INTRODUCTION

Airbnb is a popular platform for home-sharing which allows people to rent out their properties so that they can earn from it. The hosts are supposed to set a price for their listings along with pictures, general descriptions, and amenities that the property offers. The prices keep on varying in accordance to different days of the week or seasons or on factors like dates which are closer to holiday season or weekends etc.

PROJECT AIMS:

This project aims to use machine learning techniques for predicting listing's success, and also to gain insights about market dynamics of Airbnb for the city of London. As a lister who is new to airbnb market of London, it is difficult to decide on what things he/she shoud keep in mind while renting out the property. Our model is designed to make their decision easy by filtering out the key factors they should keep in mind so that they can have maximum Return on their investment.

THE DATA SET

Insideairbnb.com Inside Airbnb is a popular site that provides with scraped data related to Airbnb listings, reviews, neighborhoods etc for different cities around the world. The data set we are using for our analysis was scraped on 06th of September, 2023 and gives us information about all the listings of London that were appearing live on that date.

IMPORTING LIBRARIES AND THE DATA:

In [655…

```
# Importing required libraries
import pandas as pd
import numpy as np
from numpy.random import seed
seed(123)
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import datetime
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import explained_variance_score, mean_squared_error, r2_score
```

In [656…

```
df1=pd.read_csv('nlistings.csv')
df2=pd.read_csv('reviews2.csv')

# Check the total number of rows using .shape attribute
total_rows = df1.shape[0]
print("Total number of rows in the Listings are:", total_rows)

total_rows = df2.shape[0]
print("Total number of rows in the Reviews are:", total_rows)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_23616\1621382389.py:1: DtypeWarning: Columns (68) have mi
xed types. Specify dtype option on import or set low_memory=False.
  df1=pd.read_csv('nlistings.csv')
Total number of rows in the Listings are: 87946
Total number of rows in the Reviews are: 1581033
```

DATA CLEANING AND PRE PROCESSING:

Data pre-processing is the most crucial step in an ML project.

1. Started with dropping the columns that had free text in them, we can later augment our ML models by doing NLP for sentimental analysis by using them but for this analysis, we are not using them as it will

increase the complexity of the code and will drag us away from our assignment's core objectives.

2. The next step was to drop the columns that had all null entries in them, which makes them useless for our model building.

In [657…
```
# Dropping columns because they contain free-text
cols_to_drop = ['listing_url', 'scrape_id', 'last_scraped', 'name', 'description', 'neighborhood_
                'picture_url', 'host_id', 'host_url', 'host_name', 'host_location', 'host_about',
                'host_picture_url', 'host_neighbourhood', 'host_verifications','host_has_profile_
df = df1.drop(cols_to_drop, axis=1)
#Dropping these columns because they had all null entries
df.drop(columns=['license', 'calendar_updated','bathrooms','neighbourhood_group_cleansed'], inpla
```

In [658…
```
print(sum((df.host_listings_count == df.host_total_listings_count) == False))
df.loc[((df.host_listings_count == df.host_total_listings_count) == False)][:5]
```

43796

Out[658…

	id	source	host_since	host_response_time	host_response_rate	host_acceptance_rate	host_is_superho
1	93015	city scrape	2011-04-11	within a few hours	100%	25%	
2	13913	city scrape	2009-11-16	within a few hours	100%	88%	
3	15400	city scrape	2009-12-05	within a day	100%	41%	
5	17402	city scrape	2010-01-04	within an hour	100%	100%	
6	93783	city scrape	2011-04-12	within a day	100%	80%	

5 rows × 53 columns

host_listings_count and host_total_listings_count are similar columns having around 43,796 same values. The only place where they differ are the rows having Nans. That's why we will drop one of these columns. The columns related to different host listing counts have very high correlation among them since they are derived from one another. So dropping them is beneficial for us to avoid multicollinearity.

In [659…
```
df.drop(['host_total_listings_count', 'calculated_host_listings_count', 'calculated_host_listings
         'calculated_host_listings_count_private_rooms', 'calculated_host_listings_count_shared_r
```

In [660…
```
df.shape
```

Out[660…
```
(87946, 48)
```

Again dropping some columns which will have little to no contribution in predicting wether our listing will be successful or not.

1. Several columns related to property location have missing information and values they will be removed. We will keep 'neighborhood_cleansed' column
2. Latitude and longitude will also be used to calculate euclidean distance from the city centre to see an impact on listings success if a property is located closer to center of city.
3. Only retaining two columns related to minimum and maximum nights a guest can stay, dropping the other because of the same reason as they are derived from the main ones, and will eventually lead to multicollinearity

In [661…
```
df.drop(['minimum_minimum_nights', 'maximum_minimum_nights', 'minimum_maximum_nights', 'maximum_m
         'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm'], axis=1, inplace=True)
```

In [662…
```
df.shape
```

Out[662… (87946, 42)

Checking boolean and categorical features in our data set to see if they have sufficient information and to know if they are worth keeping or not.

In [663…
```
# Replacing columns with f/t with 0/1
df.replace({'f': 0, 't': 1}, inplace=True)

# Plotting the distribution of numerical and boolean categories
df.hist(figsize=(20,20));
```

Majority of the columns have skewed distributions, indicating that we might need to standardize or normalize them according to the situation before using them inside our model.

CLEANING INDIVIDUAL COLUMNS

Here we will convert types of different variables, clean their null values, or bin them accordingly as we don't want to lose a significant amount of data while removing them.

1. One observation was made when an attempt was made to remove the null values by using a conventional Python command for removing nulls, but none were removed.
2. Later by understanding the data it was figured out that those columns were stored as strings/objects, hence the nulls can't be removed in that way.

3. N/As can be removed using the commands, but if we have NaNs in our data set sometimes this refers to valuable information. So rather than dropping the rows with Nans it is preferred to impute them with mean/median value of the column after checking the distribution (whichever seems correct).
4. Imputing the columns which have almost one-third of data missing or labeled as NaN, it is not suggested to impute them with mean or median values as this will create bias in our results and our data will be skewed extremely as a result.
5. Another way to avoid this is to categorize them under one label so we can see their impact if any on the target variable.

HOST RESPONSE RATE:

For host_response_rate a major chunk of the data had null values, but we do not want to lose valuable information about the overall data set by dropping them. After doing an in depth analysis, it was evident that these null values were majorly present due to the listing IDs that were inactive or were new so they were not rented as of now. Eventually decided to bin them in categories rather than to impute it with any mean/median value or by replacing it with zero.

In [664…
```
print("Null values:", df.host_response_rate.isna().sum())
print(f"Proportion: {round((df.host_response_rate.isna().sum()/len(df))*100, 1)}%")
```

```
Null values: 28918
Proportion: 32.9%
```

In [665…
```
# Removing the % sign from the host_response_rate string and converting to an integer
df.host_response_rate = df.host_response_rate.str[:-1].astype('float64')

print("Mean host response rate:", round(df['host_response_rate'].mean(),0))
print("Median host response rate:", df['host_response_rate'].median())
print(f"Proportion of 100% host response rates: {round(((df.host_response_rate == 100.0).sum()/df
```

```
Mean host response rate: 93.0
Median host response rate: 100.0
Proportion of 100% host response rates: 69.7%
```

In [666…
```
# Bin into four categories
df.host_response_rate = pd.cut(df.host_response_rate, bins=[0, 50, 90, 100], labels=['0-49%', '50

# Converting to string
df.host_response_rate = df.host_response_rate.astype('str')

# Replace nulls with 'unknown'
df.host_response_rate.replace('nan', 'unknown', inplace=True)

# Category counts
df.host_response_rate.value_counts()
```

Out[666…
```
host_response_rate
90-100%    48240
unknown    28918
50-89%      8133
0-49%       2655
Name: count, dtype: int64
```

ROOM TYPE & PROPERTY TYPE

In [667…
```
df.room_type.value_counts()
```

Out[667…
```
room_type
Entire home/apt    54575
Private room       32711
Shared room          441
Hotel room           219
Name: count, dtype: int64
```

In [668…
```
df.property_type.value_counts()
```

Out[668…
```
property_type
Entire rental unit                    33700
Private room in rental unit           14455
Private room in home                  10679
Entire condo                           8696
Entire home                            7557
                                      ...
Shared room in serviced apartment         1
Yurt                                      1
Earthen home                              1
Private room in camper/rv                 1
Private room in cave                      1
Name: count, Length: 102, dtype: int64
```

Both of these variables are valuable in a sense that this gives host an insight which type of rooms or property is being listed & rented frequently, so they can understand the preference of customers. From above it is evident that privacy is the attribute they look for while renting, hence resulting in entire rental units and private rooms being booked or listed more often. Later on in the analysis we will visualize them to see their impact on other features as well.

PRICE

1. Price is a key feature in our project analysis, we can generate insights related to revenue being generated by certain types of rooms or properties, or whether which neighborhoods are good for listers to choose so they can earn good revenue.
2. Moreover, this will also be used to determine the average price of different rental units so that lister know how to price their property keeping market competitive rates in mind. Because pricing it more or less will not benefit the listers eventually in the longer run.
3. Upon analyzing the sample of our data frame we realized that this column has some outliers, and that was because of some unusual kinds of bookings that were made for longer periods which is generally not the case.
4. Hence, the decision to remove the outliers was made rather than to take its log normal because we are focusing on the generalisability of our model, which benefits an average lister.

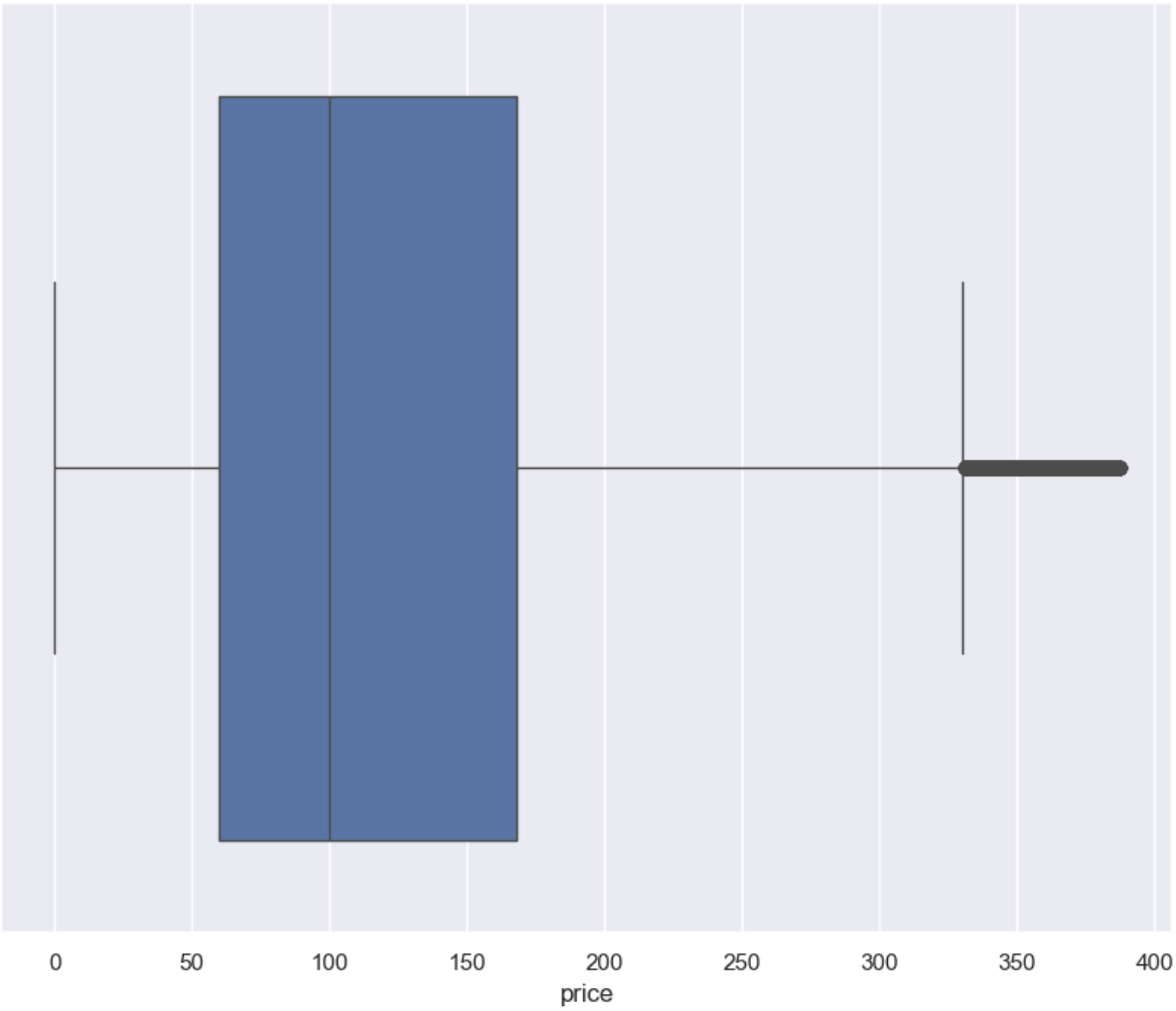
In [669…
```
# Converting the tpe of price to float from object as it was stored as string and also removed th
df["price"] = df["price"].str[1:].str.replace(",","").astype("float")
```

