1.Data Cleaning

The First part of any Data Analysis Process is Data Cleaning, in this project, I have applied Data Cleaning in two steps:

```
1. Remove Outliers from Revenue Generated & Revenue Realized Column
2. Handled NaN values on ratings_given column
```

Methods used for Data Preprocessing, Inspection and Cleaning

1. describe(), 2. min(), 3. max(), 4.mean(), 5..std(), 6..isnull(), 7. 68-95-99 rule to exclude outliers

In [1]:

```
# Import the necessary libraries

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [2]:

```
#Load the required dataset
df_bookings = pd.read_csv('fact_bookings.csv')
df_hotels = pd.read_csv('dim_hotels.csv')
df_rooms = pd.read_csv('dim_rooms.csv')
df_date = pd.read_csv('dim_date.csv')
df_agg_bookings = pd.read_csv('fact_aggregated_bookings.csv')
```

In [3]:

```
df_bookings.describe()
```

Out[3]:

	property_id	no_guests	ratings_given	revenue_generated	revenue_realized
count	134590.000000	134590.000000	56683.000000	134590.000000	134590.000000
mean	18061.113493	2.036808	3.619004	14916.013188	12696.123256
std	1093.055847	1.031766	1.235009	6452.868072	6928.108124
min	16558.000000	1.000000	1.000000	6500.000000	2600.000000
25%	17558.000000	1.000000	3.000000	9900.000000	7600.000000
50%	17564.000000	2.000000	4.000000	13500.000000	11700.000000
75%	18563.000000	2.000000	5.000000	18000.000000	15300.000000
max	19563.000000	6.000000	5.000000	45220.000000	45220.000000

In [4]:

```
df_bookings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134590 entries, 0 to 134589
Data columns (total 12 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   booking_id         134590 non-null  object 
 1   property_id        134590 non-null  int64  
 2   booking_date       134590 non-null  object 
 3   check_in_date      134590 non-null  object 
 4   checkout_date      134590 non-null  object 
 5   no_guests          134590 non-null  int64  
 6   room_category      134590 non-null  object 
 7   booking_platform   134590 non-null  object 
 8   ratings_given      56683 non-null   float64
 9   booking_status     134590 non-null  object 
 10  revenue_generated  134590 non-null  int64  
 11  revenue_realized   134590 non-null  int64  
dtypes: float64(1), int64(4), object(7)
memory usage: 12.3+ MB
```

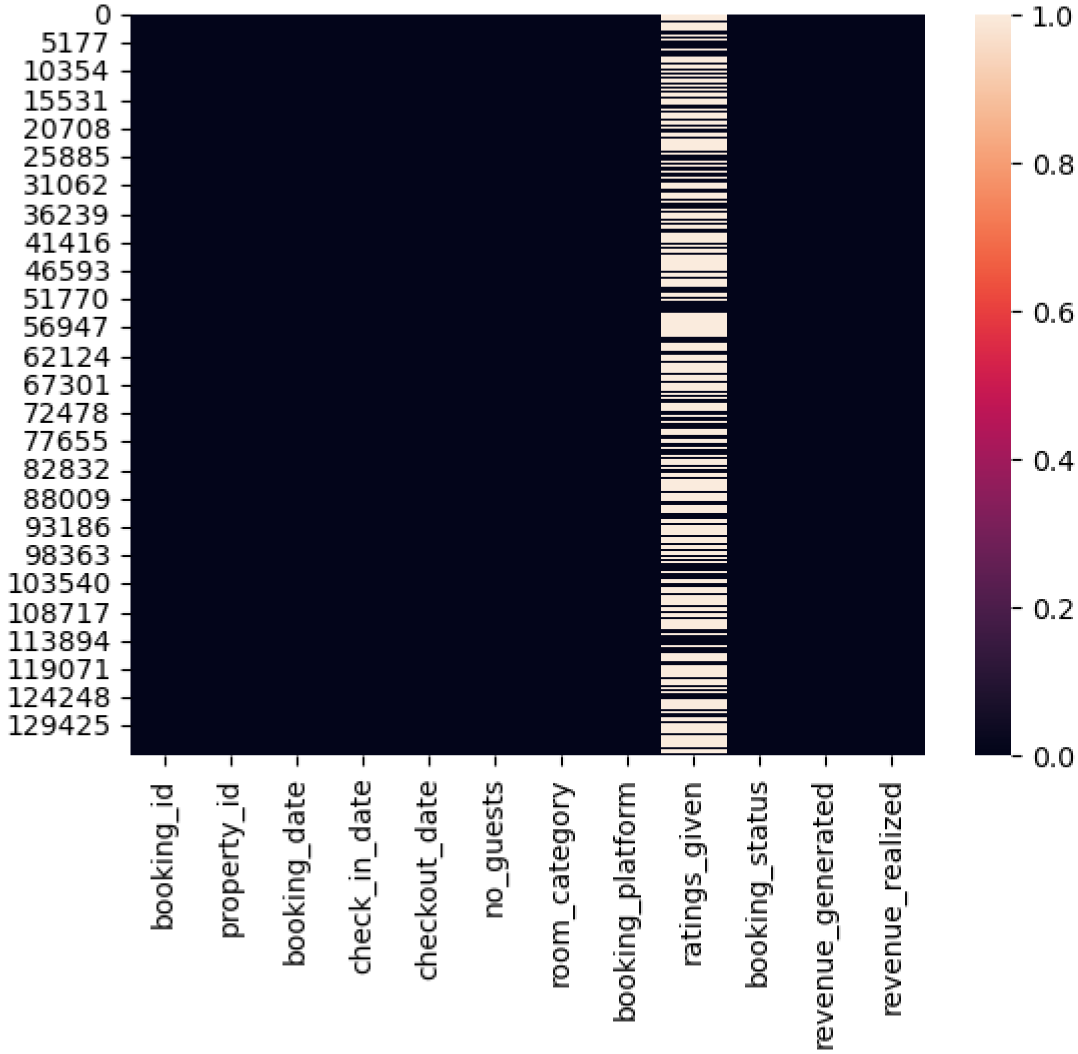
In [5]:
```
df_bookings.shape
```

Out[5]:
```
(134590, 12)
```

Use Heatmap to show null values and inconsistency

In [6]:
```
sns.heatmap(df_bookings.isnull())
```

Out[6]:
```
<Axes: >
```



I. Removing the outliers from revenue_genrated columns using 68-95-99.7 rule

