AIRBNB

Affordable accomodations, friendly staff, and many more.

In [1]:

```
# Importing Required Libraries

# import 'Numpy'
import numpy as np

# import 'Pandas'
import pandas as pd

# import subpackage of Matplotlib
import matplotlib.pyplot as plt

# import color package from matplotlib
from matplotlib.colors import ListedColormap

# import 'Seaborn'
import seaborn as sns

# to suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')

# display all columns of the dataframe
pd.options.display.max_columns = None

# import label encoder , ordinal encoder , onehot encoder
from sklearn.preprocessing import LabelEncoder,OrdinalEncoder,OneHotEncoder

# import target encoder
from category_encoders import TargetEncoder, WOEEncoder

# import stats for performing statistical tests
import scipy.stats as stats

# import nltk
import nltk

# import wordlcoud, stopwords
from wordcloud import WordCloud,STOPWORDS

# import regular expression
import re

# import ast
import ast

# import train-test split
from sklearn.model_selection import train_test_split

# import PowerTransformer
from sklearn.preprocessing import PowerTransformer

# import StandardScaler
from sklearn.preprocessing import StandardScaler

# import vif
from statsmodels.stats.outliers_influence import variance_inflation_factor

# import Tfidf vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```



```

# import KNNimputer
from sklearn.impute import KNNImputer

# import iterative imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge

# import SMOTE to create synthetic data
from imblearn.over_sampling import SMOTE

# import various functions from sklearn
from sklearn.metrics import accuracy_score,roc_curve,roc_auc_score,classification_r
from sklearn.metrics import f1_score,precision_score,recall_score

# import GridSearchCV
from sklearn.model_selection import KFold,GridSearchCV

# import statsmodels
import statsmodels.api as sma

# Import Kmeans
from sklearn.cluster import KMeans

# import Linear Regression
from sklearn.linear_model import LinearRegression

# import Logistic Regression
from sklearn.linear_model import LogisticRegression

# import DecisionTree Classifier
from sklearn.tree import DecisionTreeClassifier

# import tree to visualize DecisiontTree
from sklearn import tree

# import RandomForest Classifier
from sklearn.ensemble import RandomForestClassifier

# import AdaBoost Classifier
from sklearn.ensemble import AdaBoostClassifier

# import GradientBoosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

# import XtremeGradientBoost Classifier
from xgboost import XGBClassifier

# import Catboost Classifier
from catboost import CatBoostClassifier

# import pickle
import pickle

```

In [2]:

```
# set the plot size using 'rcParams'
# once the plot size is set using 'rcParams', it sets the size of all the forthcomin
# pass width and height in inches to 'figure.figsize'
plt.rcParams['figure.figsize'] = [15,8]

# Creating custom color
colors = ['#97c1ee','#DCDCDC','#ee9997','#eec497','#c1ee97','#86c5da','#97edee','#ee
```

Reading the dataset and viewing the first 10 rows of it.

In [3]:

```
pd.set_option('display.max_columns',35)

df_airbnb = pd.read_csv('Listings.csv', encoding = 'iso-8859-1')

df_airbnb.head()
```

Out[3]:

	listing_id	name	host_id	host_since	host_location	host_response_time	host_resp
0	281420	Beautiful Flat in le Village Montmartre, Paris	1466919	03-12-2011	Paris, Ile-de-France, France	NaN	
1	3705183	39 mÃ□Â² Paris (Sacre CÃ□ â□□ur)	10328771	29-11-2013	Paris, Ile-de-France, France	NaN	
2	4082273	Lovely apartment with Terrace, 60m2	19252768	31-07-2014	Paris, Ile-de-France, France	NaN	
3	4797344	Cosy studio (close to Eiffel tower)	10668311	17-12-2013	Paris, Ile-de-France, France	NaN	
4	4823489	Close to Eiffel Tower - Beautiful flat : 2 rooms	24837558	14-12-2014	Paris, Ile-de-France, France	NaN	

Checking the shape/dimension of the dataset

In [4]:

```
print(f'The dataset has {df_airbnb.shape[0]} rows and {df_airbnb.shape[1]} columns')
```

```
The dataset has 279712 rows and 33 columns
```

In [5]:

```
print(f'Dimension of the dataset is {df_airbnb.ndim}')
```

```
Dimension of the dataset is 2
```

Checking the datatype, number of non null values and name of each variable in the dataset.

In [6]:

```
df_airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 279712 entries, 0 to 279711
Data columns (total 33 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   listing_id                    279712 non-null  int64  
 1   name                          279539 non-null  object 
 2   host_id                       279712 non-null  int64  
 3   host_since                    279547 non-null  object 
 4   host_location                 278872 non-null  object 
 5   host_response_time            150930 non-null  object 
 6   host_response_rate            150930 non-null  float64
 7   host_acceptance_rate          166625 non-null  float64
 8   host_is_superhost             279547 non-null  object 
 9   host_total_listings_count     279547 non-null  float64
 10  host_has_profile_pic          279547 non-null  object 
 11  host_identity_verified        279547 non-null  object 
 12  neighbourhood                 279712 non-null  object 
 13  district                      37012 non-null   object 
 14  city                          279712 non-null  object 
 15  latitude                      279712 non-null  float64
 16  longitude                     279712 non-null  float64
 17  property_type                 279712 non-null  object 
 18  room_type                     279712 non-null  object 
 19  accommodates                  279712 non-null  int64  
 20  bedrooms                      250277 non-null  float64
 21  amenities                     279712 non-null  object 
 22  price                         279712 non-null  int64  
 23  minimum_nights                279712 non-null  int64  
 24  maximum_nights                279712 non-null  int64  
 25  review_scores_rating          188307 non-null  float64
 26  review_scores_accuracy        187999 non-null  float64
 27  review_scores_cleanliness     188047 non-null  float64
 28  review_scores_checkin         187941 non-null  float64
 29  review_scores_communication   188025 non-null  float64
 30  review_scores_location        187937 non-null  float64
 31  review_scores_value           187927 non-null  float64
 32  instant_bookable              279712 non-null  object 
dtypes: float64(13), int64(6), object(14)
memory usage: 70.4+ MB
```


Checking for the missing values. Displaying number of missing values per column

In [7]:

```
missing_values_before = pd.DataFrame({'Count of Missing values': df_airbnb.isnull().
                                      'Percentage of Missing values': round((df_airbnb.isnul

missing_values_before
```

Out[7]:

	Count of Missing values	Percentage of Missing values
listing_id	0	0.00
name	173	0.06
host_id	0	0.00
host_since	165	0.06
host_location	840	0.30
host_response_time	128782	46.04
host_response_rate	128782	46.04
host_acceptance_rate	113087	40.43
host_is_superhost	165	0.06
host_total_listings_count	165	0.06
host_has_profile_pic	165	0.06
host_identity_verified	165	0.06
neighbourhood	0	0.00
district	242700	86.77
city	0	0.00
latitude	0	0.00
longitude	0	0.00
property_type	0	0.00
room_type	0	0.00
accommodates	0	0.00
bedrooms	29435	10.52
amenities	0	0.00
price	0	0.00
minimum_nights	0	0.00
maximum_nights	0	0.00
review_scores_rating	91405	32.68
review_scores_accuracy	91713	32.79
review_scores_cleanliness	91665	32.77
review_scores_checkin	91771	32.81
review_scores_communication	91687	32.78
review_scores_location	91775	32.81
review_scores_value	91785	32.81
instant_bookable	0	0.00

Dropping of district , name , host_location and host_since columns

Since the district column has missing values of 86.76% we can drop that column.

In [8]:

```
1 df_airbnb.drop(columns = 'district',inplace=True)
```

In order to eliminate any potential insignificance to the target variable, we are choosing to drop the "name" column from the dataset.

In [9]:

```
1 df_airbnb.drop(columns = 'name',inplace=True)
```

To minimize redundancy in location-related variables, the decision has been made to drop the "host_location" column, as variables such as latitude, longitude, and city serve the same purpose of defining the location.

In [10]:

```
1 df_airbnb.drop(columns = 'host_location' , inplace = True)
```

In order to focus on the factors that have a more substantial impact on the target variable "instant bookable," the column "host_since" is being excluded from the dataset. This decision is based on the understanding that the "host_since" feature is not expected to contribute significantly to the prediction or analysis of the instant bookability of the listing. By removing this column, we can streamline the dataset and prioritize other variables that are more likely to influence the desired outcome.

In [11]:

```
1 df_airbnb.drop(columns = 'host_since' , inplace = True)
```

Imputing missing values

The variable host_total_listings_count gives information about how many times the host has posted a property in airbnb site. This variable should be modified as the count of listings mismatches with the host_id count.

In [12]:

```
1 mapper = df_airbnb.groupby(by = 'host_id')['host_id'].count()
2 df_airbnb['host_total_listings_count'] = df_airbnb['host_id'].map(mapper)
```

Investigating and predicting the missing host_response_time Value

For host_response_time variable values for a new customer will be considered as unknown and for null values of existing customer we could impute it with the same unknown. Hence we can impute all null values with unknown.

In [13]:

```
indices = df_airbnb[df_airbnb['host_response_time'].isnull()].index.to_list()

df_airbnb.loc[indices,['host_response_time']] = 'unknown'
```

Investigating and predicting the missing host_acceptance_rate Value

For variable like host_acceptance_rate values for a new customer will be considered as zero and for null values of existing customer we could impute it with zero. We can impute null values with zero.

In [14]:

```
indices = df_airbnb[df_airbnb['host_acceptance_rate'].isnull()].index.to_list()

df_airbnb.loc[indices , 'host_acceptance_rate'] = 0
```

Investigating and predicting the missing host_response_rate Value

For host_response_rate values for a new customer will be considered as zero and for null values of existing customer we could impute it with zero. We can impute null values with zero.

In [15]:

```
indices = df_airbnb[df_airbnb['host_response_rate'].isnull()].index.to_list()

df_airbnb.loc[indices,['host_response_rate']] = 0
```

Investigating and predicting the missing host_has_profile_pic,host_is_superhost,host_identity_verified Values

For variables like host_has_profile_pic,host_is_superhost,host_identity_verified we can impute it with false

In [16]:

```
indices = df_airbnb[df_airbnb['host_is_superhost'].isnull()].index.to_list()

df_airbnb.loc[indices,['host_is_superhost']] = 'f'
```

In [17]:

```
indices = df_airbnb[df_airbnb['host_has_profile_pic'].isnull()].index.to_list()

df_airbnb.loc[indices,['host_has_profile_pic']] = 'f'
```


In [18]:

```
indices = df_airbnb[df_airbnb['host_identity_verified'].isnull()].index.to_list()

df_airbnb.loc[indices,['host_identity_verified']] = 'f'
```

Investigating and predicting the missing bedrooms Value

To impute in bedrooms variable. We can take do groupby of room_type and take median of bedrooms for each room_type and can impute with that values

In [19]:

```
bedrooms_map = round(df_airbnb.groupby(by = 'property_type')['bedrooms'].median(),2)
indices = df_airbnb[df_airbnb['bedrooms'].isnull()].index.to_list()

df_airbnb.loc[indices,['bedrooms']] = df_airbnb['property_type'].map(bedrooms_map)
```

In [20]:

```
df_airbnb['bedrooms'].isnull().sum()
```

Out[20]:

```
1
```

In [21]:

```
df_airbnb.dropna(subset = 'bedrooms',inplace=True)
```

Investigating and predicting the missing review_scores_rating Value

In [22]:

```
df_review = df_airbnb.iloc[:,22:28]

for i in df_review.columns:
    indices = df_review[df_review[i].isnull()].index.to_list()

    mapper = df_airbnb.groupby('host_id')[i].median()

    df_airbnb.loc[indices , i] = df_airbnb['host_id'].map(mapper)
    df_review.loc[indices , i] = df_airbnb['host_id'].map(mapper)
    df_airbnb.loc[indices , 'review_scores_rating'] = df_airbnb['host_id'].map(mappe
```

In [23]:

```
df_review.isnull().sum()
```

Out[23]:

```
review_scores_accuracy         62580
review_scores_cleanliness      62540
review_scores_checkin          62639
review_scores_communication    62561
review_scores_location         62640
review_scores_value            62648
dtype: int64
```

In [24]:

```
len(df_review[df_review.isnull().all(axis=1)].index.to_list())
```

Out[24]:

```
62496
```

In [25]:

```
df_airbnb[df_review.isnull().all(axis=1)]['host_id'].nunique()
```

Out[25]:

```
54115
```

In [26]:

```
indices = df_review[df_review.isnull().all(axis=1)].index.to_list()

for i in df_review.columns:

    df_airbnb.loc[indices , i] = 0
    df_review.loc[indices , i] = 0
    df_airbnb.loc[indices , 'review_scores_rating'] = 0
```

In [27]:

```
df_review.isnull().sum()
```

Out[27]:

```
review_scores_accuracy          84
review_scores_cleanliness       44
review_scores_checkin          143
review_scores_communication     65
review_scores_location         144
review_scores_value            152
dtype: int64
```


In [28]:

```
df_airbnb['review_scores_rating'].isnull().sum()
```

Out[28]:

```
238
```

In [29]:

```
len(df_review[df_review.isnull().any(axis=1)].index.to_list())
```

Out[29]:

```
210
```

In [30]:

```
len(df_review)
```

Out[30]:

```
279711
```

In [31]:

```
len(df_airbnb)
```

Out[31]:

```
279711
```

In [32]:

```
df_review.dropna(subset = df_review.columns,inplace=True)
df_airbnb.dropna(subset = df_review.columns, inplace = True)
```

In [33]:

```
len(df_review[df_review.isnull().any(axis=1)].index.to_list())
```

Out[33]:

```
0
```

In [34]:

```
len(df_review)
```

Out[34]:

```
279501
```

In [35]:

```
len(df_airbnb)
```

Out[35]:

```
279501
```

In [36]:

```
279711 - 279501
```

Out[36]:

```
210
```

In [37]:

```
df_airbnb['review_scores_rating'].isnull().sum()
```

Out[37]:

```
28
```

In [38]:

```
df_airbnb.dropna(subset = ['review_scores_rating'],inplace = True)
```

In [39]:

```
df_airbnb.isnull().sum()
```

Out[39]:

```
listing_id                    0
host_id                       0
host_response_time            0
host_response_rate            0
host_acceptance_rate          0
host_is_superhost             0
host_total_listings_count     0
host_has_profile_pic          0
host_identity_verified        0
neighbourhood                 0
city                          0
latitude                      0
longitude                     0
property_type                 0
room_type                     0
accommodates                  0
bedrooms                      0
amenities                     0
price                         0
minimum_nights                0
maximum_nights                0
review_scores_rating          0
review_scores_accuracy        0
review_scores_cleanliness     0
review_scores_checkin         0
review_scores_communication   0
review_scores_location        0
review_scores_value           0
instant_bookable              0
dtype: int64
```

Dropping of listing_id ,host_id

listing_id and host_id variables serve as unique identifiers and do not possess substantial relevance or impact on the target variable, which in this case is the "instant bookable" attribute. By removing these columns, the dataset becomes more concise, focused, and conducive to further analysis and modeling.

In [40]:

```
df_airbnb.drop(columns = ['listing_id','host_id'] , inplace = True)
```

Checking for the missing values after treating them.Display number of missing values per column

In [41]:

```
missing_values_after = pd.DataFrame({'Count of Missing values': df_airbnb.isnull().s
                                     'Percentage of Missing values': (df_airbnb.isnull().su

missing_values_after
```

Out[41]:

	Count of Missing values	Percentage of Missing values
host_response_time	0	0.0
host_response_rate	0	0.0
host_acceptance_rate	0	0.0
host_is_superhost	0	0.0
host_total_listings_count	0	0.0
host_has_profile_pic	0	0.0
host_identity_verified	0	0.0
neighbourhood	0	0.0
city	0	0.0
latitude	0	0.0
longitude	0	0.0
property_type	0	0.0
room_type	0	0.0
accommodates	0	0.0
bedrooms	0	0.0
amenities	0	0.0
price	0	0.0
minimum_nights	0	0.0
maximum_nights	0	0.0
review_scores_rating	0	0.0
review_scores_accuracy	0	0.0
review_scores_cleanliness	0	0.0
review_scores_checkin	0	0.0
review_scores_communication	0	0.0
review_scores_location	0	0.0
review_scores_value	0	0.0
instant_bookable	0	0.0


```
Missing values has been treated
```

Creating a column named Region

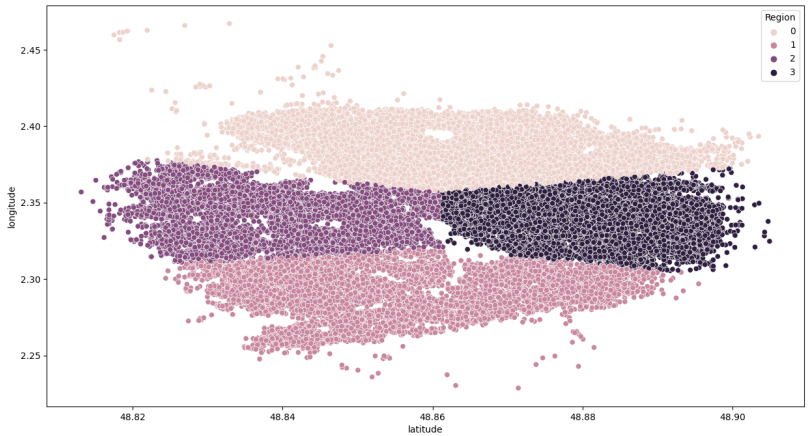
A new column named "Region" will be created by leveraging the latitude and longitude values of each city.

In [42]:

```
df_paris=df_airbnb[df_airbnb['city']=='Paris']
df_newyrk=df_airbnb[df_airbnb['city']=='New York']
df_bangkok=df_airbnb[df_airbnb['city']=='Bangkok']
df_riode=df_airbnb[df_airbnb['city']=='Rio de Janeiro']
df_sydney=df_airbnb[df_airbnb['city']=='Sydney']
df_istanbul=df_airbnb[df_airbnb['city']=='Istanbul']
df_rome=df_airbnb[df_airbnb['city']=='Rome']
df_hongkong=df_airbnb[df_airbnb['city']=='Hong Kong']
df_mexico=df_airbnb[df_airbnb['city']=='Mexico City']
df_capetwn=df_airbnb[df_airbnb['city']=='Cape Town']
```

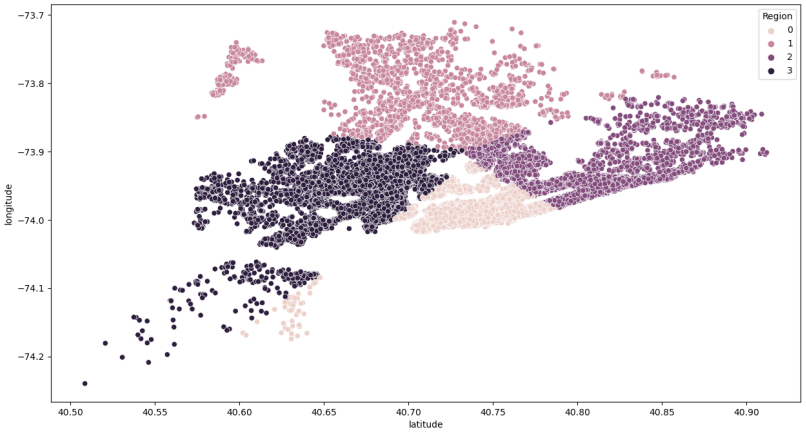
In [43]:

```
#paris
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_paris[['latitude','longitude']])
lab=kmod.labels_
df_paris['Region']=lab
sns.scatterplot(df_paris['latitude'],df_paris['longitude'],hue=df_paris['Region'])
df_paris['Region']=df_paris['Region'].map({0:'West',1:'North',2:'South',3:'East'})
```



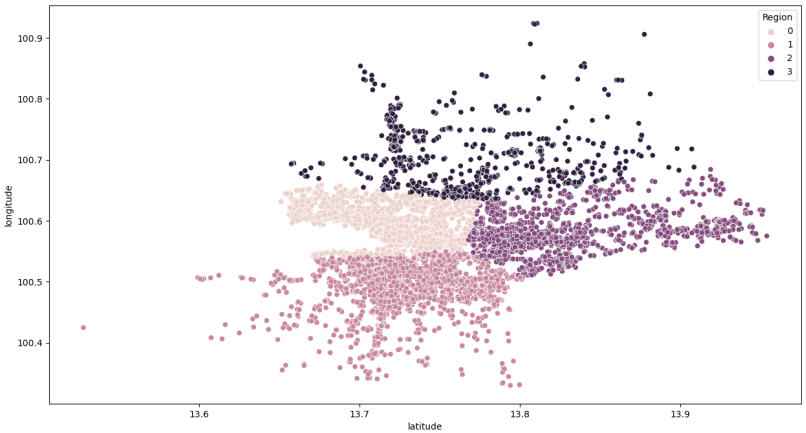
In [44]:

