AIR - BNB CASE STUDY

METHODOLOGY

By :
NIKHIL KUMAR SINGH

PROBLEM STATEMENT

- For the past few months, Airbnb has seen a major decline in revenue.
- Now that the restrictions have started lifting and people have started to travel more, Airbnb wants to make sure that it is fully prepared for this change.

OBJECTIVE

To prepare for the next best steps that Airbnb needs to take as a business, you have been asked to analyze a dataset consisting of various Airbnb listings in New York. Based on this analysis, you need to give two presentations to the following groups.

1. Presentation - I

- Data Analysis Managers: These people manage the data analysts directly for processes and their technical expertise is basic.
- Lead Data Analyst: The lead data analyst looks after the entire team of data and business analysts and is technically sound.

2. Presentation - II

- Head of Acquisitions and Operations, NYC: This head looks after all the property and host acquisitions and operations. Acquisition of the best properties, price negotiation, and negotiating the services the properties offer falls under the purview of this role.
- Head of User Experience, NYC: The head of user experience looks after the customer preferences and also handles the properties listed on the website and the Airbnb app. Basically, the head of user experience tries to optimize the order of property listing in certain neighborhoods and cities in order to get every property the optimal amount of traction.

STEPS FOLLOWED

Data Understanding, Preparation, and Pre-Processing :

- Reading Data
- Treating Missing values
- Treating outlier
- Treating some error columns.

Tools Used :

- Python Jupyter Notebook :- Mainly to perform Data Wrangling.
- Tableau :- To come up with useful insights and business recommendations.

IMPORTING NECESSARY LIBRARIES

AIR-BNB CASE STUDY

In [1]:
```
# IMPORTING NECESSARY LIBRARIES
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# IMPORTING WARNINGS
import warnings
warnings.filterwarnings("ignore")
```

READING THE DATA-SET

- First I read the data-set with help of pandas then look at the first 5 rows for understanding the data.
- Then I look at the shape of the data-set which comes out as :-
- Total 48895 rows and 16 columns.

READING & UNDERSTANDING DATA

In [3]:
```
# READING THE DATA-SET
df = pd.read_csv("AB_NYC_2019.csv")
df.head()
```

Out[3]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_revie
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	

In [4]:
```
df.shape
```

Out[4]:
```
(48895, 16)
```

UNDERSTANDING THE DATA-SET

- After looking at the shape of the data-set we will look at the overall information of the data-set to get better understanding of data and for that we use .info() function.
- By using that function we get to know about how many entries are there, all the columns name with there data-type and count of Non-nulls and overall count of data-type like how many float64, int64 and object are there.

```
In [5]: df.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
```

STASTICAL DISTRIBUTION OF DATA-SET

- After that we use .describe() function to get the overall stastical distribution of data-set.
- Numerical Column :- If we use .describe() function without any other argument we will get the stastical distribution of only numerical column i.e. float and int.
- Categorical Column :- In the 2nd command we added some arguments before .describe() function to view the stastical distribution of categorical column i.e. object.
- WE CAN SEE THAT THERE ARE SOME IRREGULARITIES IN PRICE COLUMN AS MINIMUM PRICE IS O i.e. FREE STAY AND SOME OUTLIERS VALUES ARE ALSO PRESENT IN THE ABOVE COLUMNS, SO WE WILL LOOK INTO IT.

```
In [6]: df.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.
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	7.
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	32.
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	1.
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	1.
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000	1.
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000	2.
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000	327.

- WE CAN SEE THAT THERE ARE SOME IRREGULARITIES IN PRICE COLUMN AS MINIMUM PRICE IS O i.e. `FREE STAY` AND SOME OUTLIERS VALUES ARE ALSO PRESENT IN THE ABOVE COLUMNS, SO WE WILL LOOK INTO IT.

```
In [7]: df.select_dtypes(include=['object']).describe()
```

Out[7]:

	name	host_name	neighbourhood_group	neighbourhood	room_type	last_review
count	48879	48874	48895	48895	48895	38843
unique	47896	11452	5	221	3	1764
top	Hillside Hotel	Michael	Manhattan	Williamsburg	Entire home/apt	23-06-2019
freq	18	417	21661	3920	25409	1413

MISSING VALUES

- After understanding the data we will look at the missing values in the data-set and treat them accordingly.
- By using the .isnull().sum() function we will get the count of missing values i.e. NaN values that are present in each column of the data-set.
- In 2nd command we get the percentage of missing values means which column contains how much percentage of missing values and sort it in descending order to get highest missing value column on top.
- We can see there are 4 columns that have missing values.

HANDLING MISSING VALUES

```
In [8]: df.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]: (df.isnull().sum()/len(df)*100).sort_values(ascending=False)
```

