

Alpha Team:

Ahmed Uddin Khalid Syed USN:1MS17IS008

Mohammed Salmaan Khan USN:1MS17IS065

Harsh Choudhary USN:1MS17IS046

Nayan Deep USN:1MS17IS070

Pavan Krishna USN:1MS17IS051

Aman Yadav USN:1MS17IS016

Ayush Agarwal USN:1MS17IS031

Rayjan Bhattarai USN:1MS17IS087

Anmol Kapoor USN:1MS17IS026

Gautam M G USN:1MS17IS160

AirBnB Ratings Analysis

January 6, 2021

Contents

1	AirBnB Ratings Analysis	1
1.1	Abstract	1
1.2	Data	1
1.3	Preprocessing	3
1.3.1	Data Types	3
1.3.2	Null or Missing Values	4
1.3.3	Dropping Columns	5
1.4	Statistics/Exploratory Analysis	6
1.4.1	Mean	6
1.4.2	Median	8
1.4.3	Mode	8
1.4.4	Histogram	10
1.5	Normality	12
1.5.1	Normal Distribution	12
1.5.2	QQ Plot	12
1.5.3	Kolmogorov–Smirnov(KS) test	14
1.6	Advanced Analysis	14
1.6.1	Hypothesis Testing	15
1.6.2	Covariance	18
1.6.3	Correlation	20
1.6.4	Regression Analysis	22
1.6.5	Machine Learning Algorithms	30
1.7	Advanced Visualization	39
1.7.1	Catplot	39
1.7.2	Box Plot	40
1.7.3	Count Plot	41
1.7.4	Scatter Plot	44
1.7.5	Visualization using Mapbox	45
1.8	Summary	46
1.9	Bibliography	47

Chapter 1

AirBnB Ratings Analysis

1.1 Abstract

Airbnb is an online rental for accommodation which provides us with good properties to stay. It has rentals around the world, and is spread across more than 220 countries. It has more than 7 million listings throughout the world. However, being such a huge organization, it becomes hard to keep track of which of these properties are real or value for money. There are a lot of methods to scam in such a high organization and stay unnoticed. To know the quality of a product, or in this case - the properties that are available for rent, the review section plays a vital role. From the reviews and ratings section we can analyze how well the property actually is, because it comes from a third person and not from the host. Review section gives us a first hand experience of the rental. So this minimizes the chances of the property or a listing to be fraudulent. To know which property is right for you, you need to analyze on how the ratings and reviews work. There are a lot of factors which come into play when analyzing on the basis of ratings. To analyze these factors and give us a better quality of a rental would prove value for money.

1.2 Data

We have the data set on Airbnb ratings and reviews from Kaggle.com. The data set that was employed was mainly focused on the Airbnbs in New York, Los Angeles and other cities of the world ; it is a detailed data set with 41 attributes, a few of the attributes being: price per day (which will hereafter be simply referred to as price), number of beds, property type, neighborhood, cleaning fee, host's ratings score, etc.

The Dataset comprises of three main tables:

LA Listings and NY Listings - The rentals listed in Los Angeles and New York, United States of America with 35 attributes:

- Listing id - It is used to create a join with the other files and is also the id number of the listing.
- Name - tells the name of the property
- Host ID - used to identify the host and used to create join with other files
- Host name - can be used to identify words associated with the host in reviews
- Host Response rate- number of inquiries the host has responded to, by the total inquiries the host had received.
- Host is Superhost - categorical - it describes about the hosts which are highly rated and are reliable
- Host total listings count - Total Listings listed by the host.
- Street - neighbourhood cleansed will be used instead
- City - used to identify the city of the property
- Neighbourhood - neighbourhood cleansed will be used instead
- State - used to identify the state
- Country - used to identify the city

- Latitude - we use it to visualise the data on the map
- Longitude - we use it to visualise the data on the map
- Property type - categorical variable house, townhouse, apartment, hostel, cabin
- Room type - categorical variable entire home/apt, private room or shared room
- Accommodates - discrete value describing property the number of guests the rental can accommodate
- Bathrooms - discrete value describing number of bathrooms included in the rental
- Bedrooms - discrete value describing number of bedrooms included in the rental
- Amenities - due to number of unique features (over 100) we will only concentrate on the total number of amenities
- Price - price per night for number of included guests
- Minimum nights - minimum number of nights a guest can stay for the rental
- Maximum nights: maximum number of nights a guest can stay for the rental
- Availability: We are not interested in data which depends on daily updates
- Calendar last scraped: We are not interested in data which depends on daily updates
- Number of reviews - total number of reviews in entire listing history
- Last review date: Tells the date when last time property was reviewed
- Review score Rating: Overall rating of the listing on a scale of 100.
- Review scores accuracy: How accurately does the listing page represent the property
- Review score cleanliness: Tells if the guests felt that if the space was clean
- Review scores Check-in : tells how smoothly the check-in was for guests
- Review score communication: How good was the communication, response time etc
- Review score location: How did guests feel about the neighbourhood? It can include facilities, transportation etc
- Review scores value: did the guest feel that it was worth of it's price
- Reviews per month - Average number of reviews received each month

AirBnB Reviews - This Table contains the reviews written by the reviewers of AirBnB. This table contains 6 columns:

- Listing ID - it is used to create a join with the other files
- Id - The unique review id for each review.
- Date - The Date at which the review is uploaded
- Reviewer ID - The id of the person writing the review
- Reviewer Name - The name of the person writing the review
- Comments - The Review on the listing of Airbnb

Sites from which data was extracted:

<https://www.kaggle.com/samyukthamurali/airbnb-ratings-dataset>

(1.1)

License : CC0 1.0 Universal (CC0 1.0) Public Domain Dedication

1.3 Preprocessing

Preprocessing is an essential process required to make the Data into more understandable form. As it is the process of reducing the errors in further computations and also making it simple and easy for further computations.

1.3.1 Data Types

In this section, we will be understanding what the data types are in this data set. Data can be of various forms, which can be broadly classified into two categories Numeric and Objects. Below we can see the various data types that have been used in the data set.

Listing ID	int64
Name	object
Host ID	int64
Host Name	object
Host Response Rate	float64
Host Is Superhost	bool
Host total listings count	float64
Street	object
City	object
Neighbourhood cleansed	object
State	object
Country	object
latitude	float64
longitude	float64
Property type	object
Room type	object
Accommodates	float64
Bathrooms	float64
Bedrooms	float64
Amenities	object
Price	int64
Minimum nights	int64
Maximum nights	float64
Availability 365	int64
Calendar last scraped	object
Number of reviews	int64
Last Review Date	object
Review Scores Rating	int64
Review Scores Accuracy	int64
Review Scores Cleanliness	int64
Review Scores Checkin	int64
Review Scores Communication	int64
Review Scores Location	int64
Review Scores Value	int64
Reviews per month	float64
dtype:	object

Fig. 1.1: Data Types of Columns

The above image shows the data types present in the columns. As we can see the types of data present in these columns. Data types such as int64 and float64 are of numeric data type. We can also observe 'bool' which stands for boolean and it can be either True or False.

1.3.2 Null or Missing Values

In this section, we check if the data set contains null or missing values. Null and missing values are values which may have been misplaced, or are values which have been lost. These values will cause problems in further computations. So, therefore we must either drop these values or replace them.

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

This is used to find if the data set contains any missing or null values. The output to this code is:

```
Listing ID          0
Name                31
Host ID            0
Host Name          232
Host Response Rate 45118
Host Is Superhost   0
Host total listings count 232
Street             31439
City               0
Neighbourhood cleansed 0
State              0
Country            0
latitude           0
longitude           0
Property type       0
Room type          0
Accommodates        31439
Bathrooms           31581
Bedrooms            31439
Amenities           31721
Price              0
Minimum nights      0
Maximum nights      31439
Availability 365     0
Calendar last scraped 31439
Number of reviews   0
Last Review Date    16683
Review Scores Rating 0
Review Scores Accuracy 0
Review Scores Cleanliness 0
Review Scores Checkin 0
Review Scores Communication 0
Review Scores Location 0
Review Scores Value 0
Reviews per month    0
dtype: int64
```

Fig. 1.2: Null values in the data set

As you can see the count of all the null values. So we know which all columns have null values. The next step is either to drop these columns or replacing the null values. In this section we will be

replacing the null values with the mean of those columns.

Filling null Values:

```
[ ] import math
df['Accommodates']=df['Accommodates'].fillna(math.floor(df['Accommodates'].mean()))
df['Bathrooms']=df['Bathrooms'].fillna(math.floor(df['Bathrooms'].mean()))
df['Bedrooms']=df['Bedrooms'].fillna(math.floor(df['Bedrooms'].mean()))
df['Maximum nights']=df['Maximum nights'].fillna(math.floor(df['Maximum nights'].mean()))
```

Fig. 1.3: (dataset).fillna()

1.3.3 Dropping Columns

Dropping columns is an important process of data pre-processing. As we need to remove the columns with null values and the columns which we would not require in further computation with the data.

```
df.drop(['Listing ID','Name','Host ID','Host Name','Host Response Rate','Host Is Superhost','Street','Last Review Date',
        'Calendar last scraped'], axis=1, inplace=True)
df.head(3)
```

Fig. 1.4: Dropping columns

In the above image we can see we have dropped some of the columns which had null values(i.e, 'Street'). Also the columns which we will not require further such as 'Listing ID'.

	Host total listings count	City	Neighbourhood	State	Country	latitude	longitude	Property type	Room type	Accommodates	Bathrooms	Bedrooms	Amenities	Price	Minimum nights
0	4.0	Bronx	Allerton	NY	United States	40.866889	-73.857756	House	Private room	1.0	1.0	1.0	Cable TV,Internet,Wireless Internet,Kitchen,Fr...	43	2
1	1.0	Bronx	Soundview	NY	United States	40.829392	-73.865137	House	Private room	1.0	1.0	1.0	TV,Internet,Wireless Internet,Air conditioning...	28	2
2	16.0	Bronx	Fordham	NY	United States	40.869139	-73.895096	House	Private room	4.0	3.0	2.0	Internet,Wireless Internet,Air conditioning,Ki...	80	3

Fig. 1.5: Data After Preprocessing

1.4 Statistics/Exploratory Analysis

1.4.1 Mean

Mean is the average value of a set of elements. It is used for various calculation in Machine learning algorithms. Below is the formula for finding the mean.