```
#newyork
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_newyrk[['latitude','longitude']])
lab=kmod.labels_
df_newyrk['Region']=lab
sns.scatterplot(df_newyrk['latitude'],df_newyrk['longitude'],hue=df_newyrk['Region']
df_newyrk['Region']=df_newyrk['Region'].map({0:'West',1:'East',2:'South',3:'North'})
```



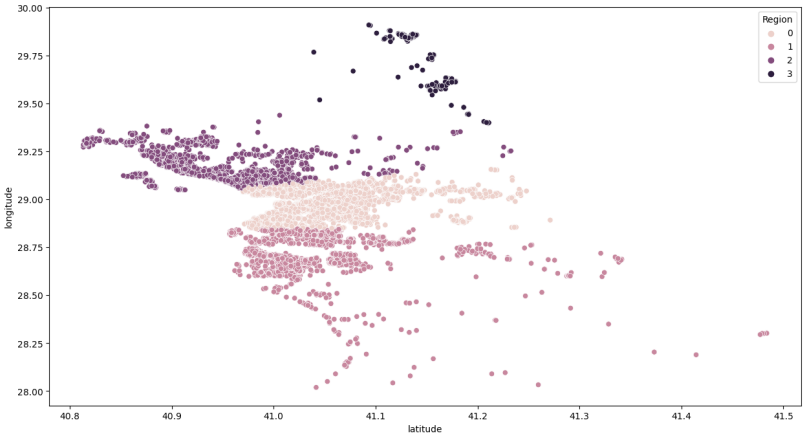
In [45]:

```
#bangkok
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_bangkok[['latitude','longitude']])
lab=kmod.labels_
df_bangkok['Region']=lab
sns.scatterplot(df_bangkok['latitude'],df_bangkok['longitude'],hue=df_bangkok['Regic
df_bangkok['Region']=df_bangkok['Region'].map({0:'East',1:'West',2:'South',3:'North'
```



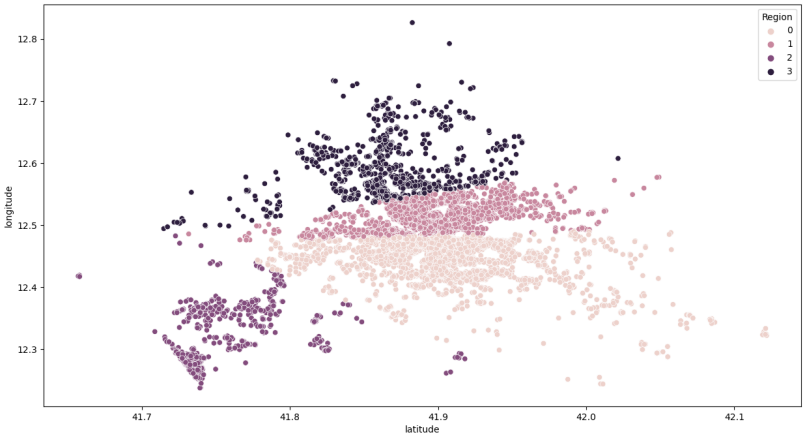
In [46]:

```
# istanbul
km=KMeans(n_clusters=4, init='k-means++')
kmod=km.fit(df_istanbul[['latitude','longitude']])
lab=kmod.labels_
df_istanbul['Region']=lab
sns.scatterplot(df_istanbul['latitude'],df_istanbul['longitude'],hue=df_istanbul['Re
df_istanbul['Region']=df_istanbul['Region'].map({0:'West',1:'South',2:'East',3:'Nort
```



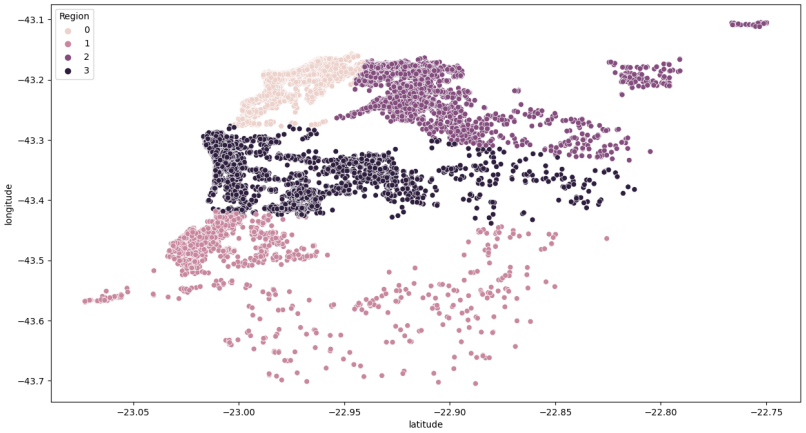
In [47]:

```
#rome
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_rome[['latitude','longitude']])
lab=kmod.labels_
df_rome['Region']=lab
sns.scatterplot(df_rome['latitude'],df_rome['longitude'],hue=df_rome['Region'])
df_rome['Region']=df_rome['Region'].map({0:'East',1:'West',2:'South',3:'North'})
```



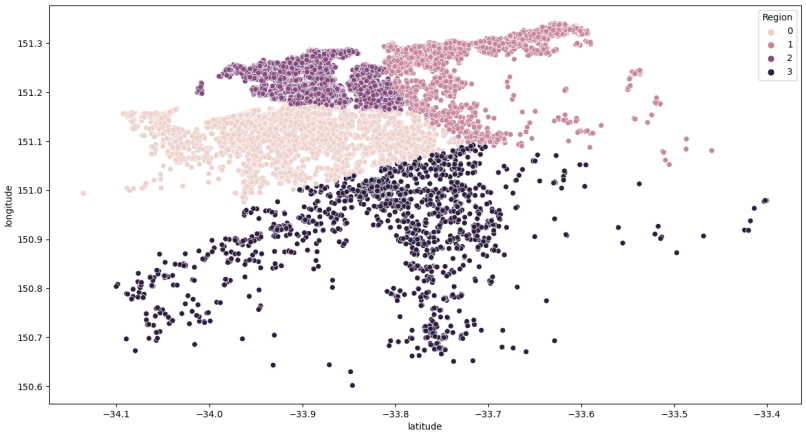
In [48]:

```
# df_riode
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_riode[['latitude','longitude']])
lab=kmod.labels_
df_riode['Region']=lab
sns.scatterplot(df_riode['latitude'],df_riode['longitude'],hue=df_riode['Region'])
df_riode['Region']=df_riode['Region'].map({0:'North',1:'West',2:'South',3:'East'})
```



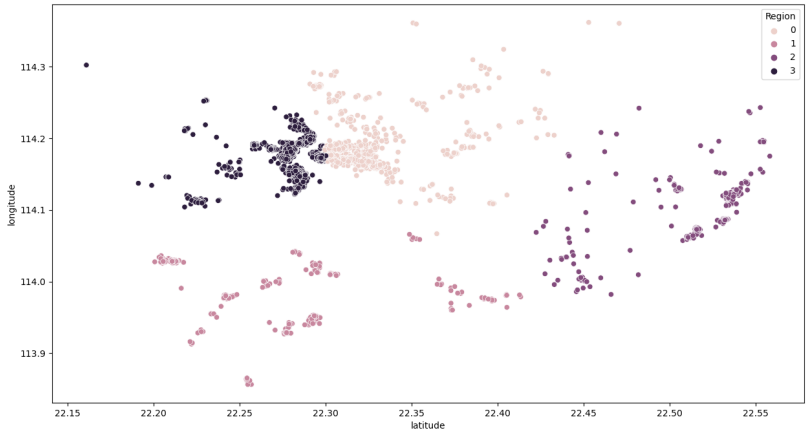
In [49]:

```
# df_sydney
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_sydney[['latitude','longitude']])
lab=kmod.labels_
df_sydney['Region']=lab
sns.scatterplot(df_sydney['latitude'],df_sydney['longitude'],hue=df_sydney['Region'])
df_sydney['Region']=df_sydney['Region'].map({0:'East',1:'West',2:'South',3:'North'})
```



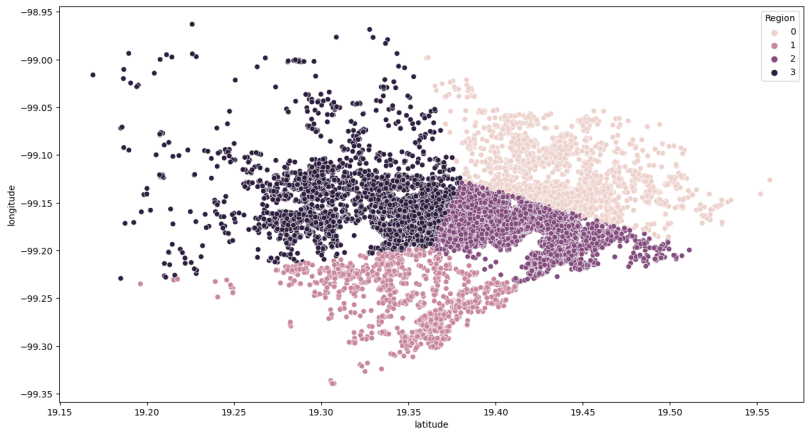
In [50]:

```
# df_hongkong
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_hongkong[['latitude','longitude']])
lab=kmod.labels_
df_hongkong['Region']=lab
sns.scatterplot(df_hongkong['latitude'],df_hongkong['longitude'],hue=df_hongkong['Re
df_hongkong['Region']=df_hongkong['Region'].map({0:'North',1:'South',2:'East',3:'Wes
```



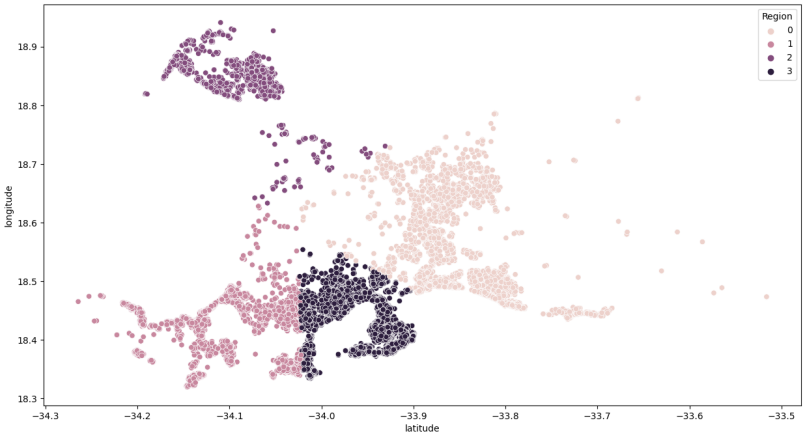
In [51]:

```
# df_mexico
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_mexico[['latitude','longitude']])
lab=kmod.labels_
df_mexico['Region']=lab
sns.scatterplot(df_mexico['latitude'],df_mexico['longitude'],hue=df_mexico['Region']
df_mexico['Region']=df_mexico['Region'].map({0:'North',1:'South',2:'West',3:'East'})
```



In [52]:

```
# df_capetown
km=KMeans(n_clusters=4, init='k-means++',random_state=20)
kmod=km.fit(df_capetwn[['latitude','longitude']])
lab=kmod.labels_
df_capetwn['Region']=lab
sns.scatterplot(df_capetwn['latitude'],df_capetwn['longitude'],hue=df_capetwn['Regic
df_capetwn['Region']=df_capetwn['Region'].map({0:'South',1:'North',2:'East',3:'West'
```



In [53]:

```
df_airbnb=pd.concat([df_paris,df_newyrk,df_bangkok,df_riode,df_sydney,df_istanbul,df
```

In [54]:

```
df_airbnb['Region'].isnull().sum()
```

Out[54]:

```
0
```


Checking for the descriptive statistics of the dataset

In [55]:

```
df_airbnb.describe(include = 'object').T
```

Out[55]:

	count	unique	top	freq
host_response_time	279473	5	unknown	128593
host_is_superhost	279473	2	f	229222
host_has_profile_pic	279473	2	t	278394
host_identity_verified	279473	2	t	201043
neighbourhood	279473	660	I Centro Storico	14868
city	279473	10	Paris	64623
property_type	279473	143	Entire apartment	138861
room_type	279473	4	Entire place	181853
amenities	279473	244843	["Long term stays allowed"]	1384
instant_bookable	279473	2	f	163924
Region	279473	4	West	103672

From above statistics we get to know :

1) There are total 279351 rows and 10 unique classes in city. In that majority class with a count of 64594 is Paris.

2) Variable neighbourhood is having 660 unique values. Out of 660 unqiue districts majority of the property is located in I Centro Storico and their count is 14868

3) Property of room_type Entire place is highest among the other 3 room_type classes with a count of 181751

4) In the target variable we can see it is a binary class and there is no class imbalance. Majority of the property has instant bookability as false with a count of 163844

5) In property_type there are 142 unique classes in that maximum count is for entire apartment of count 138786

6) For majority of the hosts there identity has been verified and they have profile picture.

Checking for the summary statistics of the dataset

In [56]:

```
1 df_airbnb.describe().T
```

Out[56]:

	count	mean	std	min	25%
host_response_rate	279473.0	0.467532	4.793144e-01	0.00000	0.00000
host_acceptance_rate	279473.0	0.493103	4.632361e-01	0.00000	0.00000
host_total_listings_count	279473.0	11.573240	4.666310e+01	1.00000	1.00000
latitude	279473.0	18.759348	3.255973e+01	-34.26440	-22.96439
longitude	279473.0	12.593154	7.307681e+01	-99.33963	-43.19803
accommodates	279473.0	3.289080	2.133675e+00	0.00000	2.00000
bedrooms	279473.0	1.464617	1.103801e+00	1.00000	1.00000
price	279473.0	609.004498	3.443148e+03	0.00000	75.00000
minimum_nights	279473.0	8.048445	3.152451e+01	1.00000	1.00000
maximum_nights	279473.0	27581.580693	7.285989e+06	1.00000	45.00000
review_scores_rating	279473.0	63.733561	4.335140e+01	0.00000	9.00000
review_scores_accuracy	279473.0	7.404756	4.076545e+00	0.00000	7.50000
review_scores_cleanliness	279473.0	7.218803	4.009373e+00	0.00000	6.00000
review_scores_checkin	279473.0	7.523238	4.114920e+00	0.00000	8.00000
review_scores_communication	279473.0	7.514853	4.115360e+00	0.00000	8.00000
review_scores_location	279473.0	7.474897	4.082281e+00	0.00000	8.00000
review_scores_value	279473.0	7.224424	3.991712e+00	0.00000	7.00000

From above statistics we get to know :

1) Average of price of properties listed is $6.08

2) Minimum nights allowed to stay is one night

3) Maximum people allowed to stay is 16 members

4) Size of bedrooms range from room type, minimum rooms available is 1 and maximum is 50

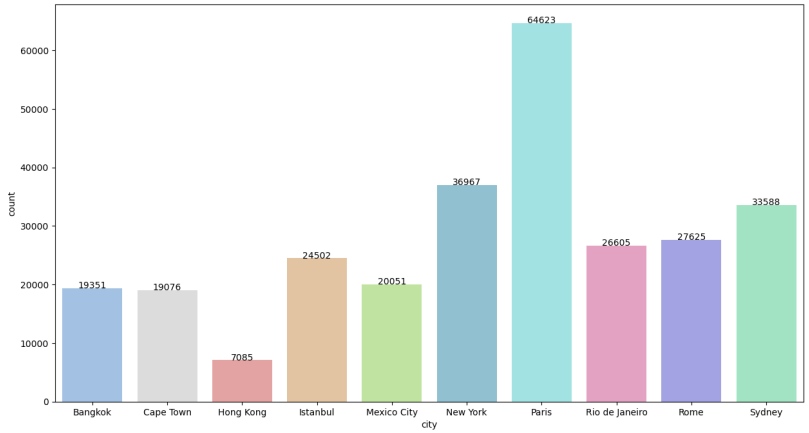
Univariate Analysis

City

In Airbnb, the city column is a categorical variable that identifies the city where a particular listing is located.

In [57]:

```
1 sns.countplot(df_airbnb['city'].sort_values(),palette = colors)
2
3 for i,v in enumerate(df_airbnb['city'].value_counts().sort_index()):
4     plt.text(x = i , y = v + 5 , s = v, ha = 'center')
```



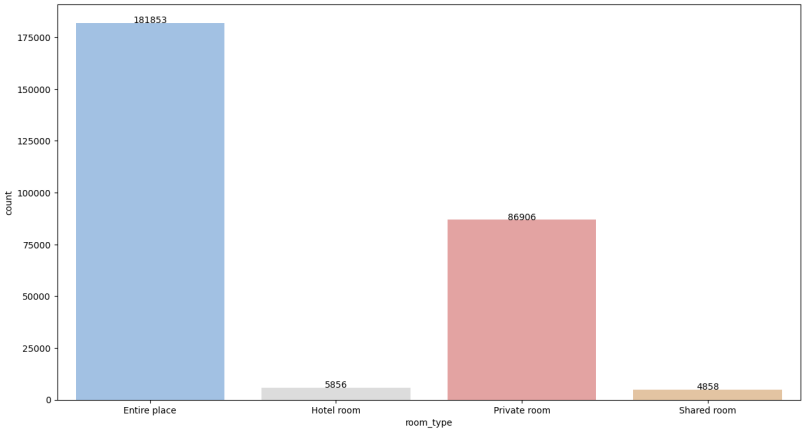
From above countplot it is clearly evident that more properties are available in Paris of count 54657 followed by NewYork 36994 and Sydney 33596. Least property is in HongKong city with total properties of 7083. Also we can see that Bangkok and Cape Town almost has same number of properties available

Room Type

Room_type describes the type of room or space that is being offered for rent. The room_type variable can be used as a feature in data analysis or modeling to predict the likelihood of a booking, and to identify the factors that are most predictive of a guest's preference for a particular room type.

In [58]:

```
sns.countplot(df_airbnb['room_type'].sort_values(),palette = colors)

for i,v in enumerate(df_airbnb['room_type'].value_counts().sort_index()):
    plt.text(x = i , y = v + 5 , s = v, ha = 'center')
```



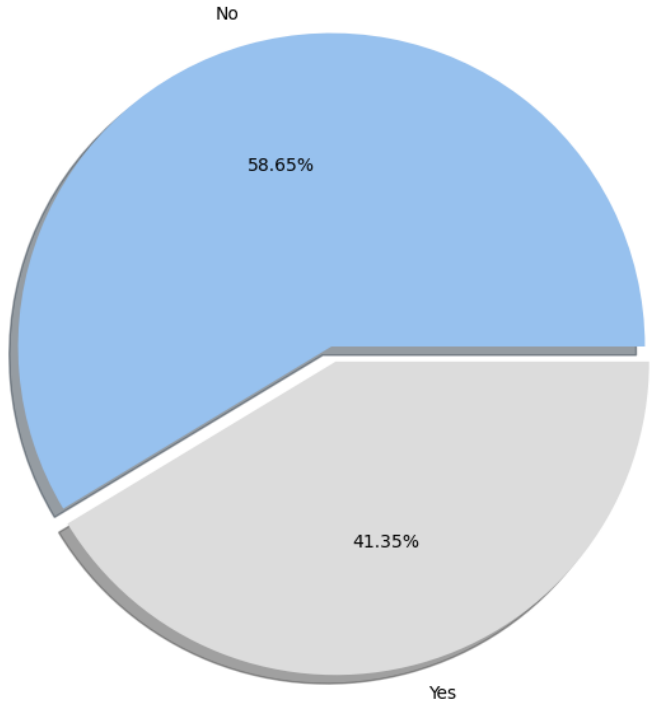
From above countplot we can infer that most of the properties available are Entire place with a count of 181886. Next to Entire place is Private room having 86945 properties. Comparatvely Shared room and Hotel room share the same number of properties

Instant_bookable

This variable can help hosts to optimize their listings for instant bookings, and to improve their overall booking rates and occupancy rates.

In [59]:

```
s = df_airbnb['instant_bookable'].value_counts()

plt.pie(s, labels = ['No','Yes'], autopct = '%.2f%%', shadow = True, explode = [0,0.
plt.show()
```



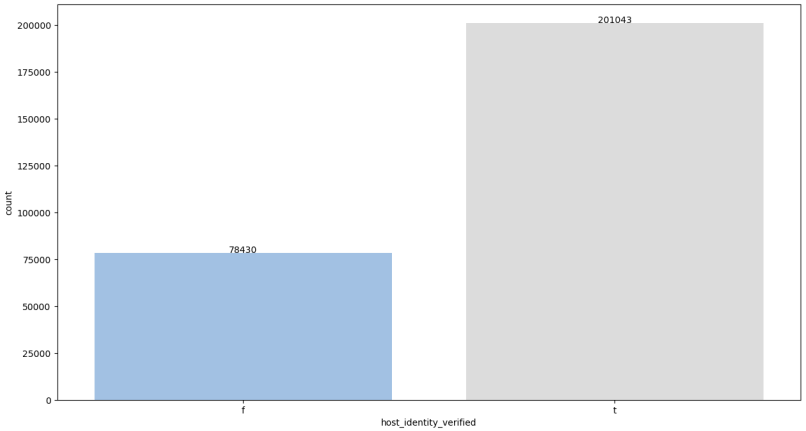
From above countplot and value_counts we can see that binary class is not in equal proportions but still it is not class imbalanced since the percentage of minority class is nearly 40%. Creating synthetic data using smote technique will not required

Host_identity_verified

This variable shows whether the host's identity has been verified by Airbnb or not. This variable can be useful for guests to assess the trustworthiness and safety of a potential host. A host with a verified identity is generally perceived as more trustworthy and reliable, as it suggests that Airbnb has confirmed their identity and has taken steps to ensure that they are who they claim to be.

In [60]:

```
sns.countplot(df_airbnb['host_identity_verified'].sort_values(),palette = colors)

for i,v in enumerate(df_airbnb['host_identity_verified'].value_counts().sort_index()):
    plt.text(x = i , y = v + 5 , s = v, ha = 'center')
```



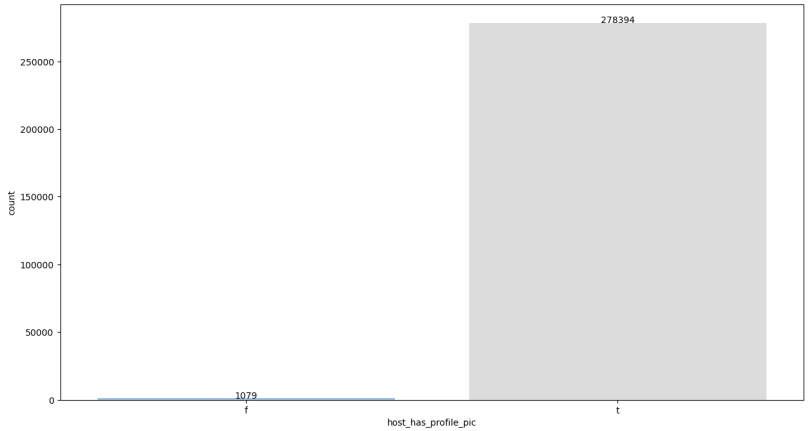
Majority of the properties posted we have host verified their idenity by Airbnb

Host_has_profile_pic

This variable shows whether the host has uploaded a profile picture or not. This variable can be useful for guests to assess the trustworthiness and professionalism of a potential host. A host with a profile picture is generally perceived as more trustworthy and reliable, as it suggests that they are invested in their Airbnb hosting and are more likely to be responsive to guests' needs.

In [61]:

```
sns.countplot(df_airbnb['host_has_profile_pic'].sort_values(),palette = colors)

for i,v in enumerate(df_airbnb['host_has_profile_pic'].value_counts().sort_index()):
    plt.text(x = i , y = v + 5 , s = v, ha = 'center')
```



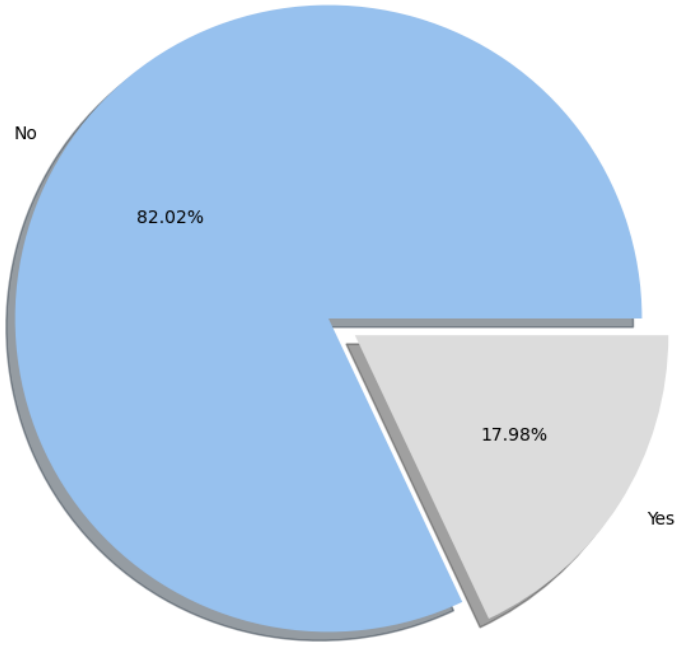
From above countplot we get to know that majority of the host has updated their profile picture

Host_is_superhost

```
"Superhost" is a highly-rated and experienced host who provides exceptional hospitality to guests. The Superhost program is designed to recognize hosts who go above and beyond to provide great experiences for their guests and to help guests find the best possible hosts
```

In [62]:

```
s = df_airbnb['host_is_superhost'].value_counts()

plt.pie(s, labels = ['No','Yes'], autopct = '%.2f%%', shadow = True, explode = [0,0.
plt.show()
```



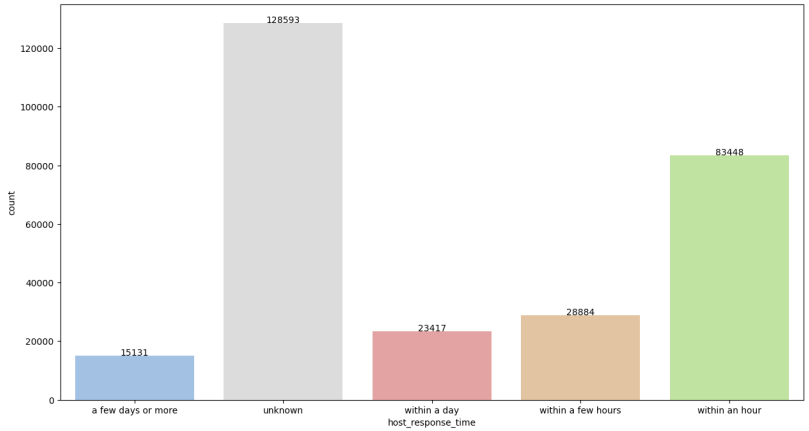
As we can see from above pie chart most of the host are not a superhost.

Host_response_time

This variable indicates how long it takes for a host to respond to a guest's booking inquiry or message. The host_response_time variable can be used as a feature in data analysis or modeling to predict the likelihood of a booking, and to identify the factors that are most predictive of a host's response time.

In [63]:

```
sns.countplot(df_airbnb['host_response_time'].sort_values(),palette = colors)

for i,v in enumerate(df_airbnb['host_response_time'].value_counts().sort_index()):
    plt.text(x = i , y = v + 5 , s = v, ha = 'center')
```



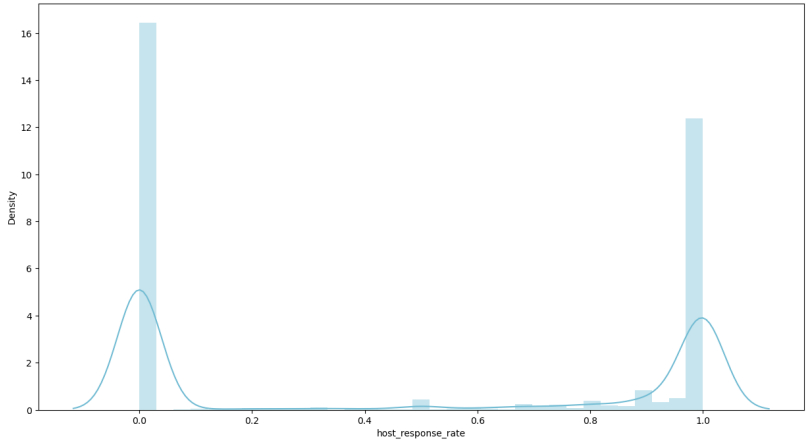
From above countplot it is clearly evident that the response rate of many hosts are unknown. But apart from unknown majority of the hosts respond within an hour or to the max they respond within a day. Only few host take a few days to respond

Host_response_rate

This variable indicates the percentage of enquiries or messages that a host responds to within a certain timeframe. And it is expressed in percentage, ranging from 0 to 100.

In [64]:

```
sns.distplot(df_airbnb['host_response_rate'],color = '#72bcd4')
plt.show()
```

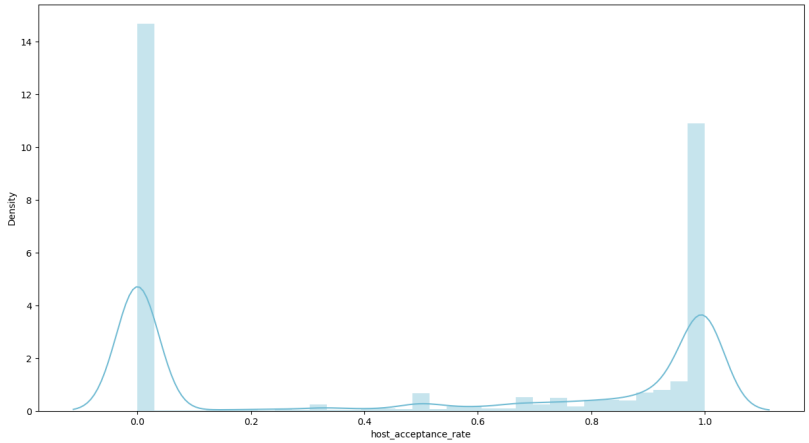


Host_acceptance_rate

This variable indicates the percentage of booking requests that a host accepts. And it is expressed in percentage, ranging from 0 to 100.

In [65]:

```
sns.distplot(df_airbnb['host_acceptance_rate'],color = '#72bcd4')
plt.show()
```

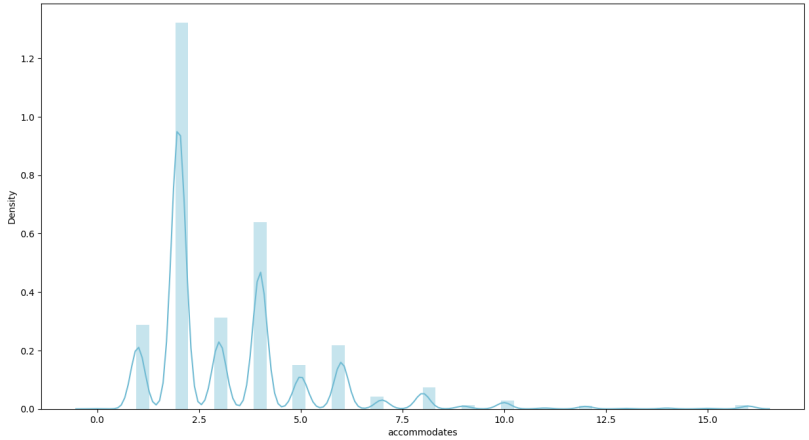


Accommodates

It indicates the maximum number of guests that a particular listing can accommodate.

In [66]:

```
sns.distplot(df_airbnb['accommodates'],color = '#72bcd4')
plt.show()
```

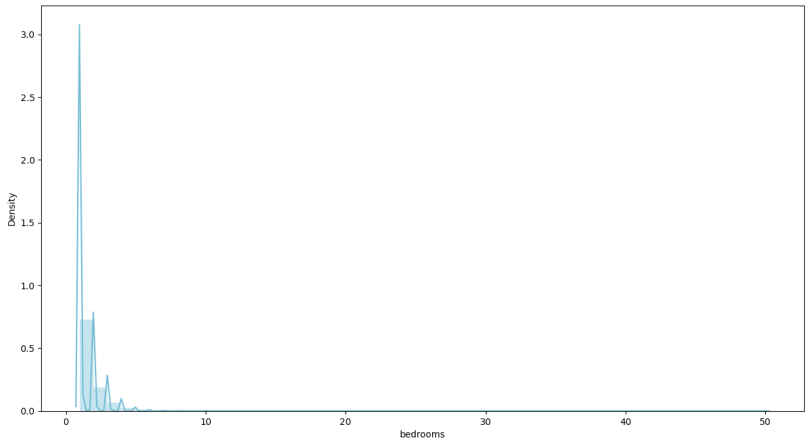


Bedrooms

This variable indicates the number of bedrooms available in a particular listing.

In [67]:

```
sns.distplot(df_airbnb['bedrooms'],color = '#72bcd4')
plt.show()
```

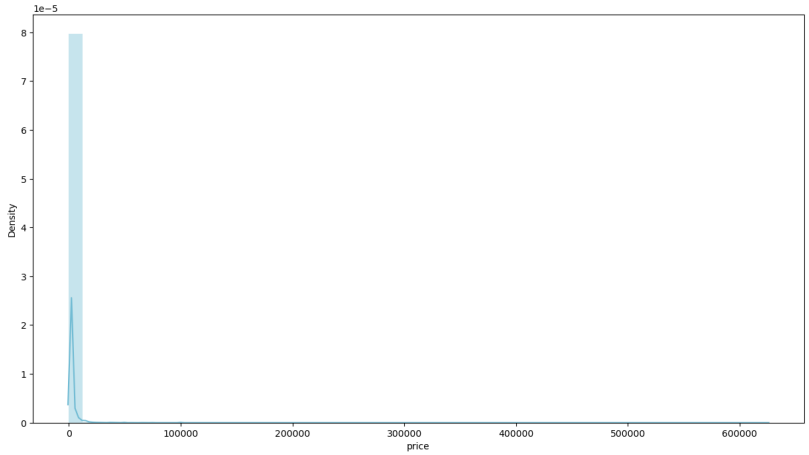


Price

It indicates the nightly price of a particular listing and it is often one of the primary factors that guests consider when choosing a listing to book.

In [68]:

```
sns.distplot(df_airbnb['price'],color = '#72bcd4')
plt.show()
```

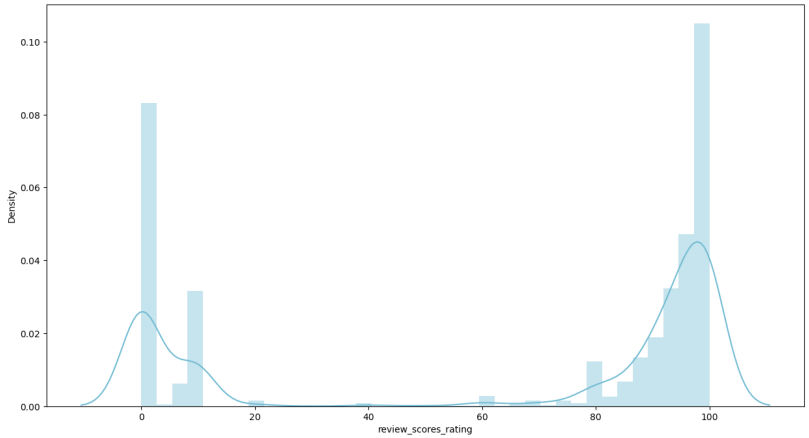


Review_scores_rating

This variable indicates the overall satisfaction rating of guests who have stayed at a particular listing and it provides insight into the quality of the listing and the experiences of previous guests.

In [69]:

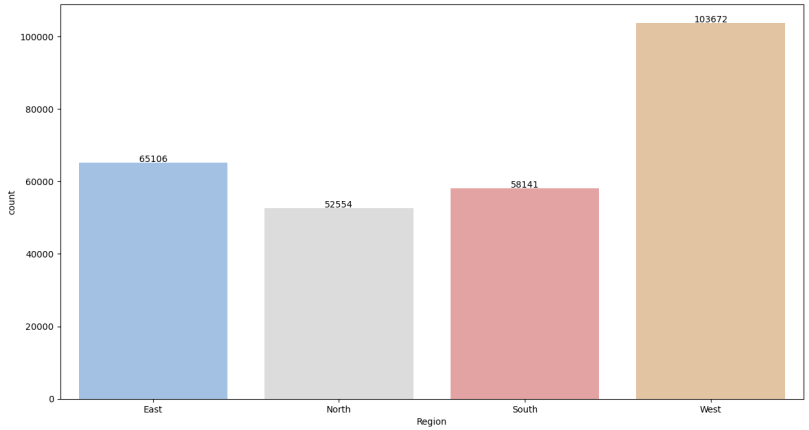
```
sns.distplot(df_airbnb['review_scores_rating'],color = '#72bcd4')
plt.show()
```



Regions

In [70]:

```
sns.countplot(df_airbnb['Region'].sort_values() , palette = colors)

for i,v in enumerate(df_airbnb['Region'].value_counts().sort_index()):
    plt.text(x = i , y = v + 200 , s = v , ha = 'center')
```

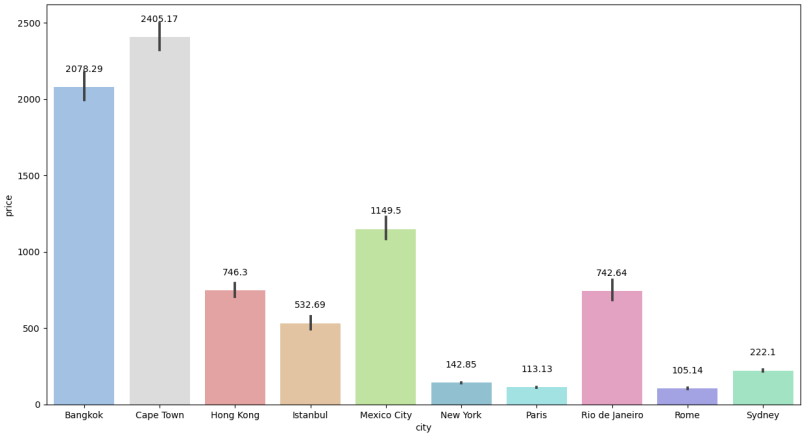


Bivariate Analysis

City vs Price

In [71]:

```
sns.barplot(x = df_airbnb['city'].sort_values(), y = df_airbnb['price'],palette = cc

for i,v in enumerate(df_airbnb.groupby('city')['price'].mean()):
    plt.text(x = i , y = v + 100, s = round(v,2) , ha = 'center')
```

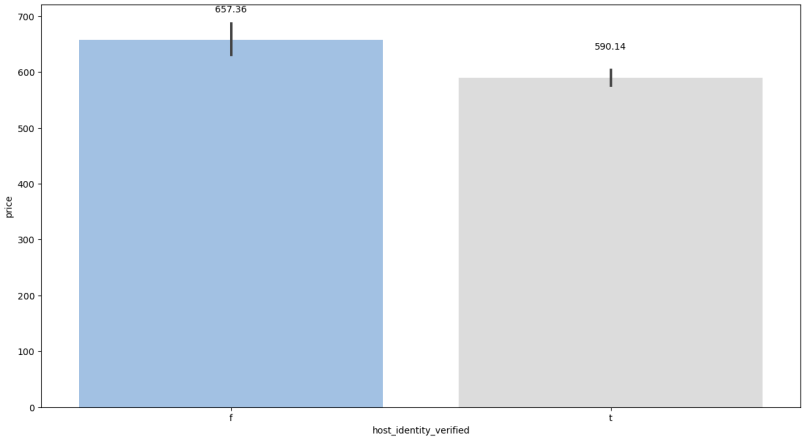


From the plot we can infer that Highest price is in Cape Town of 2405 per night followed by Bankgkok 2078 per night. Average price is in Mexico of 1149 and least price is in the cities of NewYork, Paris and Rome

Host_identity_verified vs Price

In [72]:

```
sns.barplot(x = df_airbnb['host_identity_verified'], y = df_airbnb['price'],palette

for i,v in enumerate(df_airbnb.groupby('host_identity_verified')['price'].mean()):
    plt.text(x = i , y = v + 50, s = round(v,2) , ha = 'center')
```

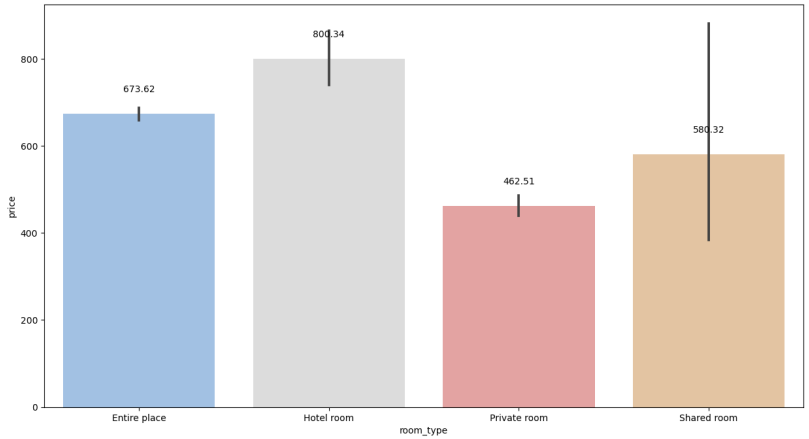


From above plot it is clearly evident that there is no much significal difference in price between having verified status or not. Price is almost same for both classes

Room_type vs Price

In [73]:

```
sns.barplot(x = df_airbnb['room_type'].sort_values(), y = df_airbnb['price'],palette

for i,v in enumerate(df_airbnb.groupby('room_type')['price'].mean()):
    plt.text(x = i , y = v + 50, s = round(v,2) , ha = 'center')
```

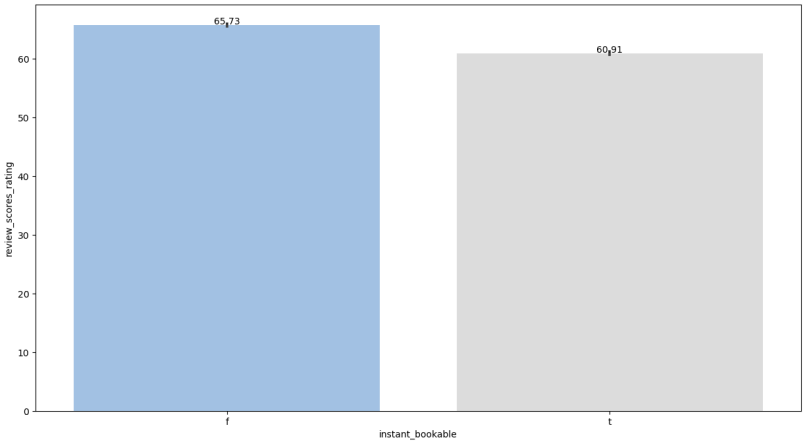


From above barplot we can see that price for Hotel room and Entire place is expensive compared to Private room and Shared room. The price of former is 800 & 673 and the latter is 462 & 580

Instant bookable vs Review score rating

In [74]:

```
sns.barplot(x = df_airbnb['instant_bookable'].sort_values(), y = df_airbnb['review_s

for i,v in enumerate(df_airbnb.groupby('instant_bookable')['review_scores_rating'].m
    plt.text(x = i , y = v + 0.1, s = round(v,2) , ha = 'center')
```

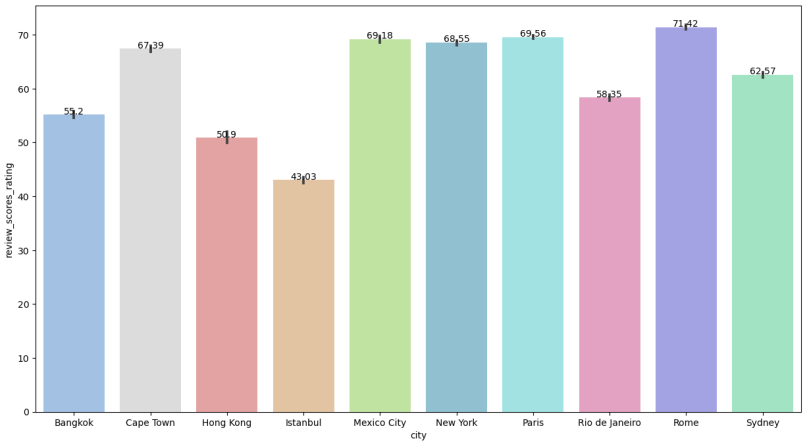


From above plot we get to know that there is not much significance difference in rating's irrespective of the instant bookable

City vs Review score rating

In [75]:

```
sns.barplot(x = df_airbnb['city'].sort_values(), y = df_airbnb['review_scores_rating

for i,v in enumerate(df_airbnb.groupby('city')['review_scores_rating'].mean()):
    plt.text(x = i , y = v + 0.1, s = round(v,2) , ha = 'center')
```

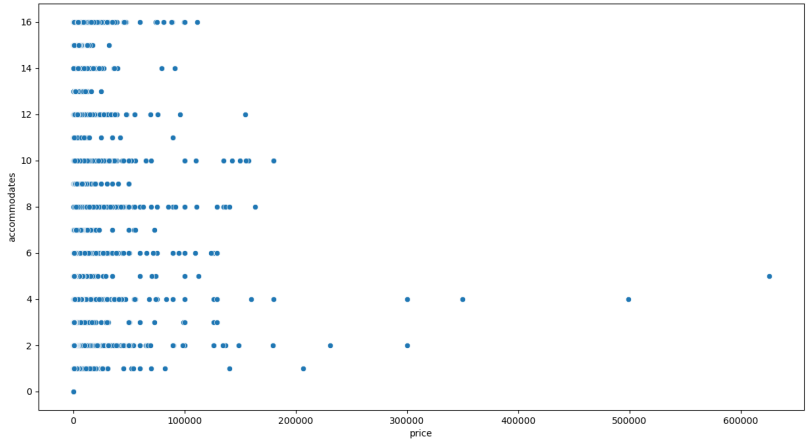


The above plot shows the distribution of rating's across different cities. We can see that there is no much diiference in the rating score among the cities

Accomodates vs price

In [76]:

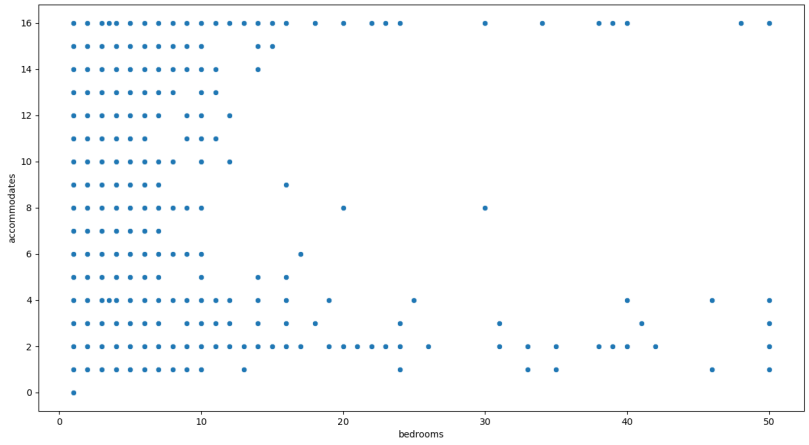
```
sns.scatterplot(x = df_airbnb['price'], y = df_airbnb['accommodates'],palette = colo
plt.show()
```



Accomodates vs Bedrooms

In [77]:

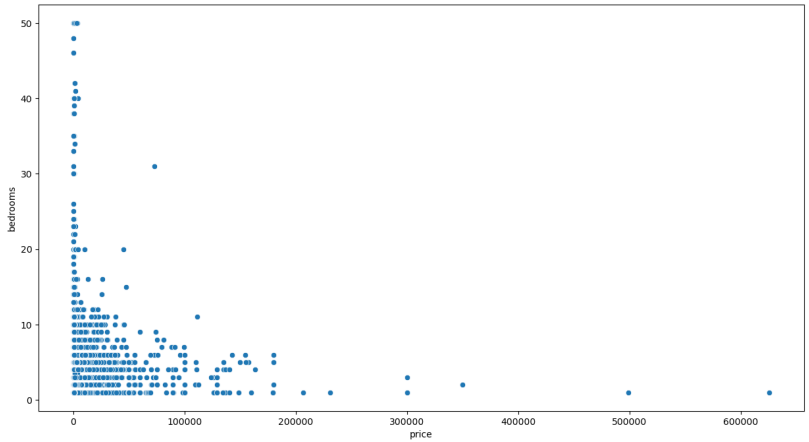
```
sns.scatterplot(x = df_airbnb['bedrooms'], y = df_airbnb['accommodates'],palette = c
plt.show()
```



Price vs Bedrooms

In [78]:

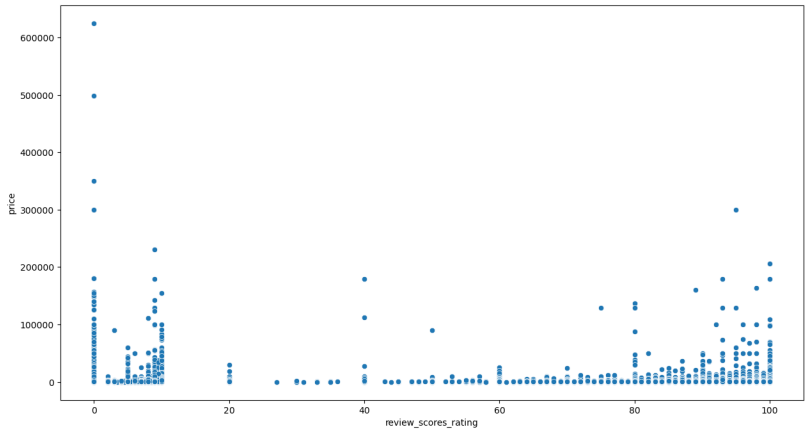
```
sns.scatterplot(x = df_airbnb['price'], y = df_airbnb['bedrooms'],palette = colors)
plt.show()
```



Price vs Review score rating

In [79]:

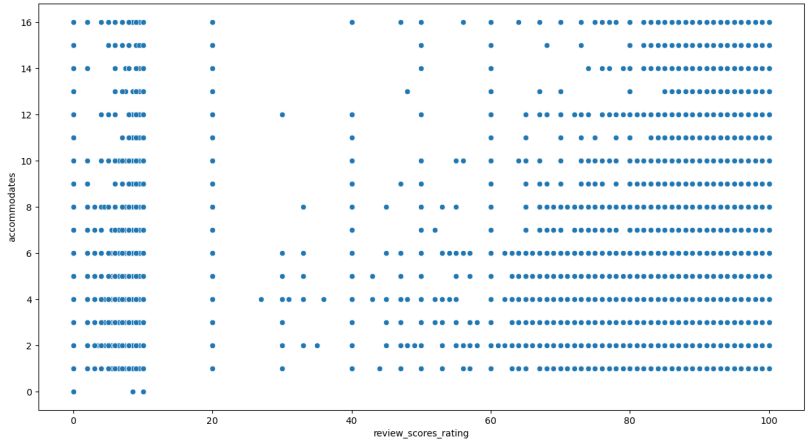
```
sns.scatterplot(x = df_airbnb['review_scores_rating'], y = df_airbnb['price'],palett
plt.show()
```



Accommodates vs Review score rating

In [80]:

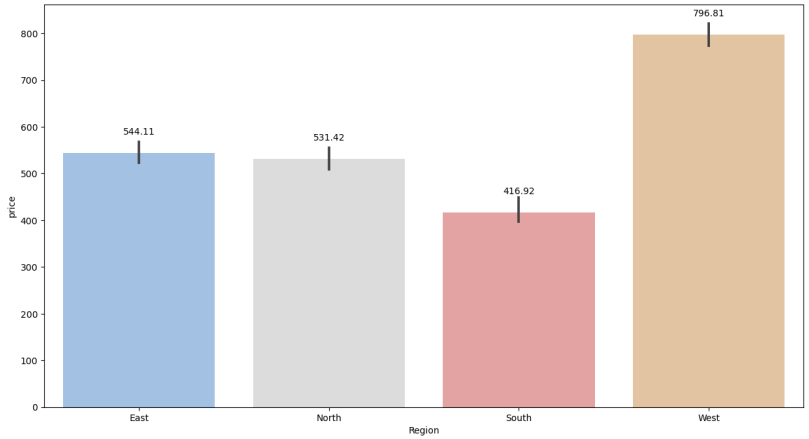
```
sns.scatterplot(x = df_airbnb['review_scores_rating'], y = df_airbnb['accommodates']
plt.show()
```



Region vs price

In [81]:

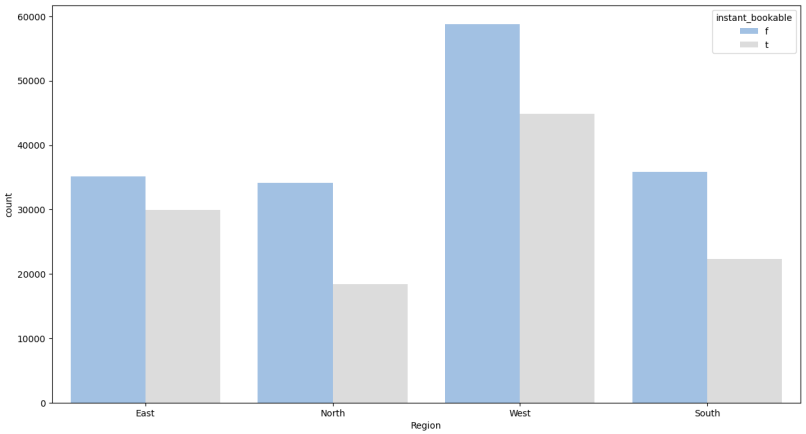
```
sns.barplot(x = df_airbnb['Region'].sort_values() , y = df_airbnb['price'], palette

for i,v in enumerate(round(df_airbnb.groupby(by = 'Region')['price'].mean().sort_inc
    plt.text(x = i , y = v + 40, s = v, ha = 'center')
```



Region vs Instant_bookable

In [82]:

```
sns.countplot(x = df_airbnb['Region'], hue = df_airbnb['instant_bookable'], palette
plt.show()
```

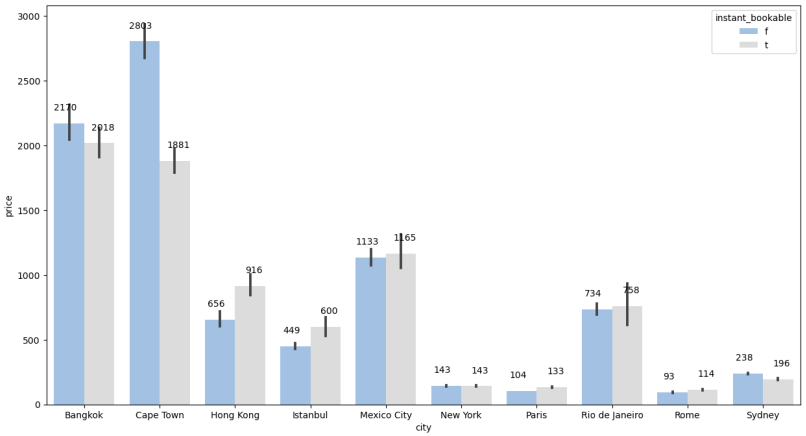


Multivariate Analysis

City vs Price vs Instant bookable

In [83]:

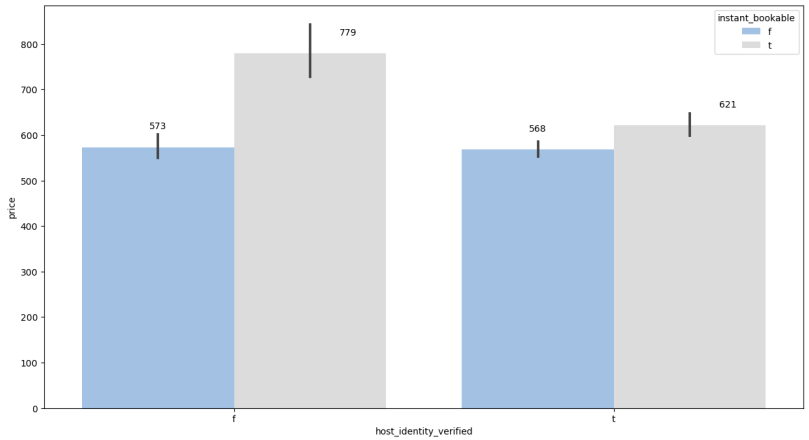
```
sns.barplot(x = df_airbnb['city'].sort_values(), y = df_airbnb['price'], hue = df_ai
            palette = colors)


for i,v in enumerate(df_airbnb.groupby(['city','instant_bookable'])['price'].mean())
    plt.text(x = i/2 - 0.25, y = v + 100, s = round(v) , ha = 'center')
```



Host_identity_verified vs Price vs Instant bookable

In [84]:

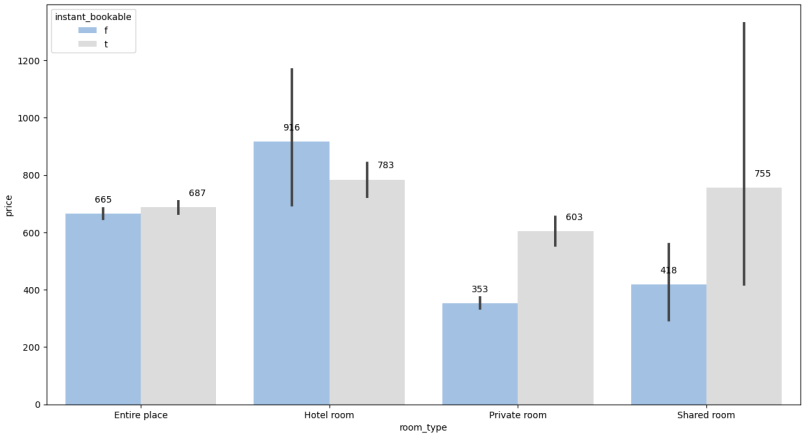
```
sns.barplot(x = df_airbnb['host_identity_verified'].sort_values(), y = df_airbnb['pr
            hue = df_airbnb['instant_bookable'],palette = colors)


for i,v in enumerate(df_airbnb.groupby(['host_identity_verified','instant_bookable']
    plt.text(x = i/2 - 0.2, y = v + 40, s = round(v) , ha = 'center')
```



Room_type vs Price vs Instant bookable

In [85]:

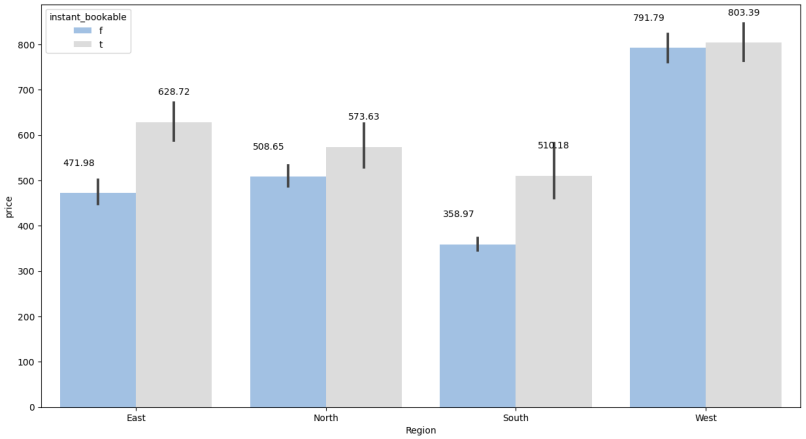
```
sns.barplot(x = df_airbnb['room_type'].sort_values(), y = df_airbnb['price'], hue =
           palette = colors)


for i,v in enumerate(df_airbnb.groupby(['room_type','instant_bookable'])['price'].me
    plt.text(x = i/2 - 0.2, y = v + 40, s = round(v) , ha = 'center')
```



Region vs Price vs Instant_bookable

In [86]:

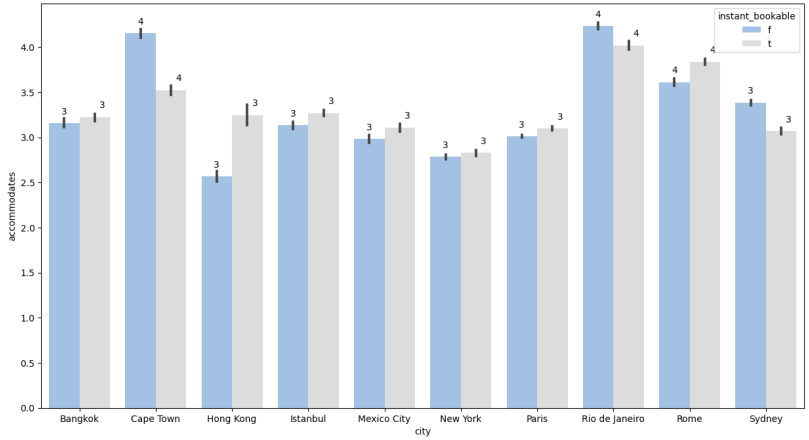
```
sns.barplot(x = df_airbnb['Region'].sort_values() , y = df_airbnb['price'], hue = df
            palette = colors)

for i,v in enumerate(round(df_airbnb.groupby(by = ['Region','instant_bookable'])['pr
    plt.text(x = i - i/2 - 0.3, y = v + 60, s = v, ha = 'center')
```



Accomodates vs City vs Instant bookable

In [87]:

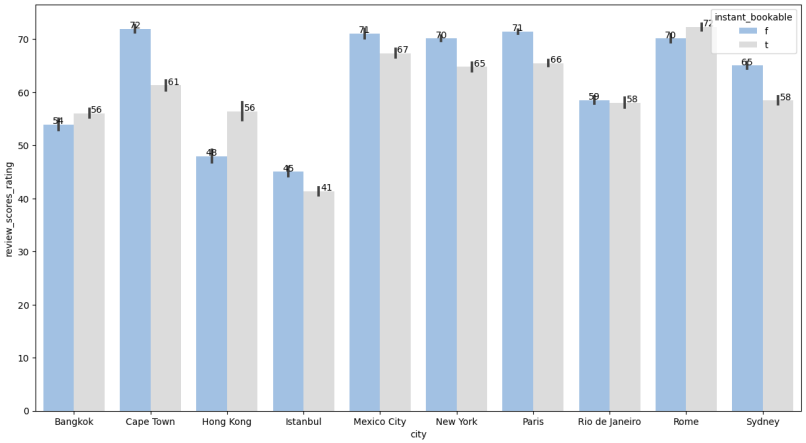
```
sns.barplot(x = df_airbnb['city'].sort_values(), y = df_airbnb['accommodates'], hue
           ,palette = colors)


for i,v in enumerate(df_airbnb.groupby(['city','instant_bookable'])['accommodates'].
    plt.text(x = i/2 - 0.2, y = v + 0.1 , s = round(v) , ha = 'center')
```



City vs Review score rating vs Instant bookable

In [88]:

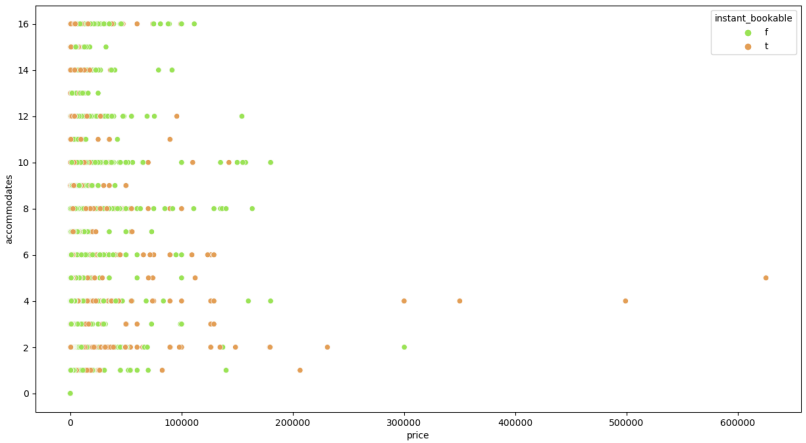
```
sns.barplot(x = df_airbnb['city'].sort_values(), y = df_airbnb['review_scores_rating
            hue = df_airbnb['instant_bookable'],palette = colors)