```
1. Check max and min revenue generated from bookings dataframe
2. Calculate Upper limit and lower limit for the column by
calculating mean and standard deviation
3. Check Outliers for all existing revenue entries
4. Clean the outliers and saved it to the same Dataframe.
```

In [7]:
```
df_bookings
```

Out[7]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests
0	May012216558RT11	16558	2022-04-27	2022-05-01	2022-05-02	3
1	May012216558RT12	16558	2022-04-30	2022-05-01	2022-05-02	2
2	May012216558RT13	16558	2022-04-28	2022-05-01	2022-05-04	2
3	May012216558RT14	16558	2022-04-28	2022-05-01	2022-05-02	2
4	May012216558RT15	16558	2022-04-27	2022-05-01	2022-05-02	4
...	...	...	...	...	...	...
134585	Jul312217564RT46	17564	2022-07-29	2022-07-31	2022-08-03	1
134586	Jul312217564RT47	17564	2022-07-30	2022-07-31	2022-08-01	4
134587	Jul312217564RT48	17564	2022-07-30	2022-07-31	2022-08-02	1
134588	Jul312217564RT49	17564	2022-07-29	2022-07-31	2022-08-01	2
134589	Jul312217564RT410	17564	2022-07-31	2022-07-31	2022-08-01	2

134590 rows × 12 columns

In [8]:
```
max_revenue = df_bookings.revenue_generated.max()
min_revenue = df_bookings.revenue_generated.min()
max_revenue, min_revenue
```

Out[8]:
```
(45220, 6500)
```

In [9]:
```
avg_revenue = df_bookings.revenue_generated.mean()
std_dev_revenue = df_bookings.revenue_generated.std()

avg_revenue,std_dev_revenue
```

Out[9]:
```
(14916.013188201203, 6452.868071768531)
```

In [10]:
```
higher_limit = avg_revenue + 3*std_dev_revenue
lower_limit = avg_revenue - 3*std_dev_revenue
higher_limit, lower_limit
```

Out[10]:
```
(34274.617403506796, -4442.59102710439)
```

In [11]:
```
df_bookings = df_bookings[(df_bookings.revenue_generated>lower_limit) & (df_booking
```

In [12]:
```
df_bookings.revenue_generated.describe()
```

Out[12]:
```
count    133070.000000
mean      14648.486999
std        5971.338752
min        6500.000000
25%        9900.000000
50%       13500.000000
75%       18000.000000
max       34200.000000
Name: revenue_generated, dtype: float64
```

II. Removing the outliers from revenue_realized columns using 68-95-99.7 rule

