Airbnb prices in european cities

IMPORT NECESSARY LIBRARIES

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SequentialFeatureSelector,Sel
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,r2_score
import warnings
```

IMPORT ALL CSV DATA FILES AND COMBINE THEM

IMPORT DATASETS

In [3]:
```
amsterdam_weekdays = pd.read_csv('amsterdam_weekdays.csv')
amsterdam_weekends = pd.read_csv('amsterdam_weekends.csv')
athens_weekdays = pd.read_csv('athens_weekdays.csv')
athens_weekends = pd.read_csv('athens_weekends.csv')
barcelona_weekdays = pd.read_csv('barcelona_weekdays.csv')
barcelona_weekends = pd.read_csv('barcelona_weekends.csv')
berlin_weekdays = pd.read_csv('berlin_weekdays.csv')
berlin_weekends = pd.read_csv('berlin_weekends.csv')
budapest_weekdays = pd.read_csv('budapest_weekdays.csv')
budapest_weekends = pd.read_csv('budapest_weekends.csv')
lisbon_weekdays = pd.read_csv('lisbon_weekdays.csv')
lisbon_weekends = pd.read_csv('lisbon_weekends.csv')
london_weekdays = pd.read_csv('london_weekdays.csv')
london_weekends = pd.read_csv('london_weekends.csv')
paris_weekdays = pd.read_csv('paris_weekdays.csv')
paris_weekends = pd.read_csv('paris_weekends.csv')
rome_weekdays = pd.read_csv('rome_weekdays.csv')
rome_weekends = pd.read_csv('rome_weekends.csv')
vienna_weekdays = pd.read_csv('vienna_weekdays.csv')
vienna_weekends = pd.read_csv('vienna_weekends.csv')
```

DIMENSIONS OF IMPORTED DATASETS

In [4]:
```
print('amsterdam_weekdays shape = ' + str(amsterdam_weekdays.shape)
print('amsterdam_weekends shape = ' + str(amsterdam_weekends.shape)
print('athens_weekdays shape = ' + str(athens_weekdays.shape))
print('athens_weekends shape = ' + str(athens_weekends.shape))
print('barcelona_weekdays shape = ' + str(barcelona_weekdays.shape)
print('barcelona_weekends shape = ' + str(barcelona_weekends.shape)
print('berlin_weekdays shape = ' + str(berlin_weekdays.shape))
print('berlin_weekends shape = ' + str(berlin_weekends.shape))
print('budapest_weekdays shape = ' + str(budapest_weekdays.shape))
print('budapest_weekends shape = ' + str(budapest_weekends.shape))
print('lisbon_weekdays shape = ' + str(lisbon_weekdays.shape))
print('lisbon_weekends shape = ' + str(lisbon_weekends.shape))
print('london_weekdays shape = ' + str(london_weekdays.shape))
print('london_weekends shape = ' + str(london_weekends.shape))
print('paris_weekdays shape = ' + str(paris_weekdays.shape))
print('paris_weekends shape = ' + str(paris_weekends.shape))
print('rome_weekdays shape = ' + str(rome_weekdays.shape))
print('rome_weekends shape = ' + str(rome_weekends.shape))
print('vienna_weekdays shape = ' + str(vienna_weekdays.shape))
print('vienna_weekends shape = ' + str(vienna_weekends.shape))
```

```
amsterdam_weekdays shape = (1103, 20)
amsterdam_weekends shape = (977, 20)
athens_weekdays shape = (2653, 20)
athens_weekends shape = (2627, 20)
barcelona_weekdays shape = (1555, 20)
barcelona_weekends shape = (1278, 20)
berlin_weekdays shape = (1284, 20)
berlin_weekends shape = (1200, 20)
budapest_weekdays shape = (2074, 20)
budapest_weekends shape = (1948, 20)
lisbon_weekdays shape = (2857, 20)
lisbon_weekends shape = (2906, 20)
london_weekdays shape = (4614, 20)
london_weekends shape = (5379, 20)
paris_weekdays shape = (3130, 20)
paris_weekends shape = (3558, 20)
rome_weekdays shape = (4492, 20)
rome_weekends shape = (4535, 20)
vienna_weekdays shape = (1738, 20)
vienna_weekends shape = (1799, 20)
```

FEATURES OF THE IMPORTED DATASETS

In [5]:
```
print(amsterdam_weekdays.columns)
print(amsterdam_weekends.columns)
print(athens_weekdays.columns)
print(athens_weekends.columns)
print(barcelona_weekdays.columns)
print(barcelona_weekends.columns)
print(berlin_weekdays.columns)
print(berlin_weekends.columns)
print(budapest_weekdays.columns)
print(budapest_weekends.columns)
print(lisbon_weekdays.columns)
print(lisbon_weekends.columns)
print(london_weekdays.columns)
print(london_weekends.columns)
print(paris_weekdays.columns)
print(paris_weekends.columns)
print(rome_weekdays.columns)
print(rome_weekends.columns)
print(vienna_weekdays.columns)
print(vienna_weekends.columns)
```