```
Out[9]: last_review                     20.558339
        reviews_per_month               20.558339
        host_name                        0.042949
        name                             0.032723
        id                               0.000000
        host_id                          0.000000
        neighbourhood_group              0.000000
        neighbourhood                    0.000000
        latitude                         0.000000
        longitude                        0.000000
        room_type                        0.000000
        price                            0.000000
        minimum_nights                   0.000000
        number_of_reviews                0.000000
        calculated_host_listings_count   0.000000
```

ASSUMPTION OF MISSING VALUE

- MISSING VALUES ARE PRESENT IN NAME, HOST_NAME, LAST_REVIEWS & REVIEWS_PER_MONTH_COLUMNS.
- IN THE ABOVE EXPLORATING PART WE CAN SEE THAT IF THE N0_OF_REVIEWS = 0 THEN IT DOES NOT MAKE SENSE TO HAVE LAST_REVIEW & REVIEWS_PER_MONTH AND ARE MARKED AS NULL. HENCE THE MISSING VALUES IN THE DATA IS FOLLOWING A PATTERN AND WILL BE TREATED ACCORDINGLY.
- ABOVE ASSUMPTION HOLDS TRUE.

- MISSING VALUES ARE PRESENT IN `NAME, HOST_NAME, LAST_REVIEWS & REVIEWS_PER_MONTH_COLUMNS`.
- IN THE ABOVE EXPLORATING PART WE CAN SEE THAT IF THE `N0_OF_REVIEWS = 0` THEN IT DOES NOT MAKE SENSE TO HAVE `LAST_REVIEW & REVIEWS_PER_MONTH` AND ARE MARKED AS NULL. HENCE THE MISSING VALUES IN THE DATA IS FOLLOWING A PAT AND WILL BE TREATED ACCORDINGLY.

- LET's CHECK WHEATHER ABOVE ASSUMPTION HOLDS TRUE OR NOT.

In [10]:
```
assumption = df.loc[(df.last_review.isnull()) &
                    (df.reviews_per_month.isnull())][['number_of_reviews',  'last_review', 'reviews_per_month']]
assumption
```

Out[10]:

	number_of_reviews	last_review	reviews_per_month
2	0	NaN	NaN
19	0	NaN	NaN
26	0	NaN	NaN
36	0	NaN	NaN
38	0	NaN	NaN
...	...	...	...
48890	0	NaN	NaN
48891	0	NaN	NaN
48892	0	NaN	NaN
48893	0	NaN	NaN
48894	0	NaN	NaN

TREATING ASSUMPTION

- First we check the number of rows and columns that are present in the assumption and we found out that the exact amount of null values are present in both the columns.
- SO WE WILL IMPUTE THE REVIEWS_PER_MONTH COLUMN WITH 0 AND DROP THE LAST_REVIEW COLUMN AS IT WILL BE OF NO USE TO US FOR THE ASSIGNMENT.
- Now only 2 columns are left which have missing values so we will look at it and treat it accordingly.

```
In [11]: # LETS CHECK THE SHAPE OF ASSUMPTION
         assumption.shape

Out[11]: (10052, 3)
```

- THE EXACT AMOUNT OF NULL VALUES ARE PRESENT IN BOTH THE COLUMNS WHICH PROVES THAT THE ABOVE ASSUMPTION THAT WAS MADE IS CLEAR AND CONCISE.
- WE WILL IMPUTE THE `REVIEWS_PER_MONTH` COLUMN WITH `0` AND DROP THE `LAST_REVIEW` COLUMN AS IT WILL BE OF NO USE TO US FOR THE ASSIGNMENT.

```
In [12]: df.reviews_per_month.fillna(0, inplace=True)

In [13]: df.isnull().sum()

Out[13]: 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                 0
         calculated_host_listings_count    0
         availability_365                  0
         dtype: int64

In [14]: df.drop(['last_review'], axis=1, inplace=True)
```

HANDLING REMAINING 2 MISSING COLUMN

- After treating assumption accordingly we look at the values of remaining 2 missing column.
- HERE WE CAN SEE 21 HOST_NAME AND 16 NAME WHICH ARE MISSING.
- SINCE THESE VALUES ARE VERY LESS IN NUMBER WE WILL IMPUTE IT WITH MODE AS THEY ARE CATEGORICAL COLUMNS AND WE USE MODE TO FILL THE NULL VALUES OF CATEGORICAL COLUMN AND MEAN OR MEDIAN TO FILL THE NULL VALUES OF NUMERICAL COLUMNS.