Equation:

$$A = \frac{1}{n} \sum_{i=1}^n a_i = \frac{a_1 + a_2 + \dots + a_n}{n} \left(\frac{1}{n} \sum_{i=1}^n x_i \right) \quad (1.2)$$

We have calculated the average values for some of the columns :

```
print("*****")
print("\t\t\t\t\tBasic Statistics\n\n")
print('Mean: \n\n')

print("Price          \tMean:",df['Price'].mean())
print("Number of reviews", "\tMean:",df['Number of reviews'].mean())
print("Availability      ", "\tMean:",df['Availability 365'].mean())
print("Accommodates       ", "\tMean:",df['Accommodates'].mean())
print("Reviews per month", "\tMean:",df['Reviews per month'].mean())
```

```
*****
                                     Basic Statistics

Mean:

Price          Mean: 154.54574977887498
Number of reviews Mean: 16.589037479042627
Availability      Mean: 131.23845859351277
Accommodates       Mean: 2.4755970375846545
Reviews per month Mean: 1.1352155143962435
```

Fig. 1.6: Mean of some columns in the data set

From the above output we can analyze that:

- **Price:** That the average prices for a property to rent in New York City are around 154 dollars.
- **Number of reviews:** The average number of reviews on each of the listings in New York is 16.
- **Availability:** The average availability of the listings in New York in a year is 131 days.
- **Accommodates:** The average number of accommodates on listings is 2.

- **Reviews per month :** The number of reviews each of the listings receives in a month on an average is 1.

1.4.2 Median

Median is the middle value of the set of the elements when put in ascending order. It separates the upper half of values from the lower half of values.

```
print('Median: \n\n')

print("Price          \tMedian:",df['Price'].median())
print("Number of reviews", "\tMedian:",df['Number of reviews'].median())
print("Availability      ", "\tMedian:",df['Availability 365'].median())
print("Accommodates      ", "\tMedian:",df['Accommodates'].median())
print("Reviews per month", "\tMedian:",df['Reviews per month'].median())
```

We found the median for the numeric values in our data.

Median:

Price	Median: 105.0
Number of reviews	Median: 4.0
Availability	Median: 79.0
Accommodates	Median: 2.0
Reviews per month	Median: 0.45

Fig. 1.7: Median of some columns in the data set

From the above output, we see that number of reviews on listings which separates the higher values from the lower values is 4. The value which separates the availability of a listing in a year from the higher values to the lower values is 79 days.

1.4.3 Mode

Mode is the most occurring value in a column. This can be used for a better understanding of a column.

```
print('Mode: \n\n')
DF=df[['Accommodates','City','Neighbourhood','Property type','Room type',
'Price','Host total listings count']]
print(DF.mode())
```

Mode:

```

Accommodates      City Neighbourhood Property type      Room type \
0              2.0  Manhattan  Williamsburg      Apartment  Entire home/apt

Price  Host total listings count
0     150                        1.0

```

Fig. 1.8: Mode

From the above output we can see that the most occurring value in Accommodates is 2. Which means most of the properties in New York can accommodate up to 2 people. Under the City column in the output, we can see that most of the listings of AirBnB are in the city of Manhattan.

1.4.4 Histogram

Histogram is a method to visualize the numerical values in a data set. It is a bar type chart which shows the count of the values in a certain range. It uses bins to define the different ranges such that each bin represents a certain range.

```
DF=df[['Bathrooms','Bedrooms','Price','Minimum nights','Maximum nights',
'Availability 365','Number of reviews','Review Scores Rating',
'Review Scores Accuracy','Review Scores Cleanliness','Review Scores Checkin',
'Review Scores Communication','Review Scores Location','Review Scores Value']]
DF2=df[['Host total listings count','latitude','longitude','Accommodates']]
fig = plt.figure(figsize = (20,15))
ax = fig.gca()
DF.hist(ax = ax)
```

Below you can the histograms being displayed in the figure:

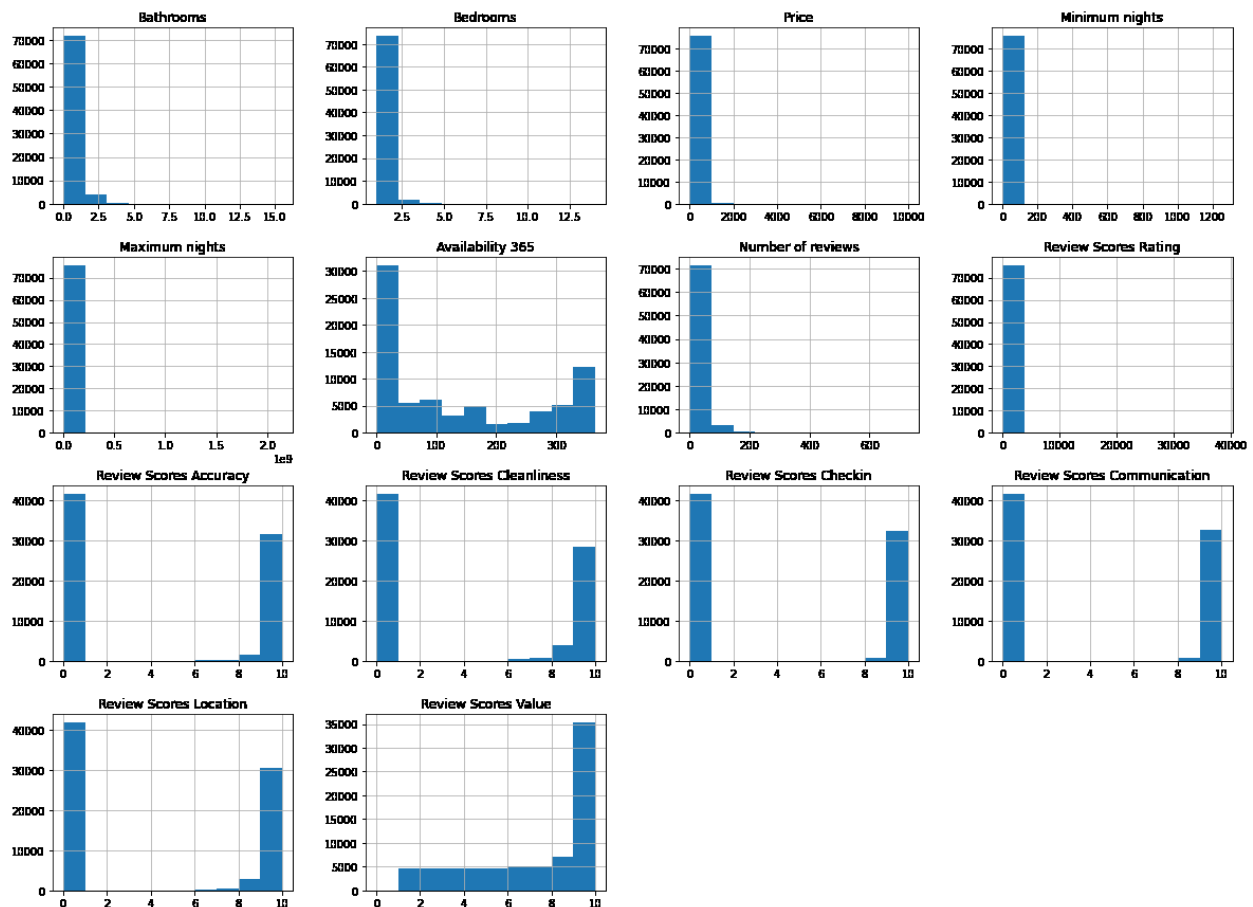


Fig. 1.9: Histogram of some columns

```
fig = plt.figure(figsize = (20,15))
ax = fig.gca()
DF2.hist(ax = ax)
```

Some more histograms on few columns such as latitude ,longitude ,accommodates and host total listings count.

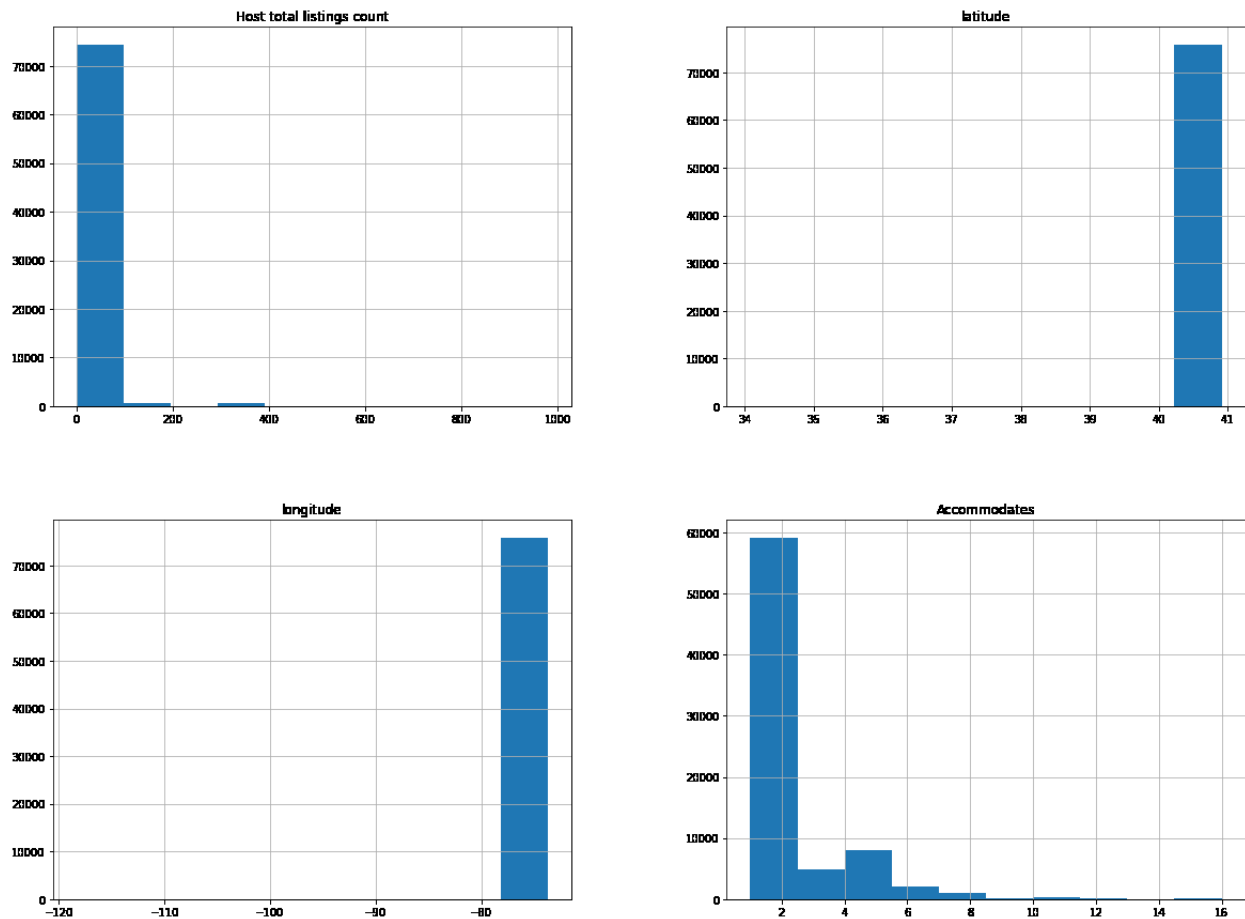


Fig. 1.10: Few more Histograms of the data set

1.5 Normality

1.5.1 Normal Distribution

There are multiple distribution types of the data. The most preferred distribution for data analysis is the normal distribution.

