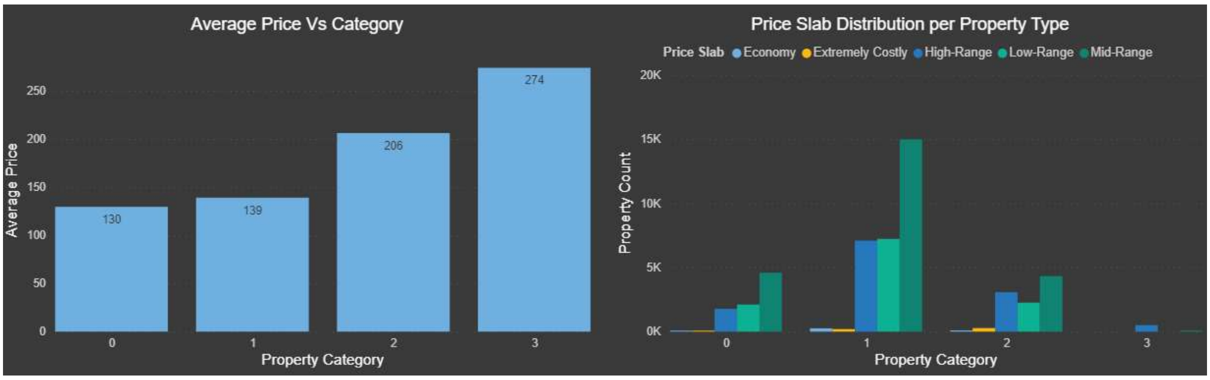
2. Distribution of various clusters:-



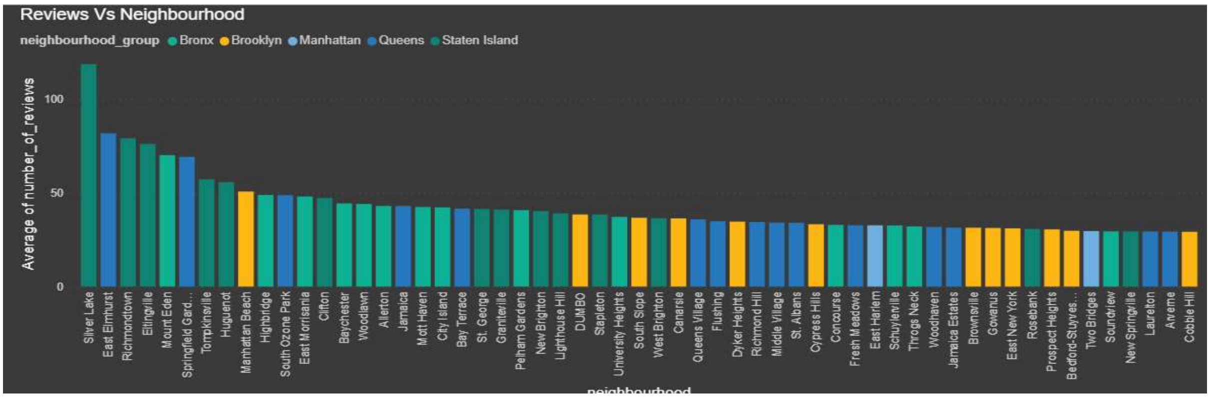
3. Distribution of reviews per cluster:-

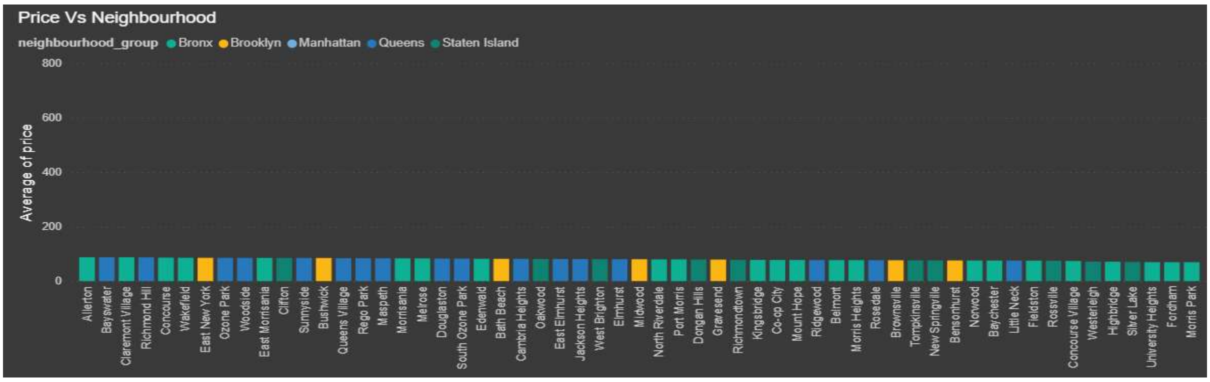


4. Price Distribution based on Cluster (Property Category). Hence, we suggested to pick cluster 0, improve cluster 1 and get rid of cluster 2 and 3:-

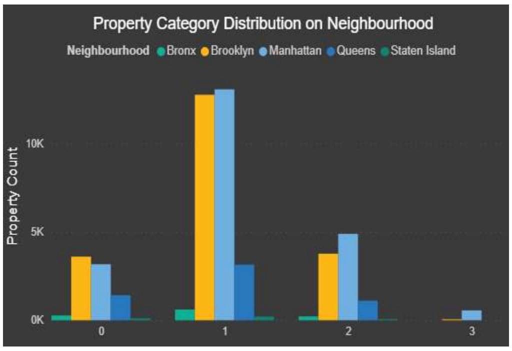


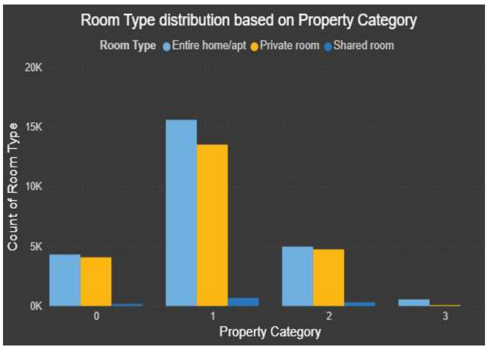
5. Why few localities in Staten Island are special – best rating and fewest prices:-