```
    1. Check max and min revenue generated from bookings dataframe
    2. Calculate Upper limit and lower limit for the column by
calculating mean and standard deviation
    3. Check Outliers for all existing revenue entries
    4. Clean the outliers and saved it to the same Dataframe.
```

In [13]:
```
df_bookings.revenue_realized.describe()
```

Out[13]:
```
count    133070.000000
mean      12468.775464
std        6537.748605
min        2600.000000
25%        7600.000000
50%       11400.000000
75%       15300.000000
max       34200.000000
Name: revenue_realized, dtype: float64
```

In [14]:
```
upper_lim_real=df_bookings.revenue_realized.mean()+3*df_bookings.revenue_realized.s
upper_lim_real
```

Out[14]:
```
32082.021279982062
```

In [15]:
```
df_bookings = df_bookings[df_bookings.revenue_realized<upper_lim_real]
```

In [16]:
```
df_bookings.revenue_realized.describe()
```

Out[16]:
```
count    129814.000000
mean      11969.040712
std        5796.851530
min        2600.000000
25%        7200.000000
50%       11050.000000
75%       15300.000000
max       31920.000000
Name: revenue_realized, dtype: float64
```

Check Is there any outliers present in Room type 'RT4' as this is Presidential Room and remove if any.

In [17]:
```
df_bookings[df_bookings.room_category=='RT4'].revenue_realized.describe()
```

Out[17]:
```
count    11297.000000
mean     19627.388510
std       7284.005363
min       7600.000000
25%      12920.000000
50%      19000.000000
75%      26600.000000
max      31920.000000
Name: revenue_realized, dtype: float64
```

In [18]:
```
19627.388510+3*7284.005363
```

Out[18]:
```
41479.404599
```

There is No outliers Present as room type level

III. Checking Nan Values