```
Index(['Unnamed: 0', 'realSum', 'room_type', 'room_shared', 'room_
private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedroo
ms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index'
,
       'rest_index_norm', 'lng', 'lat'],
      dtype='object')
Index(['Unnamed: 0', 'realSum', 'room_type', 'room_shared', 'room_
private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedroo
ms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index'
,
       'rest_index_norm', 'lng', 'lat'],
      dtype='object')
Index(['Unnamed: 0', 'realSum', 'room_type', 'room_shared', 'room_
```

- **From observing the shapes of the Individual imported data files, we can see that the number of features in all othe files are same, the number of records however, is different**
- **From observing the List of feature names , we can see that all datasets have the same number, as well as the same features, thus they can be stacked on top of each other in order to convert all these different datasets into a single dataset.**

COMBINE ALL THE DIFFERENT DATASETS INTO A SINGLE ONE

In [6]:
```
def combine(csv_1,col_1,csv_2,col_2,city):        # Combine the 2 da
    csv_1['week time'] = col_1
    csv_2['week time'] = col_2
    csv_1.drop(columns = ['Unnamed: 0'],inplace=True)
    csv_2.drop(columns = ['Unnamed: 0'],inplace=True)
    merged = pd.concat([csv_1, csv_2])
    merged['city'] = city
    return merged
```

- **In the above funtion, we combine the 'weekdays' and 'weekend' datasets of the individual datasets into 1 dataset for a particular city**
- **We also add the name of the City and put it into a new column 'city', since we will be combining all these cities datasets and thus would need to differentiate the data of cities in some way for analysis and insights**
- **We also remove the unnamed:0 feature since it is the index number of the records, thus is not very useful**

In [7]:
```
amsterdam = combine(amsterdam_weekdays,'weekdays',amsterdam_weekend
athens = combine(athens_weekdays,'weekdays',athens_weekends,'weeken
barcelona = combine(barcelona_weekdays,'weekdays',barcelona_weekend
berlin = combine(berlin_weekdays,'weekdays',berlin_weekends,'weeken
budapest = combine(budapest_weekdays,'weekdays',budapest_weekends,'
lisbon = combine(lisbon_weekdays,'weekdays',lisbon_weekends,'weeken
london = combine(london_weekdays,'weekdays',london_weekends,'weeken
paris = combine(paris_weekdays,'weekdays',paris_weekends,'weekends'
rome = combine(rome_weekdays,'weekdays',rome_weekends,'weekends','r
vienna = combine(vienna_weekdays,'weekdays',vienna_weekends,'weeken
```

In [8]:
```
cities_names = ['amsterdam', 'athens', 'barcelona', 'berlin', 'buda
cities = [amsterdam, athens, barcelona, berlin, budapest, lisbon, l
```

In [9]:
```
europe_data = pd.concat(cities, ignore_index=True)
```

- **using pandas'concat function, we vertically stacked data of all cities, to transform them into a single dataset**

CLEAN & ANALYZE THE DATA

In [10]:
```
europe_data.head()
```

Out[10]:

	realSum	room_type	room_shared	room_private	person_capacity	host_is_superhost
0	194.033698	Private room	False	True	2.0	False
1	344.245776	Private room	False	True	4.0	False
2	264.101422	Private room	False	True	2.0	False
3	433.529398	Private room	False	True	4.0	False
4	485.552926	Private room	False	True	2.0	True

5 rows × 21 columns

In [11]:
```
europe_data.tail()
```

Out[11]:

	realSum	room_type	room_shared	room_private	person_capacity	host_is_superh
51702	715.938574	Entire home/apt	False	False	6.0	Fa
51703	304.793960	Entire home/apt	False	False	2.0	Fa
51704	637.168969	Entire home/apt	False	False	2.0	Fa
51705	301.054157	Private room	False	True	2.0	Fa
51706	133.230489	Private room	False	True	4.0	T

5 rows × 21 columns

In [12]:
```
europe_data.sample(5)
```

Out[12]:

	realSum	room_type	room_shared	room_private	person_capacity	host_is_superh
48069	129.520959	Private room	False	True	3.0	Fa
20159	191.838649	Entire home/apt	False	False	4.0	T
11882	182.060391	Shared room	True	False	6.0	Fa
20356	723.264540	Entire home/apt	False	False	4.0	Fa
12043	162.428718	Private room	False	True	2.0	Fa

5 rows × 21 columns

In [13]:
```
europe_data.isna().sum()
```

Out[13]:
```
realSum                       0
room_type                     0
room_shared                   0
room_private                  0
person_capacity               0
host_is_superhost             0
multi                         0
biz                           0
cleanliness_rating            0
guest_satisfaction_overall    0
bedrooms                      0
dist                          0
metro_dist                    0
attr_index                    0
attr_index_norm               0
rest_index                    0
rest_index_norm               0
lng                           0
lat                           0
week time                     0
city                          0
dtype: int64
```

- **There are no null or NaN values in the whole dataset, which is what we want, and saves us a step in data cleaning.**

In [14]:
```
europe_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51707 entries, 0 to 51706
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   realSum                     51707 non-null  float64
 1   room_type                   51707 non-null  object 
 2   room_shared                 51707 non-null  bool   
 3   room_private                51707 non-null  bool   
 4   person_capacity             51707 non-null  float64
 5   host_is_superhost           51707 non-null  bool   
 6   multi                       51707 non-null  int64  
 7   biz                         51707 non-null  int64  
 8   cleanliness_rating          51707 non-null  float64
 9   guest_satisfaction_overall  51707 non-null  float64
 10  bedrooms                    51707 non-null  int64  
 11  dist                        51707 non-null  float64
 12  metro_dist                  51707 non-null  float64
 13  attr_index                  51707 non-null  float64
 14  attr_index_norm             51707 non-null  float64
 15  rest_index                  51707 non-null  float64
 16  rest_index_norm             51707 non-null  float64
 17  lng                         51707 non-null  float64
 18  lat                         51707 non-null  float64
 19  week time                   51707 non-null  object 
 20  city                        51707 non-null  object 
dtypes: bool(3), float64(12), int64(3), object(3)
memory usage: 7.2+ MB
```

- **We can cross check that all the values are non-null in the dataset**
- **We have features having, object (multiple), float and int datatypes. Meaning that we can be having ordinal,categorical and binary/boolean features in our data**

In [15]:
```
europe_data.describe()
```

Out[15]:

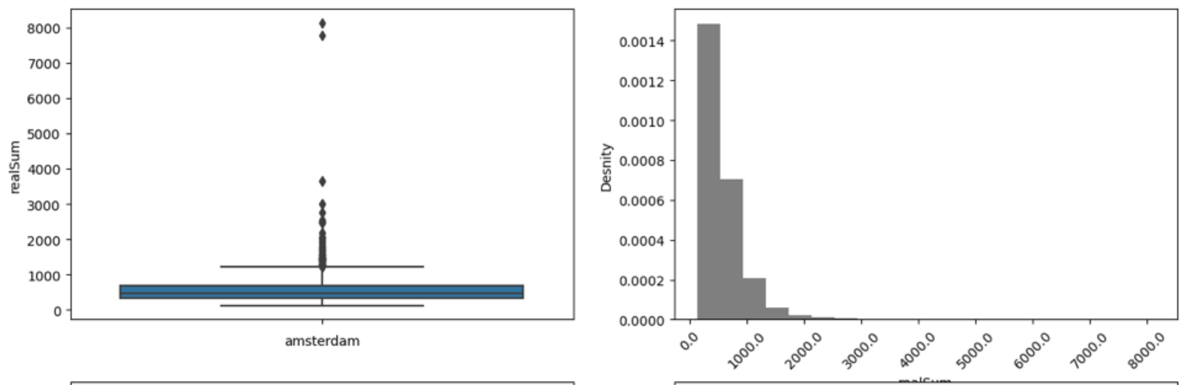
	realSum	person_capacity	multi	biz	cleanliness_rating	gues
count	51707.000000	51707.000000	51707.000000	51707.000000	51707.000000	
mean	279.879591	3.161661	0.291353	0.350204	9.390624	
std	327.948386	1.298545	0.454390	0.477038	0.954868	
min	34.779339	2.000000	0.000000	0.000000	2.000000	
25%	148.752174	2.000000	0.000000	0.000000	9.000000	
50%	211.343089	3.000000	0.000000	0.000000	10.000000	
75%	319.694287	4.000000	1.000000	1.000000	10.000000	
max	18545.450285	6.000000	1.000000	1.000000	10.000000	

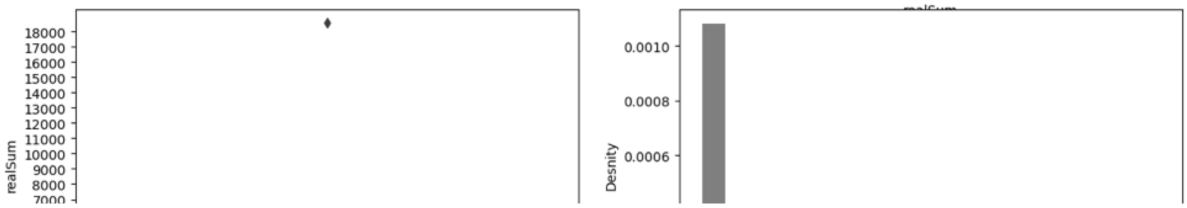
In [16]:
```
europe_data['city'].value_counts()
```

Out[16]:
```
london       9993
rome         9027
paris        6688
lisbon       5763
athens       5280
budapest     4022
vienna       3537
barcelona    2833
berlin       2484
amsterdam    2080
Name: city, dtype: int64
```

In [17]:

```
warnings.filterwarnings('ignore')

def plotter_1(city,row):
    sns.boxplot(y='realSum', data=city, ax=axs[row,0])
    axs[row,0].set_yticks(np.arange(0,max(city['realSum']),1000))
    axs[row,0].set_xlabel(cities_names[row])
    axs[row,0].set_ylabel('realSum')

    axs[row,1].hist(city['realSum'], bins=20, alpha=0.5, color='000
    axs[row,1].set_xticklabels(np.arange(0,max(city['realSum']),100
    axs[row,1].set_xticks(np.arange(0,max(city['realSum']),1000),ro
    axs[row,1].set_xlabel('realSum')
    axs[row,1].set_ylabel('Desnity')

plt.figure
fig, axs = plt.subplots(nrows=10, ncols=2, figsize=(15, 50))
fig2, axs2 = plt.subplots(nrows=2, ncols=1, figsize=(13, 5))

sns.boxplot(y='realSum', data=europe_data, ax=axs2[0])
axs2[0].set_yticks(np.arange(0,max(europe_data['realSum']),1500))
axs2[0].set_xlabel('Price Distribution of Full dataset')
axs2[0].set_ylabel('realSum')

axs2[1].hist(europe_data['realSum'], bins=20, alpha=0.5, color='000
axs2[1].set_xticklabels(np.arange(0,max(europe_data['realSum']),150
axs2[1].set_xticks(np.arange(0,max(europe_data['realSum']),1500),ro
axs2[1].set_xlabel('realSum')
axs2[1].set_ylabel('Desnity')

row = 0
for city in cities:
    plotter_1(city,row)
    row = row + 1
row = 0

plt.subplots_adjust(hspace=0.50)
plt.show()
```





In [18]:
```
print('Inter-Quartile Range of realSum : ' + str(europe_data['realS
```