In [670…
```
# Removing outliers using IQR technique
df[["price","host_response_rate","host_acceptance_rate"]].dtypes

# writing a outlier function for removing outliers in important columns.
def iqr_technique(DFcolumn):
  Q1 = np.percentile(DFcolumn, 25)
  Q3 = np.percentile(DFcolumn, 75)
  IQR = Q3 - Q1
  lower_range = Q1 - (1.5 * IQR)
  upper_range = Q3 + (1.5 * IQR)                        # interquantile range

  return lower_range,upper_range
```

In [671…
```
lower_bound,upper_bound = iqr_technique(df['price'])

df = df[(df.price>lower_bound) & (df.price<upper_bound)]
```

In [672…
```
# so the outliers are removed from price column now check with boxplot and also check shape of ne

sns.boxplot(x = df['price'])
print(df.shape)
```

```
(81450, 42)
```



In [673…
```
# so here outliers are removed, see the new max price
print(df['price'].max())
```

```
387.0
```

AVAILABILITY COLUMNS

1. Upon searching for the domain knowledge needed for Airbnb rentals in London, I came across an article that was written by a former Airbnb employee working as a Data Scientist.
2. She gave a very valid reason for dropping the availability columns in our data set.In London, a law was passed in 2015 that put restrictions on renting any Air_bnb property for more than 90 days in a calendar year.
3. Only one column of availability will be retained - for 90 days.
4. Another reason is these columns were highly correlated to each other resulting in multicollinearity which is anyway not good for our model building.

In [674…
```
df.drop(['availability_30', 'availability_60', 'availability_365'], axis=1, inplace=True)
```

FIRST AND LAST REVIEW

We might not use these columns in our air bnb success prediction model. And might later drop them because about 24% of the data inside these columns have missing values.

In [675…
```
print(f"Null values in 'first_review': {round(100*df.first_review.isna().sum()/len(df),1)}%")
print(f"Null values in 'review_scores_rating': {round(100*df.review_scores_rating .isna().sum()/l
```

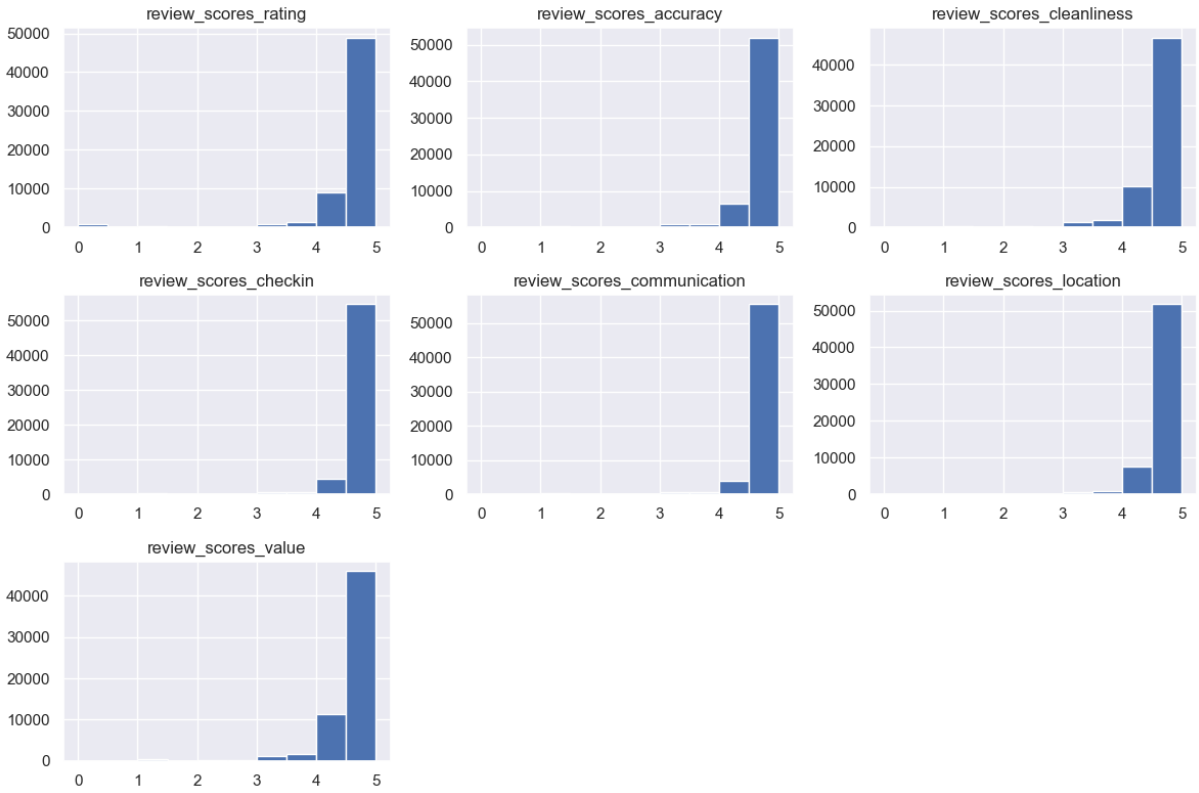
```
Null values in 'first_review': 23.8%
Null values in 'review_scores_rating': 23.8%
```

REVIEW RATING COLUMNS

There were a significant amount of NaNs present in review columns, grouping them into bins is more accurate rather than dropping the rows.

1. The histograms below were produced to decide on useful bins.
2. Majority of ratings are 4 or 5 out of 5, as also seen in the value count below.
3. 4/5 and 5/5 will be grouped together separately, and 1-3/5 will be binned separately.

In [676…
```
# Checking the distributions of the review ratings columns
variables_to_plot = list(df.columns[df.columns.str.startswith("review_scores") == True])
fig = plt.figure(figsize=(12,8))
for i, var_name in enumerate(variables_to_plot):
    ax = fig.add_subplot(3,3,i+1)
    df[var_name].hist(bins=10,ax=ax)
    ax.set_title(var_name)
fig.tight_layout()
plt.show()
```



REVIEWS PER MONTH

The reviews_per_month column also containing null values and we can simple put 0 reviews by replacing NAN's

In [677…
```
df['reviews_per_month'] = df['reviews_per_month'].replace(to_replace=np.nan,value=0).astype('int6
# the null values are replaced by 0 value
df['reviews_per_month'].isnull().sum()
```

Out[677… `0`

REMOVING COLUMNS WITH ATLEAST 70% NULL VALUES INSIDE THEM

In [678…
```
# Drop columns with a threshold for null values
# Specify a threshold to drop columns where the number of non-null values is less than the thresh
threshold = 0.7  # For example, keep columns with at least 70% non-null values
df = df.dropna(axis=1, thresh=int(threshold * len(df)))
```

In [679…
```
df.shape
```

Out[679… (81450, 36)

After initial cleaning, we are left with 36 columns, 39 of which were removed due to several reasons as stated and explained above. Now we will perform EDA on our remaining data set to gain some insights that can eventually help us to design a feature-engineered target variable or success indicator.

EXPLORATORY DATA ANALYSIS

In [680…
```
#print(df.info(show_counts=True, memory_usage=True, verbose=True))
# Run these commands to have a look at the shape of our data and to check the summary of our nume
```

In [681…
```
df.describe()
```

Out[681…

	id	host_is_superhost	host_listings_count	host_identity_verified	latitude	longitu
count	8.145000e+04	80605.000000	81445.000000	81445.000000	81450.000000	81450.0000
mean	3.514869e+17	0.152646	20.855952	0.868242	51.509547	-0.1260
std	4.119702e+17	0.359648	136.467684	0.338229	0.049806	0.1015
min	1.391300e+04	0.000000	1.000000	0.000000	51.295937	-0.4978
25%	2.156431e+07	0.000000	1.000000	1.000000	51.481640	-0.1898
50%	4.686784e+07	0.000000	2.000000	1.000000	51.514079	-0.1226
75%	8.120415e+17	0.000000	5.000000	1.000000	51.540670	-0.0649
max	9.738958e+17	1.000000	3023.000000	1.000000	51.681642	0.2957

8 rows × 25 columns

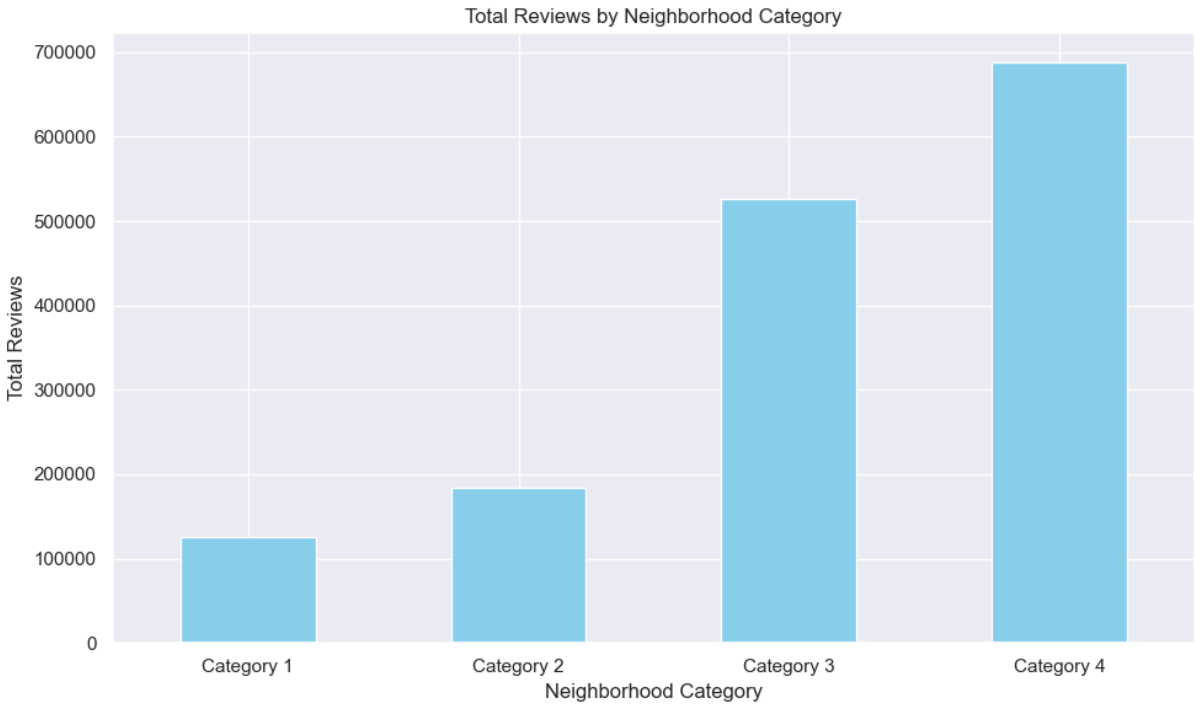
1. My approach to make way towards designing target variable was to group data into different categories based on neighborhoods.
2. Without combining them under some category the insights were not very clear. To my utmost surprise, the approach worked perfectly.
3. This also gave validation to the point that we can also do clustering analysis in future refined models to gain unique information about our data set
4. This concept was explained in our class, sometimes unsupervised learning outcomes or approach can be used to generate features with valuable information that will be later used for supervised learning models (e.g: Logistic Regression)
5. I did not perform K-mean clustering but used the idea behind it to group your data based on some variable, and categories with similar insights will appear. In future this can also be applied.
6. First step was to group the data based on neighborhoods and the average rental prices those categories have.
7. Later I decided to add more information by tagging the no of reviews along with it so we could see which category was performing better in terms of being rented out the most or popular among customers.