```
1. Check the NaN values for all the columns using isnull() function
on all columns and then sum of it.
2. Inspect the sum for all the columns and also we can use heat map
3. Rating_given column may present null values since not all
customers give rating after check out
```

In [19]:
```
df_bookings.isnull().sum()
```

Out[19]:
```
booking_id               0
property_id              0
booking_date             0
check_in_date            0
checkout_date            0
no_guests                0
room_category            0
booking_platform         0
ratings_given        75611
booking_status           0
revenue_generated        0
revenue_realized         0
dtype: int64
```

Data Transformation

After the necessary cleaning was done, now we need to transform the data based on our need. Data Transformation may include one to a number of steps, like, adding a new column to converting column values, encoding categorical values, data aggregation, merging the dataframes, data normalization, feature engineering etc.

Add column 'occ_per%' -- successful_bookings / capacityand then apply lambda function to convert into percentage form

In [20]:
```
df_agg_bookings.head()
```

Out[20]:

	property_id	check_in_date	room_category	successful_bookings	capacity
0	16559	01-May-22	RT1	25	30
1	19562	01-May-22	RT1	28	30
2	19563	01-May-22	RT1	23	30
3	17558	01-May-22	RT1	13	19
4	16558	01-May-22	RT1	18	19

In [21]:
```
df_agg_bookings['occ_per']= df_agg_bookings.successful_bookings/df_agg_bookings.car
```

In [22]:
```
df_agg_bookings['occ_per']= df_agg_bookings.occ_per.apply(lambda x: round(x*100,2))
```

In [23]:
```
df_agg_bookings.rename(columns={'occ_per':'occ_per%'},inplace=True)
```

Insights Generation

The business questions asked in the scenarios are called ad-hoc questions,in this project we will be finding solutions of total 11 ad-hoc questions.

1. WHAT IS THE AVERAGE OCCUPANCY RATE PER ROOM TYPE?

In [24]:
```
df_bookings.head(2)
```

Out[24]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_
0	May012216558RT11	16558	2022-04-27	2022-05-01	2022-05-02	3	
1	May012216558RT12	16558	2022-04-30	2022-05-01	2022-05-02	2	

In [25]:
```
df_agg_bookings.head(2)
```

Out[25]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_per%
0	16559	01-May-22	RT1	25	30	83.33
1	19562	01-May-22	RT1	28	30	93.33

In [26]:
```
df_rooms.head(2)
```

Out[26]:

	room_id	room_class
0	RT1	Standard
1	RT2	Elite

In [27]:
```
df_agg_bookings.groupby(by='room_category')['occ_per%'].mean('occ_per%').round(2)
```

Out[27]:
```
room_category
RT1    57.92
RT2    58.01
RT3    58.03
RT4    59.28
Name: occ_per%, dtype: float64
```

In [28]:
```
#Merge two dataframes into df on column -- 'room_category','room_id'
df = pd.merge(df_agg_bookings,df_rooms,left_on ='room_category', right_on = 'room_i
```

In [29]:
```
df.groupby(by=['room_class','room_category'])['occ_per%'].mean().round(2)
```

Out[29]:
```
room_class    room_category
Elite         RT2              58.01
Premium       RT3              58.03
Presidential  RT4              59.28
Standard      RT1              57.92
Name: occ_per%, dtype: float64
```

Conclusion

Room_category- RT4 /(Presidentail Room Class) have highest average occupancy rate and and other classes have almost same occupany rate

What is the average occupancy rate city wise.

Step-01 City information is in df_hotels,So, merged this column to df on yhe property_id column to get corresponding cities

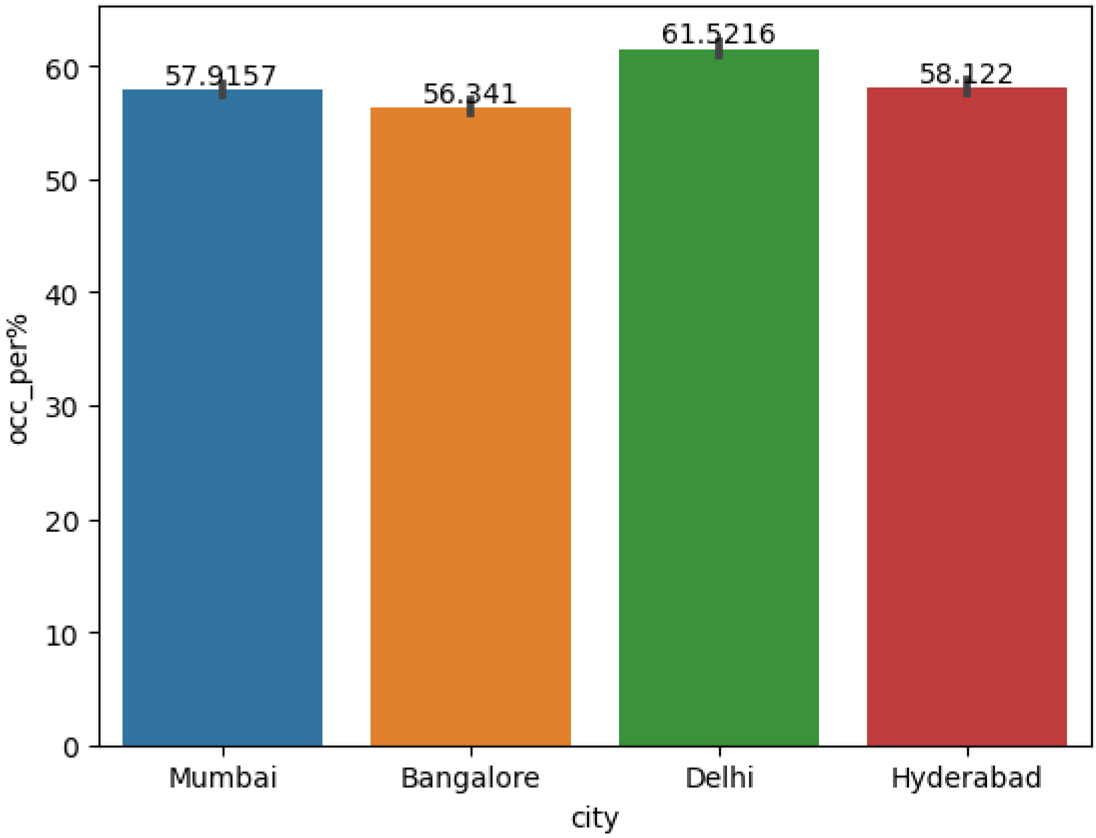
Step-02 After that group by city and calculated average occupancy percentage and rounded up

In [30]:
```
df = pd.merge(left=df, right=df_hotels, left_on='property_id', right_on ='property_
```

In [31]:
```
df.groupby('city')['occ_per%'].mean().round(2)
```

Out[31]:
```
city
Bangalore    56.34
Delhi        61.52
Hyderabad    58.12
Mumbai       57.92
Name: occ_per%, dtype: float64
```

In [32]:
```
ax=sns.barplot(data=df,x='city',y='occ_per%')
ax.bar_label(ax.containers[0])
plt.show()
```



Conclusion