for i,v in enumerate(df_airbnb.groupby(['city','instant_bookable'])['review_scores_r
    plt.text(x = i/2 - 0.2, y = v + 0.1 , s = round(v) , ha = 'center')
```



Accomodates vs price vs Instant bookable

In [89]:

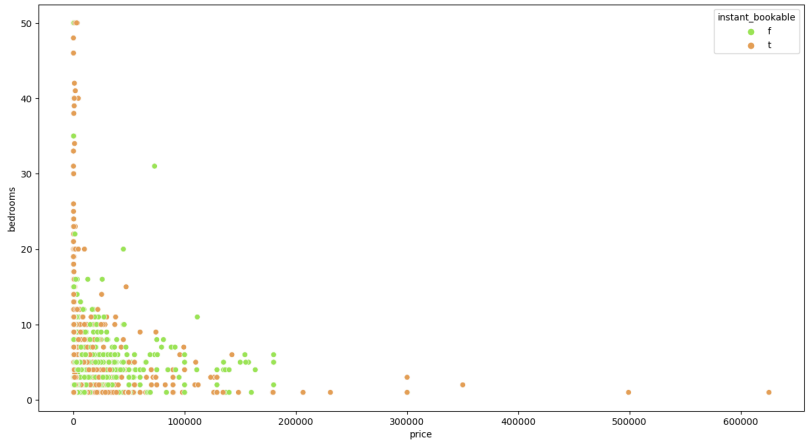
```
sns.scatterplot(x = df_airbnb['price'], y = df_airbnb['accommodates'], hue = df_airb
                ,palette = ['#9ae355','#e39e55'])

plt.show()
```



Price vs Bedrooms vs Instant bookable

In [90]:

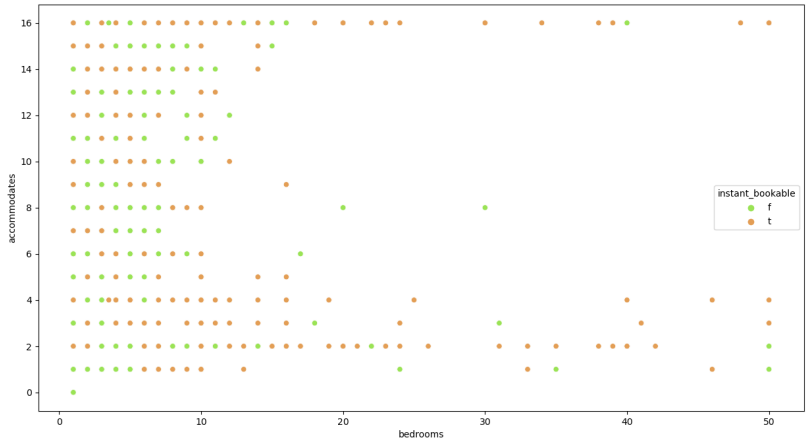
```
sns.scatterplot(x = df_airbnb['price'], y = df_airbnb['bedrooms'], hue = df_airbnb['
                ,palette = ['#9ae355','#e39e55'])

plt.show()
```



Bedrooms vs Accommodates vs Instant bookable

In [91]:

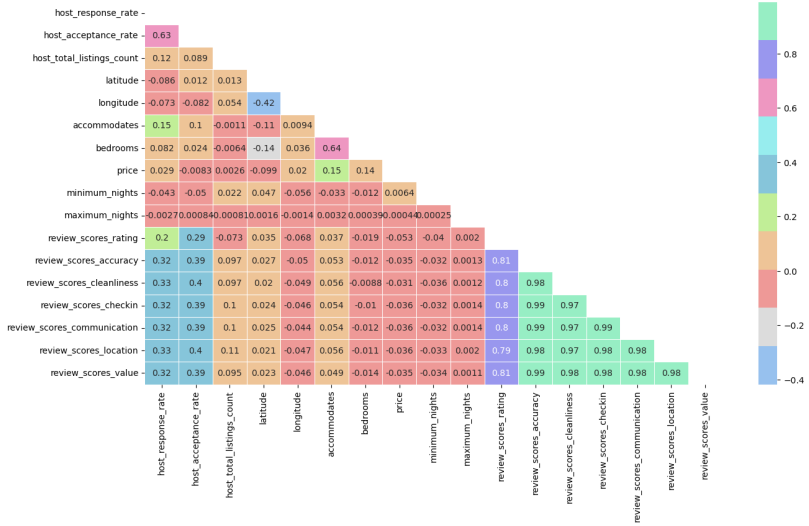
```
sns.scatterplot(x = df_airbnb['bedrooms'], y = df_airbnb['accommodates'], hue = df_a
                ,palette = ['#9ae355','#e39e55'])
plt.show()
```



Checking correlation with heatmap

In [92]:

```
sns.heatmap(df_airbnb.corr(), annot = True, cmap = colors,linewidth = 0.6,mask = np.
plt.show()
```



Performing hypothesis testing to find the significant variables

Hypothesis :

H0 (Null Hypothesis) : There is no significant relationship between the variables being tested.

Ha (Alternative Hypothesis) : There is a significant relationship between the variables being tested

Consider significance level as 0.05

In [93]:

```
num_cols = df_airbnb.select_dtypes(include = np.number).columns
num_cols
```

Out[93]:

```
Index(['host_response_rate', 'host_acceptance_rate',
       'host_total_listings_count', 'latitude', 'longitude', 'accommodate
s',
       'bedrooms', 'price', 'minimum_nights', 'maximum_nights',
       'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value'],
      dtype='object')
```


In [94]:

```
# Creating a dataframe to store the values

statistical_result = pd.DataFrame(columns = ['Columns','Pvalue','Remarks'])
```

In [95]:

```
# Numerical vs Categorical(Instant_bookable) - f_oneway

for i in num_cols:
    groups = [df_airbnb.loc[df_airbnb['instant_bookable'] == subclass , i] for subcl
              in df_airbnb['instant_bookable'].unique()]

    stat , pval = stats.f_oneway(*groups)

    statistical_result = statistical_result.append({'Columns': i , 'Pvalue': pval ,
                                                    'Remarks': 'Reject H0' if pval <
                                                   ignore_index = True)
```

In [96]:

```
cat_cols = df_airbnb.select_dtypes(exclude = np.number).columns.to_list()
print(cat_cols)
```

```
['host_response_time', 'host_is_superhost', 'host_has_profile_pic', 'host_
identity_verified', 'neighbourhood', 'city', 'property_type', 'room_type',
'amenities', 'instant_bookable', 'Region']
```

host_since is a datetime object and instant bookable is the target variable so we should remove it from cat_cols

In [97]:

```
cat_cols.remove('instant_bookable')
print(cat_cols)
```

```
['host_response_time', 'host_is_superhost', 'host_has_profile_pic', 'host_
identity_verified', 'neighbourhood', 'city', 'property_type', 'room_type',
'amenities', 'Region']
```

In [98]:

```
# Categorical vs Categorical(Instant_bookable) - chi2_contingency

for i in cat_cols:
    stat , pval , dof , expected = stats.chi2_contingency(pd.crosstab(df_airbnb[i],d

    statistical_result = statistical_result.append({'Columns':i , 'Pvalue': pval ,
                                                    'Remarks' : 'Reject H0' if pval <
                                                   ignore_index = True)
```

In [99]:

```
statistical_result
```

Out[99]:

	Columns	Pvalue	Remarks
0	host_response_rate	0.000000e+00	Reject H0
1	host_acceptance_rate	0.000000e+00	Reject H0
2	host_total_listings_count	0.000000e+00	Reject H0
3	latitude	1.218462e-01	Failed to reject H0
4	longitude	1.115112e-134	Reject H0
5	accommodates	1.290423e-06	Reject H0
6	bedrooms	4.834544e-18	Reject H0
7	price	5.595255e-13	Reject H0
8	minimum_nights	1.209669e-123	Reject H0
9	maximum_nights	6.246272e-01	Failed to reject H0
10	review_scores_rating	1.805145e-184	Reject H0
11	review_scores_accuracy	1.411651e-43	Reject H0
12	review_scores_cleanliness	3.888389e-21	Reject H0
13	review_scores_checkin	7.071339e-35	Reject H0
14	review_scores_communication	3.402854e-39	Reject H0
15	review_scores_location	1.199631e-22	Reject H0
16	review_scores_value	2.815450e-37	Reject H0
17	host_response_time	0.000000e+00	Reject H0
18	host_is_superhost	3.887066e-79	Reject H0
19	host_has_profile_pic	3.101958e-01	Failed to reject H0
20	host_identity_verified	1.208082e-03	Reject H0
21	neighbourhood	0.000000e+00	Reject H0
22	city	0.000000e+00	Reject H0
23	property_type	0.000000e+00	Reject H0
24	room_type	0.000000e+00	Reject H0
25	amenities	1.482095e-185	Reject H0
26	Region	0.000000e+00	Reject H0

Insights from statistical test

From above performed statistical tests we can conclude that all of the columns except latitude and maximum_nights has rejected null hypotheses which means the columns are significant to the target variable instant_bookable. The variables latitude , longitude , maximum_nights and host_has_profile_pic can be dropped as they are insignificant to the target variable.

In [100]:

```
df_airbnb.drop(columns = ['latitude','longitude','maximum_nights','host_has_profile_
```

Amenities Wordcloud

In [101]:

```
from nltk.corpus import stopwords
```

In [102]:

```
stop_words = list(set(stopwords.words('english') + list(STOPWORDS)))
```

In [103]:

```
df_airbnb['amenities'] = df_airbnb['amenities'].apply(lambda x : ast.literal_eval(x)
```

In [104]:

```
# Extracting words without numbers and special characters

df_airbnb['amenities'] = df_airbnb['amenities'].apply(lambda x : re.sub('[^a-z]+', '
```

In [105]:

```
# Doing Lemmatization

from nltk.stem import WordNetLemmatizer

wnl = WordNetLemmatizer()

df_airbnb['amenities'] = df_airbnb['amenities'].apply(lambda x : ' '.join([ wnl.lemm
```

In [106]:

```
# Extracting words which are not in stop words

df_airbnb['amenities'] = df_airbnb['amenities'].apply(lambda x : ' '.join([ i for i
```

In [107]:

```
# Creating a list with all rows of amenities

comment_words = []

for i in df_airbnb['amenities']:

    i = i.split()

    for j in i:
        comment_words.append(j)
```

In [108]:

```
# Converting comment_words into series and taking frequency of each words

comment_words = pd.Series(comment_words).value_counts()
```

In [109]:

```
# Generating wordcloud

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                max_words = 1000,
                min_font_size = 10).generate_from_frequencies(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [110]:

```
# Considering top 20 words

top_words = comment_words[:20].index.to_list()

print(top_words)
```

```
['allowed', 'dryer', 'tv', 'wifi', 'essential', 'kitchen', 'stay', 'long',
'term', 'alarm', 'hanger', 'hair', 'iron', 'washer', 'heating', 'hot', 'wo
rkspace', 'dedicated', 'parking', 'shampoo']
```

In [111]:

```
df_airbnb['amenities'] = df_airbnb['amenities'].map(lambda x : ' '.join([ i for i ir
```

Splitting the dataset randomly into train and test dataset using ratio of 70:30

In [112]:

```
x = df_airbnb.drop(columns = ['instant_bookable'])
y = df_airbnb['instant_bookable']

xtrain , xtest , ytrain , ytest = train_test_split(x,y,test_size = 0.30,random_state
```

Checking and treating of outliers

In [113]:

```
num_cols = xtrain.select_dtypes(include = np.number).columns
num_cols
```

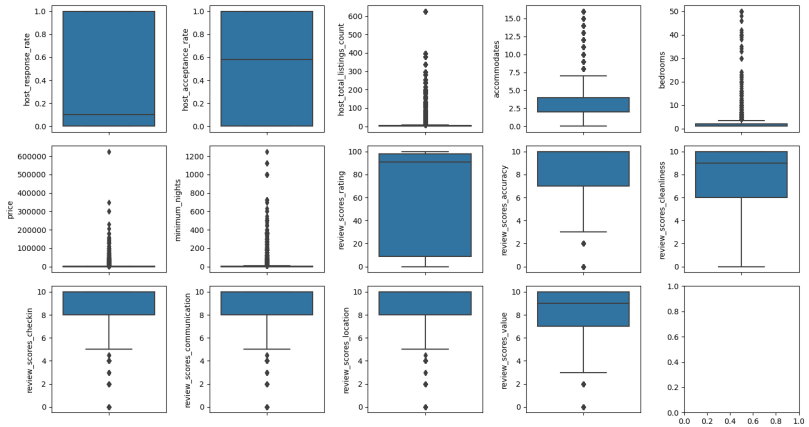
Out[113]:

```
Index(['host_response_rate', 'host_acceptance_rate',
       'host_total_listings_count', 'accommodates', 'bedrooms', 'price',
       'minimum_nights', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value'],
      dtype='object')
```

For train data

In [114]:

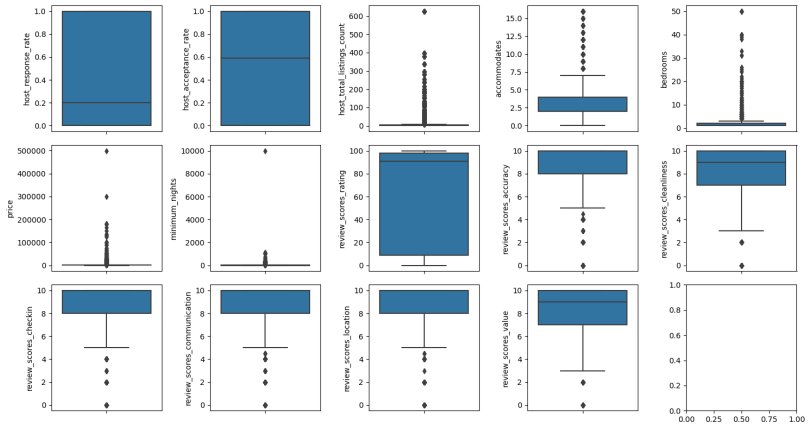
```
f , ax = plt.subplots(3,5)

for i,v in zip(num_cols,ax.flatten()):
    sns.boxplot(y = xtrain[i] , ax = v)

plt.tight_layout()
plt.show()
```



For test data

In [115]:

```
f , ax = plt.subplots(3,5)

for i,v in zip(num_cols,ax.flatten()):
    sns.boxplot(y = xtest[i] , ax = v)

plt.tight_layout()
plt.show()
```



From above box plots it is clearly evident that there are outliers . By doing IQR method we tend lose data. Hence we go forward by doing transformation technique

In [116]:

```
out_cols = ['host_total_listings_count','accommodates','bedrooms','price','minimum_r
            'review_scores_checkin','review_scores_communication','review_scores_loca
            'review_scores_cleanliness']
```

In [117]:

```
pt = PowerTransformer()

for i in out_cols:
    variable = pt.fit(xtrain[[i]])

    xtrain[i] = variable.transform(xtrain[[i]])
    xtest[i] = variable.transform(xtest[[i]])

#     saving as pickle file
#     with open(f'Transformation\\{i}.pkl','wb') as file:
#         pickle.dump(variable,file)


```

Scaling of Numerical variables

In [118]:

```
num_cols = ['host_response_rate','host_acceptance_rate','host_total_listings_count',
            'bedrooms','price','minimum_nights','review_scores_rating','review_score

ss = StandardScaler()

for i in num_cols:

    variable = ss.fit(xtrain[[i]])

    xtrain[i] = variable.transform(xtrain[[i]])
    xtest[i] = variable.transform(xtest[[i]])

#     saving as pickle file
#     with open(f'Scaling\\scaling_{i}.pkl','wb') as file:
#         pickle.dump(variable,file)


```

Encoding of Categorical variables

In [119]:

```
cat_cols = xtrain.select_dtypes(exclude = np.number).columns.to_list()
print(cat_cols)
```

```
['host_response_time', 'host_is_superhost', 'host_identity_verified', 'nei
ghbourhood', 'city', 'property_type', 'room_type', 'amenities', 'Region']
```

Host_response_time

In [120]:

```
xtrain['host_response_time'].unique()
```

Out[120]:

```
array(['unknown', 'within a day', 'within an hour', 'within a few hours',
       'a few days or more'], dtype=object)
```

Host response time variable is a ordinal categorical variable. It has a hierarchy between the subclasses. The host who responds within a few hours is more likely to get instant bookability than the host who responds a few days or more

In [121]:

```
oe = OrdinalEncoder(categories = [['unknown','a few days or more','within a day','wi

oe_response_time = oe.fit((xtrain[['host_response_time']]))

xtrain['host_response_time'] = oe_response_time.transform(xtrain[['host_response_tim

xtest['host_response_time'] = oe_response_time.transform(xtest[['host_response_time'

xtrain['host_response_time'].unique()
```

Out[121]:

```
array([0., 2., 3., 4., 1.])
```

Host_is_superhost

In [122]:

```
xtrain['host_is_superhost'].unique()
```

Out[122]:

```
array(['t', 'f'], dtype=object)
```

Host_is_superhost variable is a nominal categorical variable. There is no hierarchy between the subclasses. Hence we can replace true with 1 and false with 0

In [123]:

```
xtrain['host_is_superhost'].replace({'t':1,'f':0} , inplace = True)
xtest['host_is_superhost'].replace({'t':1,'f':0} , inplace = True)
```

Host_identity_verified

In [124]:

```
xtrain['host_identity_verified'].unique()
```

Out[124]:

```
array(['t', 'f'], dtype=object)
```

Host_identity_verified variable is a nominal categorical variable. There is no hierarchy between the subclasses. Hence we can replace true with 1 and false with 0

In [125]:

```
xtrain['host_identity_verified'].replace({'t':1,'f':0} , inplace = True)
xtest['host_identity_verified'].replace({'t':1,'f':0} , inplace = True)
```

Instant_bookable

In [126]:

```
ytrain.unique()
```

Out[126]:

```
array(['f', 't'], dtype=object)
```

Instant_bookable is a target variable. It is binary classification. Hence we can replace true with 1 and false with 0

In [127]:

```
ytrain.replace({'t':1,'f':0} , inplace = True)
ytest.replace({'t':1,'f':0} , inplace = True)
```

Neighbourhood

In [128]:

```
xtrain['neighbourhood'].nunique()
```

Out[128]:

```
651
```

In [129]:

```
xtest['neighbourhood'].nunique()
```

Out[129]:

```
608
```

Neighbourhood variable is a nominal categorical variable. There is no hierarchy between the subclasses. But there is more than 600 subclasses. Hence we can do woe encoding

In [130]:

```
neighbourhood_woe = WOEEncoder()

woe_neighborhood = neighbourhood_woe.fit(xtrain['neighbourhood'] , ytrain)

xtrain['neighbourhood'] = woe_neighborhood.transform(xtrain['neighbourhood'])
xtest['neighbourhood'] = woe_neighborhood.transform(xtest['neighbourhood'])
```

City

In [131]:

```
xtrain['city'].unique()
```

Out[131]:

```
array(['Cape Town', 'Istanbul', 'Rio de Janeiro', 'Rome', 'New York',
       'Paris', 'Sydney', 'Mexico City', 'Hong Kong', 'Bangkok'],
      dtype=object)
```

City variable is a nominal categorical variable. There is no hierarchy between the subclasses. Since there is 10 subclasses we can do woe encoding

In [132]:

```
city_woe = WOEEncoder()

woe_city = city_woe.fit(xtrain['city'] , ytrain)

xtrain['city'] = city_woe.transform(xtrain['city'])
xtest['city'] = city_woe.transform(xtest['city'])
```

Room_type

In [133]:

```
xtrain['room_type'].unique()
```

Out[133]:

```
array(['Entire place', 'Private room', 'Hotel room', 'Shared room'],
      dtype=object)
```

Room type variable is a ordinal categorical variable. It has a hierarchy between the subclasses.

In [134]:

```
oe = OrdinalEncoder(categories = [['Shared room','Private room','Hotel room','Entire

oe_room_type = oe.fit(xtrain[['room_type']])

xtrain['room_type'] = oe_room_type.transform(xtrain[['room_type']])
xtest['room_type'] = oe_room_type.transform(xtest[['room_type']])

xtrain['room_type'].unique()
```

Out[134]:

```
array([3., 1., 2., 0.])
```

Regions

In [135]:

```
xtrain['Region'].unique()
```

Out[135]:

```
array(['East', 'West', 'North', 'South'], dtype=object)
```

Regions variable is a nominal categorical variable. We can perform woe encoding technique.

In [136]:

```
regions_woe = WOEEncoder()

woe_region = regions_woe.fit(xtrain['Region'] , ytrain)

xtrain['Region'] = woe_region.transform(xtrain['Region'])
xtest['Region'] = woe_region.transform(xtest['Region'])
```

Property_type

Property type variable is a nominal categorical variable. We can perform woe encoding technique.

