# AirBnb Price Prediction

```
#Importing Necessary Library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from sklearn.preprocessing import OneHotEncoder
import plotly.express as px
from sklearn.model_selection import StratifiedKFold
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

import array
import warnings
from pylab import rcParams
from scipy.stats import f_oneway
from scipy.stats import ttest_ind
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn import tree
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import r2_score
from sklearn import metrics
from sklearn.pipeline import Pipeline
```

```
#Mount the Google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Reading the dataset
df = pd.read_csv("/content/drive/MyDrive/Emperical/AirBNB_2019.csv")
df.head()
```

```
#Understanding data types
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
```

```
#Statistical Description of dataset
df.describe()
```

| | id | host_id | latitude | longitude | price | minimum_nights | number_of_reviews | reviews_per_month | calcula |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 4.889500e+04 | 4.889500e+04 | 48895.000000 | 48895.000000 | 48895.000000 | 48895.000000 | 48895.000000 | 38843.000000 | |
| **mean** | 1.901714e+07 | 6.762001e+07 | 40.728949 | -73.952170 | 152.720687 | 7.029962 | 23.274466 | 1.373221 | |
| **std** | 1.098311e+07 | 7.861097e+07 | 0.054530 | 0.046157 | 240.154170 | 20.510550 | 44.550582 | 1.680442 | |
| **min** | 2.539000e+03 | 2.438000e+03 | 40.499790 | -74.244420 | 0.000000 | 1.000000 | 0.000000 | 0.010000 | |
| **25%** | 9.471945e+06 | 7.822033e+06 | 40.690100 | -73.983070 | 69.000000 | 1.000000 | 1.000000 | 0.190000 | |
| **50%** | 1.967728e+07 | 3.079382e+07 | 40.723070 | -73.955680 | 106.000000 | 3.000000 | 5.000000 | 0.720000 | |
| **75%** | 2.915218e+07 | 1.074344e+08 | 40.763115 | -73.936275 | 175.000000 | 5.000000 | 24.000000 | 2.020000 | |
| **max** | 3.648724e+07 | 2.743213e+08 | 40.913060 | -73.712990 | 10000.000000 | 1250.000000 | 629.000000 | 58.500000 | |

```
#Checking the Missing/Null Values
df.isnull().sum()
```

```
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
```

```
#Checking the Unique value of dataset
df.nunique()
```

```
id                                48895
name                              47905
host_id                           37457
host_name                         11452
neighbourhood_group                   5
neighbourhood                       221
latitude                          19048
longitude                         14718
```

```
room_type                           3
price                             674
minimum_nights                    109
number_of_reviews                 394
last_review                      1764
reviews_per_month                 937
calculated_host_listings_count     47
availability_365                  366
dtype: int64
```

```
#Checking the duplicate value in dataset
df.duplicated().sum()
```

```
0
```

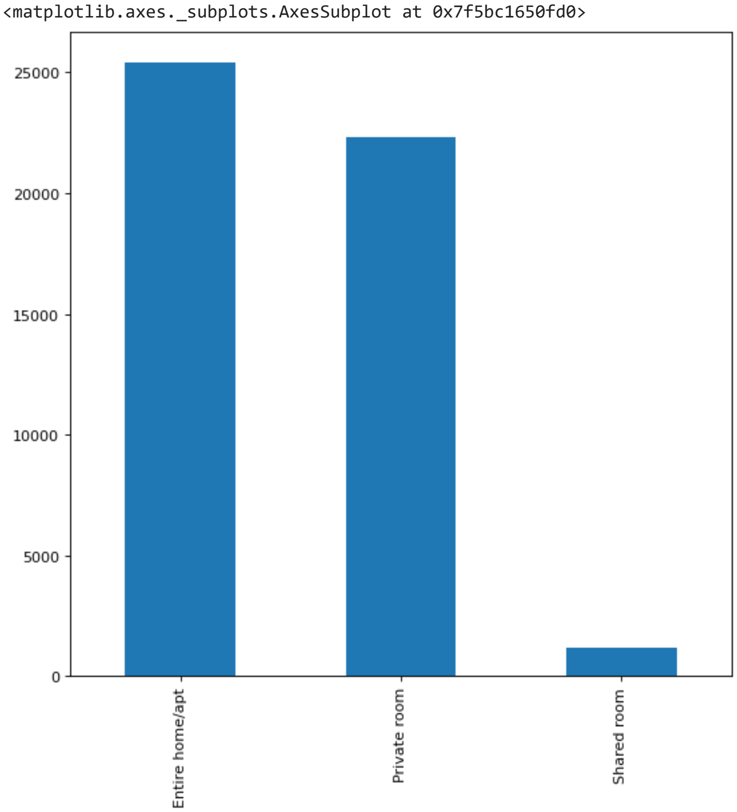
## Exploratory Data Analysis

```
# EDA for the Room types
df['room_type'].value_counts()
```

```
Entire home/apt    25409
Private room       22326
Shared room         1160
Name: room_type, dtype: int64
```

```
#Plot for the room_type
fig = plt.figure(figsize=(8,8), dpi=80)
df['room_type'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc1650fd0>
```



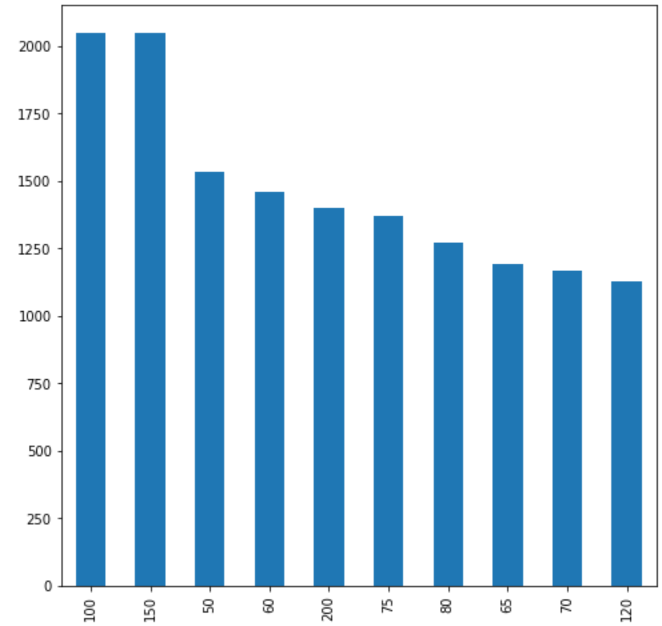

```
#EDA for Price
df.price.value_counts().iloc[:10]
```

```
100    2051
150    2047
50     1534
60     1458
200    1401
75     1370
80     1272
65     1190
70     1170
```

```
120    1130
Name: price, dtype: int64
```

```
#Visualization for price
df.price.value_counts().iloc[:10].plot(figsize=(8,8), kind = 'bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc159c820>
```




```
##Statistical Description of price
df.price.describe()
```

```
count    48895.000000
mean       152.720687
std        240.154170
min          0.000000
25%         69.000000
50%        106.000000
75%        175.000000
max      10000.000000
Name: price, dtype: float64
```

```
#Checking the dataset with high price
df[df['price'] ==10000]
```

| | id | name | host_id | host_name | neighbourhood_group | neighbourhood | latitude | longitude | room_type | price | minimum_night |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **9151** | 7003697 | Furnished room in Astoria apartment | 20582832 | Kathrine | Queens | Astoria | 40.76810 | -73.91651 | Private room | 10000 | 1( |
| **17692** | 13894339 | Luxury 1 bedroom apt. - stunning Manhattan views | 5143901 | Erin | Brooklyn | Greenpoint | 40.73260 | -73.95739 | Entire home/apt | 10000 | |
| **29238** | 22436899 | 1-BR Lincoln Center | 72390391 | Jelena | Manhattan | Upper West Side | 40.77213 | -73.98665 | Entire home/apt | 10000 | 3 |