In [682…
```
neighborhood_stats = df.groupby('neighbourhood_cleansed').agg(
    average_price=('price', 'mean'),
    total_reviews=('number_of_reviews', 'sum')
).reset_index()

# Define the categories based on average price ranges
# You can adjust the thresholds as needed
category_labels = ['Category 1', 'Category 2', 'Category 3', 'Category 4']
bins = pd.qcut(neighborhood_stats['average_price'], q=4, labels=category_labels)
```

```
# Assign categories to each neighborhood based on average price
neighborhood_stats['category'] = bins

# Calculate annual revenue for each listing
df['annual_revenue'] = df['price'] * df['number_of_reviews_ltm'] * 365

# Merge the neighborhood categories back to the main dataset
df = pd.merge(df, neighborhood_stats[['neighbourhood_cleansed', 'category']], on='neighbourhood_c
```

In [683…

```
# Plotting
plt.figure(figsize=(10, 6))

# Bar plot for total reviews by category
reviews_by_category = neighborhood_stats.groupby('category',observed=True)['total_reviews'].sum()
reviews_by_category.plot(kind='bar', color='skyblue')
plt.xlabel('Neighborhood Category')
plt.ylabel('Total Reviews')
plt.title('Total Reviews by Neighborhood Category')

plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [684…

```
# Group neighborhoods by category
neighborhoods_by_category = neighborhood_stats.groupby('category',observed=True)['neighbourhood_c

# Display neighborhoods falling under each category
for category, neighborhoods in neighborhoods_by_category.items():
    print(f"Neighborhoods in {category}:")
    print(neighborhoods)
    print()
```

```
Neighborhoods in Category 1:
['Bexley', 'Bromley', 'Croydon', 'Enfield', 'Harrow', 'Hillingdon', 'Lewisham', 'Redbridge', 'Sutt
on']

Neighborhoods in Category 2:
['Barking and Dagenham', 'Ealing', 'Greenwich', 'Haringey', 'Havering', 'Kingston upon Thames', 'N
ewham', 'Waltham Forest']

Neighborhoods in Category 3:
['Barnet', 'Brent', 'Hackney', 'Hounslow', 'Lambeth', 'Merton', 'Southwark', 'Tower Hamlets']

Neighborhoods in Category 4:
['Camden', 'City of London', 'Hammersmith and Fulham', 'Islington', 'Kensington and Chelsea', 'Ric
hmond upon Thames', 'Wandsworth', 'Westminster']
```

It was very evident that neighborhoods falling under the category 4 were performing better than other, as they were reviewed max no. of times giving us an insight that customers prefer to rent listings in these neighborhoods. A potential host can decide which neighborhoods to short list on the basis of popularity as this will significantly impact his/her listing's success

In [685…
```python
# Create a new column 'total_review' as the product of 'no_of_reviews' and 'price'

df['total_revenue'] = df['number_of_reviews'] * df['price']  # Assuming price iscurrency per revi

# Calculate annual revenue for each neighborhood based on category
neighborhood_revenue = df.groupby('category',observed=True)['annual_revenue'].sum()

# Find the category generating the highest annual revenue
highest_revenue_category = neighborhood_revenue.idxmax()
highest_revenue = neighborhood_revenue.max()

print(f"The category generating the highest annual revenue is {highest_revenue_category} with ann
```

```
The category generating the highest annual revenue is Category 4 with annual revenue of $114576190
05.00
```

The above code used reviews and price columns to calculate the revenue generated by each of these neighborhood categories. Because as lister a high ROI is a key metric to decide whether listing will fall under good or bad category Category 4 neighborhoods outperformed others in revenue generation, this fact also confirms the insight extracted earlier that category 4 is most popular among customers (highest reviews)

In [686…
```python
# Filter listings that have not generated revenue in the last 12 months
listings_without_revenue = df[df['annual_revenue'] == 0]

# Get the count of listings without revenue in the last 12 months
total_listings_without_revenue = len(listings_without_revenue)

print(f"Total listings that have not generated revenue in the last 12 months: {total_listings_wit
```

```
Total listings that have not generated revenue in the last 12 months: 39527
```

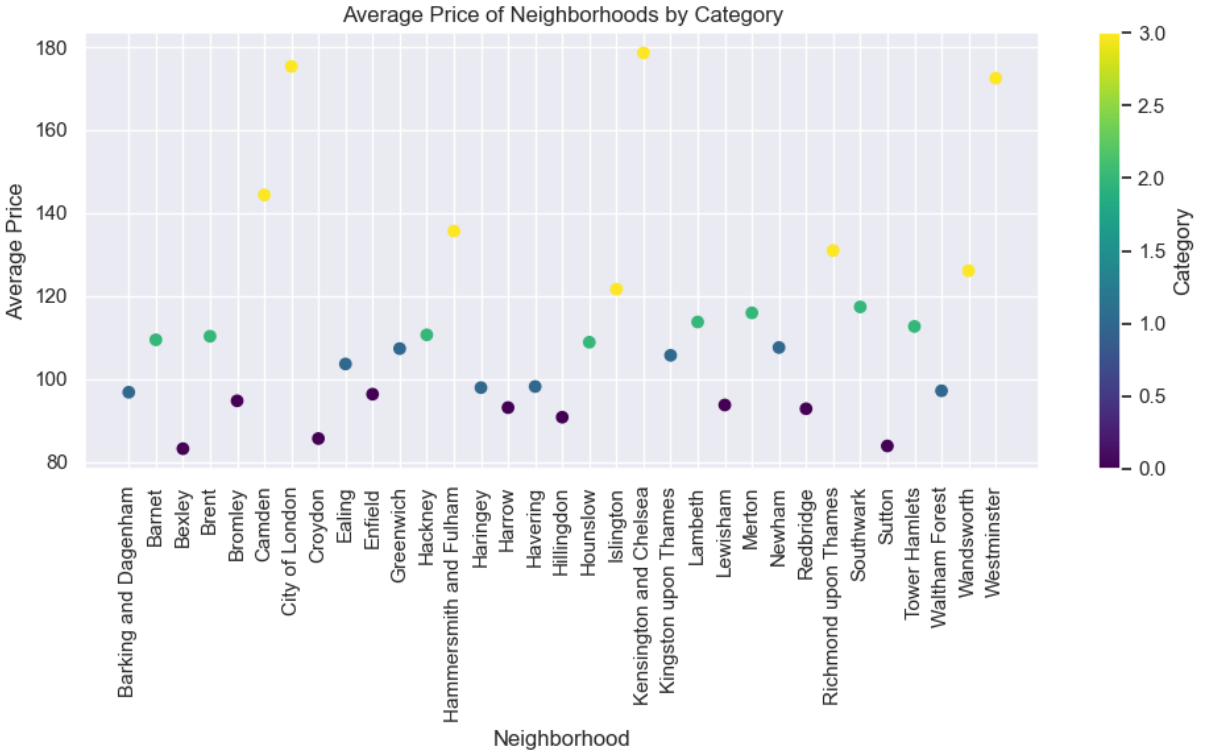
In [687…
```python
# Update your original DataFrame to only include listings with revenue in the last 12 months
#df = df[df['annual_revenue'] != 0]
```

The above step is optional. I filtered out listings that did not generate any revenue in the past 12 months, meaning either they were dormant or were new listings that have not been rented as of yet.

1. It depends on personal preference if we want to filter our data and focus only on listings that were active in the past year. Or to keep all the data.
2. I ran my model for both filtered and un-filtered data and this did not impact model performance, so I decided to keep them as the more data we have the better it is specifically for Logistic regression model.

In [688…
```python
# Group by 'neighborhood_cleansed' and calculate average price
neighborhood_avg_price = df.groupby('neighbourhood_cleansed')['price'].mean().reset_index()

# Define the categories based on average price ranges
```

```
# You can adjust the thresholds as needed
category_labels = ['Category 1', 'Category 2', 'Category 3', 'Category 4']
bins = pd.qcut(neighborhood_avg_price['price'], q=4, labels=category_labels)

# Assign categories to each neighborhood based on average price
neighborhood_avg_price['category'] = bins
neighborhood_avg_price.columns = ['neighbourhood_cleansed', 'average_price', 'category']

# Merge the neighborhood categories and average prices back to the main dataset
df = pd.merge(df, neighborhood_avg_price[['neighbourhood_cleansed','average_price']], on='neighbo
# Plotting
plt.figure(figsize=(10, 6))

# Plot average price vs neighborhood
plt.scatter(neighborhood_avg_price['neighbourhood_cleansed'], neighborhood_avg_price['average_pri
plt.colorbar(label='Category')
plt.xlabel('Neighborhood')
plt.ylabel('Average Price')
plt.title('Average Price of Neighborhoods by Category')

plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



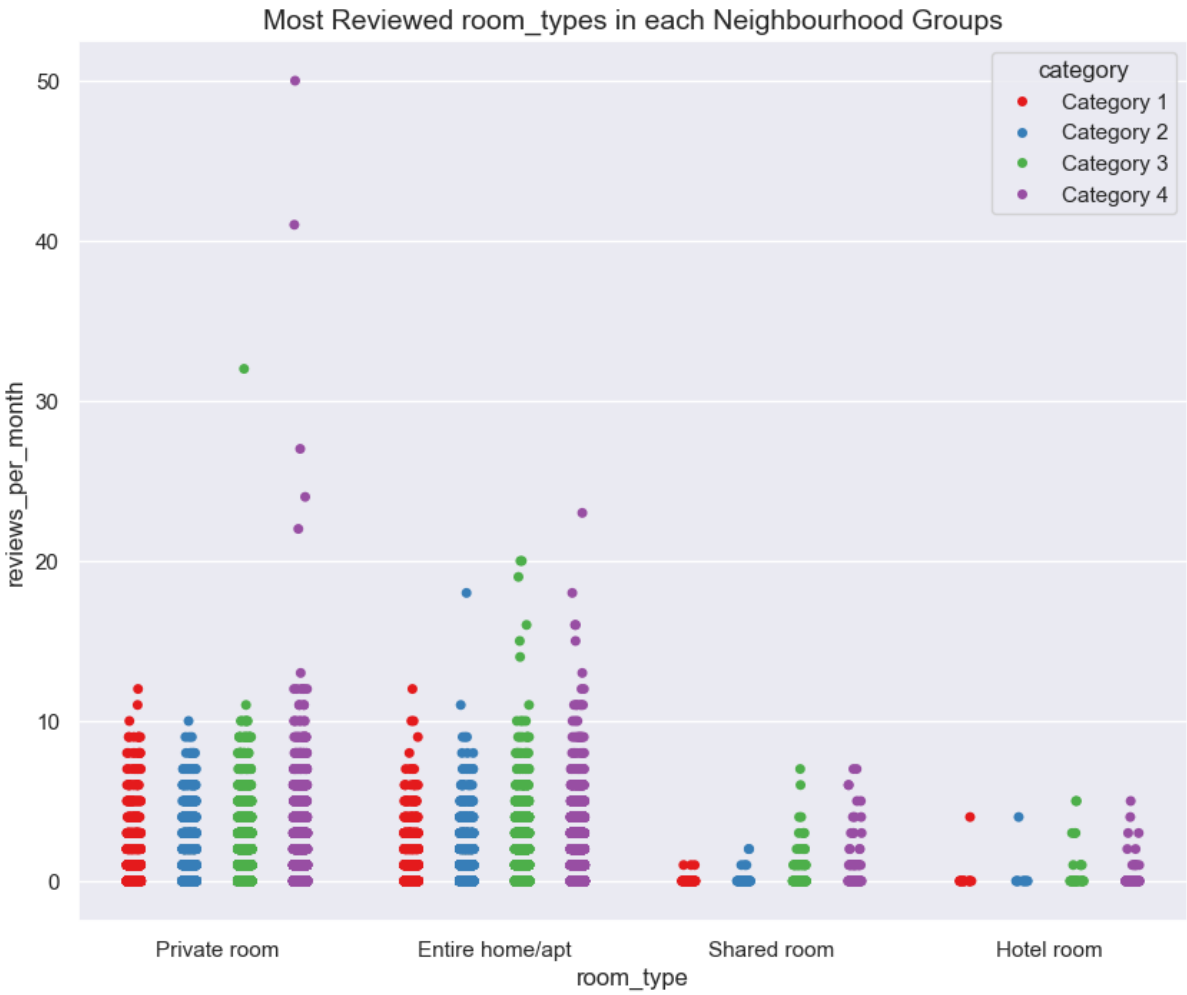
The above graph gives a perfect insight into how to decide the market competitive price for our listing based on which neighborhood category it will fall into. As a lister this is very crucial step to decide on the listing price. If you price it higher people will not rent it as other cheaper options will be available, if you price it lower your Return on investment will be on the lower side as compared to other listings. Figuring out optimal price based on which category neighborhood a listing falls into is good approach.