```
In [15]: df[df.isnull().any(axis=1)]
```

Out[15]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	numb
360	100184	Bienvenue	526653	NaN	Queens	Queens Village	40.72413	-73.76133	Private room	50	1	
2700	1449546	Cozy Studio in Flatbush	7779204	NaN	Brooklyn	Flatbush	40.64965	-73.96154	Entire home/apt	100	30	
2854	1615764	NaN	6676776	Peter	Manhattan	Battery Park City	40.71239	-74.01620	Entire home/apt	400	1000	
3703	2232600	NaN	11395220	Anna	Manhattan	East Village	40.73215	-73.98821	Entire home/apt	200	1	
5745	4183989	SPRING in the City!! Zen-Style Tranquil Bedroom	919218	NaN	Manhattan	Harlem	40.80606	-73.95061	Private room	86	3	
5775	4209595	NaN	20700823	Jesse	Manhattan	Greenwich Village	40.73473	-73.99244	Entire home/apt	225	1	

- HERE WE CAN SEE `21 HOST_NAME` AND `16 NAME` WHICH ARE MISSING.
- SINCE THESE VALUES ARE VERY LESS IN NUMBER WE WILL IMPUTE IT WITH `MODE` AS THEY ARE `CATEGORICAL COLUMNS`.

HANDLING MISSING VALUES

- Here we imputed the null values of remaining 2 column with mode.
- Then at last we again use the .isnull().sum() function to look and cross-check wheather all the null values are gone or some columns still have the null values.
- At last we should always cross-check.
- Now we have treated the null values so we will move further.

- HERE WE CAN SEE `21 HOST_NAME` AND `16 NAME` WHICH ARE MISSING.
- SINCE THESE VALUES ARE VERY LESS IN NUMBER WE WILL IMPUTE IT WITH `MODE` AS THEY ARE `CATEGORICAL COLUMNS`.

```
In [16]: df.host_name.fillna(df.host_name.mode()[0], inplace=True)

         df.name.fillna(df.name.mode()[0], inplace=True)
```

```
In [17]: df.isnull().sum()
```

```
Out[17]: id                                0
         name                              0
         host_id                           0
         host_name                         0
         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
```

CHECKING OUTLIERS

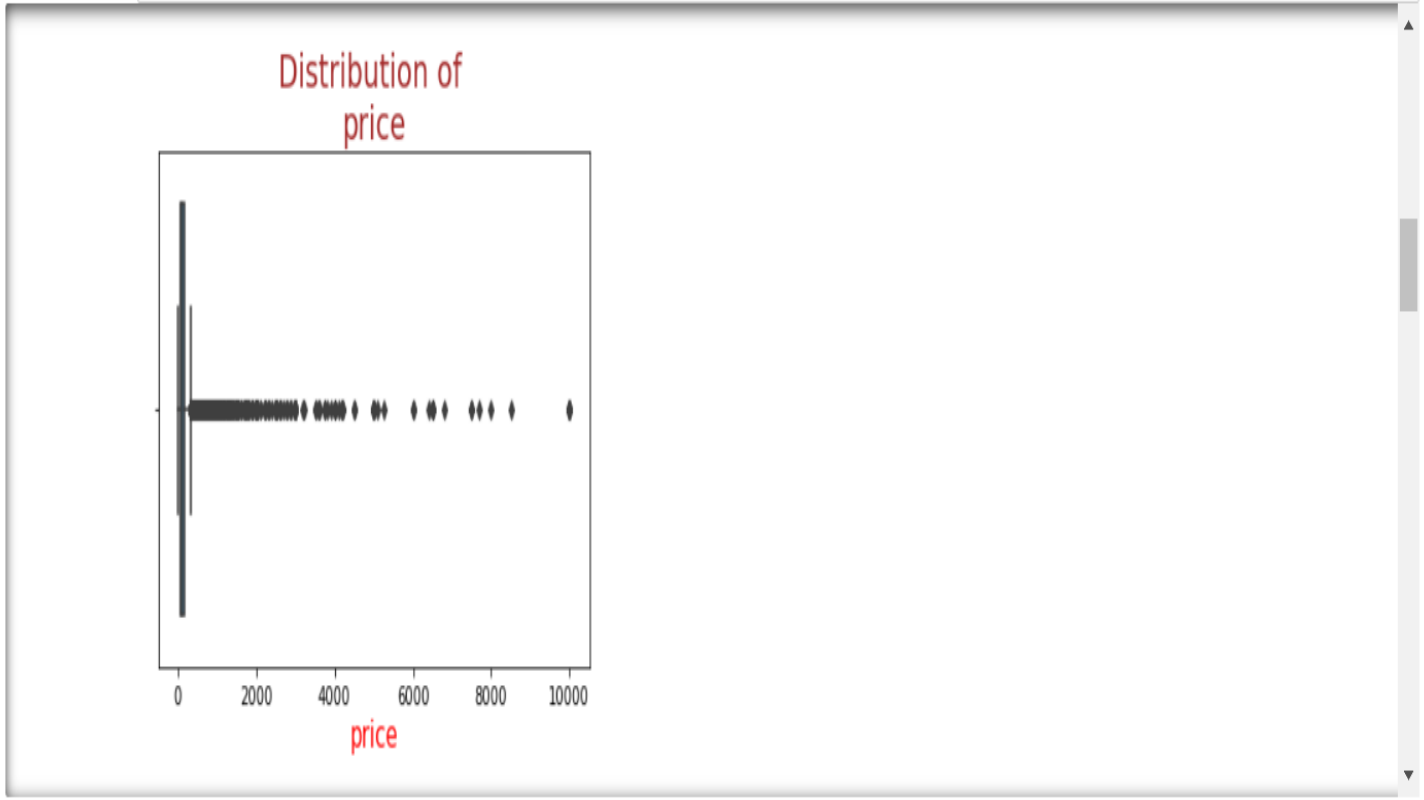
- After treating null values we will look at the outliers and treat them accordingly.
- Here in this code snippet I have created a list of all the Numerical column and running a for loop for finding the outliers in that list.
- And we found out that some columns contains outliers.

TREATING OUTLIERS

In [18]:
```
# I'AM CREATING A LIST AND RUNNING A FOR-LOOP FOR FINDING OUTLIERS IN COLUMNS