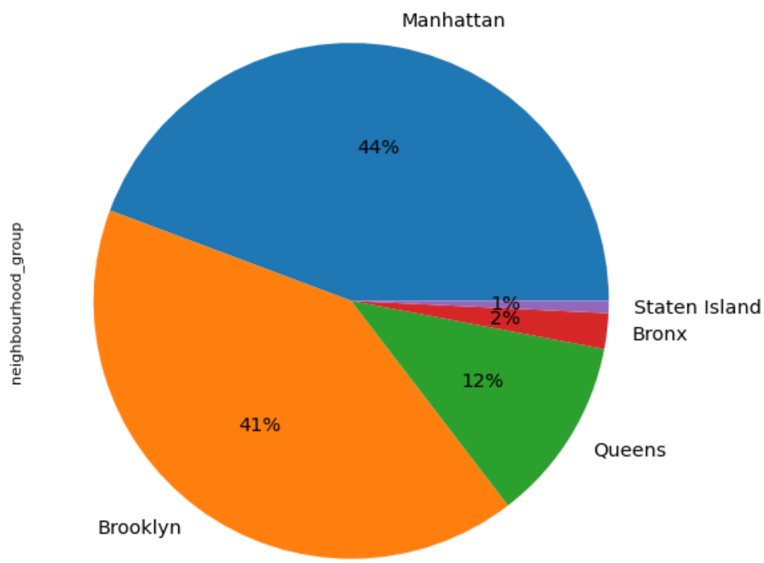
```
# EDA for neighbourhood_group
df['neighbourhood_group'].value_counts()
```

```
Manhattan        21661
Brooklyn         20104
Queens            5666
Bronx             1091
```

```
Staten Island      373
Name: neighbourhood_group, dtype: int64
```

```
#Visualization for neighbourhood_group
fig = plt.figure(figsize=(8,8), dpi=80)
df['neighbourhood_group'].value_counts().plot(kind='pie',  autopct='%1.0f%%', startangle=360, fontsize=13)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc10dd310>
```




```
#EDA for neighbourhood
df['neighbourhood'].value_counts().iloc[:5]
```

```
Williamsburg          3920
Bedford-Stuyvesant    3714
Harlem                2658
Bushwick              2465
Upper West Side       1971
Name: neighbourhood, dtype: int64
```

```
#Checking the unique value of neighbourhood
df['neighbourhood'].unique()
```

```
array(['Kensington', 'Midtown', 'Harlem', 'Clinton Hill', 'East Harlem',
       'Murray Hill', 'Bedford-Stuyvesant', "Hell's Kitchen",
       'Upper West Side', 'Chinatown', 'South Slope', 'West Village',
       'Williamsburg', 'Fort Greene', 'Chelsea', 'Crown Heights',
       'Park Slope', 'Windsor Terrace', 'Inwood', 'East Village',
       'Greenpoint', 'Bushwick', 'Flatbush', 'Lower East Side',
       'Prospect-Lefferts Gardens', 'Long Island City', 'Kips Bay',
       'SoHo', 'Upper East Side', 'Prospect Heights',
       'Washington Heights', 'Woodside', 'Brooklyn Heights',
       'Carroll Gardens', 'Gowanus', 'Flatlands', 'Cobble Hill',
       'Flushing', 'Boerum Hill', 'Sunnyside', 'DUMBO', 'St. George',
       'Highbridge', 'Financial District', 'Ridgewood',
       'Morningside Heights', 'Jamaica', 'Middle Village', 'NoHo',
       'Ditmars Steinway', 'Flatiron District', 'Roosevelt Island',
       'Greenwich Village', 'Little Italy', 'East Flatbush',
       'Tompkinsville', 'Astoria', 'Clason Point', 'Eastchester',
       'Kingsbridge', 'Two Bridges', 'Queens Village', 'Rockaway Beach',
       'Forest Hills', 'Nolita', 'Woodlawn', 'University Heights',
       'Gravesend', 'Gramercy', 'Allerton', 'East New York',
       'Theater District', 'Concourse Village', 'Sheepshead Bay',
       'Emerson Hill', 'Fort Hamilton', 'Bensonhurst', 'Tribeca',
       'Shore Acres', 'Sunset Park', 'Concourse', 'Elmhurst',
       'Brighton Beach', 'Jackson Heights', 'Cypress Hills', 'St. Albans',
       'Arrochar', 'Rego Park', 'Wakefield', 'Clifton', 'Bay Ridge',
       'Graniteville', 'Spuyten Duyvil', 'Stapleton', 'Briarwood',
       'Ozone Park', 'Columbia St', 'Vinegar Hill', 'Mott Haven',
       'Longwood', 'Canarsie', 'Battery Park City', 'Civic Center',
       'East Elmhurst', 'New Springville', 'Morris Heights', 'Arverne',
       'Cambria Heights', 'Tottenville', 'Mariners Harbor', 'Concord',
       'Borough Park', 'Bayside', 'Downtown Brooklyn', 'Port Morris',
```

```
       'Fieldston', 'Kew Gardens', 'Midwood', 'College Point',
       'Mount Eden', 'City Island', 'Glendale', 'Port Richmond',
       'Red Hook', 'Richmond Hill', 'Bellerose', 'Maspeth',
       'Williamsbridge', 'Soundview', 'Woodhaven', 'Woodrow',
       'Co-op City', 'Stuyvesant Town', 'Parkchester', 'North Riverdale',
       'Dyker Heights', 'Bronxdale', 'Sea Gate', 'Riverdale',
       'Kew Gardens Hills', 'Bay Terrace', 'Norwood', 'Claremont Village',
       'Whitestone', 'Fordham', 'Bayswater', 'Navy Yard', 'Brownsville',
       'Eltingville', 'Fresh Meadows', 'Mount Hope', 'Lighthouse Hill',
       'Springfield Gardens', 'Howard Beach', 'Belle Harbor',
       'Jamaica Estates', 'Van Nest', 'Morris Park', 'West Brighton',
       'Far Rockaway', 'South Ozone Park', 'Tremont', 'Corona',
       'Great Kills', 'Manhattan Beach', 'Marble Hill', 'Dongan Hills',
       'Castleton Corners', 'East Morrisania', 'Hunts Point', 'Neponsit',
       'Pelham Bay', 'Randall Manor', 'Throgs Neck', 'Todt Hill',
       'West Farms', 'Silver Lake', 'Morrisania', 'Laurelton',
       'Grymes Hill', 'Holliswood', 'Pelham Gardens', 'Belmont',
       'Rosedale', 'Edgemere', 'New Brighton', 'Midland Beach',
       'Baychester', 'Melrose', 'Bergen Beach', 'Richmondtown',
       'Howland Hook', 'Schuylerville', 'Coney Island', 'New Dorp Beach',
       "Prince's Bay", 'South Beach', 'Bath Beach', 'Jamaica Hills',
       'Oakwood', 'Castle Hill', 'Hollis', 'Douglaston', 'Huguenot',
       'Olinville', 'Edenwald', 'Grant City', 'Westerleigh',
       'Bay Terrace, Staten Island', 'Westchester Square', 'Little Neck',
       'Fort Wadsworth', 'Rosebank', 'Unionport', 'Mill Basin',
       'Arden Heights', "Bull's Head", 'New Dorp', 'Rossville',
       'Breezy Point', 'Willowbrook'], dtype=object)
```

```
#Word Cloud for neighbourhood
plt.subplots(figsize=(10,10))
wordcloud = WordCloud(background_color='white', width=1920,height=1080).generate(" ".join(df.neighbourhood))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```


```
#EDA for reviews_per_month'
df[df['reviews_per_month'] > 1].reviews_per_month.value_counts().sum()
```

```
15908
```

```
#Getting the reviews_per_month which have greater than 1
df[df['reviews_per_month'] > 1]['reviews_per_month'].value_counts().iloc[:5]
```

```
2.00    406
3.00    222
4.00    130
1.15     90
1.05     88
Name: reviews_per_month, dtype: int64
```

```
#Getting the maximum value for reviews_per_month
df['reviews_per_month'].max()
```

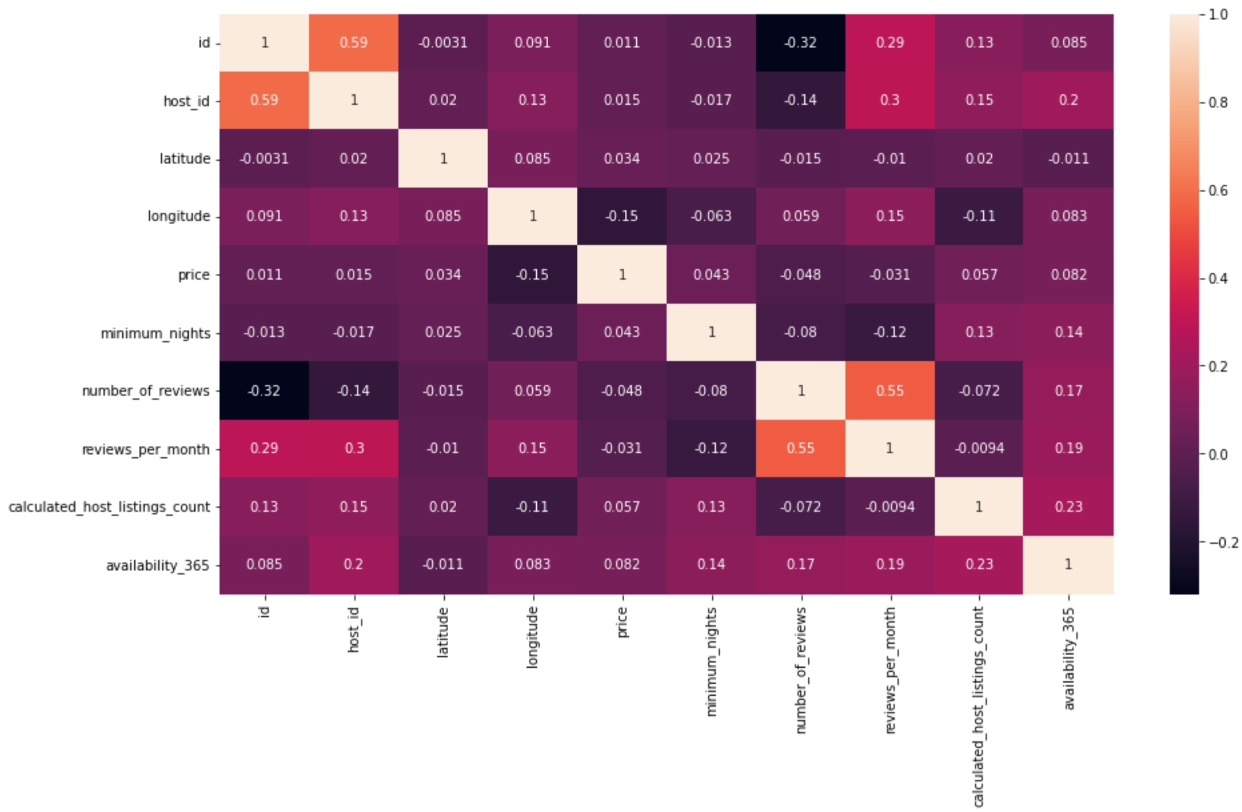
```
58.5
```