In [689…

```
import seaborn as sns
# create a figure with a default size of (10, 8)
f, ax = plt.subplots(figsize=(10, 8))

# create a stripplot that displays the number of reviews per month for each room type in the Airb
ax = sns.stripplot(x='room_type', y='reviews_per_month', hue='category', dodge=True, data=df, pal

# set the title of the plot
ax.set_title('Most Reviewed room_types in each Neighbourhood Groups', fontsize='14')
```

Out[689…

```
Text(0.5, 1.0, 'Most Reviewed room_types in each Neighbourhood Groups')
```



The above graph is also very useful for the success indicator generation and analysis. This tell us that the most popular type of rooms preferred by customers is private room or entire home/apt. We also gained an insight that category 3 and 4 neighborhoods are the ones that have shared or hotel rooms as well which are being booked often. This graph also depicts that in category 4 neighborhoods, properties (private room) were even rented (reviewed) for around 50 times as well, indicating a great market to create a new listing of private room type.

In [690…

```
# Count the number of listings in each neighborhood group and store the result in a Pandas series
counts = df['category'].value_counts()

# Reset the index of the series so that the neighborhood groups become columns in the resulting d
Top_Neighborhood_group = counts.reset_index()

# Rename the columns of the dataframe to be more descriptive
Top_Neighborhood_group.columns = ['Neighborhood_Groups', 'Listing_Counts']

# display the resulting DataFrame
Top_Neighborhood_group
```

Out[690…

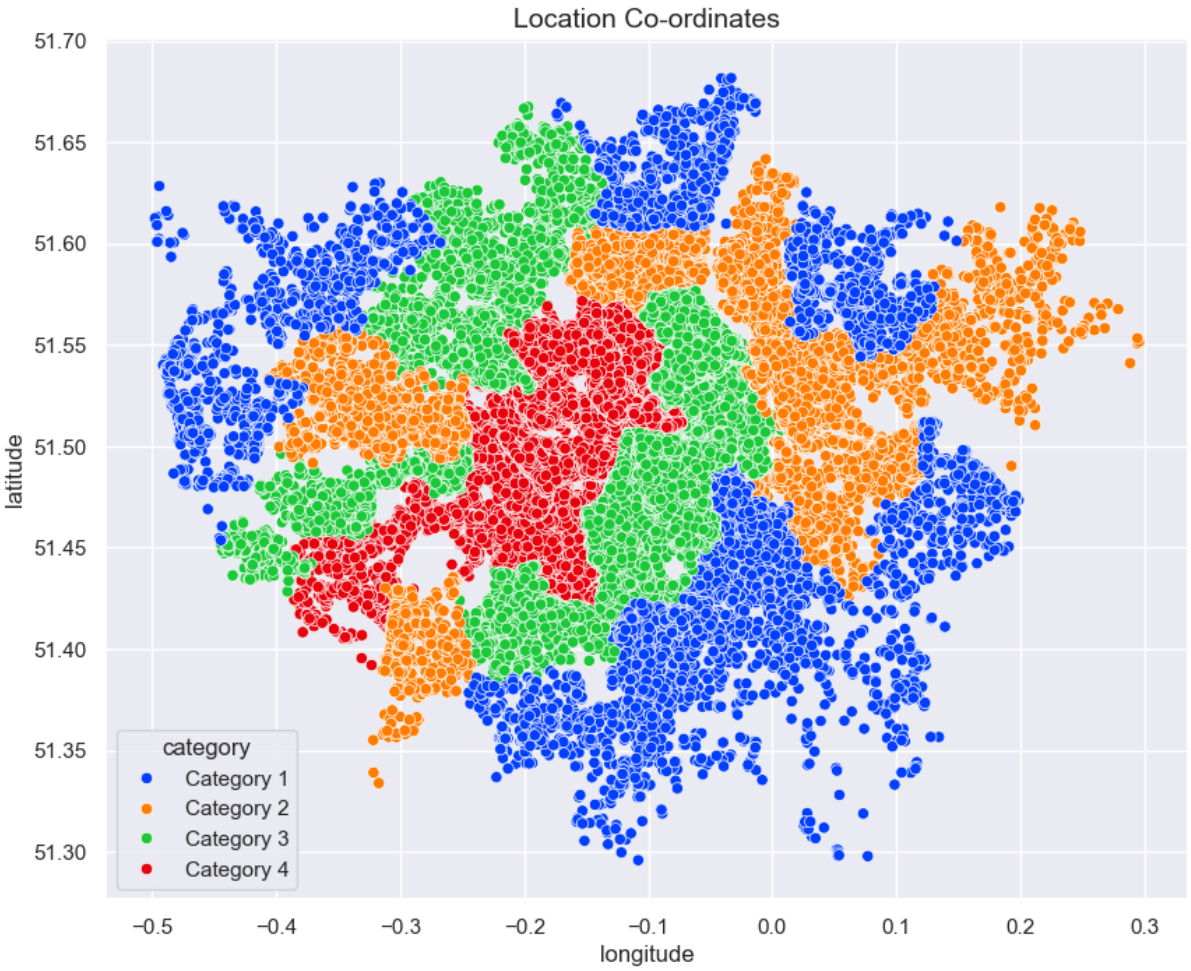
	Neighborhood_Groups	Listing_Counts
0	Category 4	31632
1	Category 3	29390
2	Category 2	11773
3	Category 1	8655

In [691…

```
#trying to find where the coordinates belong from the latitude and longitude

# set the default figure size for the seaborn library
```

```
sns.set(rc={"figure.figsize": (10, 8)})

# create a scatter plot that displays the longitude and latitude of the listings in the Airbnb da
ax = sns.scatterplot(data=df, x="longitude", y="latitude", hue='category', palette='bright')

# set the title of the plot
ax.set_title('Location Co-ordinates', fontsize='14')
```

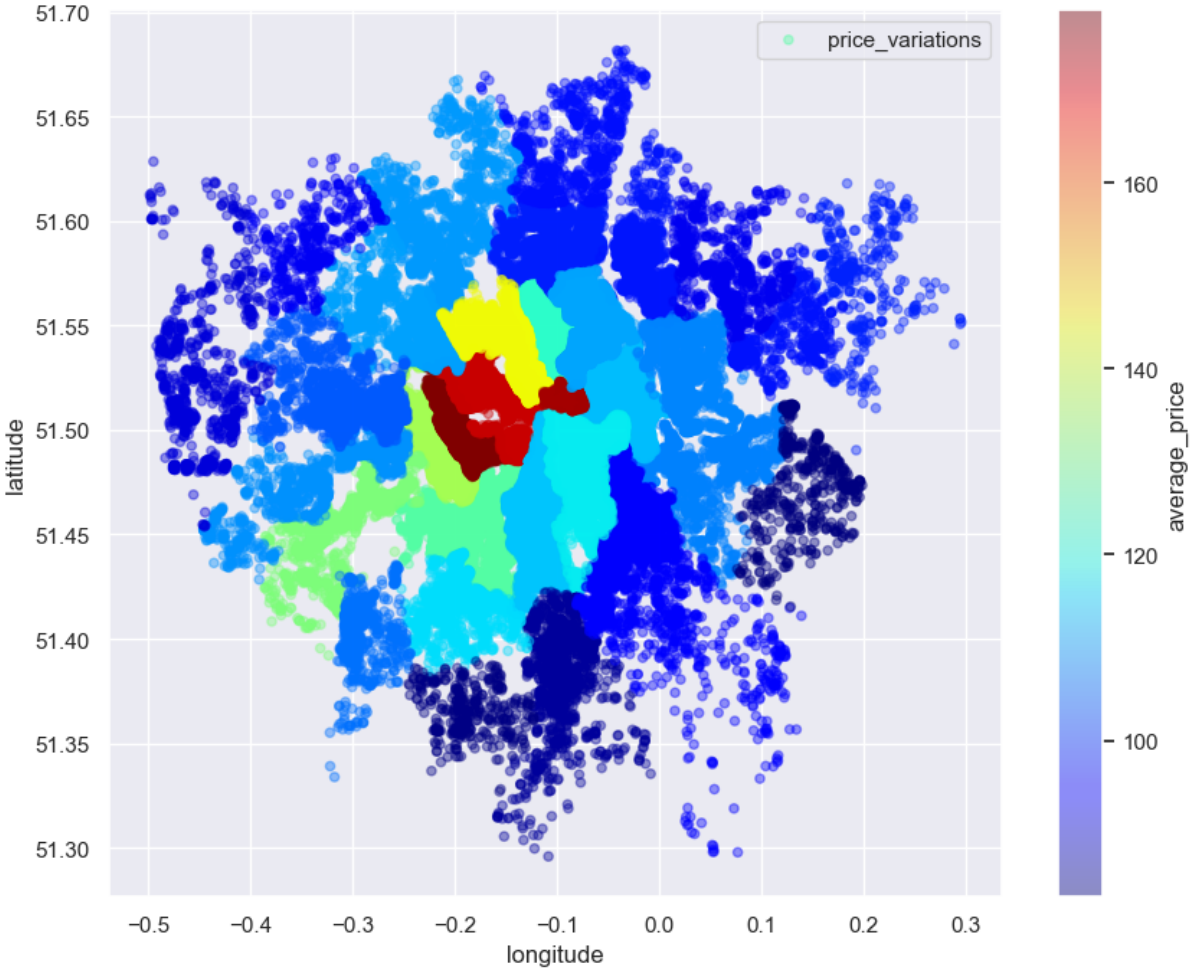
Out[691… `Text(0.5, 1.0, 'Location Co-ordinates')`



A great way to have better understanding of neighborhood categories is to visualize them on London's map. Thinking of as a lister's point of view category 4 listing neighborhoods are situated mostly near to the centre of London, making them more popular among customers. As most attractions, tourist spots, facilities like public transportation (tubes, lines, trains) lie near to it.

In [692…
```
# Let's have an idea of the price variations in neighborhood_groups

# create a scatter plot that displays the longitude and latitude of the listings in the Airbnb da
lat_long = df.plot(kind='scatter', x='longitude', y='latitude', label='price_variations', c='aver
                   cmap=plt.get_cmap('jet'), colorbar=True, alpha=0.4, figsize=(10, 8))

# add a legend to the plot
lat_long.legend()
```

Out[692… `<matplotlib.legend.Legend at 0x290216643b0>`

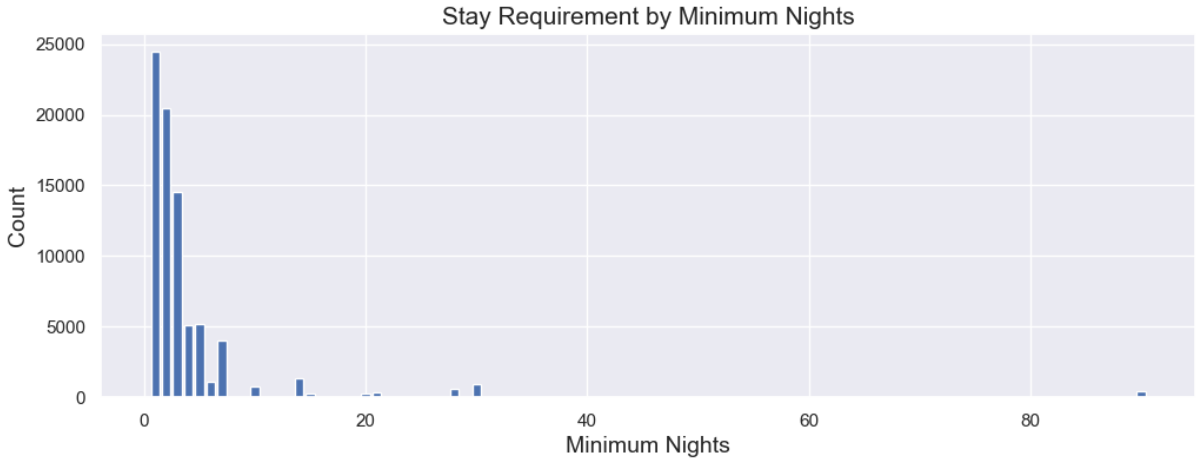


The above graph is also an excellent depiction of price variation, the more you move towards city's outskirts, the more cheaper the rates will be for airbnb listings. This also explains the fact the more facilities a customer will attain in terms of location, low crime rates or tourist spots, public transportation the higher they have to pay..