If our data already is normally distributed, it is easier for us, if not, then the data has to be converted to normal distribution.

There are various methods to check for Normal Distribution.

We have plotted a Q-Q Plot and performed the KS test to check for Normality.

We have performed these two normality tests on the column 'Availability 365' - which tells us how many days in the year the property is available to be rented

1.5.2 QQ Plot

To interpret the QQ plot, refer the below figure which shows how normally distributed data, skewed data and data peaked in the middle will look when plotted using a QQ plot

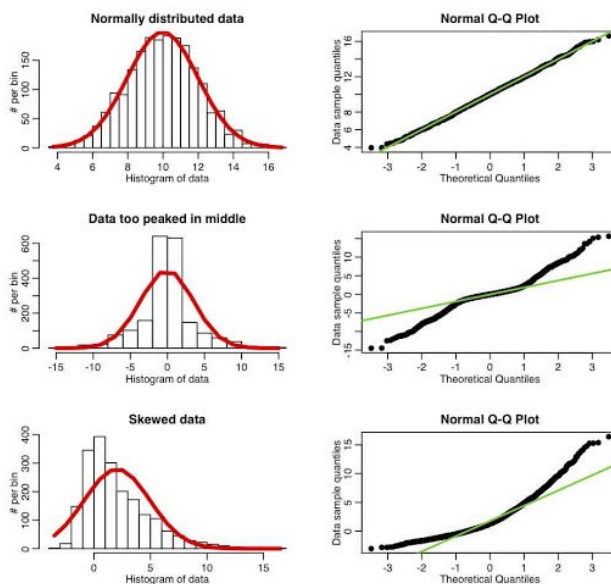


Fig. 1.11: Inference for interpreting QQ Plot

Using the code given below, we plot the QQ Plot

```
import statsmodels.api as sm
import pylab as py
sm.qqplot(df['Availability 365'],line='45')
py.show()
```

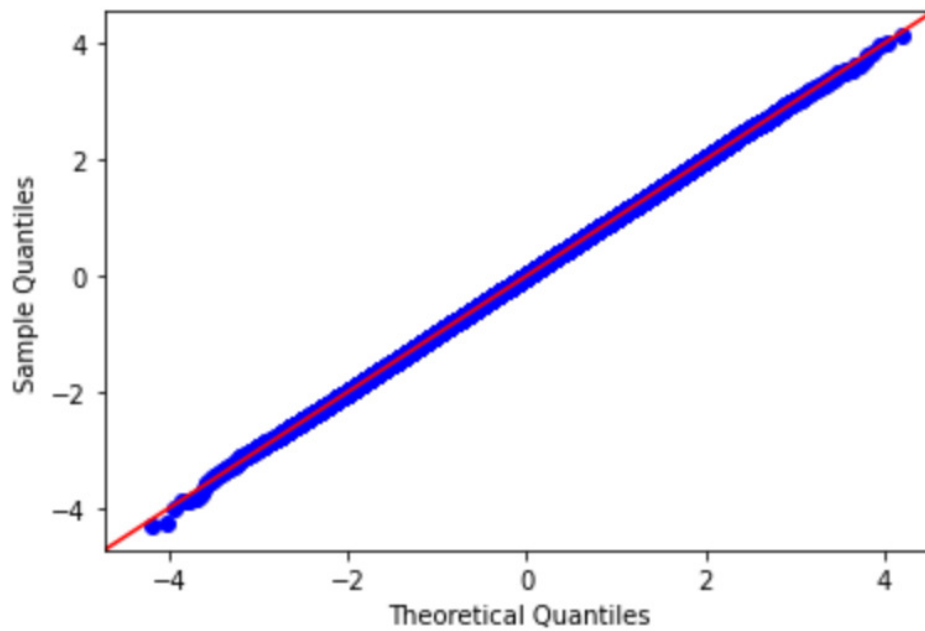


Fig. 1.12: QQ Plot showing Normal Distribution

After comparing our QQ Plot with the Inference Diagram provided above, we conclude that the column follows Normal Distribution

1.5.3 Kolmogorov–Smirnov(KS) test

If the particular column follows a normal distribution, then value of the KS statistic will be 0.

We perform hypothesis testing to decide if it follows a Normal Distribution.
We form a null hypothesis as well as an alternative hypothesis.

1. It is Normally Distributed
2. It is not Normally Distributed

Based on the P-Value , we can decide whether the difference is large enough to reject the null hypothesis:

1. If the P-Value of the KS Test is larger than 0.05, we assume a normal distribution
2. If the P-Value of the KS Test is smaller than 0.05, we do not assume a normal distribution

Here, we have applied KS test to 'Availability 365' column which tells us how many days in a year the property is available. And we observe that it follows Normal Distribution since our value is p value is more than 0.05

```
from scipy.stats import kstest, norm
df['Availability 365']=norm.rvs(size=75749)
ks_statistic, p_value=kstest(df['Availability 365'], 'norm')
print(ks_statistic, p_value)
```

Upon applying the KS Test , the following p value and ks statistic values are obtained

```
from scipy.stats import kstest, norm
df['Availability 365']=norm.rvs(size=75749)
ks_statistic, p_value=kstest(df['Availability 365'], 'norm')
print(ks_statistic, p_value)
```

0.002981125274782226 0.5112072829183858

Fig. 1.13: Obtained p value

Since the KS Statistic Value is 0 , and the obtained P-Value is greater than 0.05, we conclude that it follows a Normal Distribution

1.6 Advanced Analysis

1.6.1 Hypothesis Testing

A Hypothesis Test evaluates two mutually exclusive statements about a population to determine which statement is best supported by the sample data. Using Hypothesis Testing, we try to interpret or draw conclusions about the population using sample data.

The first step would be to establish an hypothesis: a null hypothesis and an alternative hypothesis:

1. **Null Hypothesis (H0):** A statement in which no difference or effect is expected
2. **Alternate Hypothesis (H1):** A statement that some difference or effect is expected

With respect to our dataset, we have performed two hypothesis tests.

Hypothesis Test 1:

H0 = The average prices of the houses in Manhattan is the same as the average prices of the whole of New York.

H1 = The average prices of the houses in Manhattan is smaller than the average prices of the whole of New York

We first calculate the mean of Manhattan prices and compare it with the mean of the price of whole of New York with the help of the following code:

```
from scipy import stats
from statsmodels.stats import weightstats as mstats

df = df.sample(frac=1)

manhattan_sample = df[df['City'] == 'Manhattan']
manhattan_sample = manhattan_sample[:50]
population_mean = df['Price'].mean()
manhattan_sample_mean = manhattan_sample['Price'].mean()
print('New York Mean: ', population_mean, 'Manhattan Mean: ', manhattan_sample_mean)
```

The following output is obtained:

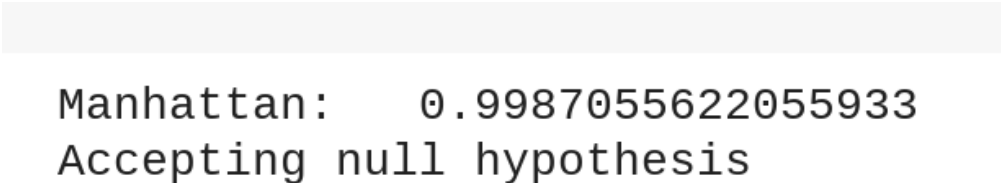
```
New York Mean:    154.54574977887498 Manhattan Mean:    203.38
```

Fig. 1.14: Manhattan Mean

Then we perform the Z test

```
zstat, pvalue = mstats.ztest(manhattan_sample['Price'],
x2=None,value=population_mean,alternative='smaller')
print('Manhattan: ',float(pvalue))
if pvalue<0.05:
    print("Rejecting null hypothesis")
else:
    print("Accepting null hypothesis")
```

For which we get the following output:



```
Manhattan: 0.9987055622055933
Accepting null hypothesis
```

Fig. 1.15: Manhattan Result

Since the p value is greater than 0.05, we accept the null hypothesis

Hypothesis Test 2:

H0 = The average prices of the houses in Brooklyn is the same as the average prices of the whole of New York.

H1 = The average prices of the houses in Brooklyn is smaller than the average prices of the whole of New York

We first calculate the mean of Brooklyn prices and compare it with the mean of the price of whole of New York with the help of the following code:

```
from scipy import stats
from statsmodels.stats import weightstats as mstats
```

```
df = df.sample(frac=1)
brooklyn_sample = df[df['City'] == 'Brooklyn']
brooklyn_sample = brooklyn_sample[:50]
population_mean = df['Price'].mean()
brooklyn_sample_mean = brooklyn_sample['Price'].mean()
print('New York Mean: ', population_mean, 'Brooklyn Mean: ', brooklyn_sample_mean)
```

The following output is obtained:

```
New York Mean:    154.54574977887498 Brooklyn Mean:    100.34
```

Fig. 1.16: Brooklyn Mean

Then we perform the Z test

```
zstat, pvalue = mstats.ztest(brooklyn_sample['Price'],
x2=None, value=population_mean, alternative='smaller')
print('Brooklyn: ', float(pvalue))
if pvalue<0.05:
    print("Rejecting null hypothesis")
else:
    print("Accepting null hypothesis")
```

For which we get the following output:

```
Brooklyn:    3.506604028935684e-07
Rejecting null hypothesis
```

Fig. 1.17: Brooklyn Result

Since the p value is lesser than 0.05(very close to 0), we reject the null hypothesis

1.6.2 Covariance

The Covariance Matrix is Basically a matrix which represents the covariance between two columns. Covariance and Correlation matrix are very similar to each other. Covariance is not bounded between -1 and 1 like correlation . Covariance between two variables is a degree by which they go together, so when one value is positive when the other value of a column is positive or when one value is negative when the other value of a column is negative then they have a very high degree of covariance.

$$cov_{x,y} = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{N - 1} \quad (1.3)$$