```
Inter-Quartile Range of realSum : 170.94211280448903
```

- **Looking at the interquartile range and the Description table of the dataset, it is evident that majority of the prices of listings in europe range from $149 to $320 while the IQR is $171.**
- **There is quite a difference between the number of records / observations in each city.**
- **From the above plots we can see that there are a lot of outliers in the data, let's make a copy of the above dataset and remove the outliers from them. We are creating new copy and will not be making changes to the original dataset so that we can analyze and play with the original dataset if we need to later.**
- **We shall oserve the distribution after removing the outliers, let's set the outlier limit for each city after observing the above plots**

REMOVING OUTLIERS AND PLOTTING PRICE DISTRIBUTION

In [19]:
```
cities_2 = [amsterdam[amsterdam['realSum'] < 2000], athens[athens['
```

In [20]:
```
europe_data_2 = pd.concat(cities_2, ignore_index=True)
```

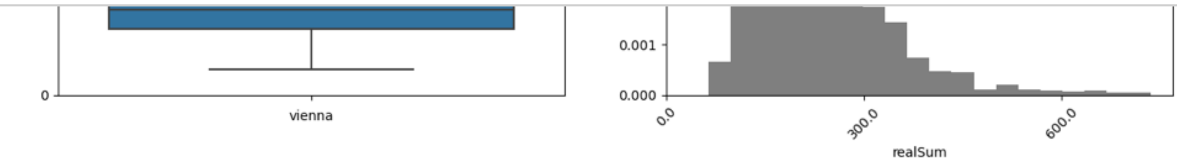
In [21]:
```
europe_data_2.describe()
```

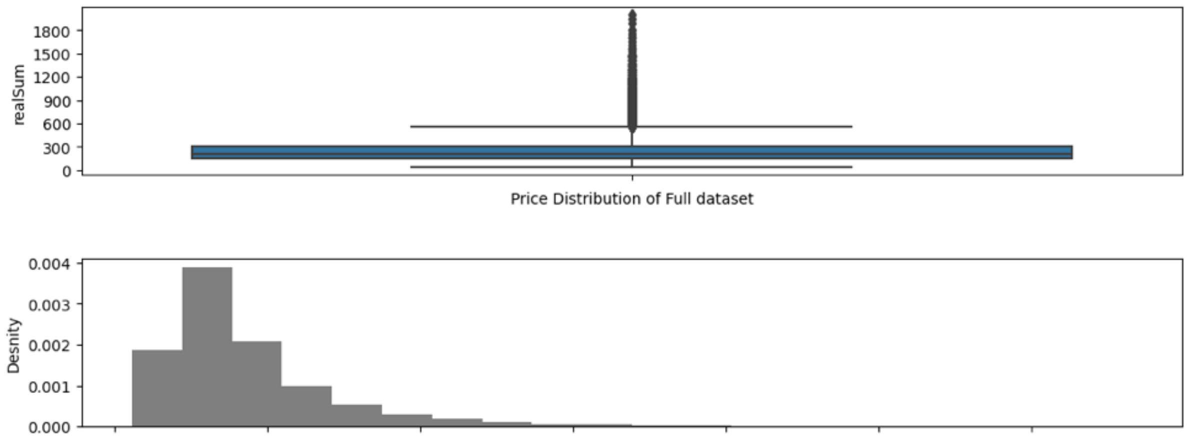
Out[21]:

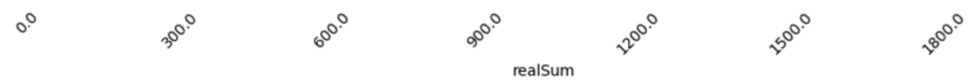
	realSum	person_capacity	multi	biz	cleanliness_rating	gues
count	51176.000000	51176.000000	51176.000000	51176.000000	51176.000000	
mean	263.785977	3.145869	0.291562	0.349637	9.388698	
std	181.924134	1.288028	0.454486	0.476860	0.955359	
min	34.779339	2.000000	0.000000	0.000000	2.000000	
25%	148.405632	2.000000	0.000000	0.000000	9.000000	
50%	208.911031	3.000000	0.000000	0.000000	10.000000	
75%	313.036525	4.000000	1.000000	1.000000	10.000000	
max	1997.515994	6.000000	1.000000	1.000000	10.000000	

In [22]:

```
warnings.filterwarnings('ignore')

def plotter_2(city,row):
    sns.boxplot(y='realSum', data=city, ax=axs[row,0])
    axs[row,0].set_yticks(np.arange(0,max(city['realSum']),300))
    axs[row,0].set_xlabel(cities_names[row])
    axs[row,0].set_ylabel('realSum')

    axs[row,1].hist(city['realSum'], bins=20, alpha=0.5, color='#00
    axs[row,1].set_xticklabels(np.arange(0,max(city['realSum']),300
    axs[row,1].set_xticks(np.arange(0,max(city['realSum']),300),rot
    axs[row,1].set_xlabel('realSum')
    axs[row,1].set_ylabel('Desnity')

plt.figure
fig, axs = plt.subplots(nrows=10, ncols=2, figsize=(15, 50))
fig2, axs2 = plt.subplots(nrows=2, ncols=1, figsize=(13, 5))
row = 0
for city in cities_2:
    plotter_2(city,row)
    row = row + 1
row = 0

sns.boxplot(y='realSum', data=europe_data_2, ax=axs2[0])
axs2[0].set_yticks(np.arange(0,max(europe_data_2['realSum']),300))
axs2[0].set_xlabel('Price Distribution of Full dataset')
axs2[0].set_ylabel('realSum')

axs2[1].hist(europe_data_2['realSum'], bins=20, alpha=0.5, color='0
axs2[1].set_xticklabels(np.arange(0,max(europe_data_2['realSum']),3
axs2[1].set_xticks(np.arange(0,max(europe_data_2['realSum']),300),r
axs2[1].set_xlabel('realSum')
axs2[1].set_ylabel('Desnity')

plt.subplots_adjust(hspace=0.50)
plt.show()
```



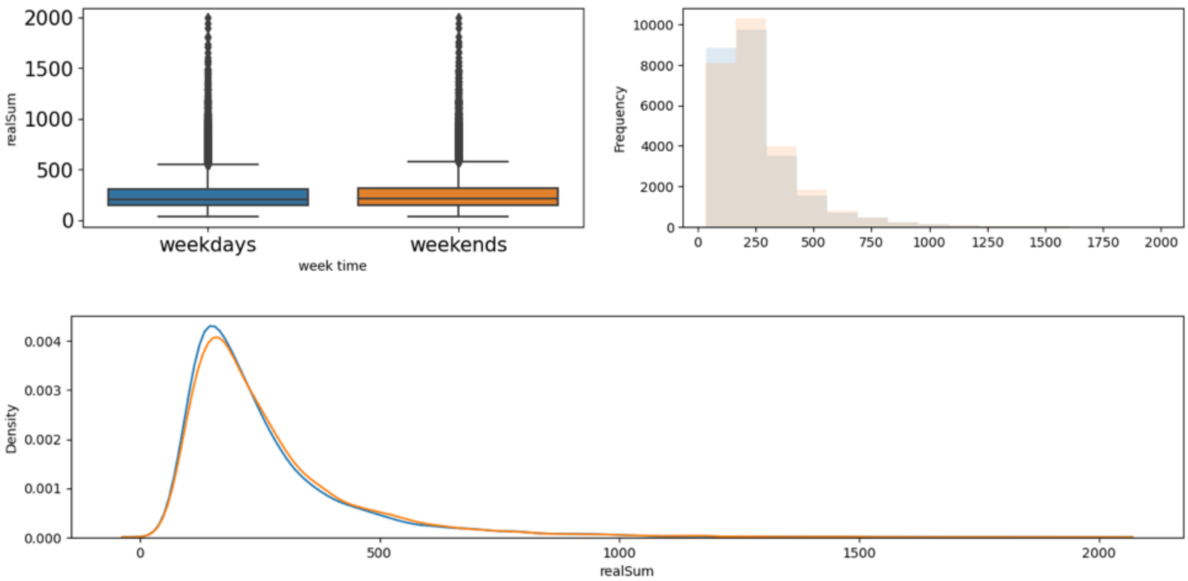




- **Removed outliers from the original dataset, not simply by looking at IQR, but by trial and test, and lowering the benchmark for the outlier to the point that there were considerable amount of observations, even in the outlier portion.**

AFFECT OF TIME OF THE WEEK ON PRICES

In [23]:
```
plt.figure
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15, 3))
fig2, axs2 = plt.subplots(nrows=1, ncols=1, figsize=(15, 3))
sns.boxplot(y='realSum', data=europe_data_2,x='week time',ax = axs[
axs[0].tick_params(axis='y', labelsize=15)
axs[0].tick_params(axis='x', labelsize=15)