In [693…

```
# Group the DataFrame by the minimum_nights column and count the number of rows in each group
min_nights_count = df.groupby('minimum_nights').size().reset_index(name = 'count')

# Sort the resulting DataFrame in descending order by the count column
min_nights_count = min_nights_count.sort_values('count', ascending=False)

# Select the top 10 rows
min_nights_count = min_nights_count.head(15)

# Reset the index
min_nights_count = min_nights_count.reset_index(drop=True)

# Display the resulting DataFrame
min_nights_count
# Extract the minimum_nights and count columns from the DataFrame
minimum_nights = min_nights_count['minimum_nights']
count = min_nights_count['count']

# Set the figure size
plt.figure(figsize=(12, 4))

# Create the bar plot
plt.bar(minimum_nights, count)

# Add axis labels and a title
plt.xlabel('Minimum Nights', fontsize='14')
plt.ylabel('Count', fontsize='14')
plt.title('Stay Requirement by Minimum Nights', fontsize='15')
```

```
# Show the plot
plt.show()
```



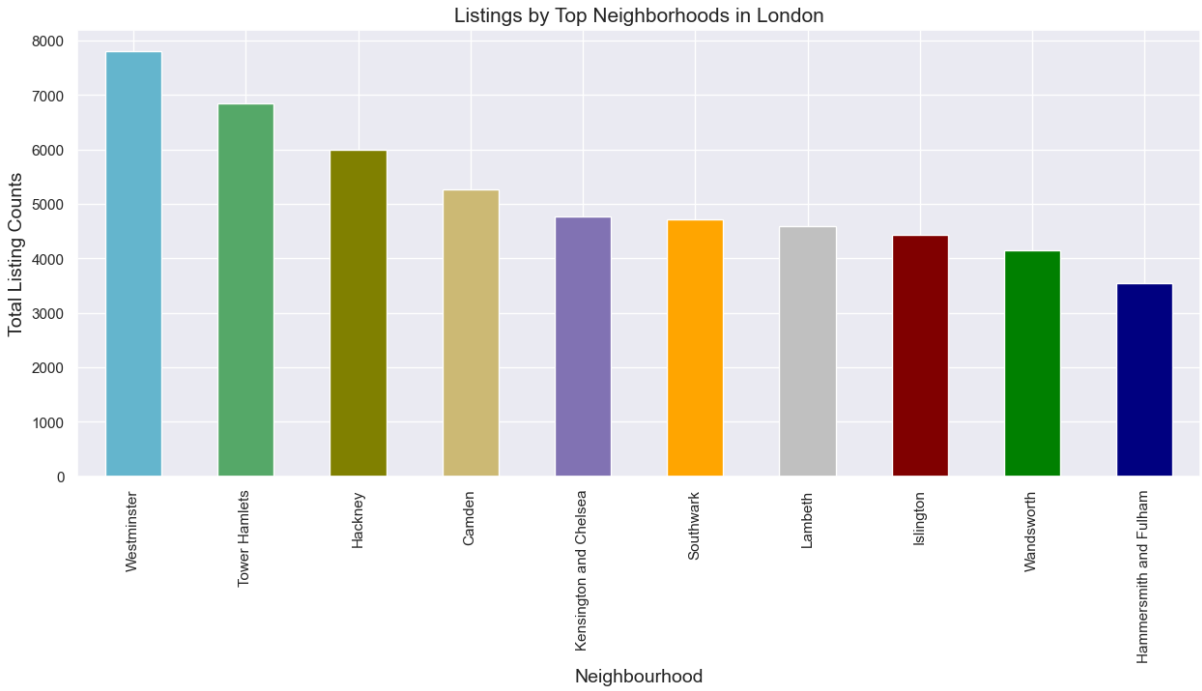
We develop an understanding from this graph that most people prefer to rent an airbnb which has roughly a threshold of 2 to 3 nights. So while launching a listing it is advisable to keep this close to minimum threshold so people find it easy to book it for shorter stays rather than bounding them in long term contracts of stay

In [694…

```
# Get the top 10 neighborhoods by listing count
top_10_neigbourhoods = df['neighbourhood_cleansed'].value_counts().nlargest(10)

# Create a list of colors to use for the bars
colors = ['c', 'g', 'olive', 'y', 'm', 'orange', '#C0C0C0', '#800000', '#008000', '#000080']

# Create a bar plot of the top 10 neighborhoods using the specified colors
top_10_neigbourhoods.plot(kind='bar', figsize=(15, 6), color = colors)

# Set the x-axis label
plt.xlabel('Neighbourhood', fontsize=14)

# Set the y-axis label
plt.ylabel('Total Listing Counts', fontsize=14)

# Set the title of the plot
plt.title('Listings by Top Neighborhoods in London', fontsize=15)
```

Out[694…

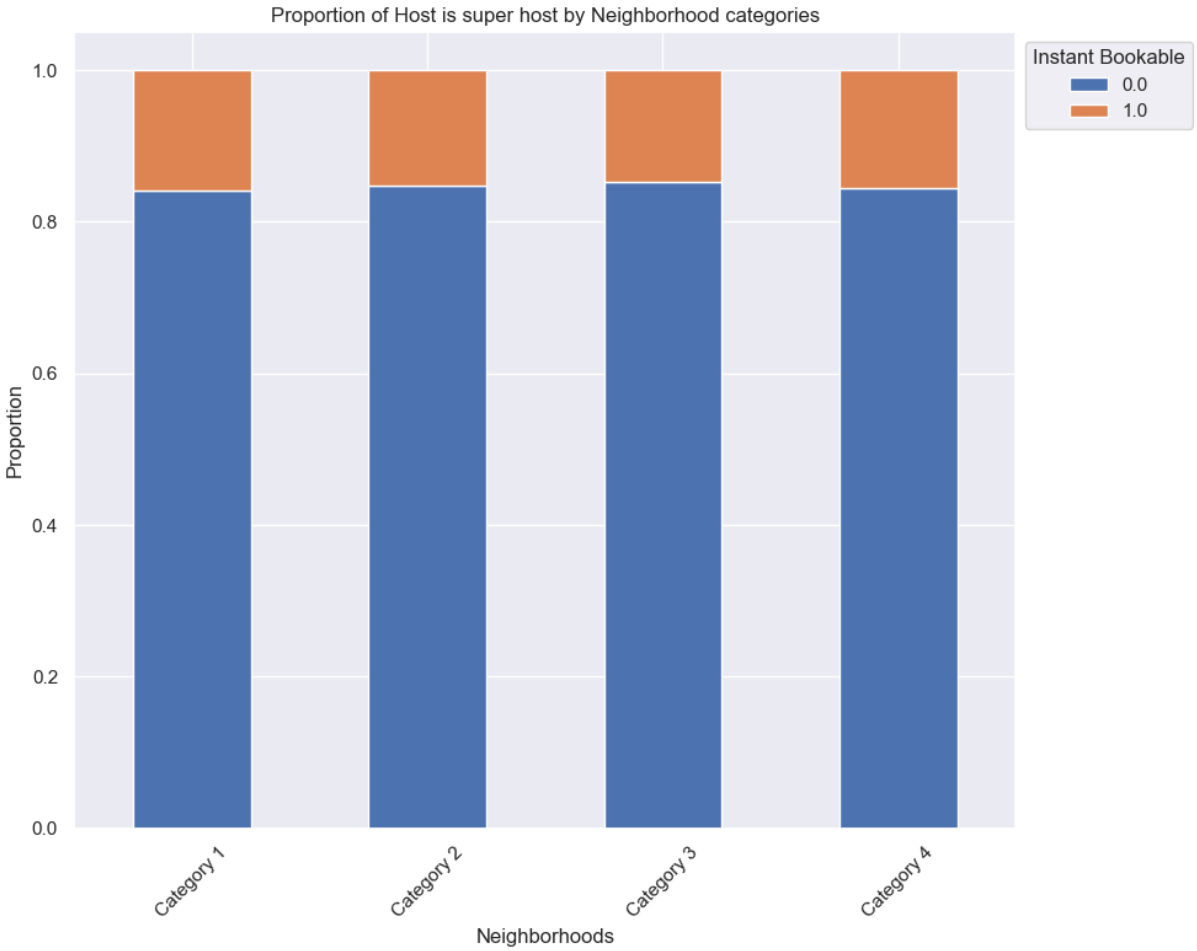
```
Text(0.5, 1.0, 'Listings by Top Neighborhoods in London')
```



CHECKING BINARY VARIABLES

In [695…
```
# Proportion of 'host_is_superhost' within different 'neighborhoods'
count_by_neighborhood = df.groupby('category',observed=True)['host_is_superhost'].value_counts(no

# Plotting the proportion for each neighborhood
plt.figure(figsize=(12, 8))
count_by_neighborhood.plot(kind='bar', stacked=True)
plt.xlabel('Neighborhoods')
plt.ylabel('Proportion')
plt.title('Proportion of Host is super host by Neighborhood categories')
plt.legend(title='Instant Bookable', bbox_to_anchor=(1, 1))
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

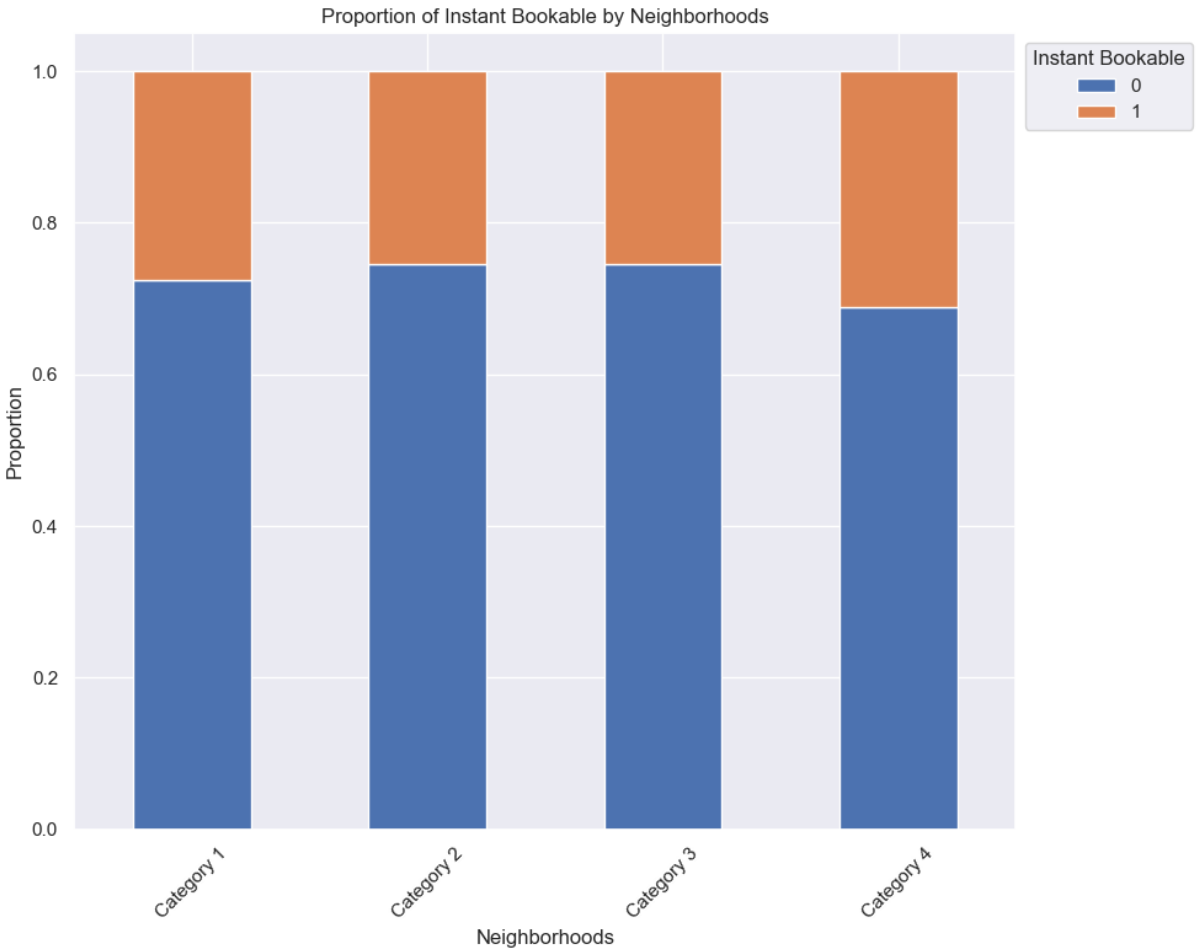
```
<Figure size 1200x800 with 0 Axes>
```



Super host status is a mark of quality for the top-rated and most experienced hosts. The above graph shows whether you have a super host badge or not it does not have much of an impact on any neighborhood category, this means being a super host does not interpret that a listing will be rented or reviewed often or will have a higher average price than other properties.

In [696…
```
# Proportion of 'instant_bookable' within different 'neighborhoods'
count_by_neighborhood = df.groupby('category', observed=True)['instant_bookable'].value_counts(no