```
#checking the maximum reviews_per_month in dataset
df[df['reviews_per_month'] == 58.5]
```

| | id | name | host_id | host_name | neighbourhood_group | neighbourhood | latitude | longitude | room_type | price | minimum_nights |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **42075** | 32678719 | Enjoy great views of the City in our Deluxe Room! | 244361589 | Row NYC | Manhattan | Theater District | 40.75918 | -73.98801 | Private room | 100 | 1 |

```
#Correlation Between Continuous Variables
corr = df.corr()
plt.figure(figsize=(15,8))
sns.heatmap(corr, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc1632460>
```

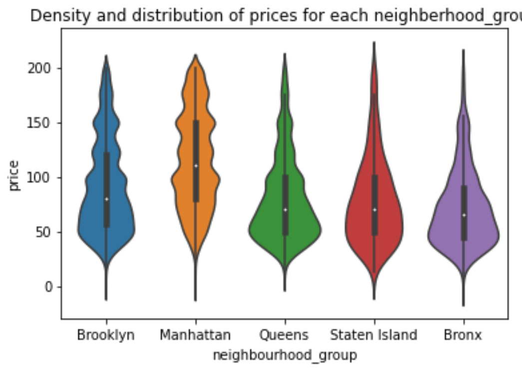



```
#Scatterplot for latitude with neighbourhood_group
plt.figure(figsize=(10,6))
sns.scatterplot(df.longitude,df.latitude,hue=df.neighbourhood_group)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. Fro
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc0528580>
```


```
#Visualization for Density of price for neighbourhood_group
v2=sns.violinplot(data=df[df.price < 200], x='neighbourhood_group', y='price')
v2.set_title('Density and distribution of prices for each neighberhood_group')
```

```
Text(0.5, 1.0, 'Density and distribution of prices for each neighberhood_group')
```

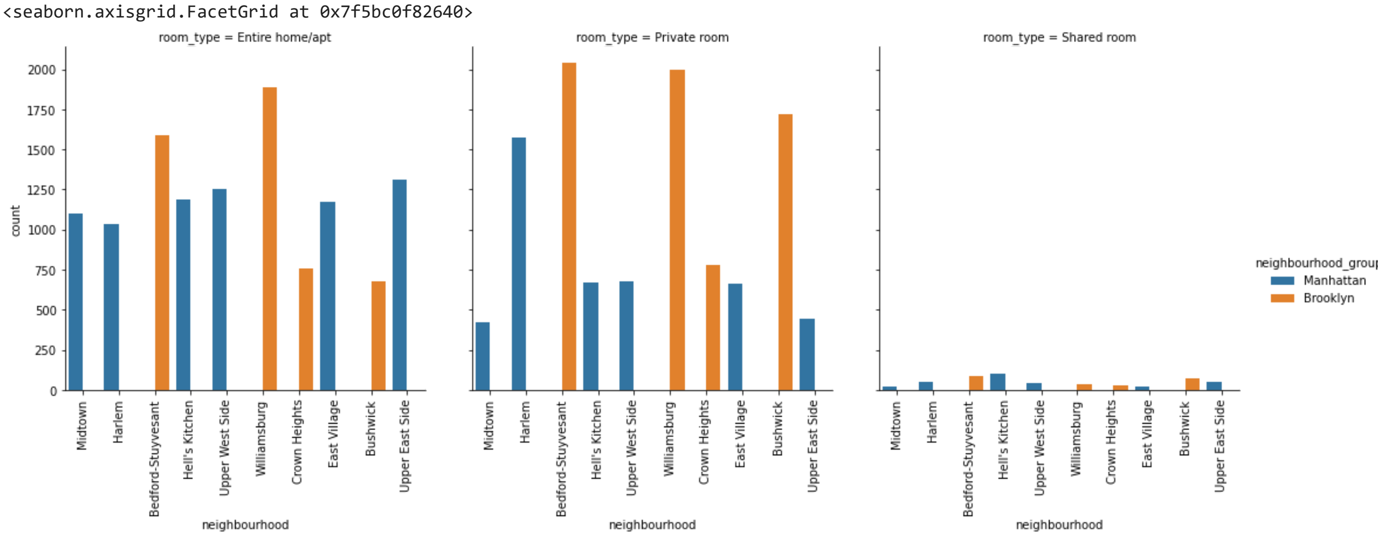



```
#Count for the neighbourhood
df['neighbourhood'].value_counts().iloc[:10]
```

```
Williamsburg          3920
Bedford-Stuyvesant    3714
Harlem                2658
Bushwick              2465
Upper West Side       1971
Hell's Kitchen        1958
East Village          1853
Upper East Side       1798
Crown Heights         1564
Midtown               1545
Name: neighbourhood, dtype: int64
```

```
#Visualization for subplot of Neighbourhood
Ngbr =df.loc[df['neighbourhood'].isin(['Williamsburg','Bedford-Stuyvesant','Harlem','Bushwick','Upper West Side','Hell\'s Kitchen','East Vill
pl =sns.catplot(x='neighbourhood', hue='neighbourhood_group', col='room_type', data=Ngbr, kind='count')
pl.set_xticklabels(rotation=90)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5bc0f82640>
```




## DATA CLEANING

```
#Unwanted columns such as id,name , host_name and last_review which is not useful for analysis so, they are removed.
df.drop(['host_id','name','latitude','longitude','id','host_name','last_review'], axis=1, inplace=True)
df.head()
```

| | neighbourhood_group | neighbourhood | room_type | price | minimum_nights | number_of_reviews | reviews_per_month | calculated_host_listings_ |
|---|---|---|---|---|---|---|---|---|
| 0 | Brooklyn | Kensington | Private room | 149 | 1 | 9 | 0.21 | |
| 1 | Manhattan | Midtown | Entire home/apt | 225 | 1 | 45 | 0.38 | |
| 2 | Manhattan | Harlem | Private room | 150 | 3 | 0 | NaN | |

```
#Droping the null values
df = df.dropna()
```

```
#Verify the Missing/Null Values after drop
df.isnull().sum()
```

```
neighbourhood_group               0
neighbourhood                     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
```

```
#Understanding data types without null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38843 entries, 0 to 48852
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   neighbourhood_group             38843 non-null  object
 1   neighbourhood                   38843 non-null  object
 2   room_type                       38843 non-null  object
 3   price                           38843 non-null  int64
 4   minimum_nights                  38843 non-null  int64
 5   number_of_reviews               38843 non-null  int64
 6   reviews_per_month               38843 non-null  float64
 7   calculated_host_listings_count  38843 non-null  int64
 8   availability_365                38843 non-null  int64
dtypes: float64(1), int64(5), object(3)
memory usage: 3.0+ MB
```

There are significant outliers in the "price" variable. In order to reduce the impact of outliers and increase the normalcy of the data, we will take the data between the 25th and 75th percentiles.

```
#Taking the dataset between the 25th and 75th percentiles to remove the outliers
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR=Q3-Q1
df = df[~((df['price']<(Q1-1.5*IQR))|(df['price']>(Q3+1.5-IQR)))]
```

```
#Understanding data types after removing the outliers
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10928 entries, 6 to 48852
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   neighbourhood_group             10928 non-null  object
 1   neighbourhood                   10928 non-null  object
 2   room_type                       10928 non-null  object
 3   price                           10928 non-null  int64
 4   minimum_nights                  10928 non-null  int64
 5   number_of_reviews               10928 non-null  int64
 6   reviews_per_month               10928 non-null  float64
```

```
 7   calculated_host_listings_count  10928 non-null  int64
 8   availability_365                10928 non-null  int64
dtypes: float64(1), int64(5), object(3)
memory usage: 853.8+ KB
```

## Statistical Analysis

```
#Shapiro Test (for checking Normality)
import scipy.stats as st
res = st.shapiro(df.price)
print("The Shapiro Test for the price is: ",res)
```

```
The Shapiro Test for the price is:  ShapiroResult(statistic=0.9598448872566223, pvalue=0.0)
/usr/local/lib/python3.8/dist-packages/scipy/stats/morestats.py:1760: UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

The null hypothesis is disproved because the p value (0.0) is less than alpha (5%, as assumed). The distribution is therefore abnormal (as cound be seen from the Distribution Plot above). Since the distribution is non-normal, the data can only theoretically be tested using non-parametric methods.