europe_data_2.groupby('week time')['realSum'].plot(kind='hist', alp

sns.kdeplot(data=europe_data_2[europe_data_2['week time'] == 'weekd
sns.kdeplot(data=europe_data_2[europe_data_2['week time'] == 'weeke
plt.subplots_adjust(hspace=0.65)
plt.show()
```



- **Surprisingly, there is almost no difference between the ranges and distribution of realSum on weekends and weekdays**

FREQUENCY DISTRIBUTION OF NUMERIC FEATURES

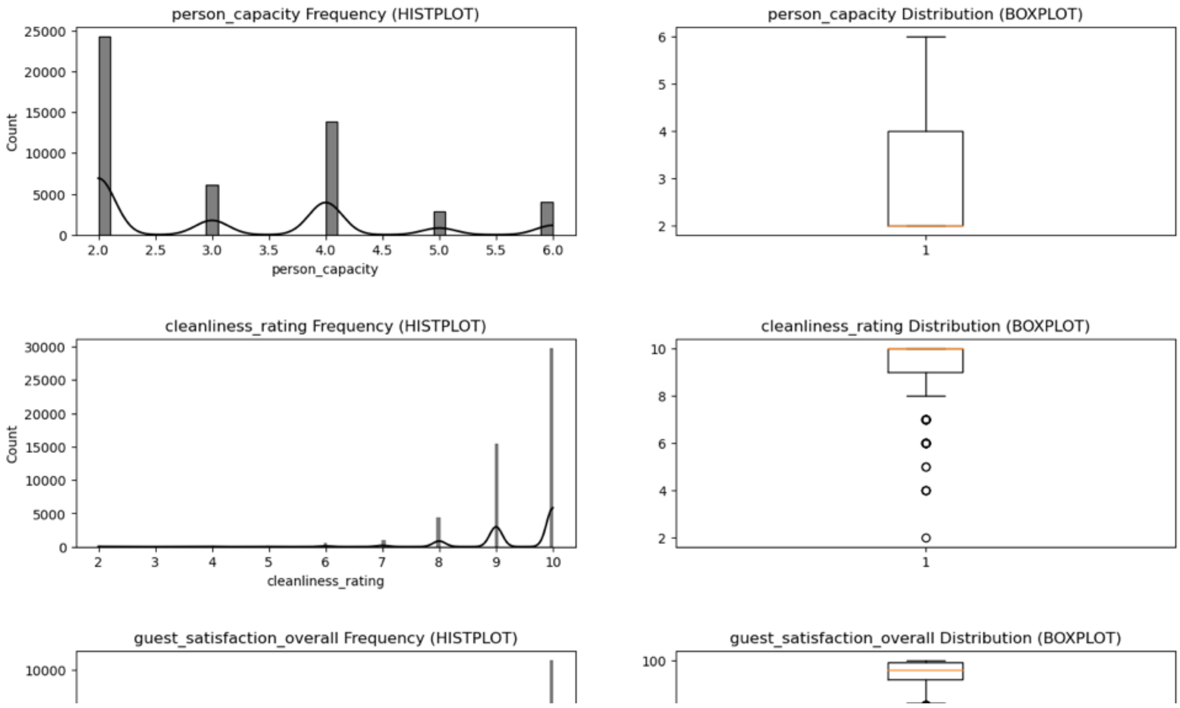
In [24]:
```
list(europe_data_2.select_dtypes(include=['int64','float64']))
```

Out[24]:
```
['realSum',
 'person_capacity',
 'multi',
 'biz',
 'cleanliness_rating',
 'guest_satisfaction_overall',
 'bedrooms',
 'dist',
 'metro_dist',
 'attr_index',
 'attr_index_norm',
 'rest_index',
 'rest_index_norm',
 'lng',
 'lat']
```

- **We list all the features with numeric data types in order to display their boxplot and frequency plot.**
- **Pandas also considers boolean features in which 1,0 represents True,False as numeric, thus we will ignore those for the following plots and analyze them later**

In [25]:
```
europe_data_2_numerical_features = list(europe_data_2.select_dtypes
```

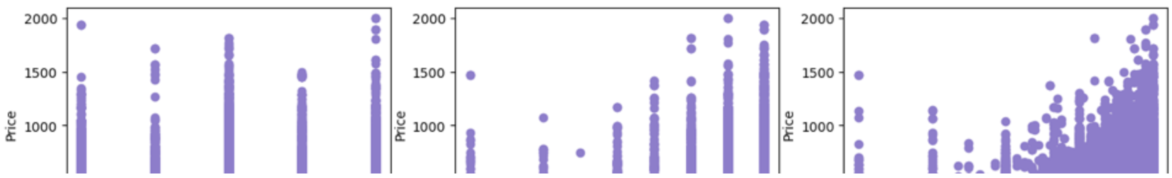
In [26]:
```
def plotter_3(feature,color,row):
    sns.histplot(data=europe_data_2[feature],ax=axes[row,0],kde=Tru
    axes[row,0].set_title(str(feature)+" Frequency (HISTPLOT)")
    axes[row,1].boxplot(amsterdam[feature])
    axes[row,1].set_title(str(feature)+" Distribution (BOXPLOT)")


plt.figure
fig, axes = plt.subplots(nrows=12, ncols=2, figsize=(15, 50))
for i in range(12):
    plotter_3( europe_data_2_numerical_features[i] , '#000000' , i)


plt.subplots_adjust(hspace=0.50)
plt.show()
```

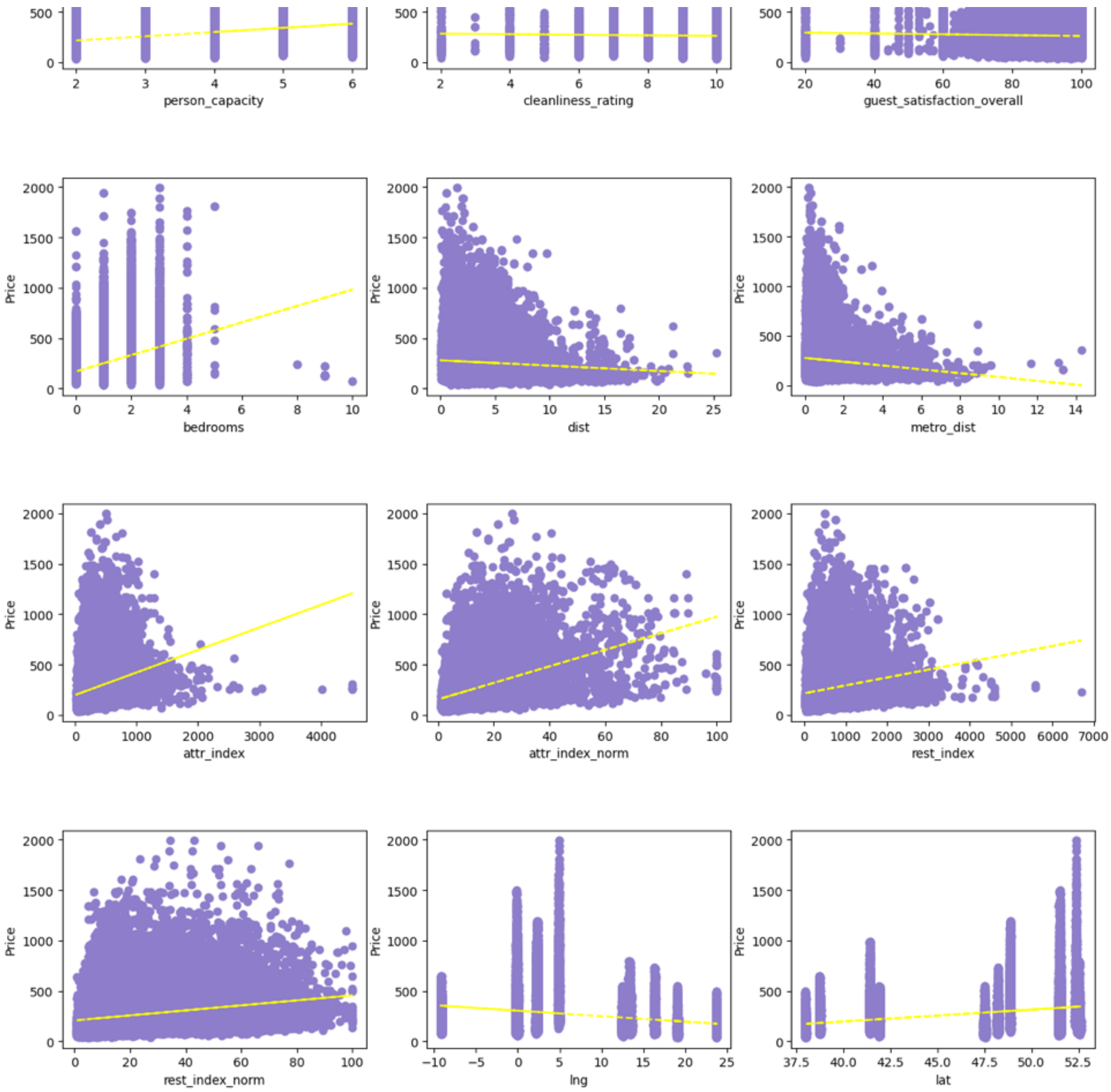


- **In european listings of Air Bnb, the people capacity in Descending order are 2,4,3,6 and 5 bedrooms .**
- **The cleanliness rating of listings can be thought of having a left skewed shape. From the boxplot it can be seen that majority of the ratings are 8-10. Thus european listings have a very high cleanliness rating on average**
- **The guest satisfaction scores follow more or less of the same pattern as cleanliness rating. wonder if there is a relation between cleanliness rating and guest satisfaction? Hold on, we'll be finding this later.**
- **In european listings of Air Bnb, the Number of Bedrooms listings have in Descending order are 1,studio,2,3 and 4 bedrooms .**
- **Most of the listings are within 7 km's of the city centre while there are relatively few listings that are 7 - 10 km's from the city centre**
- **Most of the listings are within 3 km's of the closest metro while there are relatively few listings that are 3 - 5 km's from the closest metro**

SCATTERPLOT OF NUMERIC FEATURES AND TREND LINE W.R.T REALSUM

In [27]:
```
warnings.filterwarnings('ignore')