# Plotting the proportion for each neighborhood
plt.figure(figsize=(12, 8))
count_by_neighborhood.plot(kind='bar', stacked=True)
plt.xlabel('Neighborhoods')
plt.ylabel('Proportion')
plt.title('Proportion of Instant Bookable by Neighborhoods')
plt.legend(title='Instant Bookable', bbox_to_anchor=(1, 1))
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

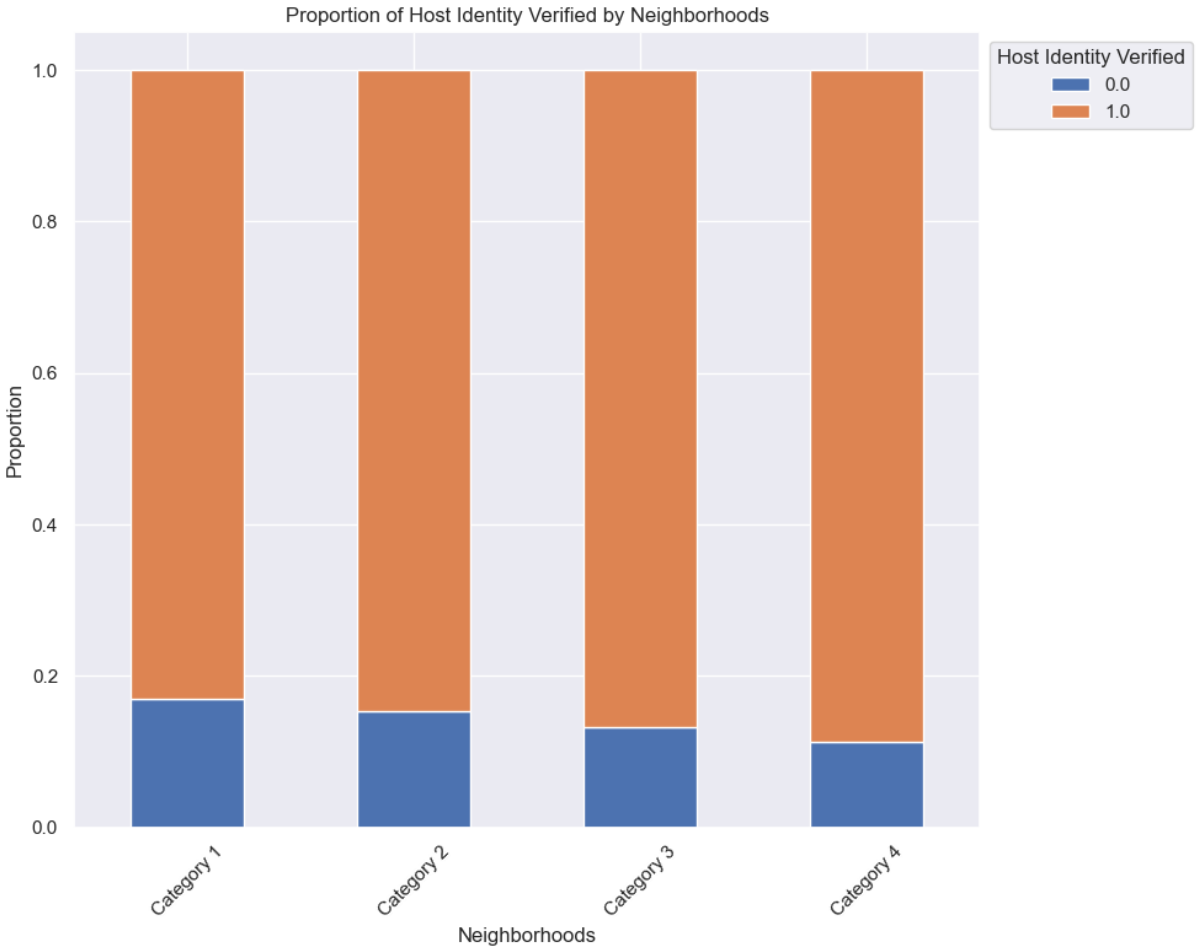


From the above graph, we gained the knowledge that instant booking might impact the listing's success slightly, as category 4 neighborhood listers offer this facility a little more often in comparison to other neighborhood categories.

In [697…

```
# Count of 'host_identity_verified' for each category in 'neighborhoods'
count_by_neighborhood = df.groupby('category',observed=True)['host_identity_verified'].value_coun

# Plotting the count for each neighborhood
plt.figure(figsize=(10, 6))
count_by_neighborhood.plot(kind='bar', stacked=True)
plt.xlabel('Neighborhoods')
plt.ylabel('Proportion')
plt.title('Proportion of Host Identity Verified by Neighborhoods')
plt.legend(title='Host Identity Verified', bbox_to_anchor=(1, 1))
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Host verification plays an important role in terms of customer's trust in the hosts. But mostly listings that are present in the data set has verified hosts. It's a general practice to add your credentials while renting out a property on the platform. So we can not infer that since category 4 neighborhoods are performing better in terms of being rented frequently, or being the ones with the highest revenue generation, their success is because of those verification credentials that were provided.

TARGET VARIABLE

We were asked to generate it by feature engineering our existing variables. The columns used were neighbourhood_cleansed, price, number_of_reviews The idea behind this target variable generation is that if a listing falls under a certain category, it will depict what are the chances of being reviewed/rented, what sort of revenue will it possibly be generating, and also a high/low occupancy rate. These three things are the ultimate success for a lister.

Another idea was to generate a score-based success metric that assigns weights to the features that have an impact on the listing's success. But it has some major short comings.

1. Firstly, without appropriate domain knowledge we can not assign weights to the features since we don't know how much importance they carry, this will produce biased results.
2. The second reason for not generating it that way because our features have multiple scales, we need to scale them down first so one category's weight might not overly impact another's
3. The third issue was that we might need to encode some features through one hot or label encoding to get them in numerical format, encoding without an in-depth domain knowledge often produces results with errors. To be on the safe side, I decided to choose the variables that needed little to no preprocessing, were already cleaned and complete and I can verify my target variable through visualizations as I did above in the EDA. Therefore I decided to make a success indicator based on the

neighbourhood category in which a property will fall. Later on in our analysis when the model will be built we will see how different variables are related to our target variable.

In [698…

```
# Define a function to label success metric based on category
def label_success_metric(row):
    category = row['category']

    if category == 'Category 4':
        return 'Good'
    elif category in ('Category 1', 'Category 2'):
        return 'Bad'
    elif category == 'Category 3':
        return 'Average'
    else:
        return 'Not categorized'

# Apply the label_success_metric function to create the success metric column
df['success_metric'] = df.apply(label_success_metric, axis=1)

# Display or use the dataset with the success metric column
print(df[['category', 'success_metric']])
```

```
        category success_metric
0      Category 4           Good
1      Category 4           Good
2      Category 4           Good
3      Category 4           Good
4      Category 4           Good
...           ...            ...
81445  Category 1            Bad
81446  Category 1            Bad
81447  Category 1            Bad
81448  Category 1            Bad
81449  Category 1            Bad

[81450 rows x 2 columns]
```

HYPOTHESIS REFINEMENT

HUNCH:

Ho: Pricing strategy impacts listing success, with competitive rates leading to higher success rates.

• **Attributes:**

'price', 'neighborhoods'

HUNCH:

Ho: The number of amenities provided influences listing success.

• **Attributes:**

'amenities'

HUNCH:

Ho: Location plays a significant role in the success of a listing.

• **Attributes:**

'neighborhood', 'latitude', 'longitude', 'property_type'

HUNCH:

Superhost status positively impacts listing success due to increased trust from guests.

• **Attributes:**

'host_is_superhost', 'host_response_rate'

HUNCH:

Ho: High ratings lead to increased listing success.

• **Attributes:**

'review_scores_rating', 'review_scores_accuracy', 'review_scores_cleanliness', 'review_scores_checkin', 'review_scores_communication', 'review_scores_location', 'review_scores_value'

HUNCH:

Ho: The number of available accommodations (e.g., an entire place, private room) affects listing success.

• **Attributes:**

'room_type', 'accommodates'

HUNCH:

Ho: Quick response time from hosts positively influences listing success.

• **Attributes:**

'host_response_time', 'host_response_rate'

The above-proposed hypothesis is refined after doing the initial data pre-processing and EDA on our data set. This is an important step in deciding which attributes to keep and which to drop for model building. Their actual impact on listing's success will be known once we will test them through our ML models. One might come up with more hypothesis proposals, as there is no end to the analysis but we have to keep it concise in accordance with our assignment outline.

FEATURE ENGINEERING

Now we will generate new features from the existing fe by manipulating them. This will be done either by aggregating columns using mean/median etc or by encoding them through different techniques if they are categorical in nature. This step is very important as we want to use the attributes mentioned in our hypothesis for testing our model.

OVERALL_REVIEW_SCORE

In [699…
```
# Calculate the mean of review-related columns to derive an overall score
df['overall_review'] = df[['review_scores_rating', 'review_scores_accuracy', 'review_scores_clean

# Displaying the updated DataFrame with the overall review score
print(df[['id', 'overall_review']])
```

```
                     id  overall_review
0                 92644        4.680000
1                 24328        4.824286
2                 80123        4.932857
3                241621        4.737143
4                263672        4.482857
...                 ...             ...
81445  965685789104358339             NaN
81446  967364380893458530             NaN
81447  967997156221529075             NaN
81448  969380028696101597             NaN
81449  972423637626394393             NaN

[81450 rows x 2 columns]
```

In [700…
```
# Remove rows with NaN values in the 'Overall Review Score' column
df = df.dropna(subset=['overall_review'])
#Unsurprisingly, there are perfect correlations between NaN reviews (i.e. listings that are not r
#different review categories, and first and last review times. NaN categories can therefore be dr
```

```
# Define bins and labels for binning
bins = [0, 3, 5]
labels = ['Low', 'High']

# Bin 'Overall Review Score' column based on specified bins and labels
df['overall_review'] = pd.cut(df['overall_review'], bins=bins, labels=labels, right=False)

# Create dummy variables (one-hot encoding) for 'Overall Review Score'
df = pd.get_dummies(df, columns=['overall_review'], drop_first=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_23616\3291565039.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  df['overall_review'] = pd.cut(df['overall_review'], bins=bins, labels=labels, right=False)
```

In [701…
```
df.shape
```

Out[701…
```
(62051, 42)
```

EUCLIDEAN DISTANCE

In [702…
```
# Define the coordinates of the city center
city_center = (51.5074, -0.1278)  # Replace with actual coordinates