## Price vs Room Type

```
#Checking the unique value of the room_type
df.room_type.unique()
```

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

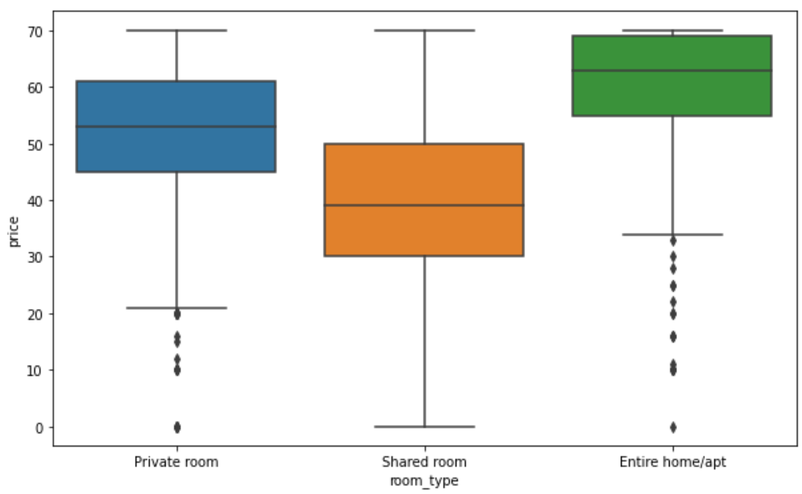
```
#Assigning the Unique values
pvt = df[df['room_type'] == 'Private room']
share = df[df['room_type'] == 'Shared room']
apt = df[df['room_type'] == 'Entire home/apt']
```

```
#Levene Test (for testing of variance)
res = st.levene(pvt.price, share.price, apt.price)
print(" The Levene Test for Price vs Room Typeis: \n ",res)
```

```
 The Levene Test for Price vs Room Typeis: 
  LeveneResult(statistic=49.601030751888054, pvalue=3.595499462501306e-22)
```

We reject hypothesis since the p value is almost zero (and consequently less than alpha = 0.05). As a result, there is a difference in the price variation across the various accommodation classifications. This is demonstrated by looking at the boxplot below.

```
#Visaulization of room_type according to price
plt.figure(figsize=(10,6))
sns.boxplot(y='price',x='room_type',data=df)
plt.show()
```



```
#Kruskal Wallis Test
res = st.kruskal(pvt.price,share.price,apt.price)
print(" The Kruskal Wallis Test for Price vs Room Type is: \n ",res)
```

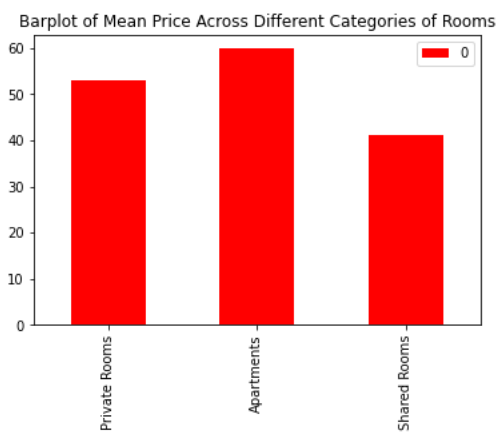
```
 The Kruskal Wallis Test for Price vs Room Type is:
  KruskalResult(statistic=743.0418084009106, pvalue=4.472201820222499e-162)
```

P value alpha is present in the test result is above the 0.05. As a result, the null hypothesis is disproved. This suggests that there is a difference in the mean price of various types of apartments. The barplot of the mean prices given below serves as confirmation.

```
ind = ['Private Rooms','Apartments','Shared Rooms']
Price_M = pd.DataFrame([pvt.price.mean(),apt.price.mean(),share.price.mean()], index=ind)
Price_M
```

| | 0 |
|---|---|
| **Private Rooms** | 52.878294 |
| **Apartments** | 59.867133 |
| **Shared Rooms** | 41.067797 |

```
#Visualization for Mean Price Across Different Categories of Rooms
Price_M.plot.bar(color='r')
plt.title("Barplot of Mean Price Across Different Categories of Rooms")
plt.show()
```




Conclusion: The two variables, price and room type, are related. Since the mean price for each type of room is different, the cost will vary depending on the choice of room.

Double-click (or enter) to edit

## One Way ANOVA

```
#Verifying the One way Anova
st.f_oneway(pvt.price,share.price,apt.price)
```

```
F_onewayResult(statistic=467.2078865807108, pvalue=2.0248725123817457e-195)
```

As a result, the null hypothesis is rejected in one direction of the ANOVA, indicating that the mean prices for the various types of rooms are not equal.

## Price vs Neighbourhood

The categorical variable of "neighbourhood group" has more than two categories, we can use either the Kruskal Wallis test or the One Way ANOVA test.

```
#Kruskal Wallis Test
r = df[df['neighbourhood_group'] == 'Brooklyn']['price']
s = df[df['neighbourhood_group'] == 'Manhattan']['price']
t = df[df['neighbourhood_group'] == 'Queens']['price']
u = df[df['neighbourhood_group'] == 'Staten Island']['price']
v = df[df['neighbourhood_group'] == 'Bronx']['price']

res = st.kruskal(r,s,t,u,v)
print(" The Kruskal Wallis Test for neighbourhood_group is: \n ",res)
```

```
 The Kruskal Wallis Test for neighbourhood_group is: 
  KruskalResult(statistic=321.98501542014543, pvalue=1.955857466328019e-68)
```

Since the P-value is close to 0. Hence, we say that there is a link between price and neighbourhood group, i.e. the price is based on the neighbourhood group that the house is offered in.

```
#One way Anova
res = st.f_oneway(r,s,t,u,v)
print(" The One way Anova for neighbourhood_group is: \n ",res)
```

```
 The One way Anova for neighbourhood_group is: 
  F_onewayResult(statistic=82.73884435819998, pvalue=2.5683471790581827e-69)
```

Hence, the P-value is less than 0 as before.So, we reject the hypothesis.

## Room Type vs Neighbourhood Group

```
#Chi Squared Test
ch = pd.crosstab(df['room_type'],df['neighbourhood_group'])
res = st.chi2_contingency(ch)
print(" The Chi Squared Test for neighbourhood_group and room_type is: \n ",res)
```

```
 The Chi Squared Test for neighbourhood_group and room_type is: 
  (125.17127733105427, 2.82913614107e-23, 8, array([[  32.97584187,  376.6704795 ,  147.40986457,  148.1295754 ,
           9.81423865],
       [ 441.09224012, 5038.42862372, 1971.78733529, 1981.41434846,
         131.27745242],
       [  29.93191801,  341.90089678,  133.80280015,  134.45607613,
           8.90830893]]))
```

Here, we get the P-value 2.899e-23 which is less than alpha 0.05 and hence, we reject the null hypothesis. Since, we can say that there is relationship between the neighbourhood_group and room_type and also shown as stacked bar plot.

```
#Stacked bar plot for neighbourhood_group and room_type
SG = pd.crosstab(df['room_type'],df['neighbourhood_group'])
SG.plot.bar(figsize=(10,6),stacked=True)
plt.show()
```


## Neighbourhood vs Neighbourhood Group

```
#Chi-Squared Test
CH = pd.crosstab(df['neighbourhood'],df['neighbourhood_group'])
st.chi2_contingency(CH)
```

```
       [2.30600293e-01, 2.63405930e+00, 1.03083821e+00, 1.03587116e+00,
        6.86310395e-02],
       [5.99560761e+00, 6.84855417e+01, 2.68017936e+01, 2.69326501e+01,
        1.78440703e+00],
       [6.64128843e+00, 7.58609078e+01, 2.96881406e+01, 2.98330893e+01,
        1.97657394e+00],
       [2.76720351e-01, 3.16087116e+00, 1.23700586e+00, 1.24304539e+00,
        8.23572474e-02],
       [4.15080527e-01, 4.74130673e+00, 1.85550878e+00, 1.86456808e+00,
        1.23535871e-01],
       [9.22401171e-02, 1.05362372e+00, 4.12335286e-01, 4.14348463e-01,
        2.74524158e-02],
       [1.01464129e+00, 1.15898609e+01, 4.53568814e+00, 4.55783309e+00,
        3.01976574e-01],
       [4.61200586e-02, 5.26811859e-01, 2.06167643e-01, 2.07174231e-01,
        1.37262079e-02],
       [4.15080527e-01, 4.74130673e+00, 1.85550878e+00, 1.86456808e+00,
        1.23535871e-01],
       [9.22401171e-02, 1.05362372e+00, 4.12335286e-01, 4.14348463e-01,
        2.74524158e-02],
       [2.76720351e-01, 3.16087116e+00, 1.23700586e+00, 1.24304539e+00,
        8.23572474e-02],
       [5.99560761e-01, 6.84855417e+00, 2.68017936e+00, 2.69326501e+00,
        1.78440703e-01],
       [4.01244510e+00, 4.58326318e+01, 1.79365849e+01, 1.80241581e+01,
        1.19418009e+00],
       [7.37920937e+00, 8.42898975e+01, 3.29868228e+01, 3.31478770e+01,
        2.19619327e+00],
       [2.76720351e-01, 3.16087116e+00, 1.23700586e+00, 1.24304539e+00,
        8.23572474e-02],
       [1.38360176e-01, 1.58043558e+00, 6.18502928e-01, 6.21522694e-01,
        4.11786237e-02],
       [1.19912152e+00, 1.36971083e+01, 5.36035871e+00, 5.38653001e+00,
        3.56881406e-01],
       [1.75717423e+01, 2.00715318e+02, 7.85498719e+01, 7.89333821e+01,
        5.22968521e+00],
       [4.61200586e-01, 5.26811859e+00, 2.06167643e+00, 2.07174231e+00,
        1.37262079e-01],
       [2.76720351e-01, 3.16087116e+00, 1.23700586e+00, 1.24304539e+00,
        8.23572474e-02],
       [2.76720351e-01, 3.16087116e+00, 1.23700586e+00, 1.24304539e+00,
        8.23572474e-02],
       [4.61200586e-02, 5.26811859e-01, 2.06167643e-01, 2.07174231e-01,
        1.37262079e-02],
       [2.30600293e-01, 2.63405930e+00, 1.03083821e+00, 1.03587116e+00,
        6.86310395e-02],
       [8.30161054e-01, 9.48261347e+00, 3.71101757e+00, 3.72913616e+00,
        2.47071742e-01],
       [3.42672035e+01, 3.91421212e+02, 1.53182559e+02, 1.53930454e+02,
        1.01985725e+01],
       [1.29136164e+00, 1.47507321e+01, 5.77269400e+00, 5.80087848e+00,
        3.84333821e-01],
       [2.58272328e+00, 2.95014641e+01, 1.15453880e+01, 1.16017570e+01,
        7.68667643e-01],
       [3.68960469e-01, 4.21449488e+00, 1.64934114e+00, 1.65739385e+00,
        1.09809663e-01],
       [4.47364568e+00, 5.11007504e+01, 1.99982613e+01, 2.00959004e+01,
        1.33144217e+00]]))
```