def plotter_4(feature,color,x,y):

    axes[x,y].scatter(y=europe_data_2["realSum"], x=europe_data_2[f
    trend_line = np.poly1d(np.polyfit(europe_data_2[feature], europ
    axes[x,y].plot(europe_data_2[feature], trend_line(europe_data_2
    axes[x,y].set_ylabel("Price")
    axes[x,y].set_xlabel(feature)

fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 17.5))
x = 0
y = 0
for i in range(12):
    plotter_4(europe_data_2_numerical_features[i] , '#8d7dca',x,y)
    y  = y + 1
    if y == 3:
        x = x + 1
        y = 0

plt.subplots_adjust(hspace=0.5)
```



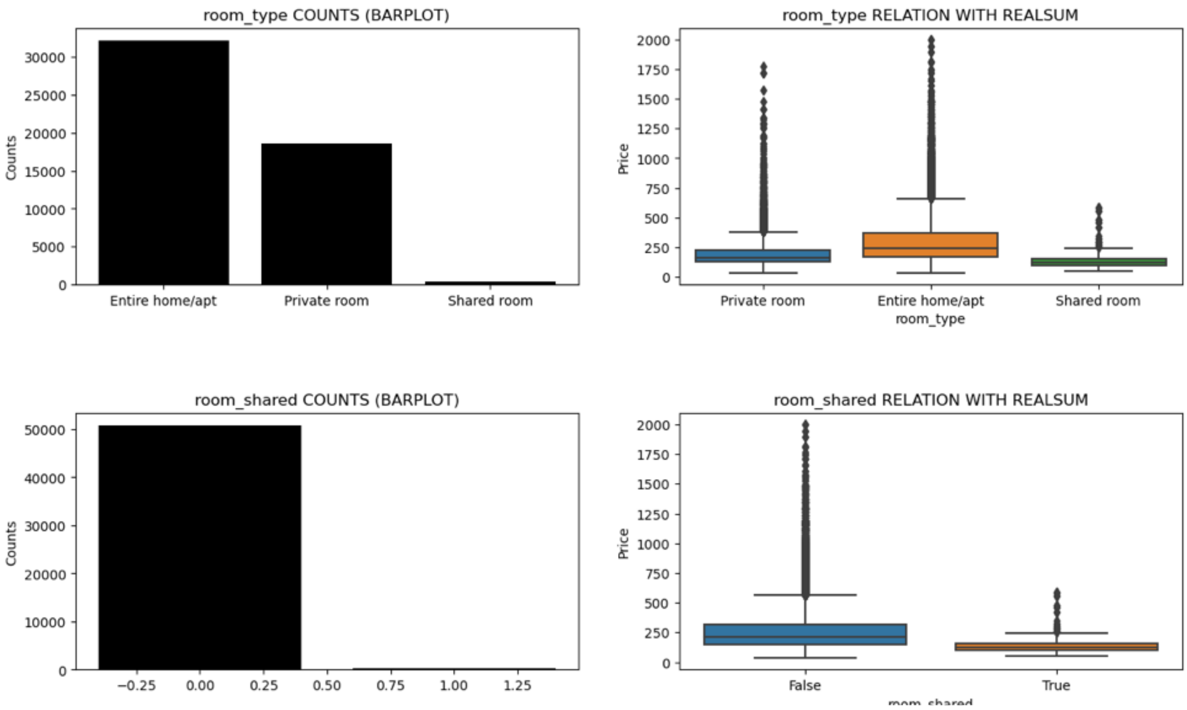


- No of bedrooms, person capacity,attraction index & restaurant index are the features that show good positive trend as their values increase
- other features such as cleanliness rating and guest satisfaction have near to neutral trend lines throughout their value range.
- Distance from city centre and nearest metro station have slighly negative trend lines as their values increase
- We will be checking these trends and corelations with the realSum later when we calculation the corelation between features

CATEGORICAL & BINARY FEATURES, THEIR COUNTS AND RELATION WITH OUTPUT THROUGH BOXPLOT

In [28]:
```
europe_data_2_categorical_features = ['room_type','room_shared','ro
```

In [29]:
```
def plotter_5(feature,color,row):
    axes[row,0].bar(x = list(europe_data_2[feature].value_counts().
    axes[row,0].set_ylabel("Counts")
    axes[row,0].set_title(str(feature)+" COUNTS (BARPLOT)")

    sns.boxplot(data=europe_data_2,x = feature,y = 'realSum',ax=axe
    axes[row,1].set_ylabel("Price")
    axes[row,1].set_title(str(feature)+" RELATION WITH REALSUM")


plt.figure
fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(15, 35))

for i in range(7):
    plotter_5(europe_data_2_categorical_features[i] , '000000' , i)

plt.subplots_adjust(hspace=0.50)
plt.show()
```



DATA PRE-PROCESSING

In [30]:
```
europe_data_2.columns
```

Out[30]:
```
Index(['realSum', 'room_type', 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedroo
ms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index'
,
       'rest_index_norm', 'lng', 'lat', 'week time', 'city'],
      dtype='object')
```

In [31]:
```
europe_data_2.head()
```

Out[31]:

	realSum	room_type	room_shared	room_private	person_capacity	host_is_superhost
0	194.033698	Private room	False	True	2.0	False
1	344.245776	Private room	False	True	4.0	False
2	264.101422	Private room	False	True	2.0	False
3	433.529398	Private room	False	True	4.0	False
4	485.552926	Private room	False	True	2.0	True

5 rows × 21 columns

REPLACE TRUE / FALSE BOOLEAN WITH 1 / 0 RESPECTIVELY

In [32]:
```
europe_data_2.replace({False: 0, True: 1},inplace=True)
europe_data_2.head()
```

Out[32]:

	realSum	room_type	room_shared	room_private	person_capacity	host_is_superhost
0	194.033698	Private room	0	1	2.0	0
1	344.245776	Private room	0	1	4.0	0
2	264.101422	Private room	0	1	2.0	0
3	433.529398	Private room	0	1	4.0	0
4	485.552926	Private room	0	1	2.0	1

5 rows × 21 columns

- **We can see that the True/ False values in room_shared, room_private and host_is_superhost is converted to 1/0**

REPLACE CATEGORICAL VLUES TO DUMMY VARIABLES

In [33]:
```
print(europe_data_2['room_type'].value_counts())
print(europe_data_2['week time'].value_counts())
print(europe_data_2['city'].value_counts())
```