In [137]:

```
property_woe = WOEEncoder()

woe_property_type = property_woe.fit(xtrain['property_type'] , ytrain)

xtrain['property_type'] = woe_property_type.transform(xtrain['property_type'])
xtest['property_type'] = woe_property_type.transform(xtest['property_type'])
```


Checking for Multicollinearity

In [138]:

```
xtrain.columns
```

Out[138]:

```
Index(['host_response_time', 'host_response_rate', 'host_acceptance_rate',
       'host_is_superhost', 'host_total_listings_count',
       'host_identity_verified', 'neighbourhood', 'city', 'property_type',
       'room_type', 'accommodates', 'bedrooms', 'amenities', 'price',
       'minimum_nights', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'Region'],
      dtype='object')
```

In [139]:

```
num_cols = ['host_response_rate','host_acceptance_rate','host_total_listings_count',
            'accommodates','bedrooms','price','minimum_nights','review_scores_rating'
            'review_scores_cleanliness','review_scores_checkin','review_scores_commur
            'review_scores_value']
```

In [140]:

```
# Creating a copy of the dataset

xtrain_copy = xtrain.copy()

# considering only numerical variables to check multicollinearity

xtrain_copy = xtrain_copy.loc[:,num_cols]
```

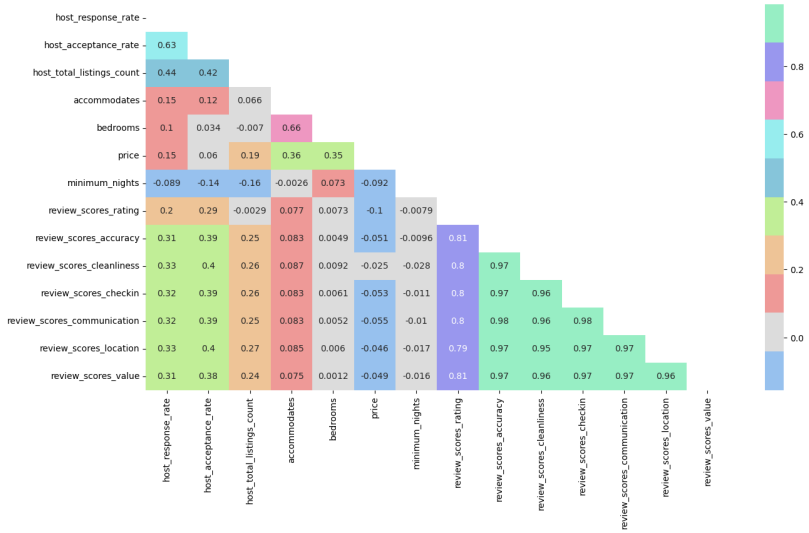
Plotting of heatmap

In [141]:

```
sns.heatmap(xtrain_copy.corr() , annot = True , mask = np.triu(xtrain_copy.corr()) ,
```

Out[141]:

<AxesSubplot:>



Calculating variance inflation factor

In [142]:

```
vif_values = []

for i in range(xtrain_copy.shape[1]):
    vif_values.append(variance_inflation_factor(xtrain_copy.values,i))

vif_df = pd.DataFrame({'Columns': xtrain_copy.columns , 'VIF': vif_values})

vif_df
```

Out[142]:

	Columns	VIF
0	host_response_rate	1.817190
1	host_acceptance_rate	1.874335
2	host_total_listings_count	1.589339
3	accommodates	1.876943
4	bedrooms	1.868194
5	price	1.281532
6	minimum_nights	1.058716
7	review_scores_rating	3.487025
8	review_scores_accuracy	35.153015
9	review_scores_cleanliness	18.993631
10	review_scores_checkin	38.780670
11	review_scores_communication	39.865239
12	review_scores_location	22.476415
13	review_scores_value	27.101451

From above dataframe we can infer that there is high multicollinearity in review related variables . First we drop review_scores_communication variable as it has the high vif value and again we check calculate the vif values

In [143]:

```
# Removing review_scores_communication to calculate vif

xtrain_copy = xtrain_copy.drop(columns = 'review_scores_communication')

vif_values = []

for i in range(xtrain_copy.shape[1]):
    vif_values.append(variance_inflation_factor(xtrain_copy.values,i))

vif_df = pd.DataFrame({'Columns': xtrain_copy.columns , 'VIF': vif_values})

vif_df
```

Out[143]:

	Columns	VIF
0	host_response_rate	1.817183
1	host_acceptance_rate	1.873861
2	host_total_listings_count	1.589095
3	accommodates	1.876657
4	bedrooms	1.868190
5	price	1.280695
6	minimum_nights	1.058607
7	review_scores_rating	3.478044
8	review_scores_accuracy	33.661268
9	review_scores_cleanliness	18.953833
10	review_scores_checkin	28.689667
11	review_scores_location	21.803260
12	review_scores_value	26.456621

We now drop the review_scores_accuracy variable as it has the vif value and again the values for remaning variables

In [144]:

```
# Removing review_scores_accuracy to calculate vif

xtrain_copy = xtrain_copy.drop(columns = 'review_scores_accuracy')

vif_values = []

for i in range(xtrain_copy.shape[1]):
    vif_values.append(variance_inflation_factor(xtrain_copy.values,i))

vif_df = pd.DataFrame({'Columns': xtrain_copy.columns , 'VIF': vif_values})

vif_df
```

Out[144]:

	Columns	VIF
0	host_response_rate	1.817171
1	host_acceptance_rate	1.873694
2	host_total_listings_count	1.588611
3	accommodates	1.875761
4	bedrooms	1.867897
5	price	1.280166
6	minimum_nights	1.057202
7	review_scores_rating	3.456973
8	review_scores_cleanliness	17.305213
9	review_scores_checkin	25.271312
10	review_scores_location	21.084957
11	review_scores_value	23.568200

We now drop the review_scores_checkin variable as it has the vif value and again the values for remaning variables

In [145]:

```
# Removing review_scores_checkin to calculate vif

xtrain_copy = xtrain_copy.drop(columns = 'review_scores_checkin')

vif_values = []

for i in range(xtrain_copy.shape[1]):
    vif_values.append(variance_inflation_factor(xtrain_copy.values,i))

vif_df = pd.DataFrame({'Columns': xtrain_copy.columns , 'VIF': vif_values})

vif_df
```

Out[145]:

	Columns	VIF
0	host_response_rate	1.816301
1	host_acceptance_rate	1.872566
2	host_total_listings_count	1.583795
3	accommodates	1.875268
4	bedrooms	1.867775
5	price	1.274894
6	minimum_nights	1.055587
7	review_scores_rating	3.437805
8	review_scores_cleanliness	16.241604
9	review_scores_location	15.729774
10	review_scores_value	21.281308

We now drop the review_scores_value variable as it has the vif value and again the values for remaning variables

In [146]:

```
# Removing review_scores_value to calculate vif

xtrain_copy = xtrain_copy.drop(columns = 'review_scores_value')

vif_values = []

for i in range(xtrain_copy.shape[1]):
    vif_values.append(variance_inflation_factor(xtrain_copy.values,i))

vif_df = pd.DataFrame({'Columns': xtrain_copy.columns , 'VIF': vif_values})

vif_df
```

Out[146]:

	Columns	VIF
0	host_response_rate	1.816299
1	host_acceptance_rate	1.872017
2	host_total_listings_count	1.582584
3	accommodates	1.873415
4	bedrooms	1.867750
5	price	1.274540
6	minimum_nights	1.055393
7	review_scores_rating	3.359656
8	review_scores_cleanliness	11.262310
9	review_scores_location	11.239094

We now drop the review_scores_cleanliness variable as it has the vif value and again the values for remaning variables

In [147]:

```
# Removing review_scores_cleanliness to calculate vif

xtrain_copy = xtrain_copy.drop(columns = 'review_scores_cleanliness')

vif_values = []

for i in range(xtrain_copy.shape[1]):
    vif_values.append(variance_inflation_factor(xtrain_copy.values,i))

vif_df = pd.DataFrame({'Columns': xtrain_copy.columns , 'VIF': vif_values})

vif_df
```

Out[147]:

	Columns	VIF
0	host_response_rate	1.815622
1	host_acceptance_rate	1.871291
2	host_total_listings_count	1.575444
3	accommodates	1.873086
4	bedrooms	1.867714
5	price	1.268624
6	minimum_nights	1.055091
7	review_scores_rating	3.153907
8	review_scores_location	3.424648

Dropping of columns to reduce multicollinearity

In [148]:

```
# Dropping columns from train data

xtrain.drop(columns = ['review_scores_communication','review_scores_accuracy','revie
                       'review_scores_value','review_scores_cleanliness'] , inplac

# Dropping columns from test data

xtest.drop(columns = ['review_scores_communication','review_scores_accuracy','review
                       'review_scores_value','review_scores_cleanliness'] , inplac
```


Performing TFIDF Vectorizer for amenities columns

In [149]:

```
vectorizer = TfidfVectorizer()

vectorizer.fit(xtrain.loc[:,'amenities'])
```

Out[149]:

```
TfidfVectorizer()
```

In [150]:

```
tf_train = vectorizer.transform(xtrain.loc[:,'amenities'])
tf_test = vectorizer.transform(xtest.loc[:,'amenities'])
```

In [151]:

```
len(vectorizer.get_feature_names_out())
```

Out[151]:

```
20
```

In [152]:

```
xtrain.drop(columns = 'amenities',inplace = True)
xtest.drop(columns = 'amenities',inplace = True)
```

In [153]:

```
xtrain = xtrain.reset_index(drop=True)
xtest = xtest.reset_index(drop = True)
```

In [154]:

```
train_df = pd.DataFrame(tf_train.toarray(), columns = vectorizer.get_feature_names_c
test_df = pd.DataFrame(tf_test.toarray(), columns = vectorizer.get_feature_names_out
```

In [155]:

```
xtrain = pd.concat([xtrain,train_df], axis =1)
xtest = pd.concat([xtest,test_df], axis = 1)
```

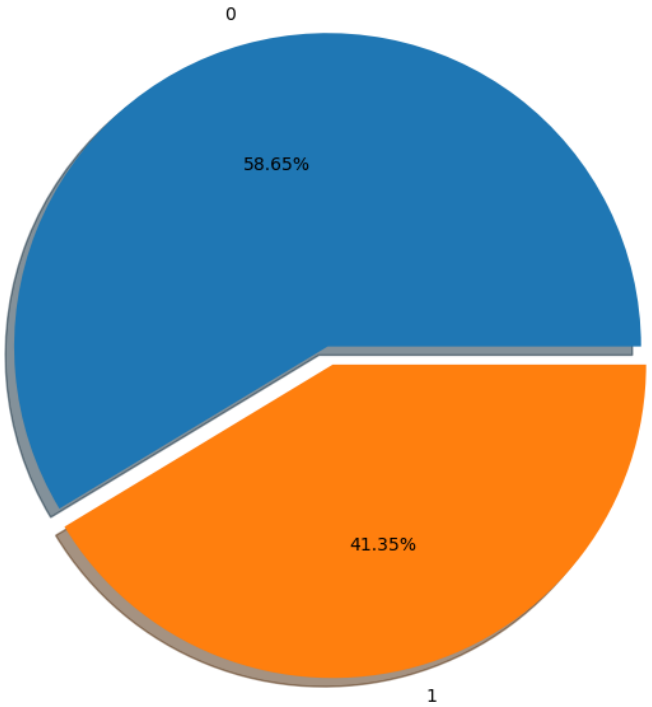
In [156]:

```
ytrain = ytrain.reset_index(drop = True)
ytest = ytest.reset_index(drop = True)
```

Checking for class imbalance in target variable

In [157]:

```
s = ytrain.value_counts()

plt.pie(s , labels = s.index , autopct = '%.2f%%' , shadow = True , explode = [0.01,
plt.show()
```



From above pie chart it is clearly evident that there is class imbalance. By using over sampling technique we can overcome this imbalance.

In [158]:

```
# For train data

sm_train = SMOTE()

xtrain , ytrain = sm_train.fit_resample(xtrain , ytrain)

# For test data

sm_test = SMOTE()

xtest , ytest = sm_test.fit_resample(xtest , ytest)

```

In [159]:

```
ytrain.value_counts(normalize = True)
```

Out[159]:

```
0    0.5
1    0.5
Name: instant_bookable, dtype: float64
```

Building a base model

Building a base model using Logistic Regression as it is having the highest explanatory power compared to other models

In [160]:

```
model_lr = sma.Logit(ytrain,sma.add_constant(xtrain)).fit()

model_lr
```

```
Optimization terminated successfully.
         Current function value: 0.613537
         Iterations 6
```

Out[160]:

```
<statsmodels.discrete.discrete_model.BinaryResultsWrapper at 0x1e00f4d4af0
>
```

Checking for summary

In [161]:

```
1 model_lr.summary()
```


Out[161]:

Logit Regression Results

Dep. Variable:	instant_bookable	**No. Observations:**	229494
Model:	Logit	**Df Residuals:**	229457
Method:	MLE	**Df Model:**	36
Date:	Thu, 15 Jun 2023	**Pseudo R-squ.:**	0.1149
Time:	21:39:03	**Log-Likelihood:**	-1.4080e+05
converged:	True	**LL-Null:**	-1.5907e+05
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.5235	0.054	-9.736	0.000	-0.629	-0.418
host_response_time	-0.1781	0.010	-18.056	0.000	-0.197	-0.159
host_response_rate	0.1122	0.015	7.294	0.000	0.082	0.142
host_acceptance_rate	0.4199	0.007	64.350	0.000	0.407	0.433
host_is_superhost	-0.1107	0.013	-8.525	0.000	-0.136	-0.085
host_total_listings_count	0.2317	0.006	38.636	0.000	0.220	0.243
host_identity_verified	-0.1267	0.011	-11.672	0.000	-0.148	-0.105
neighbourhood	0.6991	0.014	48.214	0.000	0.671	0.728
city	-0.0423	0.019	-2.287	0.022	-0.079	-0.006
property_type	0.5329	0.013	42.262	0.000	0.508	0.558
room_type	0.0252	0.006	4.240	0.000	0.014	0.037
accommodates	0.0496	0.007	7.357	0.000	0.036	0.063
bedrooms	-0.0387	0.006	-6.199	0.000	-0.051	-0.026
price	-0.0720	0.006	-11.691	0.000	-0.084	-0.060
minimum_nights	-0.1755	0.005	-34.524	0.000	-0.185	-0.166
review_scores_rating	-0.0259	0.008	-3.121	0.002	-0.042	-0.010
review_scores_location	-0.2479	0.009	-29.002	0.000	-0.265	-0.231
Region	-0.0524	0.030	-1.755	0.079	-0.111	0.006
alarm	0.4248	0.027	15.859	0.000	0.372	0.477
allowed	-0.0357	0.070	-0.510	0.610	-0.173	0.101
dedicated	2.1338	2.976	0.717	0.473	-3.700	7.968
dryer	0.3637	0.050	7.298	0.000	0.266	0.461
essential	1.1017	0.070	15.800	0.000	0.965	1.238
hair	0.4586	0.063	7.299	0.000	0.335	0.582
hanger	0.7857	0.048	16.388	0.000	0.692	0.880
heating	-0.4399	0.043	-10.262	0.000	-0.524	-0.356
hot	0.2072	0.036	5.802	0.000	0.137	0.277
iron	0.5730	0.043	13.368	0.000	0.489	0.657
kitchen	0.1051	0.068	1.534	0.125	-0.029	0.239

long	6.2646	nan	nan	nan	nan	nan
parking	0.2388	0.029	8.218	0.000	0.182	0.296
shampoo	0.2645	0.038	6.906	0.000	0.189	0.340
stay	-12.1615	3.106	-3.916	0.000	-18.249	-6.074
term	6.2646	nan	nan	nan	nan	nan
tv	0.0110	0.035	0.319	0.750	-0.057	0.079
washer	-0.1877	0.043	-4.369	0.000	-0.272	-0.104
wifi	0.0138	0.071	0.194	0.846	-0.125	0.153
workspace	-1.9248	2.977	-0.647	0.518	-7.759	3.909

Calculating various metrics to evaluate the model performance

In [162]:

```
pred_prob_train = model_lr.predict(sma.add_constant(xtrain))
pred_prob_test = model_lr.predict(sma.add_constant(xtest))
```

In [163]:

```
# Calculating youden's index to convert probability prediction to class prediction

# For train data

fpr , tpr , threshold = roc_curve(ytrain,pred_prob_train)

youden_index_train = []

for i,v in zip(fpr , tpr):
    res = v - i
    youden_index_train.append(res)

yi_train = max(np.round(youden_index_train,2))


# For test data

fpr , tpr , threshold = roc_curve(ytest,pred_prob_test)

youden_index_test = []

for i,v in zip(fpr , tpr):
    res = v - i
    youden_index_test.append(res)

yi_test = max(np.round(youden_index_test,2))

print(f'Youdens index for train data is {yi_train}')
print(f'Youdens index for test data is {yi_test}')
```

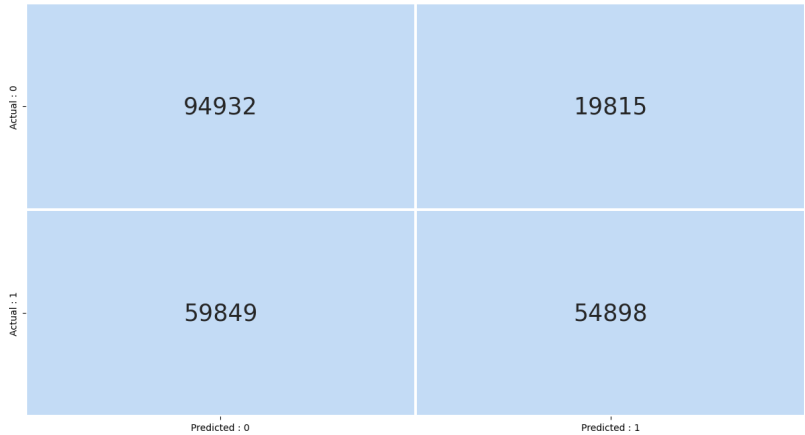
```
Youdens index for train data is 0.33
Youdens index for test data is 0.33
```


In [164]:

```
# Converting probability prediction to class prediction using 0.59 as threshold valu

pred_train = [ 1 if i > 0.59 else 0  for i in pred_prob_train]
pred_test = [ 1 if i > 0.59 else 0  for i in pred_prob_test]

```

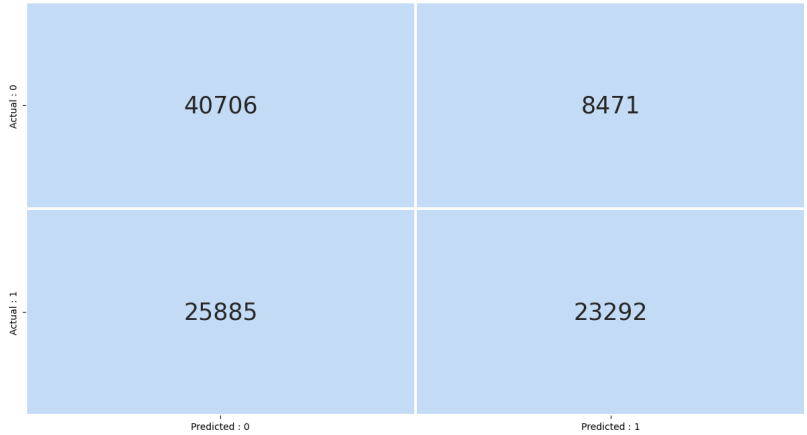
In [165]:

```
# plotting confusion matrix for train data

cm = confusion_matrix(ytrain,pred_train)
conf_matrix = pd.DataFrame(data = cm , columns = ['Predicted : 0','Predicted : 1'],
c = ['#c3dbf5']

sns.heatmap(data = conf_matrix, annot = True , cbar = False , fmt = 'd' , cmap = c ,
            annot_kws = {'size' : 25})
plt.show()
```



In [166]:

```
# plotting confusion matrix for test data

cm = confusion_matrix(ytest,pred_test)
conf_matrix = pd.DataFrame(data = cm , columns = ['Predicted : 0','Predicted : 1'],
c = ['#c3dbf5']

sns.heatmap(data = conf_matrix, annot = True , cbar = False , fmt = 'd' , cmap = c ,
           annot_kws = {'size' : 25})
plt.show()
```



In [167]:

```
# Classification report

print(f'Train report : \n{classification_report(ytrain,pred_train)}\n')
print(f'Testreport : \n{classification_report(ytest,pred_test)}')
```

```
Train report :
              precision    recall  f1-score   support

           0       0.61      0.83      0.70    114747
           1       0.73      0.48      0.58    114747

    accuracy                           0.65    229494
   macro avg       0.67      0.65      0.64    229494
weighted avg       0.67      0.65      0.64    229494


Testreport :
              precision    recall  f1-score   support

           0       0.61      0.83      0.70     49177
           1       0.73      0.47      0.58     49177

    accuracy                           0.65     98354
   macro avg       0.67      0.65      0.64     98354
weighted avg       0.67      0.65      0.64     98354
```


From above report we can conclude that our base model has performed good in both train and unseen data with accuracy of 67%. On further progress we try to improve our performance by building other models, tuning their hyperparameters and selecting columns based on feature importance score