# Calculate Euclidean distance from the city center for each listing
df['euclidean_distance'] = np.sqrt((df['latitude'] - city_center[0])**2 + (df['longitude'] - city

# Display or use the dataset with the added 'euclidean_distance' column
print(df[['latitude', 'longitude', 'euclidean_distance']])
```

```
       latitude  longitude  euclidean_distance
0      51.442010  -0.187390            0.088469
1      51.470720  -0.162660            0.050603
2      51.451730  -0.155560            0.062207
3      51.428650  -0.157250            0.084077
4      51.470220  -0.170080            0.056302
...          ...        ...                 ...
81432  51.366580  -0.192310            0.154893
81433  51.364910  -0.179484            0.151574
81438  51.380030  -0.192270            0.142757
81439  51.377965  -0.157532            0.132806
81443  51.368667  -0.190865            0.152395

[62051 rows x 3 columns]
```

TOTAL REVENUE

In [703…
```
# Create a new column 'total_review' as the product of 'no_of_reviews' and 'price'

df['total_revenue'] = df['number_of_reviews'] * df['price']  # Assuming price is in some currency
```

AMENITIES COUNT

In [704…
```
# Calculate the number of items in each list and create a new column
df['amenities_count'] = df['amenities'].apply(lambda x: len(eval(x)))
```

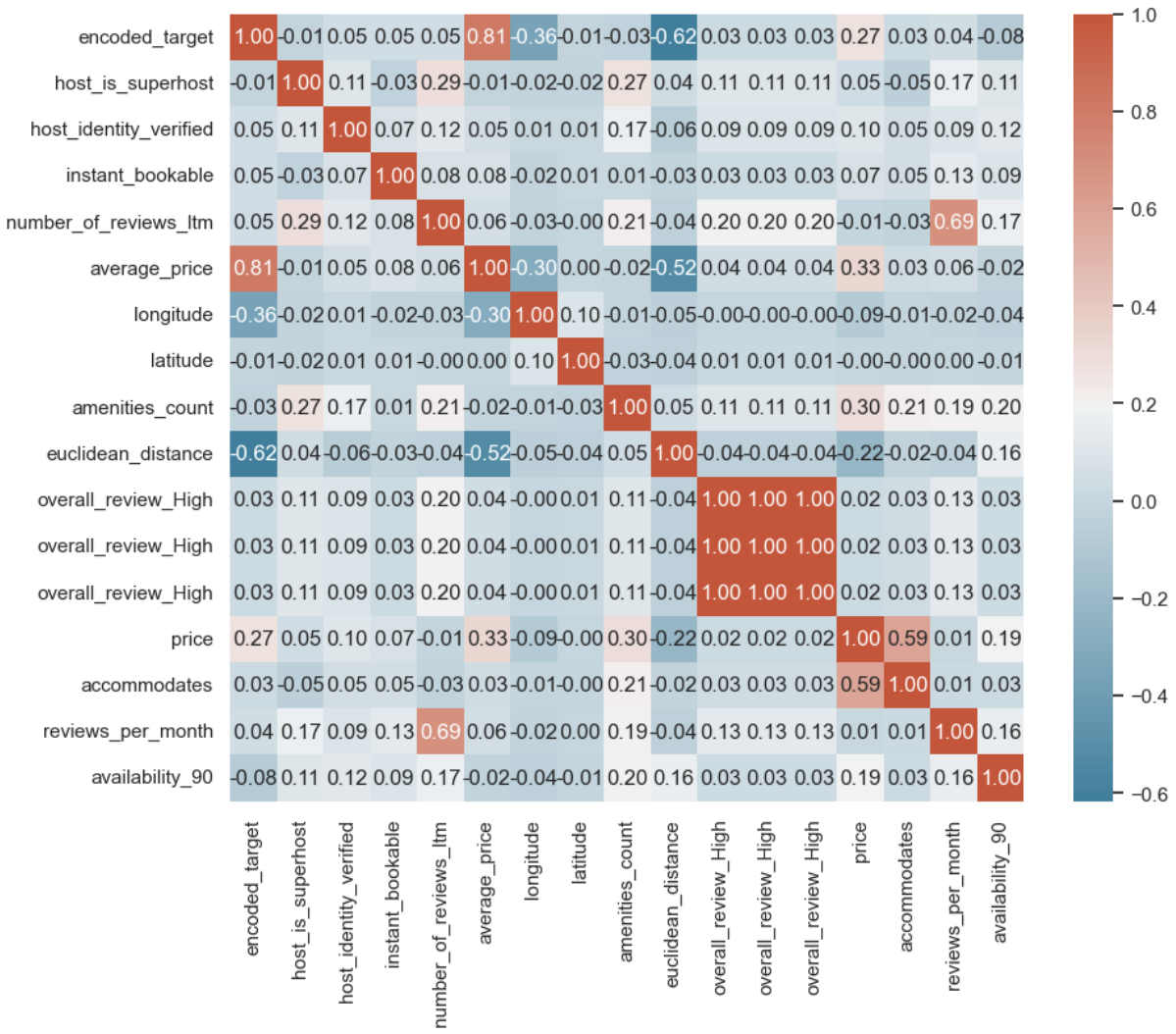
In [705…
```
# Define a mapping of classes to numerical values based on their hierarchy
class_mapping = {'Bad': 0, 'Average': 1, 'Good': 2}

# Encode the 'target' column using the defined mapping
df['encoded_target'] = df['success_metric'].map(class_mapping)
```

MODEL BUILDING

In [707…
```
new_df = df[['encoded_target','host_is_superhost','host_identity_verified','instant_bookable','nu
            ,'longitude','latitude','amenities_count','euclidean_distance','overall_review_High'
            'reviews_per_month','availability_90','number_of_reviews','success_metric']]  # Sele
new_df.shape
new_df = new_df[new_df['host_is_superhost'].notnull()]
```

In [547…
```
import seaborn as sns
corr_mat=new_df.corr(method='pearson')
cmap=sns.diverging_palette(230,20,as_cmap=True)
sns.heatmap(data=corr_mat,annot=True,fmt='.2f',cmap=cmap)
```

Out[547… <Axes: >



We made a new data frame that has features in accordance to our refined hypothesis. I plotted their heat map to analyze if they have any multicollinearity.Did manual feature selection using vif scores. After multiple trials and different combinations of features I settled for 8 features that were giving decent accuracy.

LOGISTIC REGRESSION:

In [715…
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
# Assuming you have a dataset 'df' with features and a target variable
```

```
# Replace 'features' with the columns you want to use as features and 'target' with the target co
#features = ['host_is_superhost','average_price','instant_bookable','amenities_count', 'euclidean
#target = ['success_metric']  # Replace with your target column
features= ['host_is_superhost','average_price','instant_bookable','amenities_count', 'euclidean_d
           'reviews_per_month']
target=['encoded_target']
# Assuming 'X' contains features and 'y' contains the target variable
X = new_df[features]
y = new_df[target]


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit on training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Logistic Regression model
log_reg = LogisticRegression()

# Fit the model on the training data
log_reg.fit(X_train, y_train)

# Predict on the test data
y_pred = log_reg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report and confusion matrix
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\validation.py:
1183: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy: 0.67
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.46      0.54      3062
           1       0.57      0.70      0.63      4491
           2       0.80      0.79      0.79      4718

    accuracy                           0.67     12271
   macro avg       0.68      0.65      0.66     12271
weighted avg       0.68      0.67      0.67     12271

Confusion Matrix:
[[1396 1423  243]
 [ 622 3160  709]
 [  56  952 3710]]
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logist
ic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

CHECKING RESULTS AFTER DROPPING THE FEATURE:

In [717…

```
features= ['host_is_superhost','average_price','instant_bookable','amenities_count', 'euclidean_d
target=['encoded_target']
# Assuming 'X' contains features and 'y' contains the target variable
X = new_df[features]
y = new_df[target]


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit on training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Logistic Regression model
log_reg = LogisticRegression()

# Fit the model on the training data
log_reg.fit(X_train, y_train)

# Predict on the test data
y_pred = log_reg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report and confusion matrix
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\validation.py:
1183: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy: 0.88
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      3062
           1       0.78      0.93      0.85      4491
           2       0.98      0.90      0.94      4718

    accuracy                           0.88     12271
   macro avg       0.90      0.87      0.88     12271
weighted avg       0.89      0.88      0.88     12271

Confusion Matrix:
[[2373  689    0]
 [ 189 4193  109]
 [   0  460 4258]]
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logist
ic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

OBSERVATIONS FOR LOGISTIC REGRESSION MODEL AFTER FEATURE SELECTION:

By dropping just one feature, 'reviews_per_month', the accuracy of the logistic regression model improved significantly from 67 to 88 percent.

1. This was done by doing the manual feature selection (VIF technique = variance inflation factor). A more appropriate and robust approach would be to use PCA so that features with the most significant information will be retained in the principal components and multicollinearity issues will be resolved completely.
2. Second option is to opt for Regularization techniques like Ridge or Lasso as per the model's requirements. d as 'Bad'.Average' (Row 3, Column 2): 870 instances.

RESULTS EXPLANATION OF LOGISTIC REGRESSION

WARNINGS ERRORS:

1. Covergence Warning means that our model was unable to find the optimal solution within the maximum number of iterations that were allowed to it. By setting the parameter max_iter in our logistic regression model with a higher value will allow our solver to run for more number of times and it might find an optimal solution that was generated.
2. Solver Change means we can opt for different solver like SAGA, here our model is using lbfgs What a different solver can do is they can allow us to perform regulaerization techniques like Ridge or Lasso regression and also handle penalties that comes while applying these techniques. And our model might have performed differently.

Accuracy (Overall Correct Predictions): The reported accuracy of 0.88 states the proportion of correctly predicted results over the total number of results in the dataset.

CONFUSION MATRIX

Row 1 ('Good' class):

True Positives (TP): 2373 results correctly classified as 'Good'.

False Positives (FP): 689 results were actually 'Average' but they were misclassified as 'Good'.

Row 2 ('Average' class):

True Positives (TP): 4193 results correctly classified as 'Average'.

False Positives (FP): 189 results are actually 'Good' and 109 results are actually 'Bad' but they were misclassified as 'Average'.

Row 3 ('Bad' class):

True Positives (TP): 4258 results correctly classified as 'Bad'.

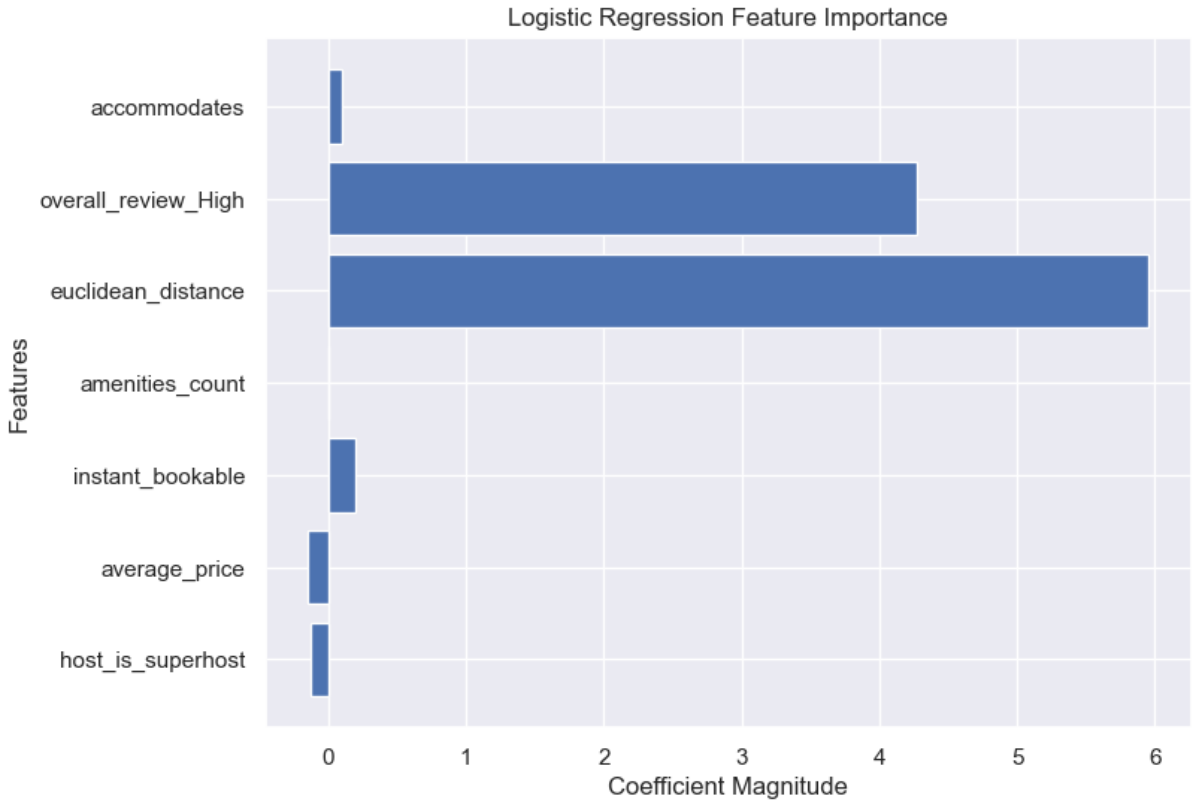
False Positives (FP): 460 results actually 'Average' but they were misclassified as 'Bad'.