outliers_checking = df[['id','host_id','price','minimum_nights','number_of_reviews','reviews_per_month',
                        'calculated_host_listings_count','availability_365']]           # LIST COLUMN

for i in outliers_checking:
    sns.boxplot(df[i])      # CREATING A BOXPLOT TO CHECK FOR OUTLIERS
    plt.xlabel(i, fontdict={"fontsize":15, "fontweight":8, "color":"red"})      # GIVING NAME FOR X-AXIS
    plt.title("Distribution of \n" +i,fontdict={"fontsize":20, "fontweight":10, "color":"brown"})   # GIVING TITLE
    plt.show()
```



TREATING OUTLIERS

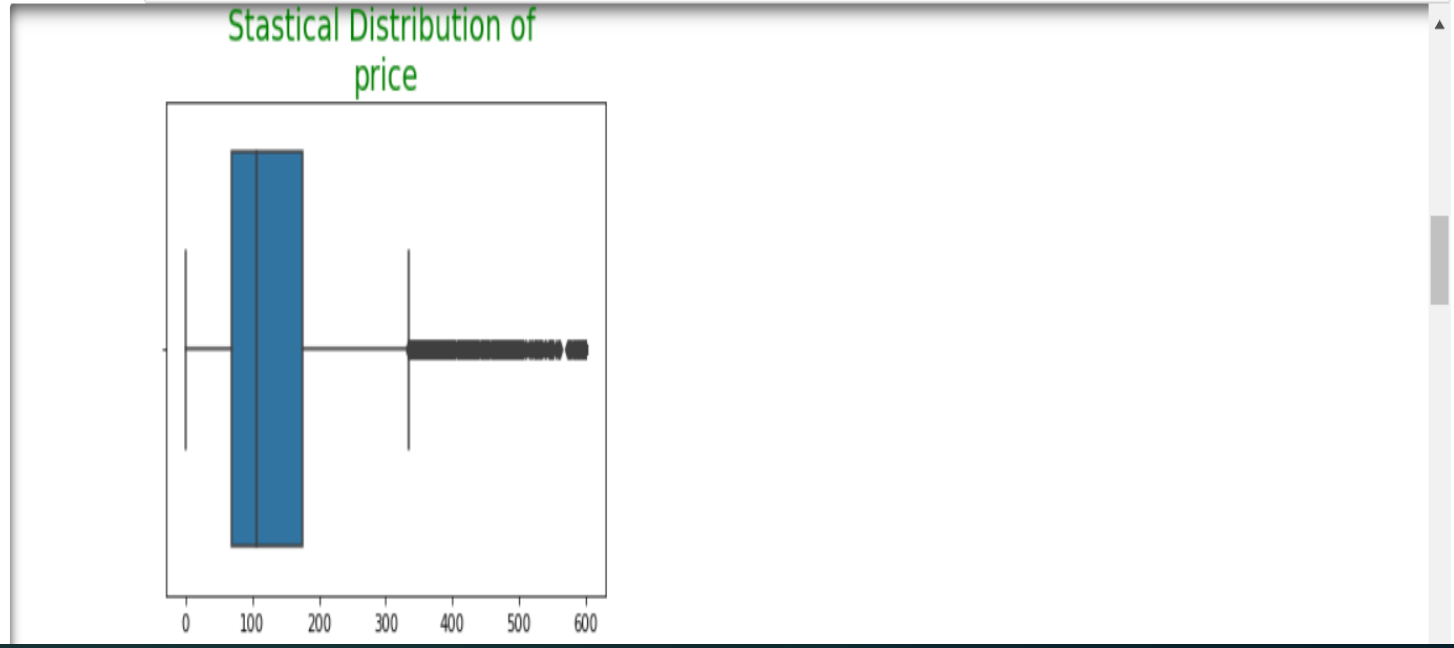
- Here in the first code snippet I have again running a for-loop for the same list to treat outliers.
- Then I have again run a for-loop to cross-check weather outliers have been treated accordingly or not.

In [19]:
```
# AGAIN RUNNING A FOR-LOOP FOR THE SAME LIST AND REMOVING THE OUTLIERS OF THAT LIST