The code for the covariance matrix is given below:

```
import pandas as pd
DF=df[['Availability 365','Review Scores Accuracy','Review Scores Value','Review Score
'Reviews per month','Price']]
cov_matrix=DF.cov()
sns.heatmap(cov_matrix,cmap='Blues_r',annot=True)
plt.figure(figsize=(18,12))
plt.show()
```

The Heatmap obtained for our data,which shows the covariance between the columns is given below:

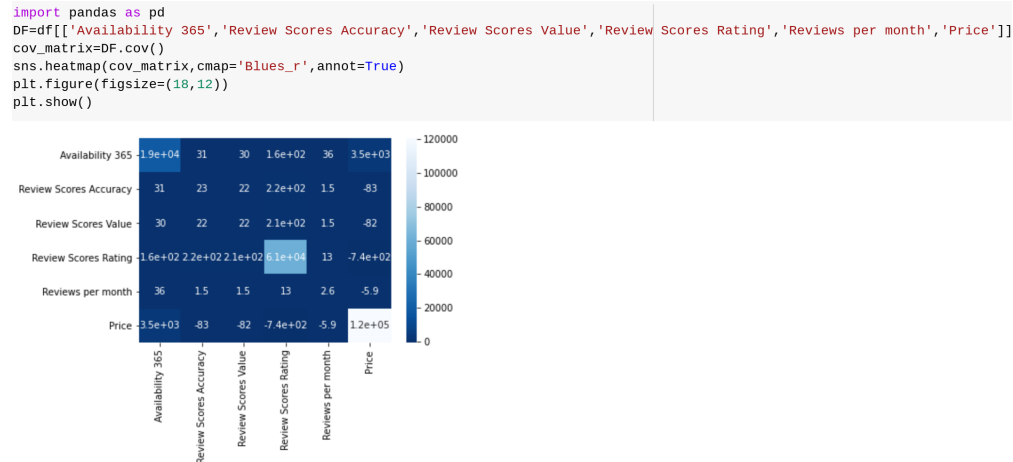


Fig. 1.18: Heatmap which shows Covariance

The covariance between the columns is represented in darker and lighter shades of the colors depending on the value, as shown in the scale of the heatmap

The cell in this matrix have a different value of covariance. So when a value in the cell of:

1. Availability ,Availability represents the variance of the Availability
2. The covariance between Availability,Reviews score rating is the covariance of those two columns

1.6.3 Correlation

Correlation is a method of finding the relationship between 2 columns.

The relationship can be weak or strong depending on the values.

The values range from -1 to 1.

If the value is closer to 1, it means that the relationship between those columns is strong, whereas if the value is closer to -1, it means that the relationship between those columns is not strong.

If there is no relationship at all between two variables, then the correlation coefficient will be 0.

$$\rho = \frac{\text{cov}(X,Y)}{\sigma_x \sigma_y}$$

The code for the heatmap is given below:

```
plt.figure(figsize=(15,8))
_=sns.heatmap(df.corr(),cmap='YlGnBu', annot = True)
```

The heatmap obtained for our data, which shows the relationship between the columns is given below:

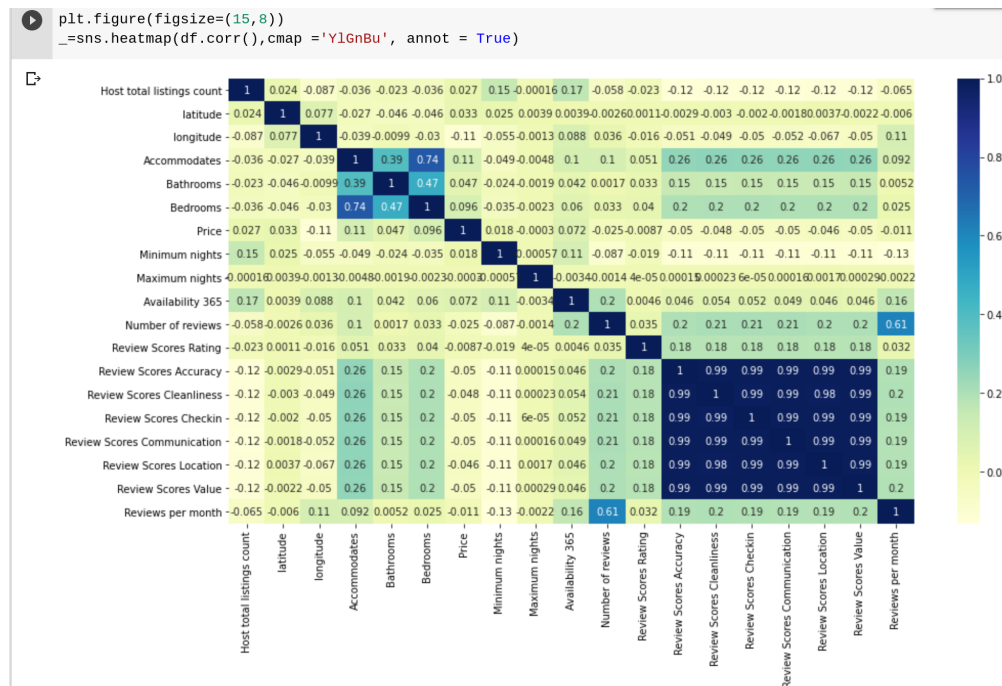


Fig. 1.19: Heatmap which shows Correlation

From the above heatmap , we can see all the relationships between the columns. From the figure, we can see that the columns that have the dark blue color are the columns that have a strong relationship.

To mention a few:

1. Review Scores Accuracy AND Review Scores Value
2. Review Scores Cleanliness AND Review Scores Checkin
3. Review Scores Communication AND Review Scores Accuracy
4. Review Scores Location AND Review Scores Value
5. Review Scores Location AND Review Scores Cleanliness

We can also see the columns that have a weak relationship by observing the values that have a lighter shade in color- like yellow.

To name a few:

1. Availability 365 AND Accomodates
2. Accomodates AND Price
3. Review Scores Accuracy AND Reviews per month
4. Reviews per month AND Maximum Nights
5. Review Scores Checkin AND Reviews per month

1.6.4 Regression Analysis

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (predictor). This technique is used for forecasting, time series modelling and finding the casual effect relationship between the variables.

Significance of Regression Analysis:

1. It indicates the significant relationships between dependent variable and independent variable.
2. It indicates the strength of impact of multiple independent variables on a dependent variable.

1.6.4.1 Linear Regression

Linear regression is used when we want to predict the value of a variable whose value is dependent on another variable. The variable we want to predict is called the dependent variable . The variable we are using to predict the other variable's value is called the independent variable. It has an equation of the form :

$$Y = a + bX \quad (1.4)$$

Where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

Now we perform linear regression to predict dependent variable which is 'Review Scores Value'. This column is dependent mainly on 5 columns in the data set. The 5 independent columns are :

- **Accuracy** : how accurate the listing has been described compared to how it turned out to be, for eg guests should be able to find up to date info and photos of the listing on AirBnB.
- **Cleanliness** : is a value which tells us how clean the property was.
- **Communication** : is how well the host communicates before and during the stay ,guests often care that their host responds quickly, reliably and frequently to their messages and questions.
- **Check-in** : is how the process of checking into the property was, and it is valued on a scale of 10.
- **Locations** : is how well the listing is located and if it is easy to locate the property mentioned on the location in the listing.

We will be using Cleanliness, Accuracy and Communication to find the Review Scores Value. Below is the code which would separate the data to build a model and test it. So we will split the data into training for the model and testing the model.

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

```
X = df[['Review Scores Accuracy', 'Review Scores Cleanliness', 'Review Scores Communicat
Y = df[ 'Review Scores Value']
X = preprocessing.normalize(X)
```

```
X = np.hstack((np.ones( (len(df['Review Scores Accuracy']) ,1)), X))

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

After running this we get the split size of the rows and columns used for testing and training:

```
(60597, 4)
(15150, 4)
(60597,)
(15150,)
```

Fig. 1.20: Splitting the dataset for training and testing

Now that we split the data, the model can be trained and tested. This gives us the shape of how many rows and columns are in X and Y. We have used the R2 score and the RMSE score to evaluate the model score. Below is the code for model training and testing:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score

lin_model = LinearRegression().fit(X_train, Y_train)
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("-----")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2*100))
print("\n")
print('Score  :',lin_model.score(X,Y))
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)
print("The model performance for testing set")
print("-----")
```

```
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2*100))
```

```
The model performance for training set
```

```
-----
RMSE is 0.676278168447478
R2 score is 97.93039596293063
```

```
Score : 0.9791621293669295
```

```
The model performance for testing set
```

```
-----
RMSE is 0.6881591286192049
R2 score is 97.85953745799264
```

Fig. 1.21: The output of the linear regression model

Here we can see we have obtained an R2 score of 97 percent in training and testing the model and we have got an rmse score of 0.67 in training and 0.68 in testing the model.

1.6.4.2 Decision Tree

Decision trees are used specific decisions and regression analysis, they are in a form of a tree structure. It basically breaks down a data set into smaller subsets based on criteria and at the same time an associated decision tree is incrementally developed. It is a tree or flowchart like structure in which each of the nodes represents a test attribute. Each tree in a regression decision tree gives an output of a Gaussian distribution as a prediction. An aggregation is performed over trees to find a Gaussian distribution closest to the combined distribution for all trees in the model.

We create a model of decision tree to find the higher price for a listing in New York. To find the higher price based on other conditions we use the column Price and few more columns such as Neighbourhood, City, Room type, Minimum nights and Number of reviews.

```
data_tree=df[['Neighbourhood','City','Room type','Minimum nights','Number of reviews'],
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
data_tree['Neighbourhood_new'] = labelencoder.fit_transform(data_tree['Neighbourhood'])
data_tree['City_new'] = labelencoder.fit_transform(data_tree['City'])
data_tree['Room_type_new'] = labelencoder.fit_transform(data_tree['Room type'])
print(data_tree.head())
data_tree=data_tree[data_tree.Price<=180]
data_tree=data_tree[data_tree.Price>=90]
```

```
len(data_tree)
```

The above code is used to change categorical data into numeric form by using 'LabelEncoder' and to display the new columns of categoric data with numeric values and the above listed columns we will be using to create the decision tree.

	Neighbourhood	City	Room type	Minimum nights	\
3866	Williamsburg	Brooklyn	Entire home/apt	2	
45583	Hell's Kitchen	Manhattan	Entire home/apt	30	
66940	Hell's Kitchen	Manhattan	Private room	5	
72058	Bushwick	Brooklyn	Private room	3	
50184	Upper East Side	Manhattan	Entire home/apt	30	

	Number of reviews	Price	Neighbourhood_new	City_new	Room_type_new
3866	24	170	220	1	0
45583	3	369	99	2	0
66940	18	180	99	2	2
72058	4	30	28	1	2
50184	0	187	207	2	0

Fig. 1.22: Columns used for creating decision tree

In the above output we can see for categorical data such as Neighbourhood, City and Room type we have give numeric labels in the new columns.