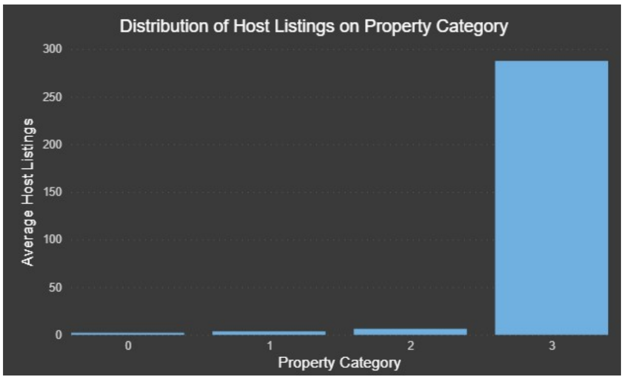


6. Insight on room types and Neighbourhood in Cluster 0 (as it has best reviews):-

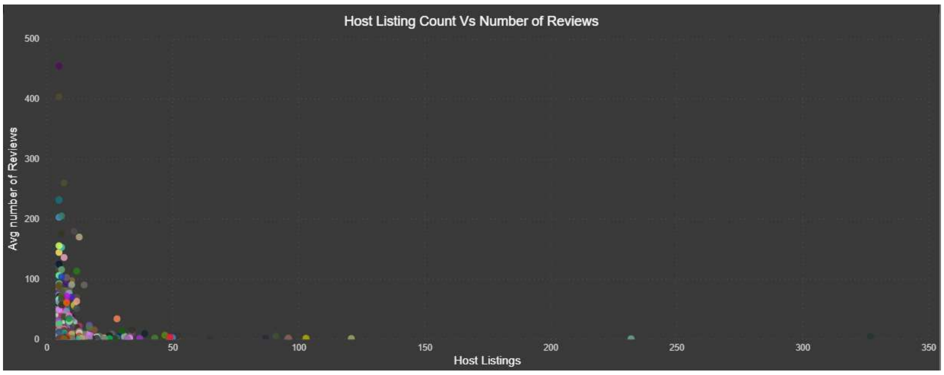




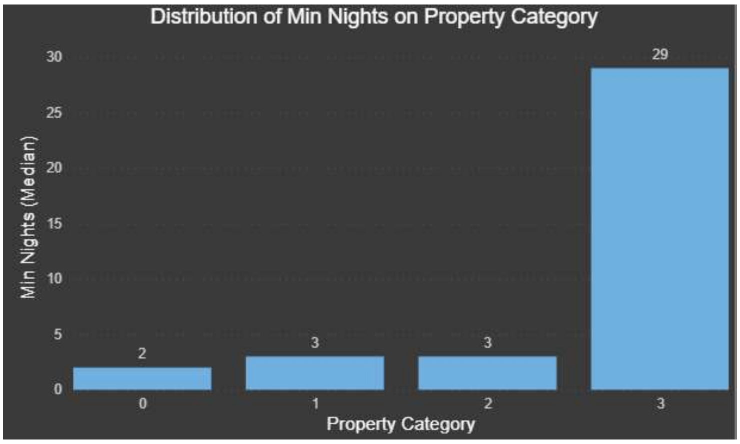
7. Why we suggest to find owners with less properties:-
   a. Cluster 0 property owners have least properties and cluster 3 got the most:-



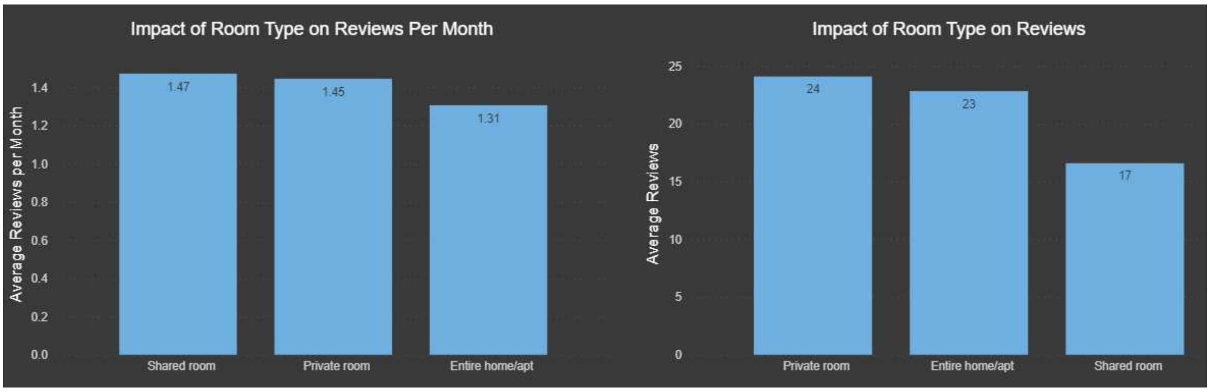
b. In the following scatter plot, owners with few properties have high reviews.



8. Properties with high requirement for minimum nights are clustered in group 3 – one with least reviews:-



9. Why we don't prefer shared properties:-



10. Cluster-wise availability round the year. Cluster 1 being worst:-