for j in outliers_checking:
    q1 = df[j].quantile(0.10)   # THIS WILL GIVE ME 10% VALUE OF DATA
    q3 = df[j].quantile(0.90)   # THIS WILL GIVE ME 90% VALUE OF DATA
    iqr = q3 - q1                           # INTER-QUARTILE RANGE FORMULA
    lower_bound = q1 - 1.5*iqr              # LOWER-BOUND FORMULA
    upper_bound = q3 + 1.5*iqr              # UPPER-BOUND FORMULA
    df[j] = np.where(df[j] < lower_bound, lower_bound, df[j])
    df[j] = np.where(df[j] > upper_bound, upper_bound, df[j])
```

In [20]:
```
# CROSS-CHECKING THE OUTLIERS

for i in outliers_checking:
    sns.boxplot(df[i])
    plt.xlabel(i, fontdict={"fontsize":15, "fontweight":8, "color":"red"})
    plt.title("Stastical Distribution of \n" +i,fontdict={"fontsize":20, "fontweight":10, "color":"green"})
    plt.show()
```



SOME PROPERTIES HAVE PRICE = 0

- At first I got all the column which has price as 0.
- Then I replaced all the values of price = 0 with the median of price.
- Then at last cross-check weather the above code snippet has been worked or not.
- At last I saved the data in csv file.