Here also, the null hypothesis is rejected as the P-value is 0 which is less than 0.05. Hence, there is relationship between Neighbourhood and Neighbourhood Group

## T-Test

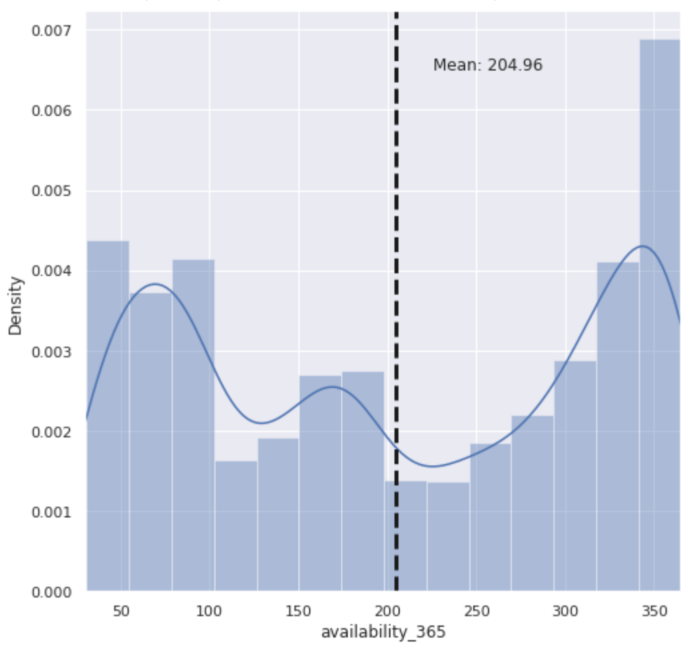
```
%matplotlib inline
warnings.filterwarnings("ignore")
```

```
rcParams['figure.figsize'] = 8,8
rcParams['font.size'] = 30
sns.set()
np.random.seed(8)

#Create a function for distrubtion plot
def plt_dstrb(inp):
    plt.figure()
    ax = sns.distplot(inp)
    ax.set_xlim([30, 365])
    plt.axvline(np.mean(inp), color="k", linestyle="dashed", linewidth=3)
    _, max_ = plt.ylim()
    plt.text(
        inp.mean() + inp.mean() / 10,
        max_ - max_ / 10,
        "Mean: {:.2f}".format(inp.mean()),
    )
    return plt.figure


#Getting data of airbnb's available over a month
df_H = df[['neighbourhood_group','availability_365']]
df_H = df_H[df_H['availability_365'] > 30]
df_H.head()
```

| | neighbourhood_group | availability_365 |
|---|---|---|
| **25** | Brooklyn | 85 |
| **28** | Manhattan | 311 |
| **30** | Manhattan | 355 |
| **31** | Manhattan | 255 |
| **32** | Brooklyn | 284 |

```
#Visaulization with mean
plt_dstrb(df_H['availability_365'])
```

```
<function matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class
'matplotlib.figure.Figure'>, clear=False, **kwargs)>
```




```
#Taking small sample size for t-test which would be invalid
sample_S = 2000
Sample_A = np.random.choice(df_H['availability_365'],sample_S)
Sample_A
```

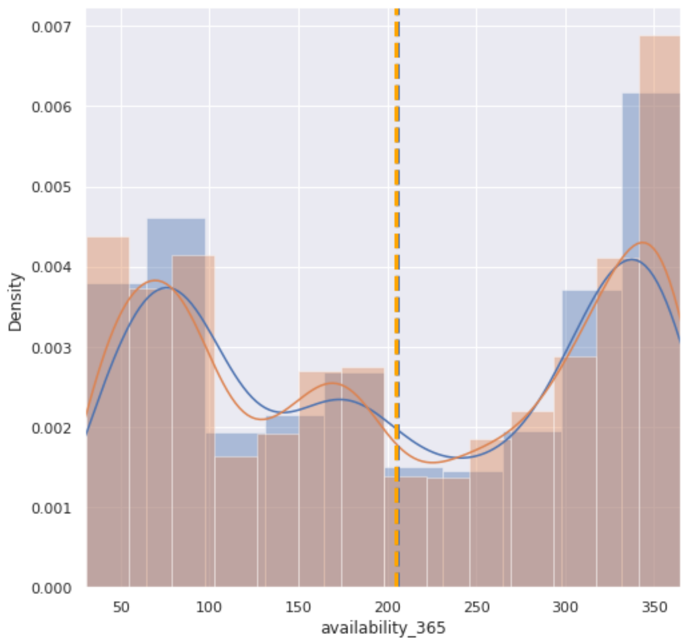
```
array([340, 247, 332, ...,  90, 310,  96])
```

```
#Mean of random value
np.mean(Sample_A)
```

```
205.6315
```

```
#plot and overlay to get the mean of each on graph
plt.figure()
ax1 = sns.distplot(Sample_A)
ax2 = sns.distplot(df_H['availability_365'])
ax1.set_xlim([30, 365])
ax2.set_xlim([30, 365])
plt.axvline(np.mean(Sample_A), color='b', linestyle='dashed', linewidth=3)
plt.axvline(np.mean(df_H['availability_365']), color='orange', linestyle='dashed', linewidth=3)
```

```
<matplotlib.lines.Line2D at 0x7f5bc0f82d60>
```




We have imported T-test using the Scipy library, which allows us to do t-tests and obtain p-values.

```
#T_test Function for comparing
def cmp_grp(arr_1, arr_2, alpha):
    stat, p = ttest_ind(arr_1, arr_2)
    print('Statistics=%.3f, p=%.3f' % (stat, p))
    if p > alpha:
        print('Same distributions hence we fail to reject H0(Null Hypothesis)')
    else:
        print('Different distributions hence we reject H0(Null Hypothesis)')

cmp_grp(df_H['availability_365'], Sample_A, 0.05)
```

```
Statistics=-0.227, p=0.821
Same distributions hence we fail to reject H0(Null Hypothesis)
```

We get the P-value 0.663 which is above the 0.05. Since, we fail to reject the Hypothesis and hence, we accept the null hypothesis.

## Encoding Data and Outlier removal.

```
#Encoding the categorical data
df.fillna({'reviews_per_month':0}, inplace=True)
#df.drop(['id','host_id','latitude','longitude','host_name','last_review','name'], axis = 1,inplace=True)
df = pd.get_dummies(df, columns=['neighbourhood_group',"room_type"], prefix = ['ng',"rt"],drop_first=True)
df.drop(["neighbourhood"], axis=1, inplace=True)
df.head()
```

| | price | minimum_nights | number_of_reviews | reviews_per_month | calculated_host_listings_count | availability_365 | ng_Brooklyn | ng_Manha |
|---|---|---|---|---|---|---|---|---|
| **6** | 60 | 45 | 49 | 0.40 | 1 | 0 | 1 | |
| **25** | 60 | 1 | 19 | 1.37 | 2 | 85 | 1 | |
| **28** | 44 | 3 | 108 | 1.11 | 3 | 311 | 0 | |
| **30** | 50 | 3 | 242 | 2.04 | 3 | 355 | 0 | |
| **31** | 52 | 2 | 88 | 1.42 | 1 | 255 | 0 | |