```
Entire home/apt    32182
Private room       18628
Shared room          366
Name: room_type, dtype: int64
weekends    25926
weekdays    25250
Name: week time, dtype: int64
london       9881
rome         8929
paris        6590
lisbon       5727
athens       5231
budapest     3979
vienna       3524
barcelona    2802
berlin       2455
amsterdam    2058
Name: city, dtype: int64
```

In [34]:
```
europe_data_2_categorical_dummies = pd.get_dummies(europe_data_2[['
europe_data_3 = pd.concat([europe_data_2_categorical_dummies, europ
```

In [35]:
```
europe_data_3
```

Out[35]:

	room_type_Private room	room_type_Shared room	week time_weekends	city_athens	city_barcelona
0	1	0	0	0	0
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0
...	...	...	...	...	...
51171	0	0	1	0	0
51172	0	0	1	0	0
51173	0	0	1	0	0
51174	1	0	1	0	0
51175	1	0	1	0	0

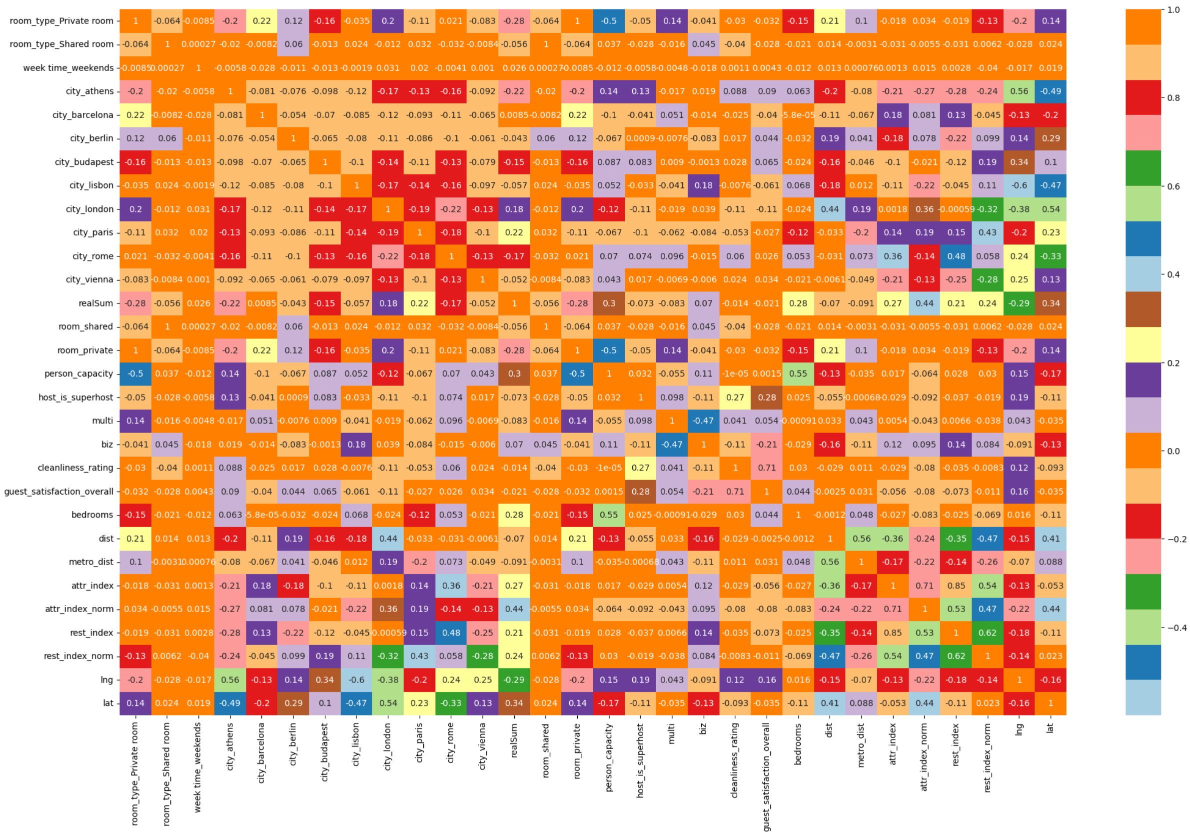
51176 rows × 30 columns

- **Using pandas'get dummies function, we will find dummy values / columns of categorical features room type, week time and city.**
- **There are slight variances and difference in median realSum of different cities, thus we have also converted the cities feature to dummies variable, thus these can be a useful feature in creating the model later.**
- **We have saved this dataframe containing the dummy variables in a variable called europe_data_3**

CHECKING FOR AND REMOVING REDUNDANT VARIABLES

In [36]:
```
plt.figure
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(25, 15))
sns.heatmap(europe_data_3.corr(),cmap=sns.color_palette("Paired",20
```

Out[36]: <AxesSubplot:>

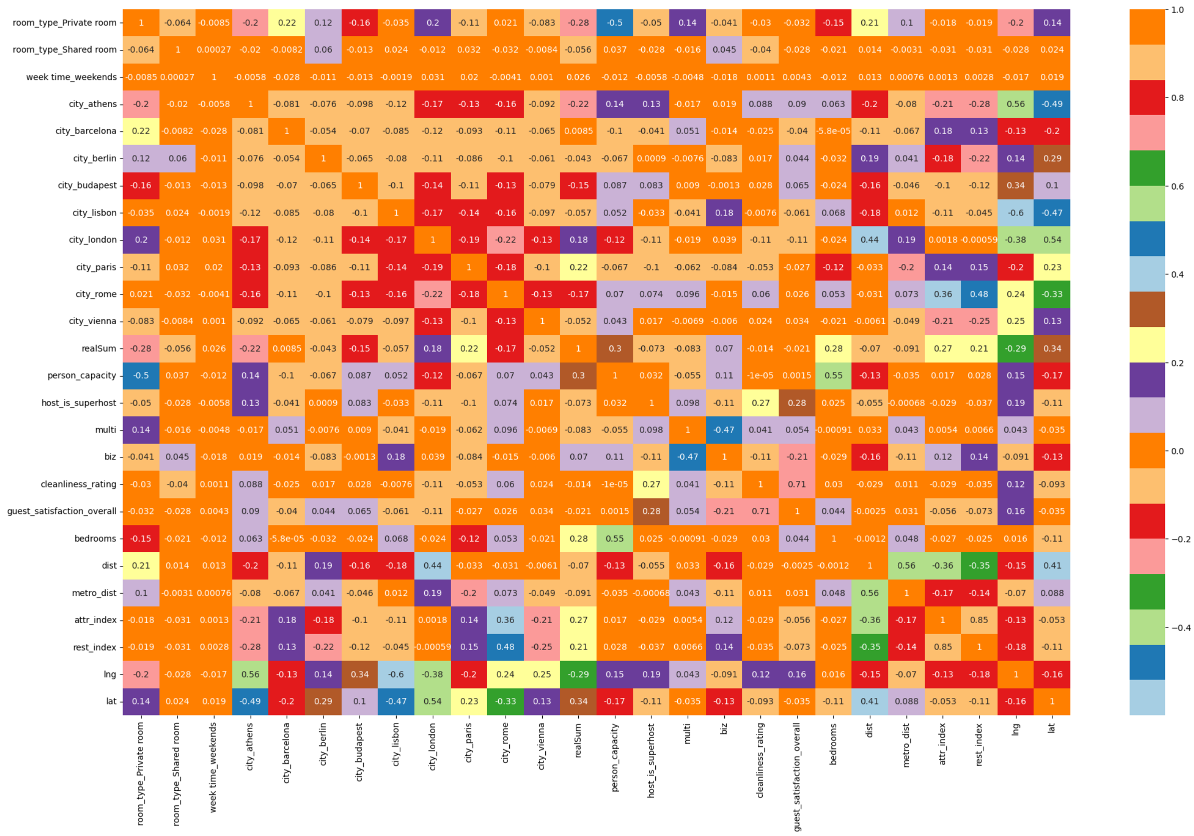


- **There are features tha have perfect corelation, thus these are the redundant features and it is safe to remove one of them. THus we will remove room_shared and room_private.**
- **We will also remove rest_index_norm and attr_index_norm , since for the moment we have non_normalized forms of these features anf we will be scaling them in the later part of the project where we will be creating the model, thus for the time being it is safe to remove these for analyzing and study purpose**

In [37]:
```
europe_data_3.drop(columns = ['rest_index_norm','attr_index_norm','
```

In [38]:
```
plt.figure
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(25, 15))
sns.heatmap(europe_data_3.corr(),cmap=sns.color_palette("Paired",20
```

Out[38]: <AxesSubplot:>



COMBINING TWO VARIABLES INTO ONE

In [39]:
```
def combine_lat_long(lng, lat):
    latitude = np.radians(lat)
    longitude = np.radians(lng)

    amsterdam_latitude = np.radians(0)
    amsterdam_longitude = np.radians(0)

    # apply Haversine formula to compute distance
    latitude_distance = amsterdam_latitude - latitude
    longitude_distance = amsterdam_longitude - longitude
    a = np.sin(latitude_distance/2)**2 + np.cos(latitude) * np.cos(
    c = 2 * np.arcsin(np.sqrt(a))
    distance = 6371 * c