After this step I have loaded the data-set in Tableau Desktop for visualizations and get & recommend insights.

- SOME PROPERTIES HAVE PRICE AS 0 i.e. FREE STAY. CLEARLY THIS IS AN ERROR. SO WE WILL REPLACE IT WITH MEDIAN VALUES.

In [21]:
```
df[df.price == 0]
```

Out[21]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	r
23161	18750597.0	Huge Brooklyn Brownstone Living, Close to it all.	8993084.0	Kimberly	Brooklyn	Bedford-Stuyvesant	40.69023	-73.95428	Private room	0.0	4.0	
25433	20333471.0	★Hostel Style Room | Ideal Traveling Buddies★	131697576.0	Anisha	Bronx	East Morrisania	40.83296	-73.88668	Private room	0.0	2.0	
25634	20523843.0	MARTIAL LOFT 3: REDEMPTION (upstairs, 2nd room)	15787004.0	Martial Loft	Brooklyn	Bushwick	40.69467	-73.92433	Private room	0.0	2.0	
25753	20608117.0	Sunny, Quiet Room in Greenpoint	1641537.0	Lauren	Brooklyn	Greenpoint	40.72462	-73.94072	Private room	0.0	2.0	
		Modern										

In [22]:
```
len(df[df.price == 0])
```

Out[22]: 11

In [23]:
```
price_0  = (df['price'] == 0 )
df.loc[price_0, 'price'] = df['price'].median()
```

In [24]:
```
df[df.price == 0]
```

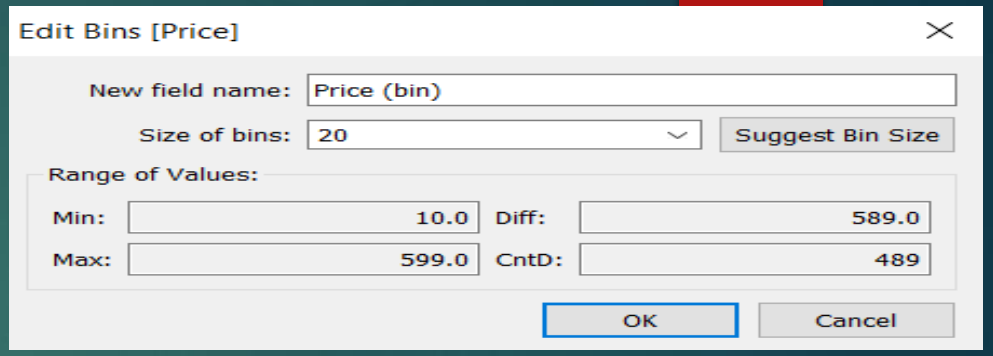
Out[24]:

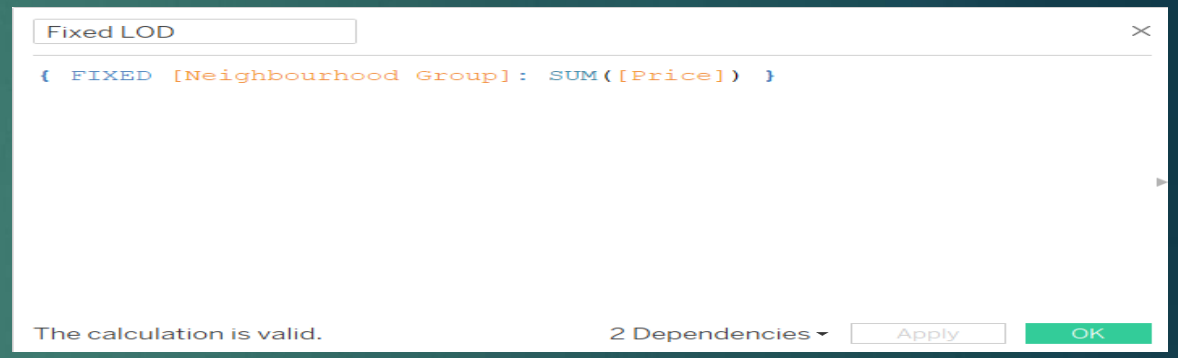
id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per

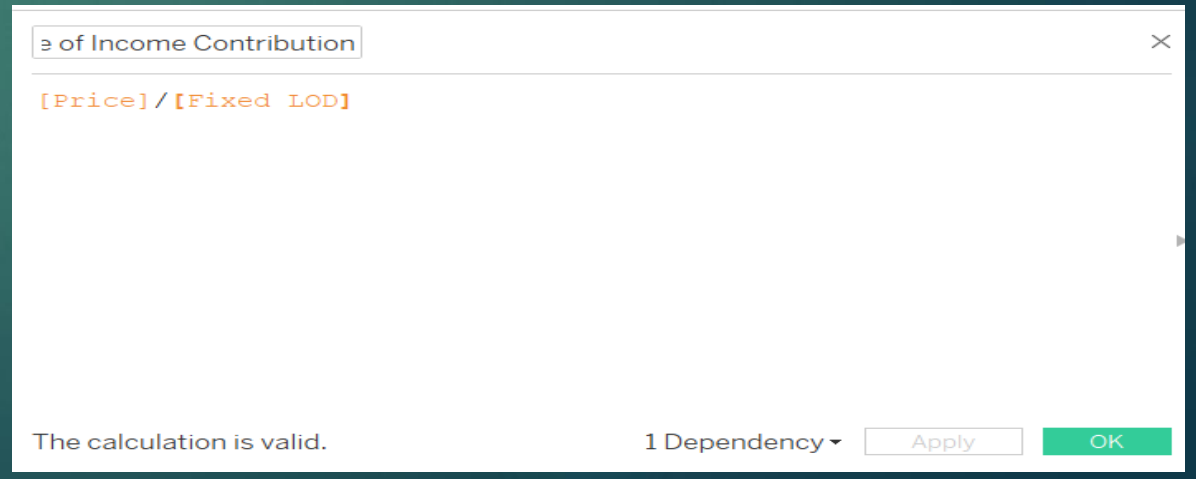
In [25]:
```
df.to_csv(r"D:\\Nikhil Documents\\AIR-BNB CASE STUDY\\Data_Cleaned_AB_NYC.csv", index = False, header=True)
```

CREATED BINS & CALCULATED FIELD IN TABLEAU

- First we have created a bin of Price column.
- Then we have created a new calculated field named as Fixed LOD using columns Neighbourhood Group and sum(price).
- And again with the help of above Fixed LOD we created a new calculated field named as Percentage of Income Contribution.



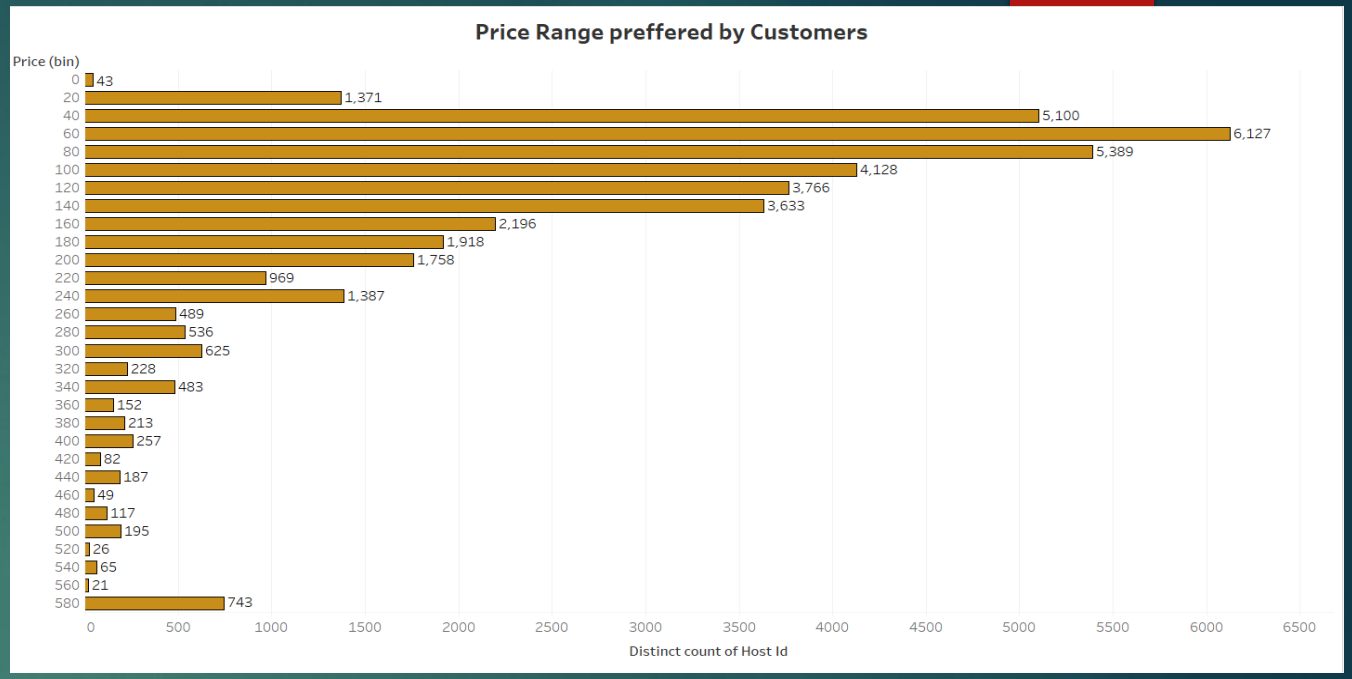




VISUALISATION

Now here with the help of bins and calculated field that we have created earlier we get the following insights :-

- Price range preferred by customers.
- Top 20 popular accommodation across NYC.



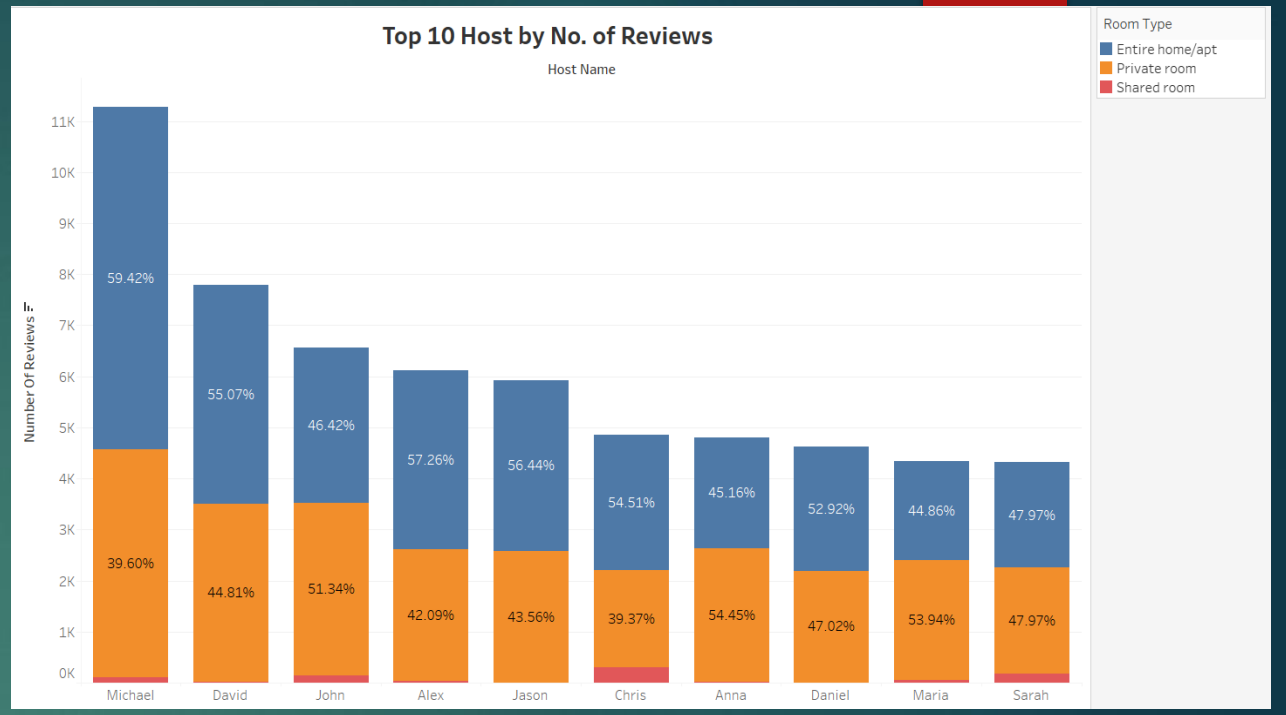
Top 20 Popular Accommodation across NYC

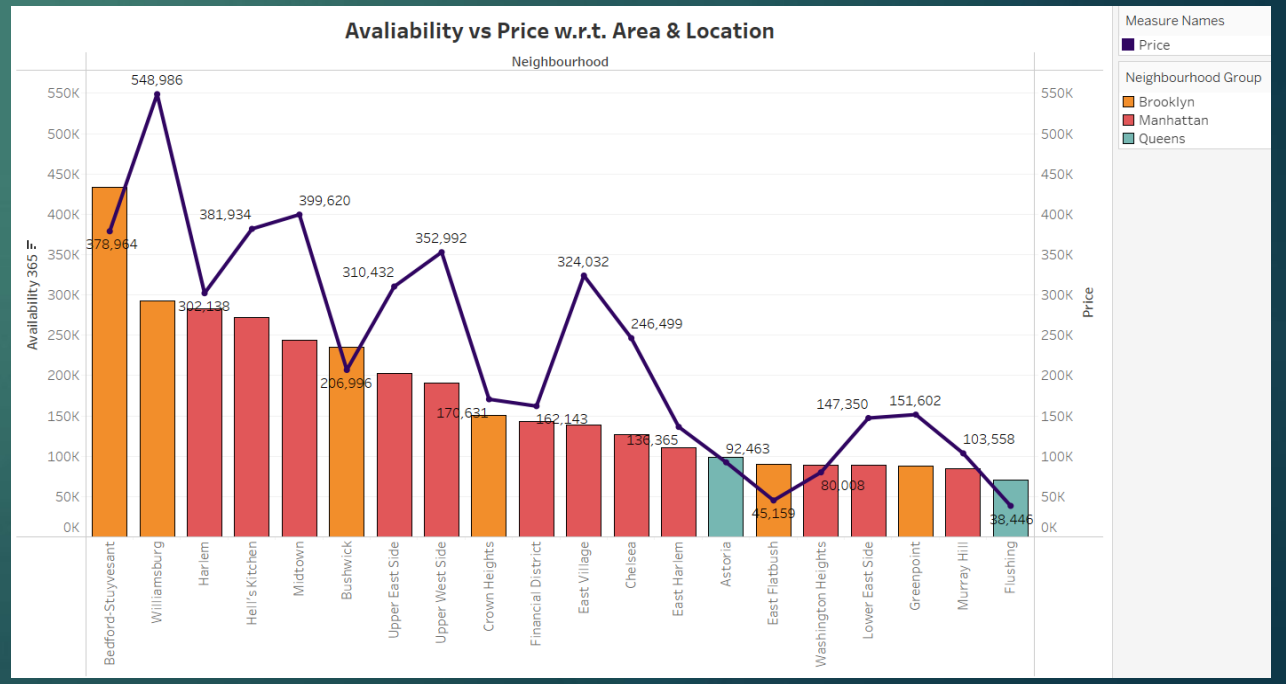
Neig..	Neighbourhood	Fixed LOD	Price	% of Total Percentage of Income Contributi..
Manhatt..	Midtown	3,867,583	399,620	5.16%
	Hell's Kitchen	3,867,583	381,934	4.93%