In [718…

```
# Assuming 'X' contains features and 'y' contains the target variable
X = new_df[features]
y = new_df[target]

# Initialize the Logistic Regression model
log_reg = LogisticRegression()

# Fit the model on the entire data
log_reg.fit(X, y)

# Get feature importances from coefficients
importance = log_reg.coef_[0]

# Plotting feature importance
plt.figure(figsize=(8, 6))
plt.barh(features, importance)
plt.xlabel('Coefficient Magnitude')
```

```
plt.ylabel('Features')
plt.title('Logistic Regression Feature Importance')
plt.show()
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\validation.py:
1183: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logist
ic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```



INTERPRETING FEATURE IMPORTANCE

1. Euclidean distance, that is proximity of measurement of distance from city's centre has the most impact on listing being successful, this output is correctly inline with our hypothesis. The more closer the listing is to city's centre the higher are chances for it being classified as successful in terms of performance.
2. overall review high means the listings that were recieving high ratings actually belongs to the category of neighborhoods that were close to the city centre and hence resulting in high occupancy or being rented frequently.
3. As we saw in our EDA, instant bookable plot showed that it was slightly impacting our listings success as category 4 neighborhood was offering this facility more than others. This also justifies our refined hypothesis.
4. Accomodates means how much people a listing can accomodate influences the listing's chance of being rented out. The more accomodation it offers the higher the chances are for it to be successful.
5. Average price is also important but not very significant, this means that pricing your listing according to the market competitive price does not play much of a role in making our listing successful. This goes against our assumed hypothesis. In future analysis we can drop this column.

DECISION TREE

In [710…

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

features= ['host_is_superhost','price','instant_bookable','amenities_count', 'euclidean_distance'
target=['encoded_target']
# Assuming 'X' contains features and 'y' contains the target variable
X = new_df[features]
y = new_df[target]

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit on training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Decision Tree Classifier model
decision_tree = DecisionTreeClassifier()

# Fit the model on the training data
decision_tree.fit(X_train, y_train)

# Predict on the test data
y_pred = decision_tree.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report and confusion matrix
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.56
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.65      0.65      3062
           1       0.48      0.47      0.47      4491
           2       0.59      0.60      0.60      4718

    accuracy                           0.56     12271
   macro avg       0.57      0.57      0.57     12271
weighted avg       0.56      0.56      0.56     12271

Confusion Matrix:
[[1985  740  337]
 [ 766 2094 1631]
 [ 315 1563 2840]]
```

Did manual feature selection to identify the features that will give better performance of decision tree model.

In [711…

```
features= ['instant_bookable','number_of_reviews',
            'longitude','latitude','overall_review_High','price']
target=['encoded_target']
# Assuming 'X' contains features and 'y' contains the target variable
X = new_df[features]
y = new_df[target]

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit on training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Decision Tree Classifier model
decision_tree = DecisionTreeClassifier()

# Fit the model on the training data
decision_tree.fit(X_train, y_train)

# Predict on the test data
y_pred = decision_tree.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report and confusion matrix
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.99
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3062
           1       0.99      0.99      0.99      4491
           2       0.99      0.99      0.99      4718

    accuracy                           0.99     12271
   macro avg       0.99      0.99      0.99     12271
weighted avg       0.99      0.99      0.99     12271

Confusion Matrix:
[[3037   18    7]
 [  29 4434   28]
 [   5   31 4682]]
```

RESULTS EXPLANATION OF DECISION TREE

The accuracy increased drastically to 99 percent but this indicates our model has overfitting the data

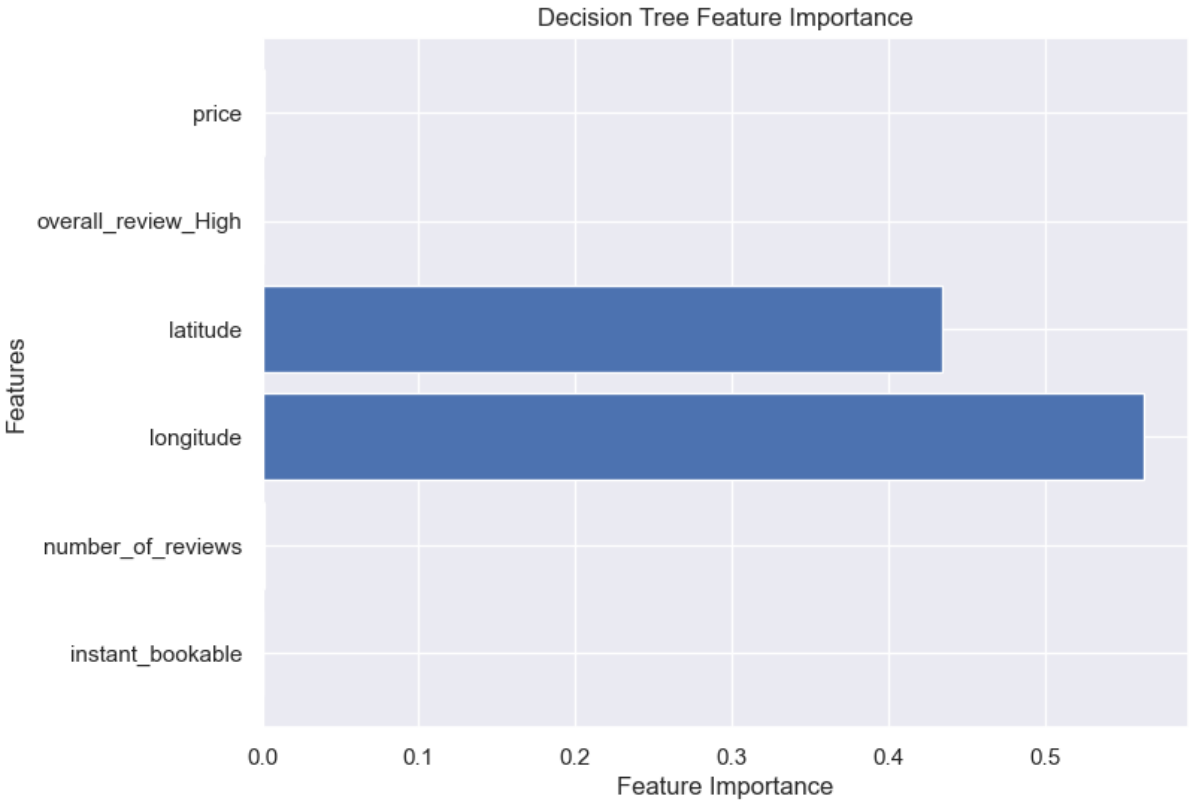
1. Decision Trees have the tendency to overfit if the complexity is not controlled.
2. By adjusting hyperparameters that control the complexity of the tree, such as max depth, minimum samples per leaf, or maximum number of features considered for splitting nodes.
3. Pruning techniques can be used to limit the growth of the tree and prevent overfitting.

The above given confusion matrix can be interpreted in the similar way as it was explained for the logistic regression model.

In [712…

```
# Assuming 'X' contains features and 'y' contains the target variable
X = df[features]
y = df[target]

# Initialize the Decision Tree Classifier model
decision_tree = DecisionTreeClassifier()

# Fit the model on the entire data
decision_tree.fit(X, y)
# Get feature importances
```

```
importance = decision_tree.feature_importances_

# Plotting feature importance
plt.figure(figsize=(8, 6))
plt.barh(features, importance)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Decision Tree Feature Importance')
plt.show()
```



INTERPRETING FEATURE IMPORTANCE

1. The longitude and latitude features are dominating over other features, emphasizing that decision tree is heavily relying on the location of listing.
2. These co ordinates can be replace by euclidean distance as well making our hypothesis valid that location proximity of listing w.r.t to centre of city plays a crucial role in success prediction

CONCLUDING REMARKS

SUMMARY OF MAIN FINDINGS & EMPHASIZE ON INSIGHTS

1. INSIGHTS FROM DATA AFTER DOING EDA:

Neighborhoods that are closer to city's centre have higher revenue generation Neighborhoods that are rented more often lie in the centre of city Super host status does not guarantee or improve the chances of our listing being ccupied Instant booking option does imoact this Thse all insights were verified from our graphs and later by our model results as well.

2. INSIGHTS FROM FEATURE IMPORTANCE OF LOGISTIC REGRESSION

Features such as location , a listing thats been reviewed higher and no of people it can accomodates plays an important role in listing's success.

3. INSIGHTS FROM FEATURE IMPORTANCE OF DECISION TREE

Here location also turned out to be the key feature, but our model was over fitting so we need to tune it down and then identify more features that are contributing towards listing's success.

4. INSIGHTS FROM THE REVIEWS AND NEIGHBORHOOD DATA SETS

We were given two other data sets as well for analysis.

1. Neighborhood Data set: This was of no use as it had only one column and can not be merged. Moreover we had similar columns in listings data set that were providing sufficient information to us so it was not included.
2. Reviews Data set: It was said in assignment's outline that this can be used for analyzing the number of reviews a listings gets and later can be used as a proximity to measure its success. We had similar column in listings data set so there was no point of including it. Later on after merging the data sets in a separate file, I realized that by merging the two data frames what information could be added was seasonality trends. It might be helpful to analyze what are the potential time/months/seasons of the year when certain properties are being reviewed more often.

CRUCIAL RESULTS & THEIR SIGNIFICANCE

The most crucial result we derived from the above analysis was about using decision tree models.

1. I used conventional method of test-train split for generalizing it over unseen data. But it is advisable that one should tune it after using this method.
2. The other possibility can be to use k fold cross validation.

One more crucial insight I observed was that I went for manual feature selection for removing multicollinearity so that model's performance can be improved. The better way to do is using PCA technique or to opt for Regularization. This point can be a take away from the analysis that might be due to manual feature selection the decision tree model was overfitting.

LIMITATIONS AND AREAS OF IMPROVEMENT FOR PROPOSED MODEL

1. We did encoding of different categorical variables in our data set without having sufficient domain knowledge about London's air bnb market. The outcomes we think were right might not be a good interpretation of that. Sometimes opting for incorrecting encoding techniques also leads to bias in resulting models. So we can not be 100% sure about our model's obtained performance.
2. The used models have their limitations as discussed above several times, in the longer run more efficient models like Random forest or XG Gradient Boosting or Deep Learning/ data mining techniques can be used for more robust performance in the future.
3. As stated above time series trend analysis can be done by merging the reviews data set.
4. We dropped some columns that had text information in them, later by doing text analysis ('Sentiment Analysis') using NLP techniques we might be able to derive more features that have an impact on a listing's success.
5. We encoded amenities by counting them, this was a very limited approach to incorporate it into our model, I came across an analysis where an airbnb employee encoded them with appropriate knowledg into different bins. Those bins were representing out of all the amenities which were actually of an impact toward's listings success. That was actually the reason our models were not really picking up amenities count as an important feature. Because sometimes properties even with 10 very essential amenities are performing better than the ones that have a count of over 40 amenities that were not so very useful

There are some warnings while running the results, by replacing the existing commands with the suggested ones to match future versions of python these can be avoided.