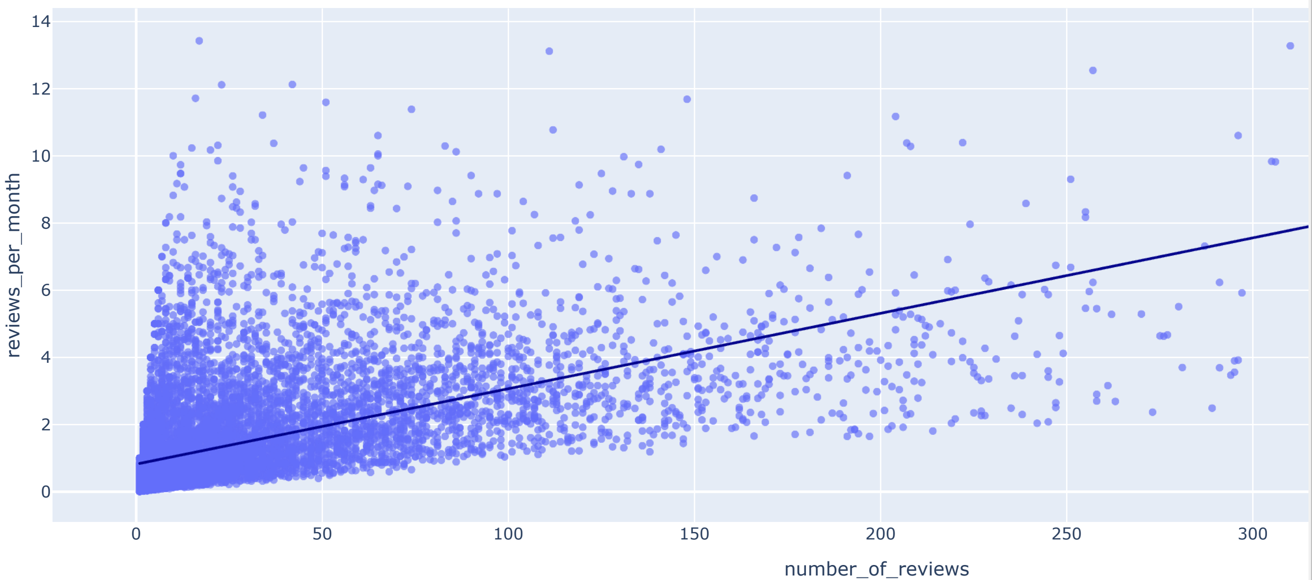
Linear Regression

```
def linear(a,b):
  global X,y,predictions,residue
  X = Reg_df[a].values.reshape(-1,1)
  y = Reg_df[b].values.reshape(-1,1)


  #3 Feature Scaling
  from sklearn.preprocessing import StandardScaler
  sc_X = StandardScaler()
  sc_y = StandardScaler()

  X = sc_X.fit_transform(X)
  y = sc_X.fit_transform(y)

  from sklearn.linear_model import LinearRegression
  reg = LinearRegression()
  reg.fit(X, y)

  print("The linear model is: Y = {:.2} + {:.2}X".format(reg.intercept_[0], reg.coef_[0][0]))

  print("Regression Intercept : ",reg.intercept_[0])


  predictions = reg.predict(X)
  rms = mean_squared_error(y, predictions, squared=False)

  fig = px.scatter(
    Reg_df, x=a, y=b, opacity=0.65,
    trendline='ols', trendline_color_override='darkblue')
  fig.show()

  # Plot the residuals after fitting a linear model
  sns.residplot(x=X, y=y, lowess=True, color="g")

  print("RMSE is: ", rms)
  x2 = sm.add_constant(X)

  est = sm.OLS(y, x2)
  #OLS is Ordinary Least Squares

  #est.TAB
  est2 = est.fit()
  print(est2.summary())

  residue = y - predictions


Reg_df = df[['number_of_reviews','reviews_per_month']]
Reg_df= Reg_df[Reg_df['number_of_reviews']<400]
Reg_df = Reg_df[Reg_df['reviews_per_month']<15]
Reg_df
```

| | number_of_reviews | reviews_per_month |
|---|---|---|
| **6** | 49 | 0.40 |
| **25** | 19 | 1.37 |
| **28** | 108 | 1.11 |
| **30** | 242 | 2.04 |
| **31** | 88 | 1.42 |
| **...** | ... | ... |
| **48534** | 1 | 1.00 |

```
linear('number_of_reviews','reviews_per_month')
df_n = pd.DataFrame(predictions, columns = ['Predictions'])
df_reg_new = pd.concat([Reg_df,df_n],axis = 1)
df_n = pd.DataFrame(residue, columns = ['Residue'])
df_reg_new = pd.concat([df_reg_new,df_n],axis = 1)
df_reg_new[['number_of_reviews','reviews_per_month','Predictions','Residue']].head()
```

```
Regression Intercept :  -4.172376831695239e-17
```




```
RMSE is:  0.8213420228839912
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.325
```

## Multiple Linear Regression

```
No. Observations:               10899   AIC:                         2.664e+04
```

```
#Dividing into Target value
df_train = df.drop(['price'],axis=1)
df_test = df['price'].values.reshape(-1,1)

df_train_Sc = StandardScaler()
df_test_Sc = StandardScaler()

df_train = df_train_Sc.fit_transform(df_train)
df_test = df_train_Sc.fit_transform(df_test)
```

```
Skew:                           2.278   Prob(JB):                         0.00
```

```
#Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_train, df_test, test_size=0.30, random_state=42)
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(7649, 11)
(7649, 1)
(3279, 11)
(3279, 1)
```

```
# instantiate
linreg = LinearRegression()

# fit the model to the training data (learn the coefficients)
linreg.fit(X_train, y_train)

# print the intercept and coefficients
print("intercept is: ",linreg.intercept_)

print("coefficients are: ",linreg.coef_)
```

```
intercept is:  [0.00334485]
coefficients are:  [[-0.04613508  0.00968174  0.03002607 -0.08917603  0.0708094   0.21510916
```

```
    0.31905844  0.13936999 -0.00727299 -0.21025422 -0.39427262]]
```

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

```
print("R-Square Value",r2_score(y_test,y_pred))
print("\n")
print ("mean_absolute_error :",metrics.mean_absolute_error(y_test, y_pred))
print("\n")
print ("mean_squared_error : ",metrics.mean_squared_error(y_test, y_pred))
print("\n")
print ("root_mean_squared_error : ",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R-Square Value 0.13935718762733007


mean_absolute_error : 0.7604323398842785


mean_squared_error :  0.8615707446682286


root_mean_squared_error :  0.9282083519707354
```

```
my_pipeline = Pipeline(steps=[('model', LinearRegression())])

from sklearn.model_selection import cross_val_score

# Multiply by -1 since sklearn calculates *negative* scores
scores1 =  1 * cross_val_score(my_pipeline, X, y,
                              cv=10,
                              scoring='r2')
scores2 = -1 * cross_val_score(my_pipeline, X, y,
                              cv=10,
                              scoring='neg_mean_absolute_error')
scores3 = -1 * cross_val_score(my_pipeline, X, y,
                              cv=10,
                              scoring='neg_root_mean_squared_error')

print("R squared scores:\n", scores1)
print("Average R squared score (across experiments):",scores1.mean())


print("RMSE scores:\n", scores3)
print("Average RMSE score (across experiments):",scores3.mean())
```

```
R squared scores:
 [-2.68983001  0.35656977  0.52437256  0.77114788  0.73031733  0.63570778
  0.46359052  0.20194511  0.00656353 -0.72625699]
Average R squared score (across experiments): 0.027412748488921234
RMSE scores:
 [1.21413378 0.5732473  0.48369005 0.42590084 0.50297081 0.58300332
 0.74908638 0.91220924 1.16354122 1.59120783]
Average RMSE score (across experiments): 0.8198990775340705
```

```
x2 = sm.add_constant(X)
est = sm.OLS(y, x2)
#OLS is Ordinary Least Squares
#est.TAB
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.325
Model:                            OLS   Adj. R-squared:                  0.325
Method:                 Least Squares   F-statistic:                     5256.
Date:                Sun, 04 Dec 2022   Prob (F-statistic):               0.00
Time:                        01:21:10   Log-Likelihood:                -13320.
No. Observations:               10899   AIC:                         2.664e+04
Df Residuals:                   10897   BIC:                         2.666e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const      -5.802e-17      0.008  -7.37e-15      1.000      -0.015       0.015
x1             0.5704      0.008     72.500      0.000       0.555       0.586
==============================================================================
Omnibus:                     5379.693   Durbin-Watson:                   1.198
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            38332.273
Skew:                           2.278   Prob(JB):                         0.00
Kurtosis:                      10.978   Cond. No.                         1.00
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

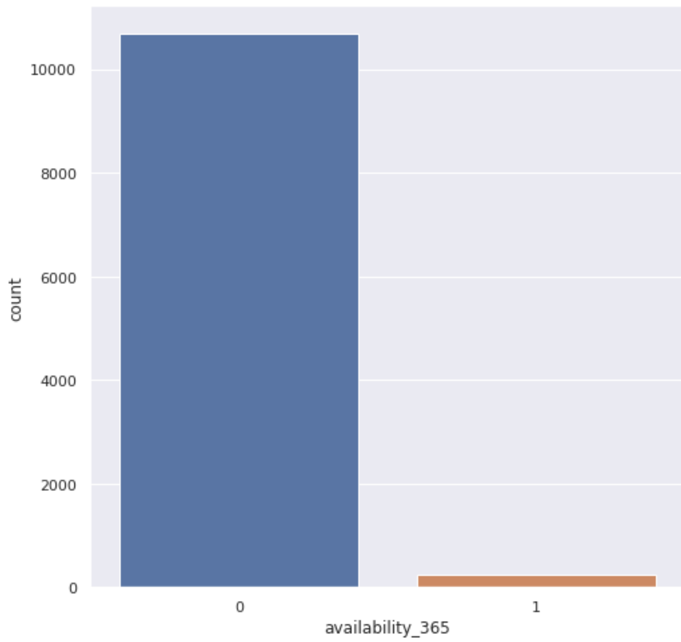
## Model Implementation for Classification