	Upper West Side	3,867,583	352,992	4.56%
	East Village	3,867,583	324,032	4.18%
	Upper East Side	3,867,583	310,432	4.01%
	Harlem	3,867,583	302,138	3.90%
	Chelsea	3,867,583	246,499	3.18%
Brooklyn	Williamsburg	2,374,577	548,986	11.54%
	Bedford-Stuyvesant	2,374,577	378,964	7.97%
	Bushwick	2,374,577	206,996	4.35%
	Crown Heights	2,374,577	170,631	3.59%
	Greenpoint	2,374,577	151,602	3.19%
Queens	Astoria	540,627	92,463	8.54%
	Long Island City	540,627	66,345	6.13%
	Flushing	540,627	38,446	3.55%
Bronx	Longwood	92,908	5,618	3.02%
Staten Island	St. George	37,044	5,270	7.10%
	Tompkinsville	37,044	3,200	4.31%
	Stapleton	37,044	2,672	3.60%
	Arrochar	37,044	2,389	3.22%

VISUALISATION

Here we get two insights :-

- Top 10 host by Number of Reviews.
- Availability vs Price w.r.t Area & Location (Dual Axis Chart).





RECOMMENDATION & PRESENTATION

- After doing all the analysis in the Data-set we made the presentation using the above given insights and visualization keeping the pyramid principle and the best business practices in mind and we have also given some recommendations on that presentation for the growth of the business.