Building different models and evaluating using appropriate technique

In [168]:

```
# Creating a user defined function to store values of accuracy , f1 score , auc_scor

performance_df = pd.DataFrame(columns = ['Model_Name','Train_Accuracy','Train_F1scor
                                         'Test_F1score','Precision_Score','Recall_Sc

def model_performance(model, name, xtrain = xtrain , xtest = xtest):
    global performance_df

    # predicting train and test data

    pred_train = model.predict(xtrain)
    pred_test = model.predict(xtest)
    pred_prob = model.predict_proba(xtest)[:,1]

    # calculating metrics for both train and test data

    acc_train = round(accuracy_score(ytrain,pred_train),2)*100
    acc_test = round(accuracy_score(ytest,pred_test),2)*100
    f1_train = round(f1_score(ytrain,pred_train),2)
    f1_test = round(f1_score(ytest,pred_test),2)
    precision = round(precision_score(ytest,pred_test),2)
    recall = round(recall_score(ytest,pred_test),2)
    auc_score = round(roc_auc_score(ytest,pred_prob),4)

    # defining function for remarks

    def remark(train_acc,test_acc):
        if abs(train_acc - test_acc) > 3 or train_acc > 95:
            return 'Over Fit'
        elif train_acc < 65 or test_acc < 65:
            return 'Under Fit'
        else:
            return 'Good Fit'

    # adding train and test scores in performance_df dataframe

    performance_df = performance_df.append({'Model_Name':name,'Train_Accuracy':acc_t
                                            'Test_Accuracy':acc_test,'Test_F1score':f
                                             'Precision_Score':precision , 'Recall_Sc
                                            'AUC_Score':auc_score,'Remarks':remark(ac

    # plotting roc_curve and calculating auc_score

    fpr , tpr , threshold = roc_curve(ytest,pred_prob)

    plt.plot(fpr,tpr)
    plt.plot([0.0,1.0],'r--')
    plt.text(x = 0.1, y = 1.0, s = ('auc_score',round(roc_auc_score(ytest,pred_test)

    print('Train Report :\n',classification_report(ytrain,pred_train))
    print('Test Report :\n',classification_report(ytest,pred_test))
```


In [169]:

```
# Creating a user defined function to highlight the rows which are good fit

def highlight_row(df):
    color_green = ['background-color : lightgreen']*len(df)
    color_white = ['background-color : white']*len(df)

    if df['Remarks'] == 'Good Fit':
        return color_green
    else:
        return color_white
```

In [170]:

```
# Appending values of our base model to dataframe

performance_df = performance_df.append({'Model_Name':'Base Model',
                                        'Train_Accuracy':round(accuracy_score(ytrain
                                        'Train_F1score':round(f1_score(ytrain,pred_t
                                        'Test_Accuracy':round(accuracy_score(ytest,p
                                        'Test_F1score':round(f1_score(ytest,pred_tes
                                        'Precision_Score':round(precision_score(ytes
                                        'Recall_Score':round(recall_score(ytest,prec
                                        'AUC_Score':roc_auc_score(ytest,pred_prob_te
                                        'Remarks':'Base'},ignore_index=True)
```

In [171]:

```
performance_df
```

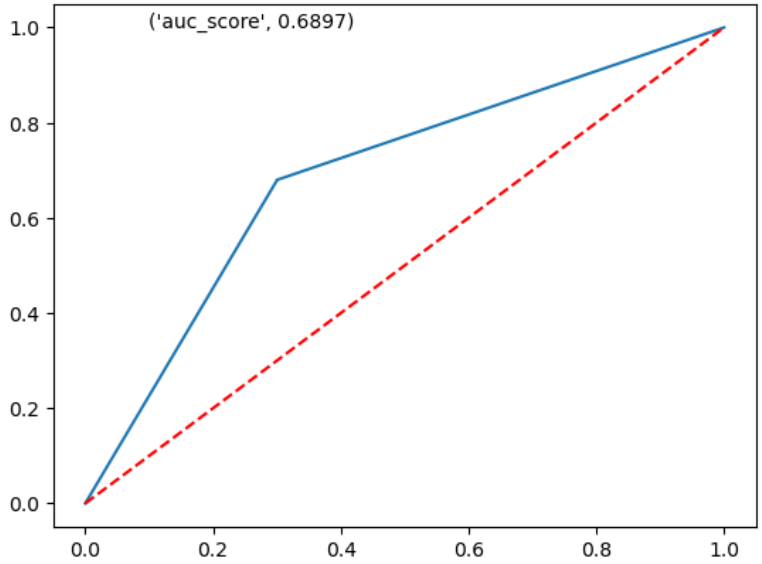
Out[171]:

	Model_Name	Train_Accuracy	Train_F1score	Test_Accuracy	Test_F1score	Precision_Scor
0	Base Model	65.0	0.58	65.0	0.58	0.7

In [6]:

```
# DecisionTree Model without tuning

model_dt = DecisionTreeClassifier().fit(xtrain,ytrain)

model_performance(model_dt,'DecisionTree Model without tuning')
```

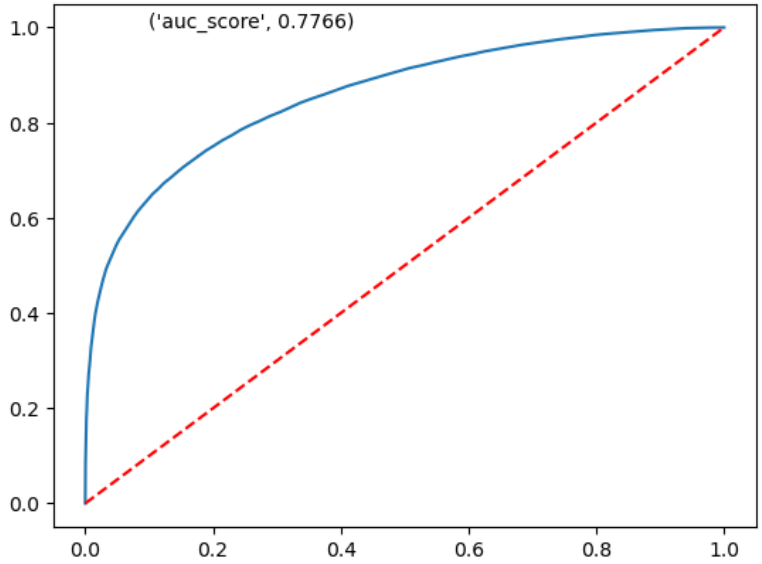
```
Train Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    114747
           1       1.00      1.00      1.00    114747

    accuracy                           1.00    229494
   macro avg       1.00      1.00      1.00    229494
weighted avg       1.00      1.00      1.00    229494

Test Report :
              precision    recall  f1-score   support

           0       0.69      0.70      0.69     49177
           1       0.69      0.68      0.69     49177

    accuracy                           0.69     98354
   macro avg       0.69      0.69      0.69     98354
weighted avg       0.69      0.69      0.69     98354
```



In [7]:

```
# Random Forest without tuning

model_rf = RandomForestClassifier().fit(xtrain,ytrain)

model_performance(model_rf,'Random Forest without tuning')
```

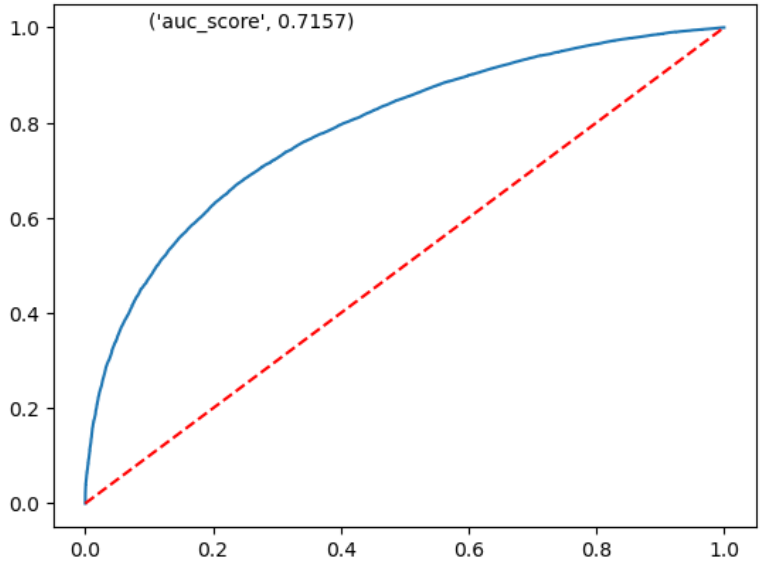
```
Train Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    114747
           1       1.00      1.00      1.00    114747

    accuracy                           1.00    229494
   macro avg       1.00      1.00      1.00    229494
weighted avg       1.00      1.00      1.00    229494

Test Report :
              precision    recall  f1-score   support

           0       0.75      0.83      0.79     49177
           1       0.81      0.72      0.76     49177

    accuracy                           0.78     98354
   macro avg       0.78      0.78      0.78     98354
weighted avg       0.78      0.78      0.78     98354
```



In [8]:

```
# AdaBoost without tuning

model_ab = AdaBoostClassifier().fit(xtrain,ytrain)

model_performance(model_ab,'AdaBoost without tuning')
```

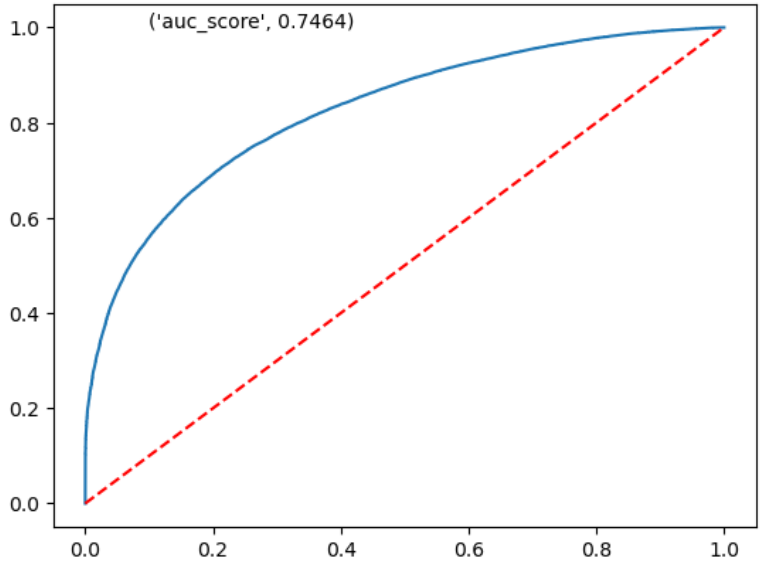
```
Train Report :
              precision    recall  f1-score   support

           0       0.70      0.74      0.72    114747
           1       0.73      0.69      0.71    114747

    accuracy                           0.71    229494
   macro avg       0.72      0.71      0.71    229494
weighted avg       0.72      0.71      0.71    229494

Test Report :
              precision    recall  f1-score   support

           0       0.71      0.74      0.72     49177
           1       0.73      0.69      0.71     49177

    accuracy                           0.72     98354
   macro avg       0.72      0.72      0.72     98354
weighted avg       0.72      0.72      0.72     98354
```



In [9]:

```
# Gradient Boosting without tuning

model_gb = GradientBoostingClassifier().fit(xtrain,ytrain)

model_performance(model_gb,'GradientBoosting without tuning')
```

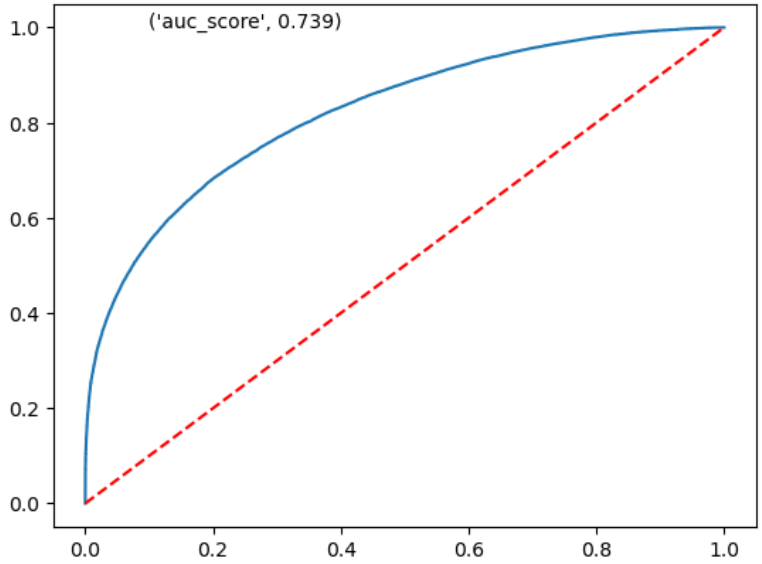
```
Train Report :
              precision    recall  f1-score   support

           0       0.73      0.79      0.75    114747
           1       0.77      0.70      0.73    114747

    accuracy                           0.74    229494
   macro avg       0.75      0.74      0.74    229494
weighted avg       0.75      0.74      0.74    229494

Test Report :
              precision    recall  f1-score   support

           0       0.73      0.79      0.76     49177
           1       0.77      0.71      0.74     49177

    accuracy                           0.75     98354
   macro avg       0.75      0.75      0.75     98354
weighted avg       0.75      0.75      0.75     98354
```



In [10]:

```
from sklearn.neural_network import MLPClassifier

model_nn = MLPClassifier().fit(xtrain,ytrain)

model_performance(model_nn , 'Neural Network')
```

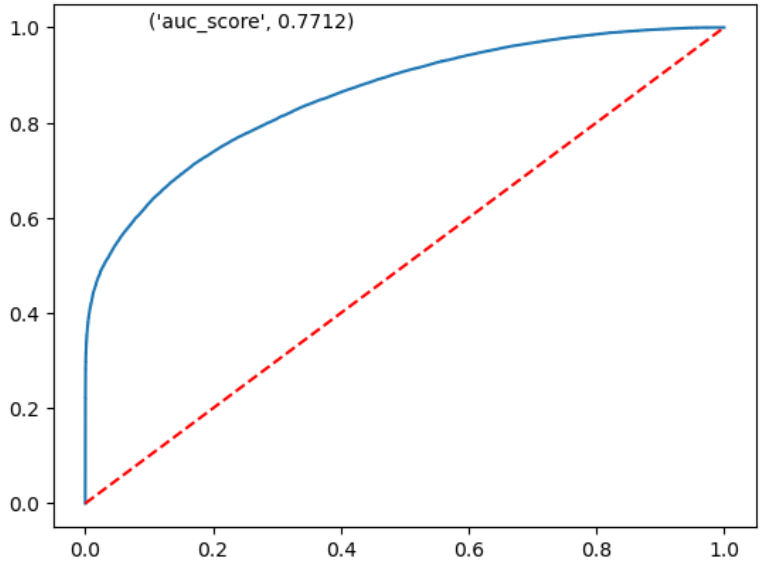
```
Train Report :
              precision    recall  f1-score   support

           0       0.75      0.77      0.76    114747
           1       0.76      0.74      0.75    114747

    accuracy                           0.75    229494
   macro avg       0.75      0.75      0.75    229494
weighted avg       0.75      0.75      0.75    229494

Test Report :
              precision    recall  f1-score   support

           0       0.73      0.76      0.74     49177
           1       0.75      0.72      0.73     49177

    accuracy                           0.74     98354
   macro avg       0.74      0.74      0.74     98354
weighted avg       0.74      0.74      0.74     98354
```



In [11]:

```
# XGB without tuning

model_xgb = XGBClassifier().fit(xtrain,ytrain)

model_performance(model_xgb,'Xtreme Gradient Boosting without tuning')
```

```
Train Report :
              precision    recall  f1-score   support

           0       0.76      0.84      0.80    114747
           1       0.82      0.73      0.77    114747

    accuracy                           0.79    229494
   macro avg       0.79      0.79      0.79    229494
weighted avg       0.79      0.79      0.79    229494

Test Report :
              precision    recall  f1-score   support

           0       0.74      0.83      0.78     49177
           1       0.81      0.71      0.76     49177

    accuracy                           0.77     98354
   macro avg       0.77      0.77      0.77     98354
weighted avg       0.77      0.77      0.77     98354
```



In [12]:

```
model_catboost = CatBoostClassifier().fit(xtrain,ytrain)

model_performance(model_catboost , 'Catboost')
```

```
Learning rate set to 0.10495
0:      learn: 0.6711363        total: 262ms    remaining: 4m 21s
1:      learn: 0.6534581        total: 316ms    remaining: 2m 37s
2:      learn: 0.6391070        total: 367ms    remaining: 2m 2s
3:      learn: 0.6272722        total: 414ms    remaining: 1m 43s
4:      learn: 0.6173311        total: 469ms    remaining: 1m 33s
5:      learn: 0.6094159        total: 525ms    remaining: 1m 27s
6:      learn: 0.6025543        total: 580ms    remaining: 1m 22s
7:      learn: 0.5945876        total: 633ms    remaining: 1m 18s
8:      learn: 0.5872053        total: 687ms    remaining: 1m 15s
9:      learn: 0.5799279        total: 741ms    remaining: 1m 13s
10:     learn: 0.5764627        total: 794ms    remaining: 1m 11s
11:     learn: 0.5709717        total: 852ms    remaining: 1m 10s
12:     learn: 0.5672991        total: 915ms    remaining: 1m 9s
13:     learn: 0.5630559        total: 969ms    remaining: 1m 8s
14:     learn: 0.5597751        total: 1.02s    remaining: 1m 7s
15:     learn: 0.5553205        total: 1.07s    remaining: 1m 6s
16:     learn: 0.5531131        total: 1.13s    remaining: 1m 5s
17:     learn: 0.5505073        total: 1.18s    remaining: 1m 4s
```

In [13]:

```
performance_df.style.apply(highlight_row,axis=1)
```

Out[13]:

	Model_Name	Train_Accuracy	Train_F1score	Test_Accuracy	Test_F1score	Precision_S
0	Base Model	66.000000	0.660000	66.000000	0.650000	0.67
1	DecisionTree Model without tuning	100.000000	1.000000	69.000000	0.690000	0.69
2	Random Forest without tuning	100.000000	1.000000	78.000000	0.760000	0.81
3	AdaBoost without tuning	71.000000	0.710000	72.000000	0.710000	0.73
4	GradientBoosting without tuning	74.000000	0.730000	75.000000	0.740000	0.77
5	Neural Network	75.000000	0.750000	74.000000	0.730000	0.75
6	Xtreme Gradient Boosting without tuning	79.000000	0.770000	77.000000	0.760000	0.81
7	Catboost	80.000000	0.790000	78.000000	0.760000	0.81

After assessing various models, it was observed that some models exhibited a significant drop in performance when applied to unseen data, indicating overfitting. However, there were models that consistently performed well on both training and unseen data. Notably, the Catboost model outperformed other models in terms of performance(78%). Hence, based on its superior performance and generalization

In [ ]:

```
# Freeze the model

import pickle

model = open("model_catboost.pickle", "wb")
pickle.dump(model_catboost, model)
model.close()
```

In [1]:

```
%%writefile app.py

import streamlit as st
import pandas as pd
import numpy as np
import pickle
import base64


with open("image1.jpg", "rb") as image_file:
    encoded_string = base64.b64encode(image_file.read())
st.markdown(
f"""
<style>
.stApp {{
    background-image: url({encoded_string.decode()});
    background-size: cover
}}
</style>
""",
unsafe_allow_html=True
)

st.title("Check your Instant Bookable status")

st.markdown("Tell us about your property 🏠")

# Loading all encoding variables

# Amenities
with open('amenities_TFIDF.pkl','rb') as file:
    en_amenities = pickle.load(file)

# encoder_city

with open('encoder_city.pkl','rb') as file:
    en_city = pickle.load(file)

# encoder_host_response_time.pkl
with open('encoder_host_response_time.pkl','rb') as file:
    en_response_time = pickle.load(file)

#encoder_neighbourhood.pkl
with open('encoder_neighbourhood.pkl','rb') as file:
    en_neighbourhood = pickle.load(file)

# encoder_property_type.pkl
with open('encoder_property_type.pkl','rb') as file:
    en_property_type = pickle.load(file)

# encoder_region.pkl

with open('encoder_region.pkl','rb') as file:
    en_region = pickle.load(file)

# encoder_room_type.pkl
with open('encoder_room_type.pkl','rb') as file:
    en_room_type = pickle.load(file)