```
df_classif = df
```

```
df_classif['availability_365'] = df_classif['availability_365'].apply(lambda x: 1 if x == 365 else 0)
```

```
sns.countplot(df_classif['availability_365'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5be00cbf10>
```

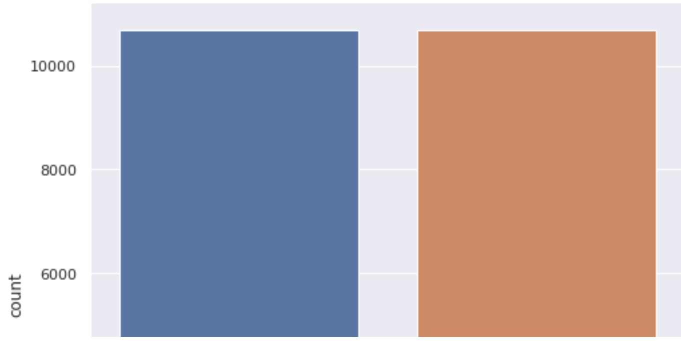



```
X = df_classif.drop(['availability_365'],axis=1)
y = df_classif['availability_365'].values
```

```
from imblearn.over_sampling import SMOTE
import imblearn
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

```
sns.countplot(y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bb96fc6a0>
```




```
len(y)
```

```
21370
```

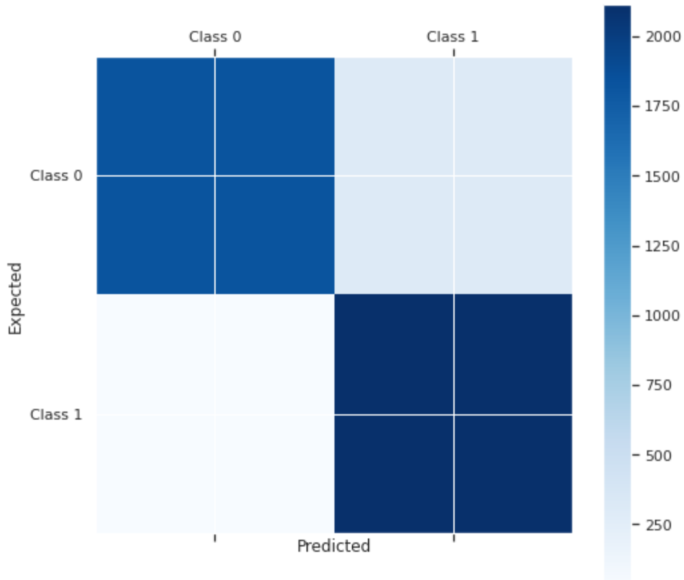
```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

# Used for classification of dataset.
def classif_results():
  conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
  print('Confusion matrix:\n', conf_mat)

  labels = ['Class 0', 'Class 1']
  fig = plt.figure()
  ax = fig.add_subplot(111)
  cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
  fig.colorbar(cax)
  ax.set_xticklabels([''] + labels)
  ax.set_yticklabels([''] + labels)
  plt.xlabel('Predicted')
  plt.ylabel('Expected')
  plt.show()

  print("Accuracy", metrics.accuracy_score(y_test, y_pred))

  from sklearn.metrics import classification_report
  print(classification_report(y_test, y_pred))


  auc = roc_auc_score(y_test, y_pred)
  print("AUC Score: ")
  print(auc)

  fpr, tpr, thresholds = roc_curve(y_test, y_pred)
  plot_roc_curve(fpr, tpr)
```

## K-Nearest Neighbour

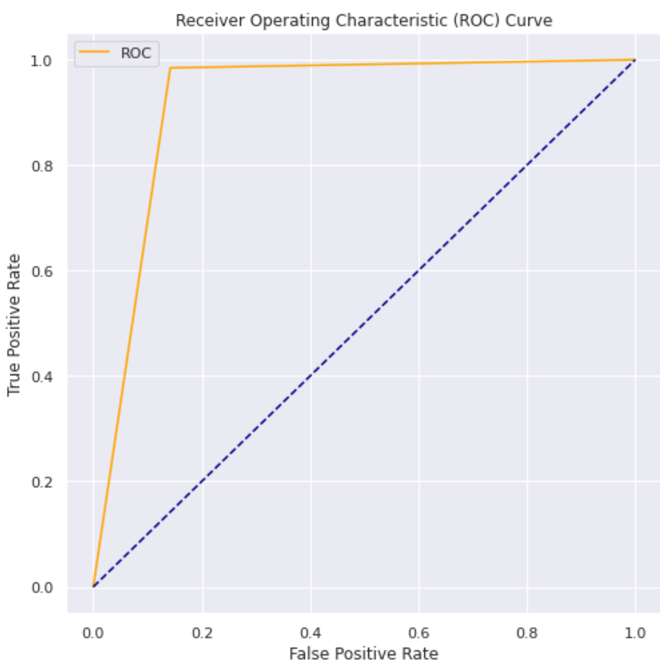
```
#Calling the KNN using Sklearn library
classifier = KNeighborsClassifier()

#Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

#Train the dataset
classifier.fit(X_train,y_train)

#Predict the value
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
```

```
classif_results()
```

```
Confusion matrix:
 [[1824  302]
 [  33 2115]]
```




```
Accuracy 0.9216190921853065
              precision    recall  f1-score   support

           0       0.98      0.86      0.92      2126
           1       0.88      0.98      0.93      2148

    accuracy                           0.92      4274
   macro avg       0.93      0.92      0.92      4274
weighted avg       0.93      0.92      0.92      4274

AUC Score:
0.9212930359423367
```




```
error = []
# Calculating error for K values between 1 and 30
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

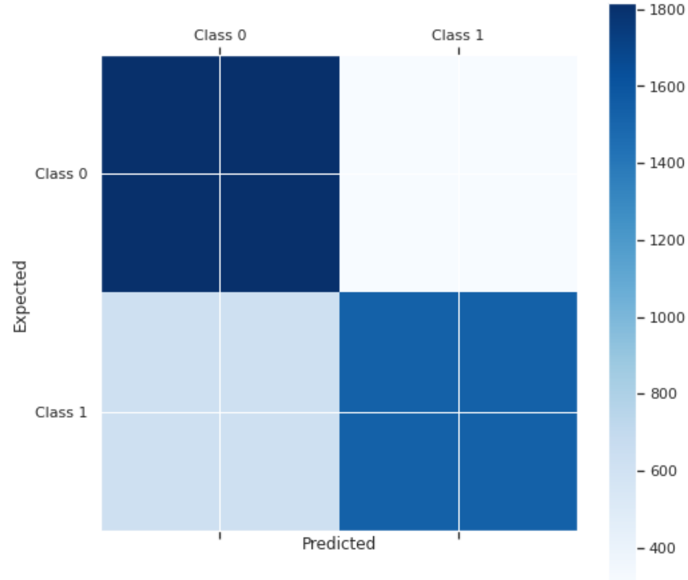
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 30), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
print("Minimum error:-",min(error),"at K =",error.index(min(error)))
```

```
Minimum error:- 0.04866635470285447 at K = 1
```

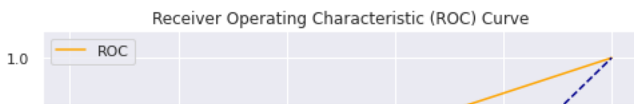

## Logistic Regression

```
#Calling the LR using Sklearn library
classifier = LogisticRegression()

#Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

#Train the model
classifier.fit(X_train,y_train)

#Predict the testing response
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)


classif_results()
```

```
Confusion matrix:
 [[1817  309]
 [ 617 1531]]
```



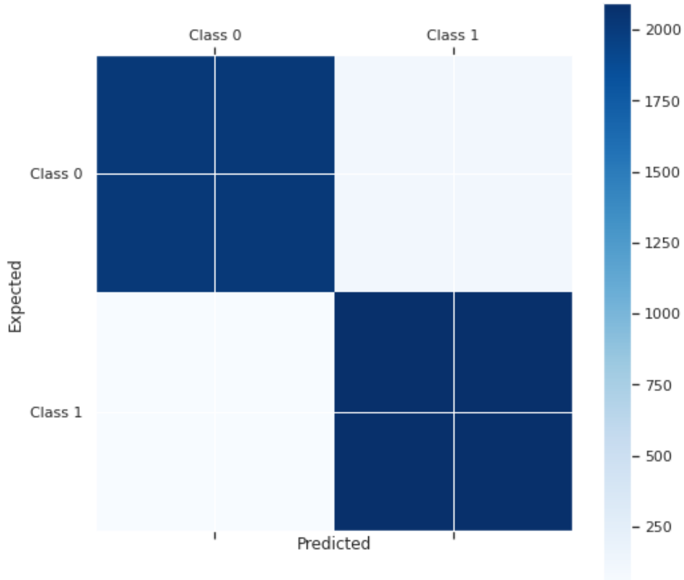

```
Accuracy 0.7833411324286382
              precision    recall  f1-score   support

           0       0.75      0.85      0.80      2126
           1       0.83      0.71      0.77      2148

    accuracy                           0.78      4274
   macro avg       0.79      0.78      0.78      4274
weighted avg       0.79      0.78      0.78      4274

AUC Score:
0.783706342157311
```