Now we take out the Median Absolute Deviation(MAD) for this data which is : _____

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
x_train,x_test,y_train,y_test=train_test_split(data_tree[['Neighbourhood_new','City_new','Price']],y_train,y_test)
Reg_tree=DecisionTreeRegressor(criterion='mse',max_depth=3,random_state=0)
Reg_tree=Reg_tree.fit(x_train,y_train)
y=y_test['Price']
predict=Reg_tree.predict(x_test)
print("median absolute deviation (MAD): ",np.mean(abs(np.multiply(np.array(y_test.T-pr
```

```
median absolute deviation (MAD): 0.1742786812184003
```

Fig. 1.23: MAD

Now we will create a decision tree those columns. To create the decision tree we run the code below: _____

```
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont
```

```

from IPython.display import Image as PImage
from sklearn.tree import export_graphviz
with open("tree1.dot", 'w') as f:
    f = export_graphviz(Reg_tree,out_file=f,max_depth = 3,impurity = True,feature_names=feature_names,
    check_call(['dot', '-Tpng', 'tree1.dot', '-o', 'tree1.png'])
img = Image.open("tree1.png")
draw = ImageDraw.Draw(img)
img.save('sample-out.png')
PImage("sample-out.png")

```

The Decision Tree consists of nodes which have test criterion from which the nodes make smaller subsets. In the below figure we can see the test criterion, the MSE score, the samples it has been divided to and the average price of that sample.

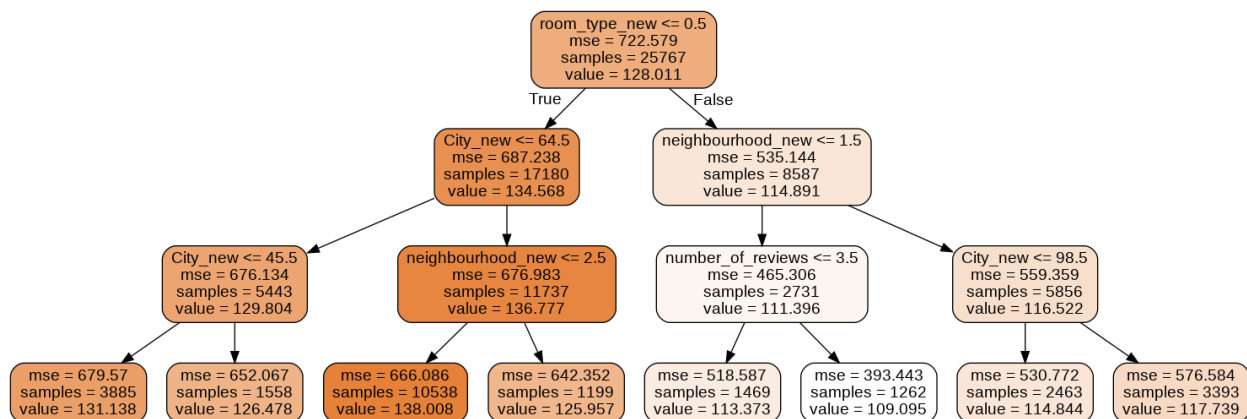


Fig. 1.24: Decision Tree for Price in Airbnb

In the figure, we can see different shades of color used for showing the different values of average prices. For a higher value it would be darker (i.e 135) and lower value it would be brighter(i.e 109). After we create a tree we now check R2 Score and RMSE Score of the model.

```

from sklearn.metrics import mean_squared_error,r2_score

rmse = (np.sqrt(mean_squared_error(y_test, predict)))
r2=r2_score(y_test,predict)

print("The model performance for testing set")
print("-----")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2*100))

```

The model performance for testing set

RMSE is 24.943854260635522

R2 score is 15.392469154546639

Fig. 1.25: Decision Tree R2 and Rmse score

1.6.4.3 Random Forest Tree

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. Random forest is a machine learning technique that is used for performing both regression and classification tasks using multiple decision trees and a statistical technique calling bagging. Random Forest is a technique in which multiple decision trees determine the final output rather than relying on individual decision trees. In this method we use the Mean Squared Error(MSE):

$$\text{MSE} = \frac{1}{N} \sum (f_i - y_i)^2 \quad (1.5)$$

Where N will be the number of the values in the data set, f_i is the value returned by the model and y_i is the actual value of the data point i .

Below we see the code used to compute the Random Forest Regressor. We take the same columns used in the linear regression to predict the value of 'Review Score Value'. We split the data into training and testing of the model.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
X = df[['Review Scores Accuracy', 'Review Scores Cleanliness', 'Review Scores Communicat
Y = df[ 'Review Scores Value']
X = preprocessing.normalize(X)
X = np.hstack((np.ones( (len(df['Review Scores Accuracy']) ,1)), X))
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
regressor = RandomForestRegressor(max_depth=8, n_estimators = 100, random_state = 0)
regressor.fit(X_train, Y_train)
y_pred = regressor.predict(X_test)
print("*****")
print("\t\t\t\t\tRandom Forest Regressor")
print("R Score: ", r2_score(Y_test, y_pred))
print("Rmse :", mean_squared_error(Y_test, y_pred))
```

After compiling the above code we get the output as:



R Score: 0.9837930433956219
 Rmse : 0.3569946250469154

Fig. 1.26: Random Forest Regressor R2 and Rmse score

In this model the R2 Score came 98 and RMSE score of 0.35.

1.6.4.4 Ridge Regression

Ridge regression is a way to create a model when the number of predictor variables in a set exceeds the number of observations. This is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values. The cost function of ridge regression is :

$$\sum (y_i - \bar{y}_i)^2 = \sum (y_i - \sum w_j * x_{ij})^2 + \lambda \sum w_j^2 \quad (1.6)$$

We perform ridge regression to predict the value of 'Review Scores Value' using the independent columns accuracy, communication, cleanliness, check-in and location.

```
from sklearn.linear_model import Ridge
import mglearn
X = df[['Review Scores Accuracy', 'Review Scores Cleanliness', 'Review Scores Communicat
'Review Scores Checkin', 'Review Scores Location']]
Y = df[ 'Review Scores Value']
X = preprocessing.normalize(X)
X = np.hstack((np.ones( (len(df['Review Scores Accuracy']) ),1)), X))

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
ridge = Ridge(alpha=1.0).fit(X_train, Y_train)
print("\n\nridge.coef_: {}".format(ridge.coef_))
print("ridge.intercept_: {}".format(ridge.intercept_))
print("Training set score:{:.2f}".format(ridge.score(X_train, Y_train)))
print("Test set score: {:.2f}".format(ridge.score(X_test, Y_test)))
```

As you can see in the code we use a test size of 0.20 of the whole data set and split the it into training and testing data. We also take the alpha value as 1. After compiling this code we get the output as:


```

ridge.coef_: [ 0.          3.97998228  5.47441147 -2.02100366  6.85003753  6.8834813 ]
ridge.intercept_: -0.006698983546624682
Training set score:0.98
Test set score: 0.98

```

Fig. 1.27: Ridge Regressor score

As you can see in the output image the training and testing score of ridge regression model is 0.98.

1.6.4.5 Lasso Regression

In statistics and machine learning, lasso (least absolute shrinkage and selection operator) is a regression method that predicts the a value using variable selections in order to give better accuracy. Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point.Lasso Regression uses L1 regularization technique.

$$\sum (y_i - \bar{y}_i)^2 = \sum (y_i - \sum w_j * x_{ij})^2 + \lambda \sum |w_j| \quad (1.7)$$

It is used when we have more number of features because it automatically performs feature selection. So we use all the 5 columns of accuracy, communication, cleanliness, check-in and location to predict the Review Scores Value

```

from sklearn.linear_model import Lasso
X = df[['Review Scores Accuracy','Review Scores Cleanliness','Review Scores Communication','Review Scores Location','Review Scores Value']]
Y = df['Review Scores Value']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
lasso = Lasso().fit(X_train, Y_train)

print("lasso.coef_: {}".format(lasso.coef_))
print("lasso.intercept_: {}".format(lasso.intercept_))
print("Training set score:{:.2f}".format(lasso.score(X_train, Y_train)))
print("Test set score: {:.2f}".format(lasso.score(X_test,Y_test)))
print("Number of features used: {}".format(np.sum(lasso.coef_!= 0)))

```

After performing the above code we get the output as: put as:

```

lasso.coef_: [0.21536533 0.04752254 0.06218553 0.35673392 0.25254788]
lasso.intercept_: 0.18182123714914855
Training set score:0.99
Test set score: 0.99
Number of features used: 5

```

Fig. 1.28: Lasso score

1.6.5 Machine Learning Algorithms

(Write something on ML Algos)

1.6.5.1 Naive Bayes

Naive Bayes is an algorithm technique in machine learning which is used for classification and prediction. It is based on Bayes Theorem which is :

$$P(A|B) = \frac{P(A) * P(B|A)}{P(B)} \quad (1.8)$$

Gaussian Naive Bayes

We will use the Gaussian model of Naive Bayes to predict values.

$$P(xi|y) = \frac{1}{\sqrt{2 * \pi * \sigma^2 * y}} \exp\left(-\frac{(xi - \gamma y)^2}{2 * \sigma^2 * y}\right) \quad (1.9)$$

We will be using this model to predict Review Scores Value from the data set using columns such as accuracy, cleanliness, check in, communication and location. Below you can see the code for splitting the data for training and testing:

```
import numpy as np
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
X = df[['Review Scores Accuracy', 'Review Scores Cleanliness', 'Review Scores Communicat
, 'Review Scores Checkin', 'Review Scores Location']]
Y = df[ 'Review Scores Value']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_stat
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

The Output for the code is:

```
(53024, 5) (22725, 5)
(53024,) (22725,)
```

Fig. 1.29: Training and Testing size

Now that we know the testing and training size we can use it to fit the model to predict:

```

scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print(X_train.shape, X_test.shape)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

```

Now that we have fit the Gaussian Model we can start the prediction:

```

y_pred = gnb.predict(X_test)
y_pred

```

Some of the predicted values of Review Scores Value are :

```
array([0, 0, 9, ..., 0, 0, 0])
```

Fig. 1.30: Predicted Review score values

To compute the model score accuracy and the training - testing score :

```

from sklearn.metrics import accuracy_score
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))
print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))

```

The output after running this is:

```

Model accuracy score: 0.8563
Training set score: 0.8544
Test set score: 0.8563

```

Fig. 1.31: Naive Bayes score

We can see that the model score accuracy is 0.85 which is good score. The training and testing score is also 0.85.

1.6.5.2 KNN

K Nearest Neighbours Algorithm is an easy-to-implement supervised machine learning algorithm that can be as a classifier as well as a regressor. As we are testing for prediction models we will

be using KNN regression for prediction. In this we take a sample of 200 rows of the data for this model. We will be predicting 'Review Scores Value' by taken columns Review Scores Accuracy, Review Scores Cleanliness, and Review Scores Communication :

```

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
X = df[['Review Scores Accuracy','Review Scores Cleanliness','Review Scores Communicat
Y = df[ 'Review Scores Value'][:200]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_stat

test_scores = []
train_scores = []

for i in range(1,15):

    knn = KNeighborsClassifier(i)
    knn.fit(X_train,y_train)

    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(X_test,y_test))
max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score]
print('Max train score {} % and k = {}'.format(max_train_score*100,
list(map(lambda x: x+1,  train_scores_ind))))

```

After running the above code we get a training score of :

```
Max train score 87.85714285714286 % and k = [3]
```

Fig. 1.32: Training Score

```

max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100,
list(map(lambda x: x+1,  test_scores_ind))))

```

```
Max test score 85.0 % and k = [7, 8, 9, 10, 11, 12, 13, 14]
```

Fig. 1.33: Testing Score

In the above output we can see that we are getting the test score as 85 per cent and train score of 87 per cent.

1.6.5.3 K Means Clustering

K Means clustering algorithm is a way of grouping values which are closer to the nearest mean. K means is a unsupervised learning technique which works iteratively to assign each to one of the K groups present. In this section we will be clustering the latitude and longitude of this data set. Before we begin the process of clustering it is important to see what we will be clustering :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
from scipy.spatial import distance
from sklearn.cluster import KMeans

DF=df[['latitude','longitude']]
X=DF.iloc[:,[0,1]].values
len_X= len(X)
wcss = []

plt.scatter(df['latitude'],df['longitude'])
```

By running the above code we will get a scatter plot of the listed properties of New York based on their latitude and longitude.

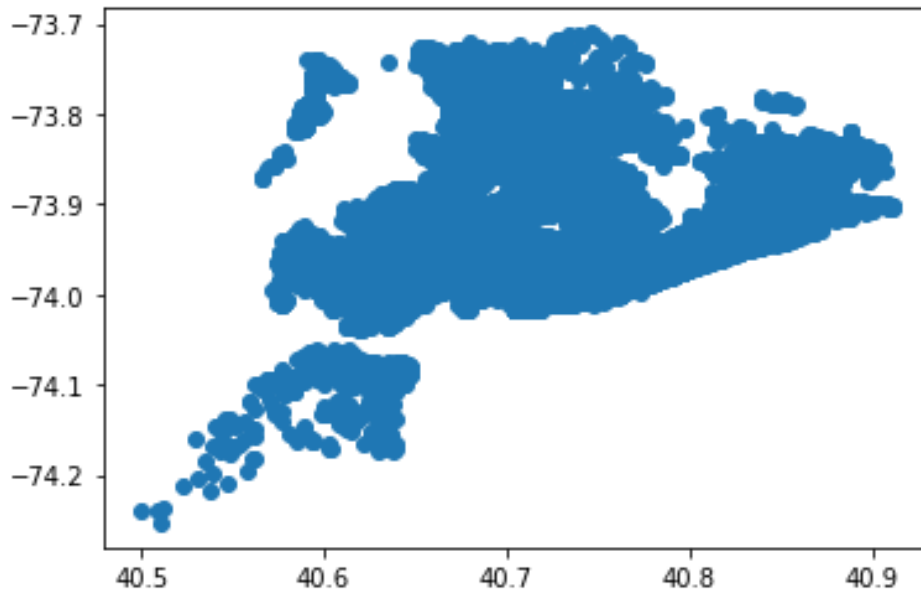


Fig. 1.34: Scatter plot for kmeans

Here we can see the scatter plot of what we are going to be clustering. The next step of the process is to find how many number of clusters are required for clustering. There are two methods by which you can find the number of clusters. The first method is the Elbow method :

```
for i in range(1,11):

kmeans = KMeans(n_clusters = i, init='k-means++',max_iter = 300 ,n_init =10)
kmeans.fit(X)
wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title("the Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("wcss")
plt.show()
```

After running this code we get the following output:

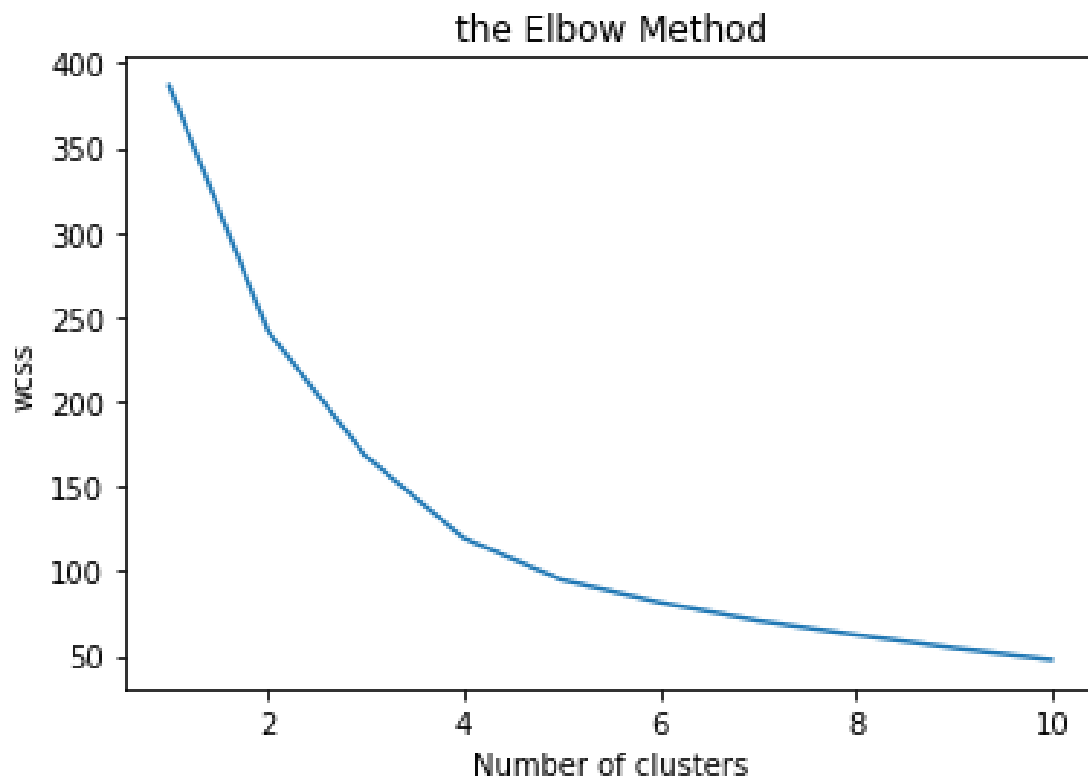


Fig. 1.35: Elbow Method

In the above graph we can observe that the final point of the elbow is on the number 5. So our k value is 5 and we will be clustering them in 5 clusters. We can reconfirm by performing the second method which is Bayesian information criterion(BIC) is performed below :

```
def Bic_func(kmeans,r=len_X):

centers = [kmeans.cluster_centers_]
labels = kmeans.labels_
m = kmeans.n_clusters
n = np.bincount(labels)
N, d = X.shape
cl_var = (1.0 / (N - m) / d) * sum([sum(distance.cdist
(X[np.where(labels == i)], [centers[0][i]], 'euclidean')**2) for i in range(m)])

const_term = 0.5 * m * np.log(N) * (d+1)
bic = np.sum([n[i] * np.log(n[i]) -n[i] * np.log(N) -((n[i] * d) / 2) *
np.log(2*np.pi*cl_var) -((n[i] - 1) * d/ 2) for i in range(m)]) - const_term

return bic
```

```

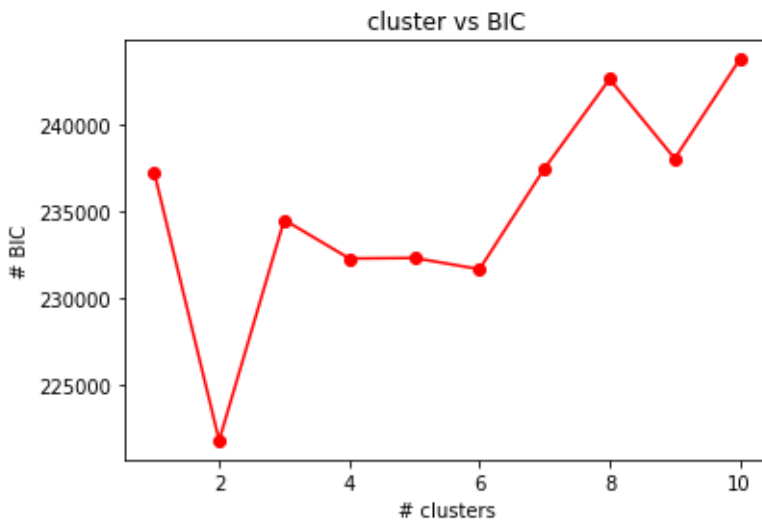
def best_k(BIC):
    temp= []
    for i in range(0,10):
        tem=BIC[i]/100
        temp.append(int(tem))
    for j in range(1,10):
        if(temp[j]==temp[j-1]):
            return j+1

BIC=[]
for i in range (1,11):
    kmeans = KMeans(n_clusters = i, init='k-means++',max_iter = 300 ,n_init =10)
    kmeans.fit(X)
    Bic=Bic_func(kmeans,r=len_X)
    BIC.append(Bic)
    print(BIC)
ks=range(1,11)
plt.plot(ks,BIC,'r-o')
plt.title("cluster vs BIC")
plt.xlabel("# clusters")
plt.ylabel("# BIC")
plt.show()
k=best_k(BIC)
print("Best K value = ",k)

```

After performing the above BIC function above we get the output:


```
[237289.10905541526, 221850.63257082034, 234527.71055305484, 232309.15843670134,
```



```
Best K value = 5
```