```



```

# Loading transformation
with open('accommodates.pkl','rb') as file:
    trans_accommodates = pickle.load(file)

with open('bedrooms.pkl','rb') as file:
    trans_bedroom = pickle.load(file)

with open('host_total_listings_count.pkl','rb') as file:
    trans_host_listing_count = pickle.load(file)

with open('minimum_nights.pkl','rb') as file:
    trans_min_nights = pickle.load(file)

with open('price.pkl','rb') as file:
    trans_price = pickle.load(file)

with open('review_scores_location.pkl','rb') as file:
    trans_re_location = pickle.load(file)

# Scaling

with open('scaling_host_acceptance_rate.pkl','rb') as file:
    scal_accet_rate = pickle.load(file)

with open('scaling_host_response_rate.pkl','rb') as file:
    scal_response_rate = pickle.load(file)

with open('scaling_longitude.pkl','rb') as file:
    scal_longitude = pickle.load(file)

with open('scaling_review_scores_rating.pkl','rb') as file:
    scal_re_rating = pickle.load(file)


#Load model
with open('model_catboost.pickle','rb') as file:
    cat_boost = pickle.load(file)


# Getting user input

host_response_time = st.selectbox('Select response time',('unknown', 'within a day',
                                                          'within a few hours','a fe
host_response_rate = st.slider('Your response rate',1,100,1)
host_acceptance_rate = st.slider('Your acceptance rate',1,100,1)
host_is_superhost = st.selectbox('Your superhost status',('Yes','No'))
host_total_listings_count = st.number_input('Enter no. of properties listed',1,50,1
host_identity_verified = st.selectbox('Do you want id verified status',('Yes','No')
neighbourhood = st.selectbox('Select your neighbourhood',('Buttes-Montmartre', 'Ely
       'Popincourt', 'Buttes-Chaumont', 'Opera', 'Gobelins',
       'Hotel-de-Ville', 'Pantheon', 'Enclos-St-Laurent',
       'Batignolles-Monceau', 'Luxembourg', 'Reuilly', 'Menilmontant',
       'Observatoire', 'Palais-Bourbon', 'Bourse', 'Louvre',
       'Lower East Side', 'Harlem', 'Crown Heights', 'Nolita', 'Midtown',
       'Greenwich Village', 'Williamsburg', 'East Village',
       'Upper West Side', "Hell's Kitchen", 'Park Slope', 'Chinatown',
       'Upper East Side', 'Columbia St', 'Inwood', 'Bedford-Stuyvesant',
       'Astoria', 'Little Italy', 'Fort Greene', 'Bushwick',
       'East Harlem', 'Washington Heights', 'Woodside', 'Greenpoint',
       'Ditmars Steinway', 'Sheepshead Bay', 'West Village',
```

```
        'Murray Hill', 'Prospect Heights', 'Brooklyn Heights',
        'Richmond Hill', 'Roosevelt Island', 'Flatiron District',
        'Ridgewood', 'Kips Bay', 'Chelsea', 'Battery Park City',
        'Long Island City', 'Flatbush', 'SoHo', 'Theater District',
        'Clinton Hill', 'Tribeca', 'Borough Park', 'Sunset Park',
        'Canarsie', 'Gramercy', 'Queens Village', 'DUMBO', 'Glendale',
        'Flushing', 'Jamaica', 'Prospect-Lefferts Gardens',
        'Downtown Brooklyn', 'Sunnyside', 'Windsor Terrace',
        'Financial District', 'East New York', 'Carroll Gardens',
        'Jackson Heights', 'Jamaica Estates', 'Morris Heights',
        'Boerum Hill', 'Vinegar Hill', 'Fort Hamilton', 'Woodhaven',
        'Maspeth', 'East Elmhurst', 'Flatlands', 'Ozone Park', 'Gowanus',
        'Cobble Hill', 'Morrisania', 'South Slope', 'St. George',
        'East Morrisania', 'Civic Center', 'Forest Hills', 'Cypress Hills',
        'Morningside Heights', 'Bayside', 'East Flatbush', 'Bay Ridge',
        'Kensington', 'Williamsbridge', 'Bensonhurst', 'NoHo', 'Midwood',
        "Prince's Bay", 'New Dorp', 'Stuyvesant Town', 'Two Bridges',
        'Concourse', 'Elmhurst', 'Kew Gardens', 'Norwood', 'Coney Island',
        'Kew Gardens Hills', 'Rego Park', 'Manhattan Beach',
        'Brighton Beach', 'Red Hook', 'Hunts Point', 'Mott Haven',
        'Gravesend', 'Longwood', 'Mount Eden', 'Middle Village',
        'Dyker Heights', 'Corona', 'Randall Manor', 'Bath Beach',
        'Parkchester', 'Kingsbridge', 'Hollis', 'Highbridge', 'Navy Yard',
        'Riverdale', 'Wakefield', 'Bronxdale', 'Arverne', 'Brownsville',
        'Mariners Harbor', 'Tompkinsville', 'Port Morris', 'Briarwood',
        'Tremont', 'Great Kills', 'Springfield Gardens', 'Arden Heights',
        'St. Albans', 'Spuyten Duyvil', 'Eastchester', 'Arrochar',
        'Mount Hope', 'Far Rockaway', 'Castleton Corners', 'Dongan Hills',
        'Van Nest', 'Stapleton', 'Laurelton', 'Unionport',
        'Rockaway Beach', 'South Ozone Park', 'Jamaica Hills',
        'Pelham Gardens', 'North Riverdale', "Bull's Head", 'Douglaston',
        'Morris Park', 'City Island', 'Oakwood', 'Bellerose',
        'College Point', 'Grymes Hill', 'Rosedale', 'Claremont Village',
        'Todt Hill', 'University Heights', 'Soundview', 'Howard Beach',
        'Woodlawn', 'Fieldston', 'Edenwald', 'Fresh Meadows',
        'Silver Lake', 'Graniteville', 'Grant City', 'Pelham Bay',
        'Lighthouse Hill', 'Baychester', 'Westerleigh', 'Concord',
        'New Springville', 'Cambria Heights', 'Clason Point', 'Allerton',
        'Willowbrook', 'Belle Harbor', 'Holliswood', 'Shore Acres',
        'Marble Hill', 'Little Neck', 'Sea Gate', 'West Brighton',
        'Concourse Village', 'New Brighton', 'Bayswater', 'Howland Hook',
        'Woodrow', 'Throgs Neck', 'Emerson Hill', 'Eltingville', 'Clifton',
        'Fordham', 'Whitestone', 'Westchester Square', 'Port Richmond',
        'Midland Beach', 'Rosebank', 'Edgemere', 'Mill Basin',
        'Co-op City', 'Olinville', 'Belmont', 'Bay Terrace', 'Melrose',
        'West Farms', 'New Dorp Beach', 'Bergen Beach', 'Huguenot',
        'Schuylerville', 'South Beach', 'Gerritsen Beach', 'Country Club',
        'Richmondtown', 'Fort Wadsworth', 'Tottenville', 'Rossville',
        'Castle Hill', 'Vadhana', 'Khlong Toei', 'Lat Krabang',
        'Rat Burana', 'Sathon', 'Bang Sue', 'Yan na wa', 'Chatu Chak',
        'Bang Kapi', 'Phasi Charoen', 'Bang Phlat', 'Khlong San',
        'Ratchathewi', 'Huai Khwang', 'Bang Na', 'Phra Khanong',
        'Din Daeng', 'Pra Wet', 'Bang Kho laen', 'Thon buri', 'Bang Rak',
        'Sai Mai', 'Parthum Wan', 'Bangkok Noi', 'Wang Thong Lang',
        'Don Mueang', 'Bang Khen', 'Bang Khae', 'Phra Nakhon', 'Suanluang',
        'Bueng Kum', 'Phaya Thai', 'Khan Na Yao', 'Bangkok Yai',
        'Chom Thong', 'Thawi Watthana', 'Taling Chan', 'Lak Si',
        'Bang Khun thain', 'Saphan Sung', 'Nong Khaem', 'Khlong Sam Wa',
        'Lat Phrao', 'Samphanthawong', 'Dusit', 'Pom Prap Sattru Phai',
        'Min Buri', 'Thung khru', 'Nong Chok', 'Bang Bon', 'Leblon',
        'Ipanema', 'Copacabana', 'Humaita', 'Recreio dos Bandeirantes',
```



```
        'Leme', 'Barra da Tijuca', 'Tijuca', 'Botafogo', 'Laranjeiras',
        'Centro', 'Flamengo', 'Catete', 'Vila Isabel', 'Gloria',
        'Jacarepagua', 'Camorim', 'Santa Teresa', 'Lagoa', 'Campo Grande',
        'Rio Comprido', 'Gavea', 'Sao Francisco Xavier', 'Sao Conrado',
        'Maracana', 'Cidade de Deus', 'Praca da Bandeira', 'Pechincha',
        'Jardim Botanico', 'Freguesia (Jacarepagua)', 'Grajau', 'Urca',
        'Cosme Velho', 'Riachuelo', 'Benfica', 'Curicica', 'Taquara',
        'Engenho de Dentro', 'Higienopolis', 'Engenho da Rainha',
        'Quintino Bocaiuva', 'Senador Vasconcelos', 'Cosmos', 'Encantado',
        'Sampaio', 'Pilares', 'Piedade', 'Cidade Nova', 'Andarai',
        'Engenho Novo', 'Sao Cristovao', 'Itanhanga', 'Mangueira',
        'Gardenia Azul', 'Praca Seca', 'Rocha', 'Campinho', 'Guaratiba',
        'Joa', 'Realengo', 'Todos os Santos', 'Guadalupe',
        'Parada de Lucas', 'Pavuna', 'Estacio', 'Jardim Guanabara', 'Anil',
        'Freguesia (Ilha)', 'Vargem Pequena', 'Meier', 'Saude',
        'Vargem Grande', 'Vidigal', 'Rocinha', 'Inhoaiba', 'Santa Cruz',
        'Bancarios', 'Lins de Vasconcelos', 'Zumbi', 'Madureira',
        'Paqueta', 'Tanque', 'Vicente de Carvalho', 'Abolicao',
        'Portuguesa', 'Cachambi', 'Iraja', 'Jardim Carioca',
        'Tomas Coelho', 'Penha Circular', 'Barra de Guaratiba',
        'Del Castilho', 'Sepetiba', 'Ribeira', 'Vila da Penha',
        'Ricardo de Albuquerque', 'Vigario Geral', 'Bonsucesso', 'Ramos',
        'Monero', 'Taua', 'Bras de Pina', 'Gamboa', 'Maria da Graca',
        'Bangu', 'Vasco da Gama', 'Santissimo', 'Jardim Sulacap', 'Penha',
        'Magalhaes Bastos', 'Bento Ribeiro', 'Alto da Boa Vista',
        'Pedra de Guaratiba', 'Paciencia', 'Catumbi', 'Pitangueiras',
        'Cacuia', 'Vila Valqueire', 'Honorio Gurgel', 'Galeao',
        'Vila Militar', 'Padre Miguel', 'Grumari', 'Manguinhos',
        'Rocha Miranda', 'Cascadura', 'Olaria', 'Coelho Neto', 'Vaz Lobo',
        'Anchieta', 'Praia da Bandeira', 'Marechal Hermes', 'Inhauma',
        'Jacare', 'Parque Anchieta', 'Cordovil', 'Cocota',
        'Cidade Universitaria', 'Gericino', 'Santo Cristo', 'Barros Filho',
        'Vista Alegre', 'Cavalcanti', 'Senador Camara',
        'Complexo do Alemao', 'Deodoro', 'Vila Kosmos', 'Osvaldo Cruz',
        'Mare', 'Acari', 'Agua Santa', 'Waverley', 'North Sydney',
        'Parramatta', 'Ryde', 'Sydney', 'Rockdale', 'Manly', 'Warringah',
        'Randwick', 'Hurstville', 'Pittwater', 'Canterbury',
        'City Of Kogarah', 'Fairfield', 'Woollahra', 'Auburn', 'Ashfield',
        'Bankstown', 'Sutherland Shire', 'Willoughby', 'Hornsby',
        'Marrickville', 'Liverpool', 'Leichhardt', 'Mosman', 'Botany Bay',
        'Canada Bay', 'Burwood', 'Blacktown', 'Strathfield',
        'The Hills Shire', 'Hunters Hill', 'Ku-Ring-Gai', 'Lane Cove',
        'Holroyd', 'Penrith', 'Camden', 'Campbelltown', 'Adalar',
        'Uskudar', 'Sisli', 'Beyoglu', 'Kadikoy', 'Sariyer', 'Besiktas',
        'Pendik', 'Maltepe', 'Bakirkoy', 'Bagcilar', 'Fatih', 'Eyup',
        'Beylikduzu', 'Basaksehir', 'Esenyurt', 'Bayrampasa',
        'Buyukcekmece', 'Kucukcekmece', 'Umraniye', 'Avcilar', 'Kartal',
        'Zeytinburnu', 'Kagithane', 'Atasehir', 'Silivri', 'Gaziosmanpasa',
        'Beykoz', 'Sancaktepe', 'Sultangazi', 'Bahcelievler', 'Gungoren',
        'Cekmekoy', 'Tuzla', 'Sultanbeyli', 'Arnavutkoy', 'Sile',
        'Esenler', 'Catalca', 'I Centro Storico', 'II Parioli/Nomentano',
        'IV Tiburtina', 'V Prenestino/Centocelle',
        'VII San Giovanni/Cinecitta', 'XIII Aurelia', 'III Monte Sacro',
        'X Ostia/Acilia', 'XV Cassia/Flaminia', 'VIII Appia Antica',
        'XIV Monte Mario', 'XII Monte Verde', 'IX Eur',
        'XI Arvalia/Portuense', 'VI Roma delle Torri', 'Southern',
        'Yau Tsim Mong', 'Islands', 'Wan Chai', 'Central & Western',
        'Sham Shui Po', 'Kowloon City', 'Eastern', 'Yuen Long',
        'Tsuen Wan', 'Wong Tai Sin', 'North', 'Sha Tin', 'Sai Kung',
        'Kwun Tong', 'Kwai Tsing', 'Tuen Mun', 'Tai Po',
        'A\x81lvaro Obregon', 'Coyoacan', 'Benito Juarez', 'Cuauhtemoc',
```

```
        'Azcapotzalco', 'Miguel Hidalgo', 'Cuajimalpa de Morelos',
        'Iztacalco', 'Venustiano Carranza', 'Tlahuac', 'Tlalpan',
        'Iztapalapa', 'La Magdalena Contreras', 'Gustavo A. Madero',
        'Xochimilco', 'Milpa Alta', 'Ward 115', 'Ward 3', 'Ward 77',
        'Ward 55', 'Ward 64', 'Ward 54', 'Ward 59', 'Ward 61', 'Ward 113',
        'Ward 105', 'Ward 8', 'Ward 15', 'Ward 100', 'Ward 74', 'Ward 27',
        'Ward 84', 'Ward 71', 'Ward 107', 'Ward 83', 'Ward 11', 'Ward 62',
        'Ward 73', 'Ward 23', 'Ward 53', 'Ward 21', 'Ward 69', 'Ward 1',
        'Ward 57', 'Ward 58', 'Ward 70', 'Ward 60', 'Ward 85', 'Ward 4',
        'Ward 86', 'Ward 2', 'Ward 10', 'Ward 103', 'Ward 109', 'Ward 112',
        'Ward 48', 'Ward 66', 'Ward 5', 'Ward 102', 'Ward 43', 'Ward 14',
        'Ward 67', 'Ward 7', 'Ward 20', 'Ward 63', 'Ward 46', 'Ward 44',
        'Ward 26', 'Ward 30', 'Ward 72', 'Ward 56', 'Ward 9', 'Ward 104',
        'Ward 116', 'Ward 94', 'Ward 65', 'Ward 78', 'Ward 22', 'Ward 17',
        'Ward 108', 'Ward 51', 'Ward 32', 'Ward 49', 'Ward 92', 'Ward 19',
        'Ward 75', 'Ward 110', 'Ward 68', 'Ward 12', 'Ward 76', 'Ward 93',
        'Ward 16', 'Ward 31', 'Ward 6', 'Ward 50', 'Ward 91', 'Ward 81',
        'Ward 29', 'Ward 25', 'Ward 28', 'Ward 82', 'Ward 111', 'Ward 101',
        'Ward 41', 'Ward 96', 'Ward 38', 'Ward 45', 'Ward 18', 'Ward 40'))


city = st.selectbox('Select city',('Paris', 'New York', 'Bangkok', 'Rio de Janeiro',
        'Istanbul', 'Rome', 'Hong Kong', 'Mexico City', 'Cape Town'))

property_type = st.selectbox('Select your property type',('Entire apartment', 'Enti
        'Entire condominium', 'Private room in apartment',
        'Private room in condominium', 'Entire guest suite', 'Earth house',
        'Entire townhouse', 'Room in serviced apartment',
        'Private room in bed and breakfast', 'Entire serviced apartment',
        'Private room in house', 'Entire villa', 'Tiny house',
        'Private room in guest suite', 'Private room in loft',
        'Room in boutique hotel', 'Entire place', 'Room in hotel',
        'Entire floor', 'Entire guesthouse', 'Houseboat', 'Entire cottage',
        'Boat', 'Private room in guesthouse', 'Entire home/apt',
        'Room in bed and breakfast', 'Room in aparthotel',
        'Private room in villa', 'Private room in cabin',
        'Shared room in apartment', 'Private room in townhouse',
        'Private room in chalet', 'Entire bed and breakfast', 'Cave',
        'Shared room in condominium', 'Private room in boat',
        'Shared room in serviced apartment', 'Shared room in hostel',
        'Entire bungalow', 'Private room',
        'Private room in serviced apartment',
        'Private room in earth house', 'Campsite', 'Shared room in loft',
        'Shared room in cabin', 'Room in hostel',
        'Shared room in bed and breakfast', 'Private room in hostel',
        'Shared room in boutique hotel', 'Private room in houseboat',
        'Shared room in house', 'Shared room in townhouse',
        'Shared room in igloo', 'Treehouse',
        'Private room in casa particular', 'Shared room in tiny house',
        'Shared room in guesthouse', 'Shared room in guest suite',
        'Entire chalet', 'Private room in nature lodge', 'Island',
        'Dome house', 'Camper/RV', 'Barn', 'Casa particular',
        'Private room in resort', 'Private room in tiny house',
        'Private room in camper/rv', 'Private room in barn',
        'Private room in tent', 'Private room in bungalow',
        'Private room in dome house', 'Private room in castle',
        'Shared room in floor', 'Shared room in bungalow',
        'Private room in in-law', 'Private room in farm stay',
        'Private room in lighthouse', 'Bus', 'Shared room in island',
        'Private room in cottage', 'Entire resort',
        'Shared room in earth house', 'Private room in dorm',
```



```
        'Room in resort', 'Private room in floor', 'Private room in train',
        'Lighthouse', 'Castle', 'Farm stay', 'Private room in island',
        'Entire dorm', 'Entire cabin', 'Private room in kezhan',
        'Shared room in dorm', 'Shared room in kezhan', 'Entire hostel',
        'Shared room in chalet', 'Shared room in cave',
        'Shared room in villa', 'Private room in hut',
        'Shared room in parking space', 'Shared room in dome house',
        'Shared room', 'Shared room in casa particular',
        'Private room in tipi', 'Room in nature lodge',
        'Private room in holiday park', 'Pension',
        'Shared room in cottage', 'Shared room in farm stay', 'Hut',
        'Shared room in nature lodge', 'Private room in treehouse',
        'Shared room in castle', 'Entire vacation home', 'Yurt',
        'Room in apartment', 'Private room in minsu',
        'Private room in pension', 'Private room in yurt', 'Tent', 'Train',
        'Shared room in tent', 'Shared room in boat',
        'Private room in bus', 'Holiday park', 'Room in casa particular',
        'Shared room in yurt', 'Private room in pousada',
        'Shared room in pension', 'Shared room in aparthotel',
        'Shared room in hotel', 'Room in pension', 'Igloo', 'Tipi',
        'Shared room in hut', 'Entire in-law', 'Private room in cave',
        'Room in guesthouse', 'Room in heritage hotel'))
room_type = st.selectbox('Select Room type',('Entire place', 'Private room', 'Hotel

accommodates = st.number_input('No. of people can be accommodated',1,16,1)

bedrooms = st.number_input('No.of bedrooms',1,20,1)

price = st.number_input('Price')

min_nights = st.slider('Minimum nights allowed',1,50,1)

region = st.selectbox('Enter your region',('East', 'North', 'West', 'South'))

amenities = st.multiselect('Select amenities',('long term stay','parking','hanger',
                                               'iron','washer','water heating','worl
                                               'fire alarm'))


# Applying encoding ,transformation,scaling on user data

encode_res_time = en_response_time.transform(pd.DataFrame([host_response_time]))
scale_res_rate = scal_response_rate.transform(pd.DataFrame([host_response_rate]))
scale_act_rate = scal_accet_rate.transform(pd.DataFrame([host_acceptance_rate]))
encode_superhost = [1 if host_is_superhost == "Yes" else 0]
trans_list_count = trans_host_listing_count.transform(pd.DataFrame([host_total_list
encode_id_verified = [1 if host_identity_verified == "Yes" else 0]
encode_neighbourhood = en_neighbourhood.transform(pd.DataFrame({"neighbourhood" :
                                                                [neighbourhood]}))
encode_city = en_city.transform(pd.DataFrame({"city" :[city]}))
encode_property_type = en_property_type.transform(pd.DataFrame({"property_type" :
                                                                [property_type]}))
encode_room_type = en_room_type.transform(pd.DataFrame([room_type]))
trans_acc = trans_accommodates.transform(pd.DataFrame([accommodates]))
trans_bed = trans_bedroom.transform(pd.DataFrame([bedrooms]))
trans_pri = trans_price.transform(pd.DataFrame([price]))
trans_min = trans_min_nights.transform(pd.DataFrame([min_nights]))
encode_region = en_region.transform(pd.DataFrame({'Region':[region]}))


data_1 = {'host_response_time' : encode_res_time[0],
```

```
        'host_response_rate': scale_res_rate[0]/100,
        'host_acceptance_rate':scale_act_rate[0]/100,
       'host_is_superhost' : encode_superhost[0],
       'host_total_listings_count':trans_list_count[0],
       'host_identity_verified':encode_id_verified[0],
       'neighbourhood' : encode_neighbourhood.iloc[0,0],
       'city' : encode_city.iloc[0,0],
       'property_type' : encode_property_type.iloc[0,0],
       'room_type' : encode_room_type[0],
       'accommodates' : trans_acc[0],
       'bedrooms' : trans_bed[0],
       'price' : trans_pri[0],
       'minimum_nights' : trans_min[0],
       'review_scores_rating' : scal_re_rating.transform(pd.DataFrame([55]))[0],
       'review_scores_location' : trans_re_location.transform(pd.DataFrame([50]))[0
       'Region' : encode_region.iloc[0,0]}
data_1 = pd.DataFrame(data_1)

data_2 = en_amenities.transform([' '.join(amenities)]).toarray()
data_2 = pd.DataFrame(data_2,columns=en_amenities.get_feature_names_out())

df = pd.concat([data_1,data_2],axis=1)

# Model prediction
prediction = cat_boost.predict(df)
if st.button('Know your status'):
    if prediction ==1 :
        st.success('Your property is instantly bookable')
    else:
        st.error('Please increase the quality of the property to get instant bookabl
```

Overwriting app.py