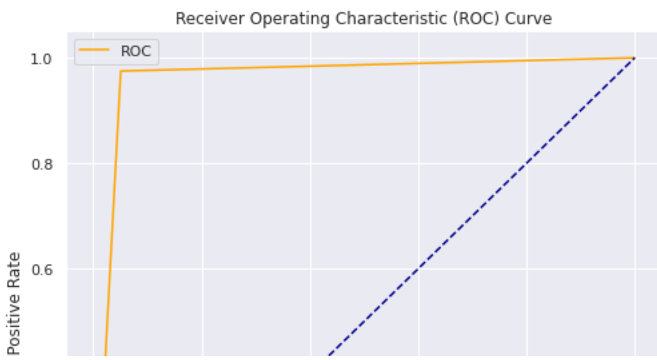

## Decision tree

```
#Calling the Decision Tree using Sklearn library
classifier = tree.DecisionTreeClassifier()

#Training the Model
classifier.fit(X_train,y_train)

#SPlit the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

#Predict the value
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)


classif_results()
```

```
Confusion matrix:
 [[2018  108]
 [  54 2094]]
```



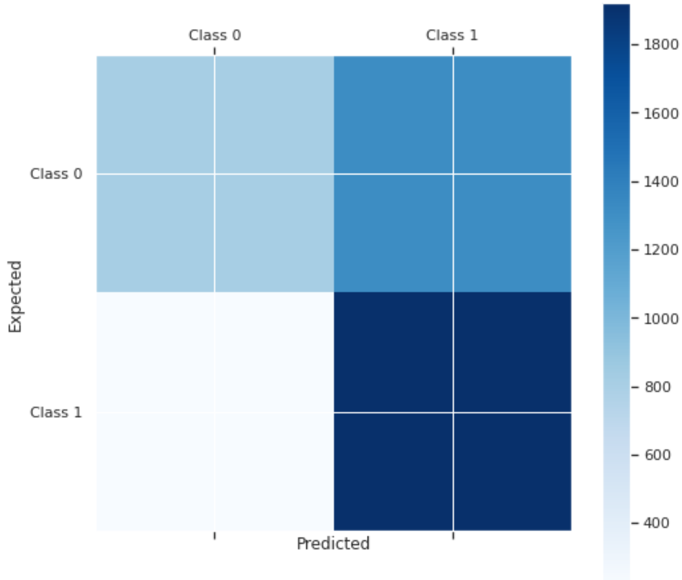

```
Accuracy 0.9620963968179691
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      2126
           1       0.95      0.97      0.96      2148

    accuracy                           0.96      4274
   macro avg       0.96      0.96      0.96      4274
weighted avg       0.96      0.96      0.96      4274

AUC Score:
0.9620303557445199
```



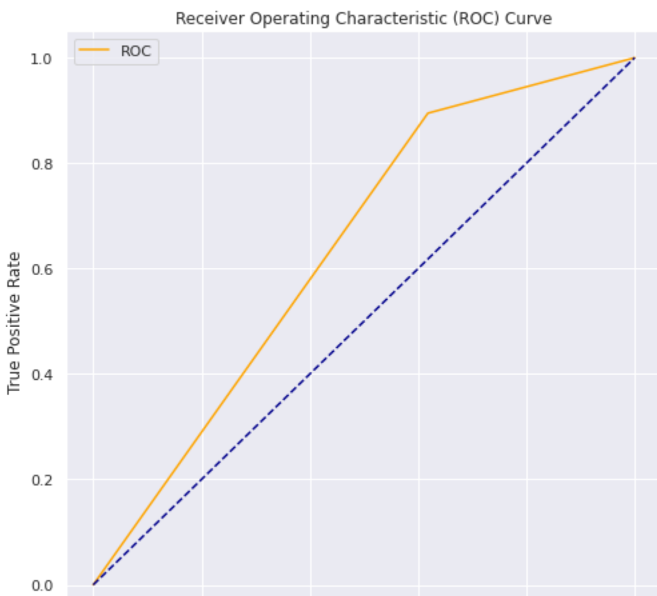

## Naive Bayes

```
#Calling the Naive Bayes using Sklearn library
classifier = GaussianNB()

#Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

#Training the Naive Bayes model
classifier.fit(X_train,y_train)

#Predicting the testing dataset
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)


classif_results()
```

```
Confusion matrix:
 [[ 812 1314]
 [ 226 1922]]
```




```
Accuracy 0.6396817969115582
              precision    recall  f1-score   support

           0       0.78      0.38      0.51      2126
           1       0.59      0.89      0.71      2148

    accuracy                           0.64      4274
   macro avg       0.69      0.64      0.61      4274
weighted avg       0.69      0.64      0.61      4274

AUC Score:
0.6383618794354196
```

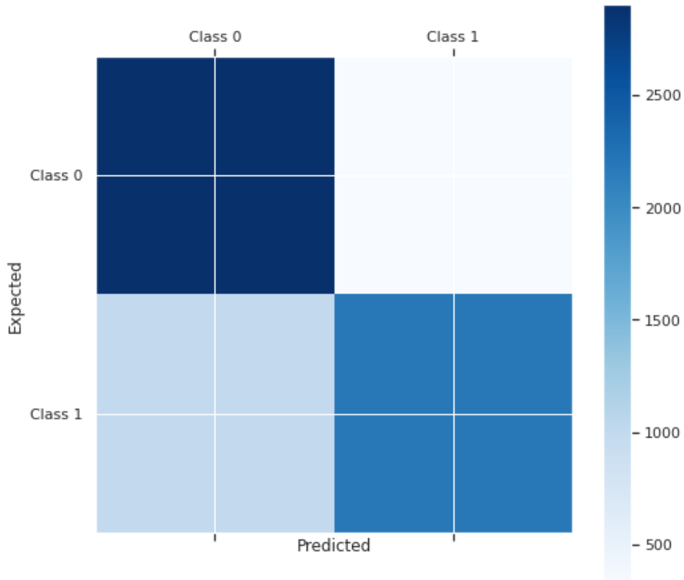



## Support Vector Machine (SVM)

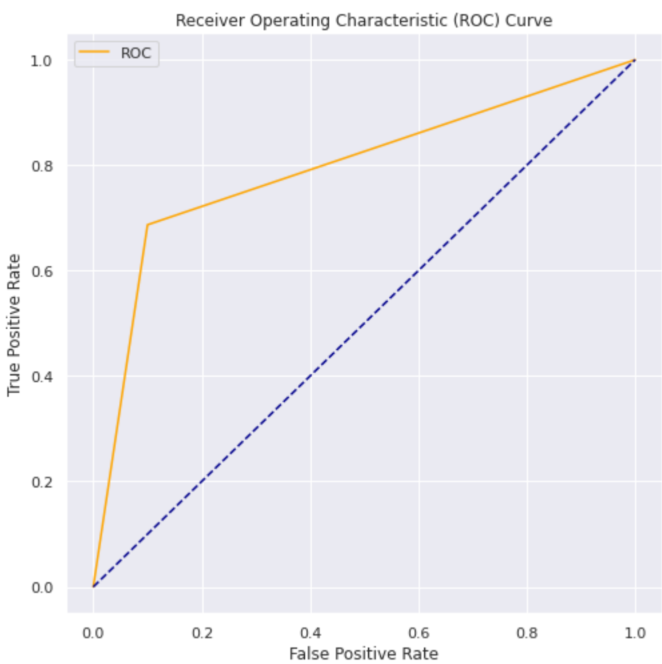
```
# import support vector classifier
from sklearn import svm

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear

#Train the model using the training sets
clf.fit(X_train, y_train)
```

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)

cm = confusion_matrix(y_test,y_pred)

classif_results()
```

```
Confusion matrix:
 [[2904  323]
 [ 997 2187]]
```




```
Accuracy 0.7941038839494619
              precision    recall  f1-score   support

           0       0.74      0.90      0.81      3227
           1       0.87      0.69      0.77      3184

    accuracy                           0.79      6411
   macro avg       0.81      0.79      0.79      6411
weighted avg       0.81      0.79      0.79      6411

AUC Score:
0.7933894468468777
```




## Model Comparision

```
lr_model = LogisticRegression()
knn_model = KNeighborsClassifier()
nb_model = GaussianNB()
des_model = tree.DecisionTreeClassifier()
sv_model =  svm.SVC()

models = [

    {
        'label': 'Naive Bayes',
        'model': nb_model
    },
    {
        'label': 'Logistic Regression',
        'model': lr_model
    },
      {
        'label': 'KNN',
        'model': knn_model
    },

    {
        'label': 'Decision Tree ',
        'model': des_model
    },
    {
        'label': 'SVM',
        'model': sv_model
    }
]

from sklearn.metrics import roc_curve, roc_auc_score, auc


plt.clf()
plt.figure(figsize=(8,8))
for m in models:
    m['model'].probability = True
    probas = m['model'].fit(X_train,y_train).predict_proba(X_test)
    fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])
    roc_auc  = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=0, fontsize='small')
plt.show()
```

<Figure size 576x576 with 0 Axes>