    return distance
```

- **The function above combines the features latitude and longitude into a single feature, which is calculated using the haversine distance formula between the latitude and longitude of the listing and co-ordinates of null island (The point on earth where the latitude and longitude is 0)**
- **The next thing to do would be to remove the features latitude and longitude (lat and lng), which would have already been converted to a single metric, haversine distance**

In [40]:
```
europe_data_3['Haversine Distance'] = combine_lat_long(europe_data_
```

In [41]:
```
europe_data_3.head()
```

Out[41]:

	room_type_Private room	room_type_Shared room	week time_weekends	city_athens	city_barcelona	city_t
0	1	0	0	0	0	
1	1	0	0	0	0	
2	1	0	0	0	0	
3	1	0	0	0	0	
4	1	0	0	0	0	

5 rows × 27 columns

In [42]:
```
europe_data_3.drop(columns=['lng','lat'],inplace=True)
```

In [43]:
```
europe_data_3.head()
```

Out[43]:

	room_type_Private room	room_type_Shared room	week time_weekends	city_athens	city_barcelona	city_t
0	1	0	0	0	0	
1	1	0	0	0	0	
2	1	0	0	0	0	
3	1	0	0	0	0	
4	1	0	0	0	0	

5 rows × 25 columns

SCALING THE FEATURES

In [44]:
```
Standard_Scaler = StandardScaler()
```

In [45]:
```
europe_data_3.shape
```

Out[45]:
```
(51176, 25)
```

In [46]:
```
europe_data_3.columns
```

Out[46]:
```
Index(['room_type_Private room', 'room_type_Shared room', 'week ti
me_weekends',
       'city_athens', 'city_barcelona', 'city_berlin', 'city_budap
est',
       'city_lisbon', 'city_london', 'city_paris', 'city_rome', 'c
ity_vienna',
       'realSum', 'person_capacity', 'host_is_superhost', 'multi',
'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedroo
ms', 'dist',
       'metro_dist', 'attr_index', 'rest_index', 'Haversine Distan
ce'],
      dtype='object')
```

In [47]:
```
features_to_scale = ['person_capacity','cleanliness_rating','guest_
features_not_to_scale = ['room_type_Private room', 'room_type_Share
                         'city_barcelona', 'city_berlin', 'city_bud
                         'city_rome', 'city_vienna','realSum','host_
```

In [48]:
```
scaled_features = pd.DataFrame(Standard_Scaler.fit_transform(europe_
scaled_features.head()
```

Out[48]:

	person_capacity	cleanliness_rating	guest_satisfaction_overall	bedrooms	dist	met
0	-0.889640	0.639872	0.044035	-0.241981	0.761073	2.
1	0.663137	-1.453601	-0.850222	-0.241981	-1.130383	-0.
2	-0.889640	-0.406864	-0.626657	-0.241981	1.063629	3.
3	0.663137	-0.406864	-0.291311	1.378486	-1.173566	-0.
4	-0.889640	0.639872	0.602945	-0.241981	-1.106878	-0.

In [49]:
```
europe_data_final = pd.concat([scaled_features.reset_index(drop=Tru
europe_data_final.head()
```

Out[49]:

	person_capacity	cleanliness_rating	guest_satisfaction_overall	bedrooms	dist	met
0	-0.889640	0.639872	0.044035	-0.241981	0.761073	2.
1	0.663137	-1.453601	-0.850222	-0.241981	-1.130383	-0.
2	-0.889640	-0.406864	-0.626657	-0.241981	1.063629	3.
3	0.663137	-0.406864	-0.291311	1.378486	-1.173566	-0.
4	-0.889640	0.639872	0.602945	-0.241981	-1.106878	-0.

5 rows × 25 columns

In [50]:
```
print('SHape of europe_data_3 : ' + str(europe_data_3.shape))
print('SHape of europe_data_final : ' + str(europe_data_final.shape
```