Fig. 1.36: BIC Function

Here from the above output we can confirm that the k clusters required is 5.
Now we will be proceeding to clustering

```
kmeans= KMeans(n_clusters = 5 , init = 'k-means++',max_iter = 300 ,n_init=10)
y_kmeans = kmeans.fit_predict(X)
fig=plt.figure(figsize=(12,8))
ax=plt
ax.scatter(X[y_kmeans == 0 ,0], X[y_kmeans == 0,1],s =20, c = 'red',label = 'Cluster 1')

ax.scatter(X[y_kmeans == 1 ,0], X[y_kmeans == 1,1] ,s =20, c = 'green',label = 'Cluster 2')
ax.scatter(X[y_kmeans == 2 ,0], X[y_kmeans == 2,1] ,s =20, c = 'yellow',label = 'Cluster 3')
ax.scatter(X[y_kmeans == 3 ,0], X[y_kmeans == 3,1] ,s =20, c = 'grey',label = 'Cluster 4')
ax.scatter(X[y_kmeans == 4 ,0], X[y_kmeans == 4,1] ,s =20, c = 'purple',label = 'Cluster 5')
ax.scatter(kmeans.cluster_centers_[0,0] , kmeans.cluster_centers_[0,1],s = 60,c='Black')
plt.title('Clustering on AirBnB locations')

plt.legend()
plt.show()
```

From running the above code we get the following clustered output:

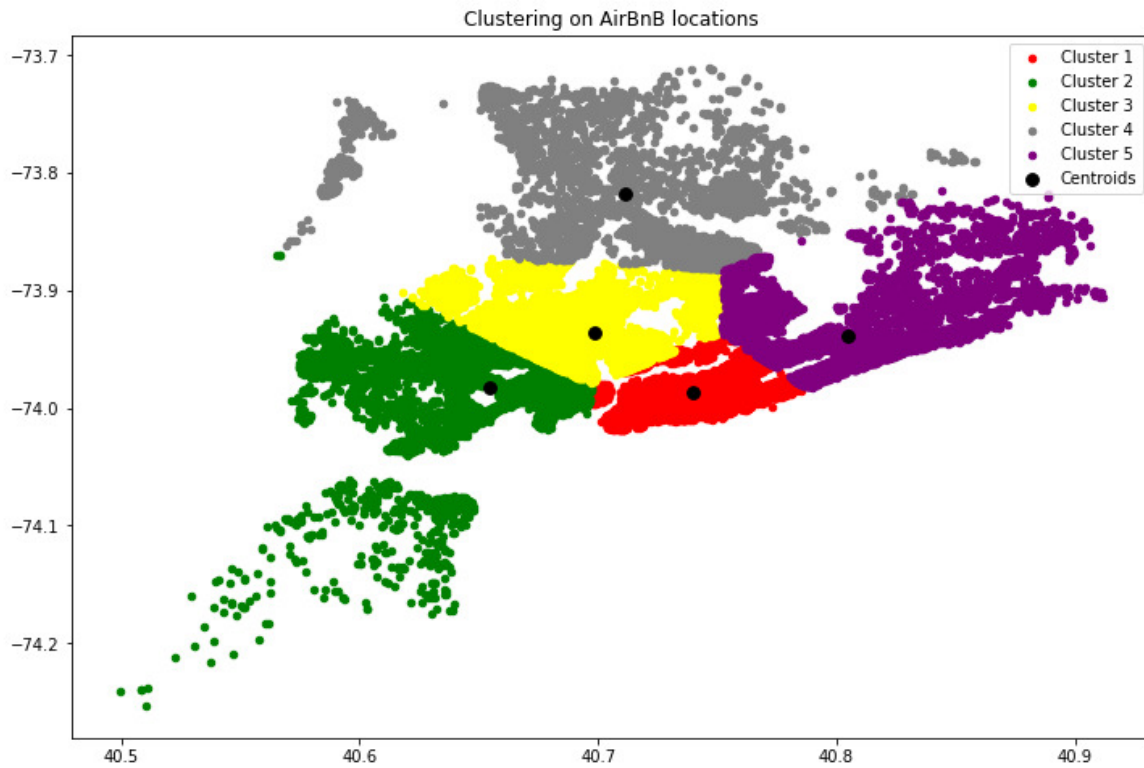


Fig. 1.37: Clustered K means output

From the above image we can see the clustered output of the latitude and longitude. Based on the density of the listings these clusters have been formed.

1.7 Advanced Visualization

Interesting details can be found by visualizing the data, details which would've been missed if the visualization was not done.

There are a multiple ways to visualize the data. We will be showing some of them.

1.7.1 Catplot

The catplot is a recent addition to seaborn. It is used to plot categorical variables easily. The code for catplot is given below:

```
sns.catplot(x='City', y='Number of reviews', data=df)
```

The following catplot is obtained when we check for the number of reviews city-wise:

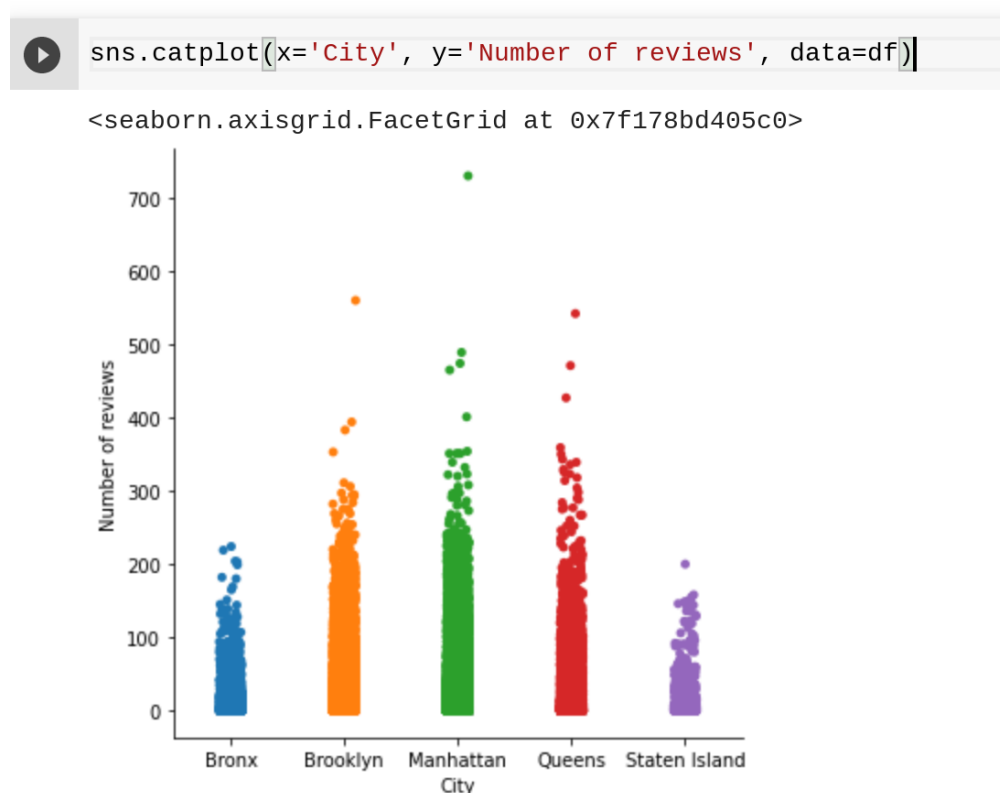


Fig. 1.38: Catplot showing number of reviews city-wise

The following catplot is obtained when we check for the number of reviews neighbourhood-wise:

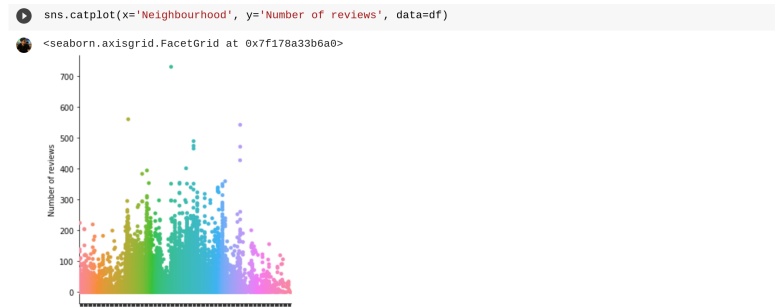


Fig. 1.39: Catplot showing number of reviews neighbourhood-wise

1.7.2 Box Plot

A boxplot is used to see how the values in the data are spread out. It shows the data in quartiles. It has 5 quartiles, namely - minimum, first quartile, median, third quartile, and maximum. It is also used to identify the outliers in the data by showing the values below the minimum value and the values above the maximum value.

The code for it is simple, and is shown below along with the output - that shows how many number of days in a year the property is available ,city-wise

```
[ ] plt.figure(figsize=(6,6))  
    ax = sns.boxplot(data=df, x='City', y='Availability 365', palette='plasma')
```

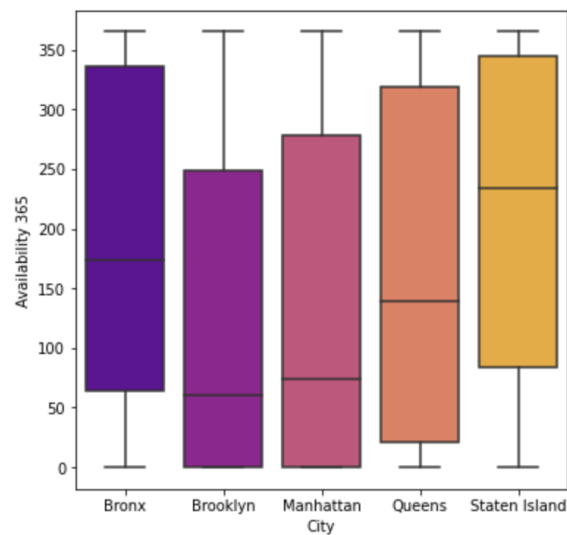


Fig. 1.40: Boxplot showing Availability 365, city-wise

1.7.3 Count Plot

A countplot is used for calculating the count in categorical variables.