```
1. Delhi has the highest average occupancy rate followed by all
other with almost same average occupancy rate.
```

Q.3 When was the occupancy better? weekday or weekend?

In [33]:
```
df_date.date.nunique()
```
Out[33]:
```
92
```

In [34]:
```
df_agg_bookings.check_in_date.nunique()
```
Out[34]:
```
92
```

Step-01 First we need to check from which table i can get date information and i found that there is separate table which has info about dates only with the column of weekday/weekend.

Step-02 Merge df_dates dataframe with df on 'date' and 'check-in date' column and then group by with day_type(weekend/weekday) and get desired output.

In [35]:
```
df_date
```
Out[35]:

	date	mmm yy	week no	day_type
0	01-May-22	May 22	W 19	weekend
1	02-May-22	May 22	W 19	weekeday
2	03-May-22	May 22	W 19	weekeday
3	04-May-22	May 22	W 19	weekeday
4	05-May-22	May 22	W 19	weekeday
...	...	...	...	...
87	27-Jul-22	Jul 22	W 31	weekeday
88	28-Jul-22	Jul 22	W 31	weekeday
89	29-Jul-22	Jul 22	W 31	weekeday
90	30-Jul-22	Jul 22	W 31	weekend
91	31-Jul-22	Jul 22	W 32	weekend

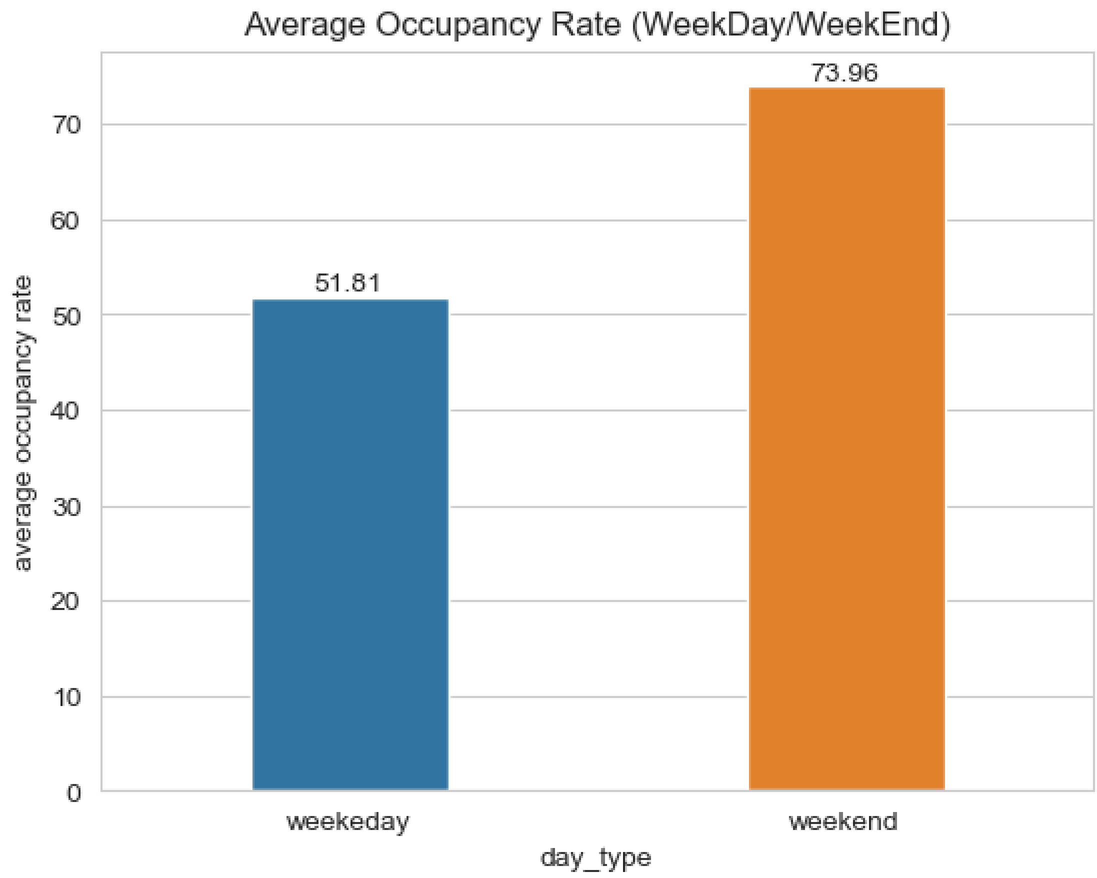
92 rows × 4 columns

In [36]:
```
df = pd.merge(left=df, right=df_date, left_on='check_in_date', right_on='date', how
```

In [37]:
```
avg_occ_pt_day = df.groupby('day_type')['occ_per%'].mean().round(2).reset_index()
```

In [38]:
```
#Set Seaborn Style
sns.set_style("whitegrid")
```