```
SHape of europe_data_3 : (51176, 25)
SHape of europe_data_final : (51176, 25)
```

- **There is a blueprint of creating models that we will follow**
  - **Scale features (Already done)**
  - **Select Regression model**
  - **Select best features using backward sequential feature selection (If less time is required to perform)**
  - **Fit the model and print training evaluation metrics**
  - **Predict using the model and print prediction evaluation metrics**
- **Print training and prediction evaluation metrics of both best features and all features**

REGRESSION MODELS

CREATING INPUTS AND OUTPUTS

In [51]:
```
X_train , X_test , Y_train , Y_test = train_test_split(europe_data_
                                                        europe_data_
                                                        random_state
                                                        test_size=0.
                                                        stratify=eur
'city_barcelona', 'city_berlin', 'city_budapest','city_lisbon', 'ci
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(43499, 24)
(43499,)
(7677, 24)
(7677,)
```

SEQUENTIAL FEATURE SELECTION FUNCTION

In [52]:
```
def sequential_feature_selection(model,X_train,Y_train,X_test):
    sfs = SequentialFeatureSelector(model,  direction='backward', s
    sfs.fit(X_train, Y_train)
    X_train_selected = sfs.transform(X_train)
    X_test_selected = sfs.transform(X_test)
    return X_train_selected, X_test_selected
```

LINEAR REGRESSION

In [53]:
```
LR = LinearRegression()
LR_2 = LinearRegression()
```

In [54]:
```
X_train_selected, X_test_selected = sequential_feature_selection(Li
```

In [55]:
```
LR.fit(X_train,Y_train)
LR_2.fit(X_train_selected, Y_train)
```

Out[55]:
```
LinearRegression()
```

In [56]:
```
LR_fit_evaulation = {'Linear Regression Fitting Evaluation (All Fea
    {'No. of Features':LR.n_features_in_,
     'R_squared score (Train)':LR.score(X_train,Y_train),
     'R_squared score (Test)':LR.score(X_test,Y_test)},
                     'Linear Regression Fitting Evaluation (Best Fe
    {'No. of Features':LR.n_features_in_,
     'R_squared score (Train)':LR_2.score(X_train_selected,Y_train)
     'R_squared score (Test)':LR_2.score(X_test_selected,Y_test)},
                                                  }
LR_fit_evaulation = pd.DataFrame(LR_fit_evaulation)
LR_fit_evaulation
```

Out[56]:

	Linear Regression Fitting Evaluation (All Features)	Linear Regression Fitting Evaluation (Best Features)
No. of Features	24.000000	24.000000
R_squared score (Train)	0.578271	0.552031
R_squared score (Test)	0.574260	0.546680

In [57]:
```
LR_TrainSet_Prediction = LR.predict(X_train)
LR_TestSet_Prediction = LR.predict(X_test)
LR_TrainSet_Prediction_2 = LR_2.predict(X_train_selected)
LR_TestSet_Prediction_2 = LR_2.predict(X_test_selected)
```

In [58]:
```
LR_predict_evaluation = {'Linear Regression Predictions Evaluation
    {'Train MSE': mean_squared_error(Y_train,LR_TrainSet_Prediction
     'Test MSE' : mean_squared_error(Y_test,LR_TestSet_Prediction),
     'Train RMSE': mean_squared_error(Y_train,LR_TrainSet_Predictio
     'Test RMSE' : mean_squared_error(Y_test,LR_TestSet_Prediction,
                         'Linear Regression Predictions Evaluation
    {'Train MSE': mean_squared_error(Y_train,LR_TrainSet_Prediction
     'Test MSE' : mean_squared_error(Y_test,LR_TestSet_Prediction_2
     'Train RMSE': mean_squared_error(Y_train,LR_TrainSet_Predictio
     'Test RMSE' : mean_squared_error(Y_test,LR_TestSet_Prediction_
    }
LR_predict_evaluation = pd.DataFrame(LR_predict_evaluation)
LR_predict_evaluation
```

Out[58]:

	Linear Regression Predictions Evaluation (All Features)	Linear Regression Predictions Evaluation (Best Features)
Train MSE	14042.389487	14916.126366
Test MSE	13601.088712	14482.212767
Train RMSE	118.500589	122.131594
Test RMSE	116.623706	120.342066

RIDGE REGRESSION

In [59]:
```
R = Ridge()
R_2 = Ridge()
```

In [60]:
```
X_train_selected, X_test_selected = sequential_feature_selection(Ri
```

In [61]:
```
R.fit(X_train,Y_train)
R_2.fit(X_train_selected, Y_train)
```

Out[61]:
```
Ridge()
```

In [62]:
```
R_fit_evaluation = {'Ridge Regression Fitting Evaluation (All Featu
    {'No. of Features':R.n_features_in_,
     'R_squared score (Train)':R.score(X_train,Y_train),
     'R_squared score (Test)':R.score(X_test,Y_test)},
                    'Ridge Regression Fitting Evaluation (Best Fea
    {'No. of Features':R_2.n_features_in_,
     'R_squared score (Train)':R_2.score(X_train_selected,Y_train),
     'R_squared score (Test)':R_2.score(X_test_selected,Y_test)},
                                                 }
R_fit_evaluation = pd.DataFrame(R_fit_evaluation)
R_fit_evaluation
```

Out[62]:

	Ridge Regression Fitting Evaluation (All Features)	Ridge Regression Fitting Evaluation (Best Features)
No. of Features	24.000000	12.000000
R_squared score (Train)	0.578270	0.552030
R_squared score (Test)	0.574318	0.546739

In [63]:
```
R_TrainSet_Prediction = R.predict(X_train)
R_TestSet_Prediction = R.predict(X_test)
R_TrainSet_Prediction_2 = R_2.predict(X_train_selected)
R_TestSet_Prediction_2 = R_2.predict(X_test_selected)
```

In [64]:
```
R_predict_evaluation = {'Ridge Regression Predictions Evaluation (A
    {'Train MSE': mean_squared_error(Y_train,R_TrainSet_Prediction)
     'Test MSE' : mean_squared_error(Y_test,R_TestSet_Prediction),
     'Train RMSE': mean_squared_error(Y_train,R_TrainSet_Prediction
     'Test RMSE' : mean_squared_error(Y_test,R_TestSet_Prediction,s
                         'Ridge Regression Predictions Evaluation (
    {'Train MSE': mean_squared_error(Y_train,R_TrainSet_Prediction_
     'Test MSE' : mean_squared_error(Y_test,R_TestSet_Prediction_2)
     'Train RMSE': mean_squared_error(Y_train,R_TrainSet_Prediction
     'Test RMSE' : mean_squared_error(Y_test,R_TestSet_Prediction_2
    }
R_predict_evaluation = pd.DataFrame(R_predict_evaluation)
R_predict_evaluation
```

Out[64]:

	Ridge Regression Predictions Evaluation (All Features)	Ridge Regression Predictions Evaluation (Best Features)
Train MSE	14042.424456	14916.150775
Test MSE	13599.242913	14480.328158
Train RMSE	118.500736	122.131694
Test RMSE	116.615792	120.334235

LASSO REGRESSION

In [65]:
```
L = Lasso()
L_2 = Lasso()
```

In [66]:
```
X_train_selected, X_test_selected = sequential_feature_selection(La
```

In [67]:
```
L.fit(X_train,Y_train)
L_2.fit(X_train_selected, Y_train)
```

Out[67]:
```
Lasso()
```

In [68]:
```
L_fit_evaulation = {'Lasso Regression Fitting Evaluation (All Featu
    {'No. of Features':L.n_features_in_,
     'R_squared score (Train)':L.score(X_train,Y_train),
     'R_squared score (Test)':L.score(X_test,Y_test)},
                     'Lasso Regression Fitting Evaluation (Best Fea
    {'No. of Features':L_2.n_features_in_,
     'R_squared score (Train)':L_2.score(X_train_selected,Y_train),
     'R_squared score (Test)':L_2.score(X_test_selected,Y_test)},
                                                  }
L_fit_evaulation = pd.DataFrame(L_fit_evaulation)
L_fit_evaulation
```

Out[68]:

	Lasso Regression Fitting Evaluation (All Features)	Lasso Regression Fitting Evaluation (Best Features)
No. of Features	24.00000	12.000000
R_squared score (Train)	0.54641	0.536776
R_squared score (Test)	0.55148	0.539752

In [69]:
```
L_TrainSet_Prediction = L.predict(X_train)
L_TestSet_Prediction = L.predict(X_test)
L_TrainSet_Prediction_2 = L_2.predict(X_train_selected)
L_TestSet_Prediction_2 = L_2.predict(X_test_selected)
```

In [70]:
```
L_predict_evaluation = {'Lasso Regression Predictions Evaluation (A
    {'Train MSE': mean_squared_error(Y_train,L_TrainSet_Prediction)
     'Test MSE' : mean_squared_error(Y_test,L_TestSet_Prediction),
     'Train RMSE': mean_squared_error(Y_train,L_TrainSet_Prediction
     'Test RMSE' : mean_squared_error(Y_test,L_TestSet_Prediction,s
                         'Lasso Regression Predictions Evaluation (
    {'Train MSE': mean_squared_error(Y_train,L_TrainSet_Prediction_
     'Test MSE' : mean_squared_error(Y_test,L_TestSet_Prediction_2)
     'Train RMSE': mean_squared_error(Y_train,L_TrainSet_Prediction
     'Test RMSE' : mean_squared_error(Y_test,L_TestSet_Prediction_2
    }
L_predict_evaluation = pd.DataFrame(L_predict_evaluation)
L_predict_evaluation
```

Out[70]:

	Lasso Regression Predictions Evaluation (All Features)	Lasso Regression Predictions Evaluation (Best Features)
Train MSE	15103.272609	15424.065072
Test MSE	14328.871467	14703.545037
Train RMSE	122.895373	124.193660
Test RMSE	119.703264	121.258175

DECISION TREE REGRESSION :

In [71]:
```
DTR = DecisionTreeRegressor()
```

In [72]:
```
DTR.fit(X_train,Y_train)
```

Out[72]:
```
DecisionTreeRegressor()
```

In [73]:
```
DTR_fit_evaluation = {'Decision Tree Regression Fitting Evaluation
    {'No. of Features':DTR.n_features_in_,
     'R_squared score (Train)':DTR.score(X_train,Y_train),
     'R_squared score (Test)':DTR.score(X_test,Y_test)}}
DTR_fit_evaluation = pd.DataFrame(DTR_fit_evaluation)
DTR_fit_evaluation
```

Out[73]:

	Decision Tree Regression Fitting Evaluation (All Features)
No. of Features	24.000000
R_squared score (Test)	0.779395
R_squared score (Train)	1.000000

In [74]:
```
DTR_TrainSet_Prediction = DTR.predict(X_train)
DTR_TestSet_Prediction = DTR.predict(X_test)
# DTR_TrainSet_Prediction_2 = DTR_2.predict(X_train_selected)
# DTR_TestSet_Prediction_2 = DTR_2.predict(X_test_selected)
```

In [75]:
```
DTR_predict_evaluation = {'Decision Tree Regression Predictions Eva
    {'Train MSE': mean_squared_error(Y_train,DTR_TrainSet_Predictio
     'Test MSE' : mean_squared_error(Y_test,DTR_TestSet_Prediction)
     'Train RMSE': mean_squared_error(Y_train,DTR_TrainSet_Predicti
     'Test RMSE' : mean_squared_error(Y_test,DTR_TestSet_Prediction
#                          'Decision Tree Regression Predictions Eva
#    {'Train MSE': mean_squared_error(Y_train,DTR_TrainSet_Predicti
#     'Test MSE' : mean_squared_error(Y_test,DTR_TestSet_Prediction
#     'Train RMSE': mean_squared_error(Y_train,DTR_TrainSet_Predict
#     'Test RMSE' : mean_squared_error(Y_test,DTR_TestSet_Predictio
    }
DTR_predict_evaluation = pd.DataFrame(DTR_predict_evaluation)
DTR_predict_evaluation
```

Out[75]:

	Decision Tree Regression Predictions Evaluation (All Features)
Test MSE	7.047648e+03
Test RMSE	8.395027e+01
Train MSE	7.451371e-30
Train RMSE	2.729720e-15

RANDOM FOREST REGRESSION :

In [76]:
```
RFR = RandomForestRegressor(n_estimators = 50,max_depth=60)
# RFR_2 = RandomForestRegressor(n_estimators = 50)
```

In [77]:
```
# X_train_selected, X_test_selected = sequential_feature_selection(
```

In [78]:
```
RFR.fit(X_train,Y_train)
# RFR_2.fit(X_train_selected, Y_train)
```

Out[78]:
```
RandomForestRegressor(max_depth=60, n_estimators=50)
```

In [79]:
```
RFR_fit_evaluation = {'Random Forest Regression Fitting Evaluation
    {'No. of Features':RFR.n_features_in_,
     'R_squared score (Train)':RFR.score(X_train,Y_train),
     'R_squared score (Test)':RFR.score(X_test,Y_test)},
#                       'Random Forest Regression Fitting Evaluation
#    {'No. of Features':RFR_2.n_features_in_,
#     'R_squared score (Train)':RFR_2.score(X_train_selected,Y_trai
#     'R_squared score (Test)':RFR_2.score(X_test_selected,Y_test)}
                                                  }
RFR_fit_evaluation = pd.DataFrame(RFR_fit_evaluation)
RFR_fit_evaluation
```

Out[79]:

	Random Forest Regression Fitting Evaluation (All Features)
No. of Features	24.000000
R_squared score (Test)	0.857424
R_squared score (Train)	0.978897

In [80]:
```
RFR_TrainSet_Prediction = RFR.predict(X_train)
RFR_TestSet_Prediction = RFR.predict(X_test)
# RFR_TrainSet_Prediction_2 = RFR_2.predict(X_train_selected)
# RFR_TestSet_Prediction_2 = RFR_2.predict(X_test_selected)
```

In [81]:
```
RFR_predict_evaluation = {'Ranndom Forest Regression Predictions Ev
    {'Train MSE': mean_squared_error(Y_train,RFR_TrainSet_Predictio
     'Test MSE' : mean_squared_error(Y_test,RFR_TestSet_Prediction)
     'Train RMSE': mean_squared_error(Y_train,RFR_TrainSet_Predicti
     'Test RMSE' : mean_squared_error(Y_test,RFR_TestSet_Prediction
 #                          'Ranndom Forest Regression Predictions Ev
 #   {'Train MSE': mean_squared_error(Y_train,DRFRTrainSet_Predicti
 #    'Test MSE' : mean_squared_error(Y_test,RFR_TestSet_Prediction
 #    'Train RMSE': mean_squared_error(Y_train,RFR_TrainSet_Predict
 #    'Test RMSE' : mean_squared_error(Y_test,RFR_TestSet_Predictio
    }
RFR_predict_evaluation = pd.DataFrame(RFR_predict_evaluation)
RFR_predict_evaluation
```

Out[81]:

	Ranndom Forest Regression Predictions Evaluation (All Features)
Test MSE	4554.886312
Test RMSE	67.489898
Train MSE	702.687026
Train RMSE	26.508244