The code for it is simple and is shown below:

```
sns.countplot(df['City'], palette="plasma")  
fig = plt.gcf()  
fig.set_size_inches(6,6)  
plt.title('City')
```

For which the get the following output:

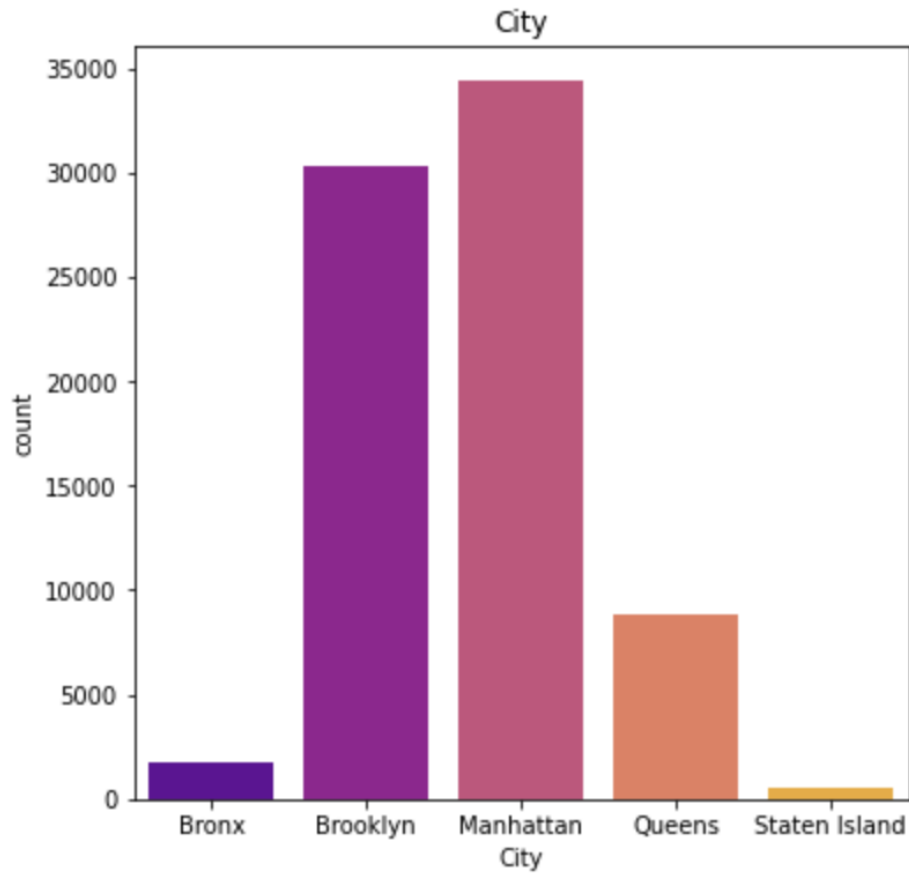
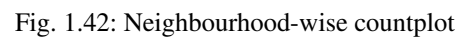


Fig. 1.41: City-wise countplot

From the above output, we conclude that Manhattan City has the most number of properties listed under AirBnB in New York

Reason why Staten island has comparatively lesser listings:
Staten Island is one of the most expensive places to stay in the U.S.A, hence there are very less properties listed under it.



In the above plot, we have taken a sample of the neighbourhoods' and counted them



From the above countplot we conclude that the most number of properties listed under Airbnb in New York are Entire homes and the least in number are the hotel rooms

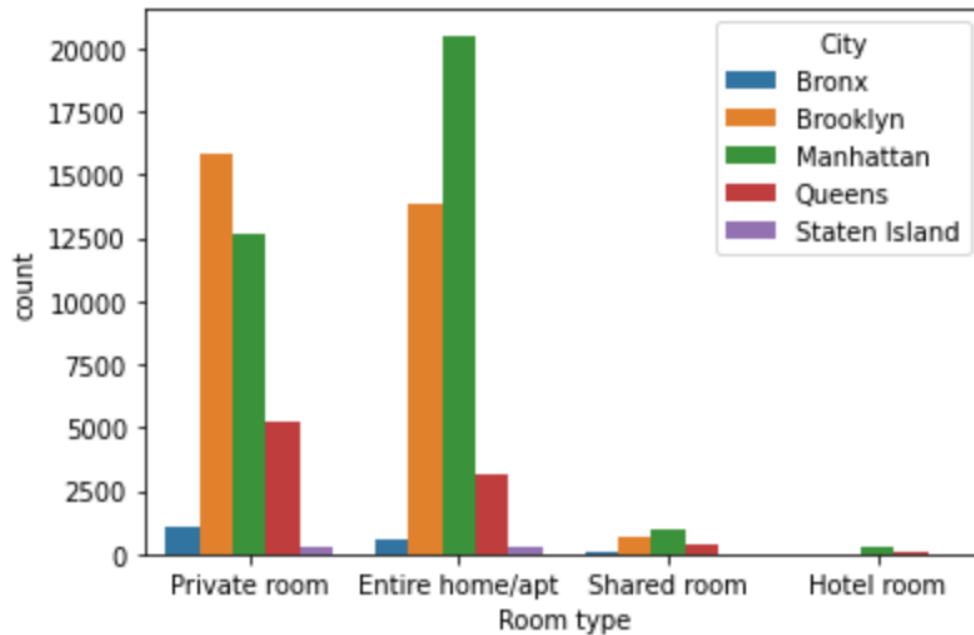


Fig. 1.44: Countplot showing the room-type of the properties in the cities

From the above countplot, we conclude that :

1. In private rooms, Brooklyn has the most private rooms and Staten Island has the least private rooms.
2. In entire home/apt , Manhattan has the highest and Staten Island has the lowest
3. In Shared rooms, Manhattan has the most and Staten island has the least
4. In hotel rooms, Manhattan has the most

Reason why Staten island has comparatively lesser listings:
Staten Island is one of the most expensive places to stay in the U.S.A, hence there are very less properties listed under it.

1.7.4 Scatter Plot

Scatter plots are used to plot data points on a horizontal and a vertical axis in the attempt to show how much one variable is affected by another

A scatter plot that we have plotted for our dataset

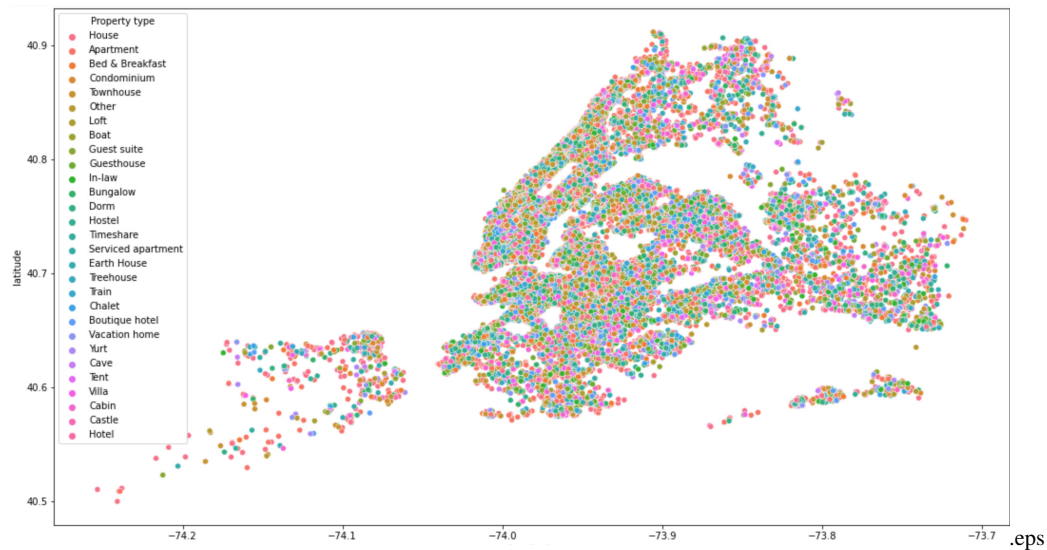


Fig. 1.45: Scatter plot showing the various property types plotted with the latitude and longitude

1.7.5 Visualization using Mapbox

Using Mapbox, we have created a menu driven map which will display the type of property that we want to view. For this to happen, an account had to be made in mapbox, and a token was issued. We used this token in order to do this and hence apply it for our data.

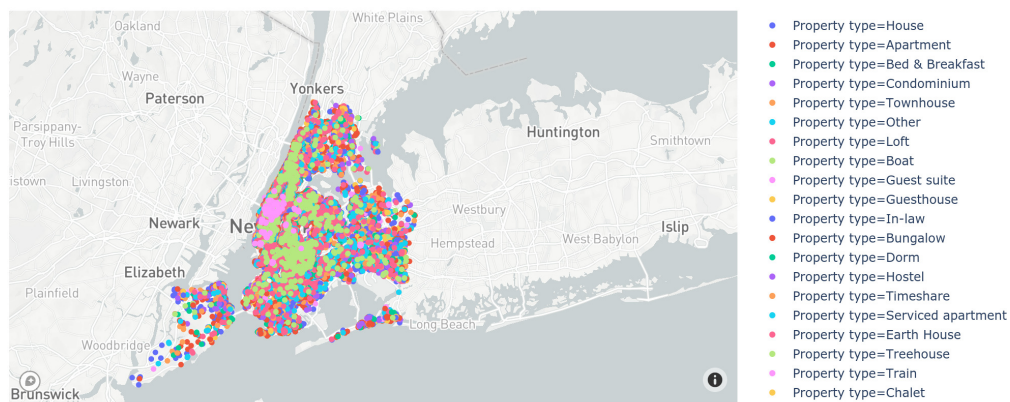


Fig. 1.46: Scatter plot showing the various property types plotted with the latitude and longitude

1.8 Summary

In this document we had done analysis on the Airbnb ratings data set which consisted of 2 different types of files. The first type was on the listings and their ratings which had 3 different files based on the location (New York, Los Angeles and other cities). The second file type was based on reviews put up on the online platform by different people sharing their experience on these rentals. We had selected the file of New York listings for further analysis.

The first and most essential step before working on any data is the data preprocessing procedure. It is important to understand the data. To do data preprocessing we must check on the columns with their data types, find the null or missing values present in the data set and deal with the outliers.

The basic summary statistics are essential to know for analysis of the data and for many of the machine learning algorithms. Mean, median, mode ,variance and histograms are some the basic summary statistics which give an idea of how the data is for further computations. Like we have seen that the average prices of Airbnb properties in the New York are 154 dollars.

The distribution type of a data defines how the data has been distributed in a column. There are many types of distribution of data. In this document we had done normality tests using KS test and QQ plot. KS test is an hypothesis test which decides if the distribution is normal by taking the p value. QQ plot is a method of plotting a column to see if its normally distributed. We had computed these tests on the availability of the listings.

We had performed a hypothesis z test saying that the average prices in Manhattan were greater than the average prices in the whole state of New York. In the regression analysis we had done prediction on the review scores value by taking the the other review score ratings. We used different machine learning algorithms such as KNN algorithm and Naive Bayes algorithm for prediction. We have also used K means clustering to cluster on the basis of latitude and longitude with showed us the density of Airbnb listings throughout New York.

In the Advanced Visualization section we have use different methods to visualize data. For categorical data such as Cities and Neighbourhood we had used count plots and cat plots. We has used scatter plots and map boxes to show different types of properties and room types throughout New York.

1.9 Bibliography

Sites from which data was extracted:

https : //www.kaggle.com (1.10)

https : //www.towardsdatascience.com (1.11)

https : //www.google.com (1.12)