In [39]:
```
ax=sns.barplot(x=avg_occ_pt_day['day_type'],y=avg_occ_pt_day['occ_per%'],data=avg_c
for i in ax.containers:
    ax.bar_label(i,)
plt.title('Average Occupancy Rate (WeekDay/WeekEnd)')
plt.ylabel('average occupancy rate')
plt.show()
```

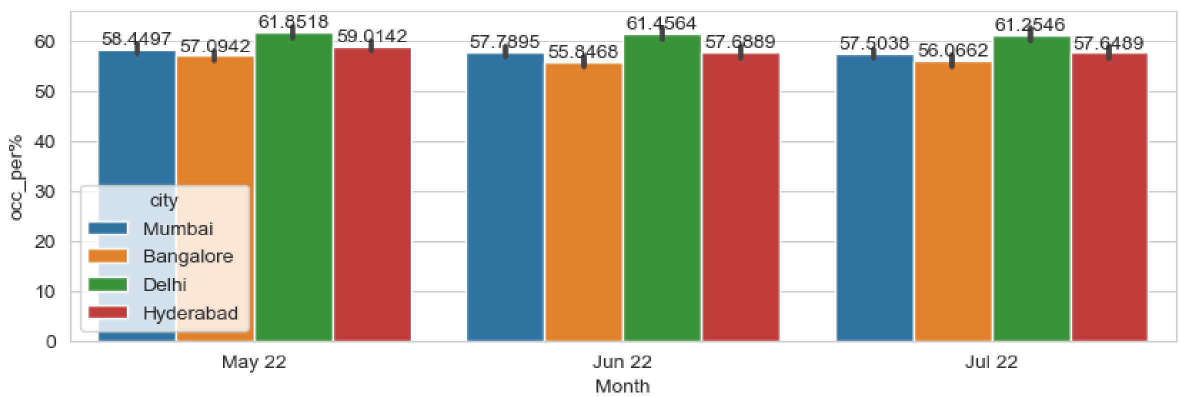


Conclusion

The Weekend days have much higher occupancy Rate(>70%) than that of week-days, probably because of holidays

Q.4 Show the City Wise Occupancy Rates for Different Months

In [56]:
```
plt.figure(figsize=(10,3))
ax=sns.barplot(x=df['mmm yy'],y=df['occ_per%'],hue=df['city'],width=0.85)
for i in ax.containers:
    ax.bar_label(i,)
plt.xlabel('Month')
plt.show()
```



In [41]:
```
df.groupby(by = ['mmm yy','city'])['occ_per%'].mean('occ_per%').round(2)
```

Out[41]:
```
mmm yy  city
Jul 22  Bangalore     56.07
        Delhi         61.25
        Hyderabad     57.65
        Mumbai        57.50
Jun 22  Bangalore     55.85
        Delhi         61.46
        Hyderabad     57.69
        Mumbai        57.79
May 22  Bangalore     57.09
        Delhi         61.85
        Hyderabad     59.01
        Mumbai        58.45
Name: occ_per%, dtype: float64
```

Conclusion

1. Delhi has the most highest occupancy rate for all three months, followed by hyderabad and mumbai.

2. Banglore has the least occupancy rate among all the cities in all three months.

Q.7 Calculate the Revenue realized city wise

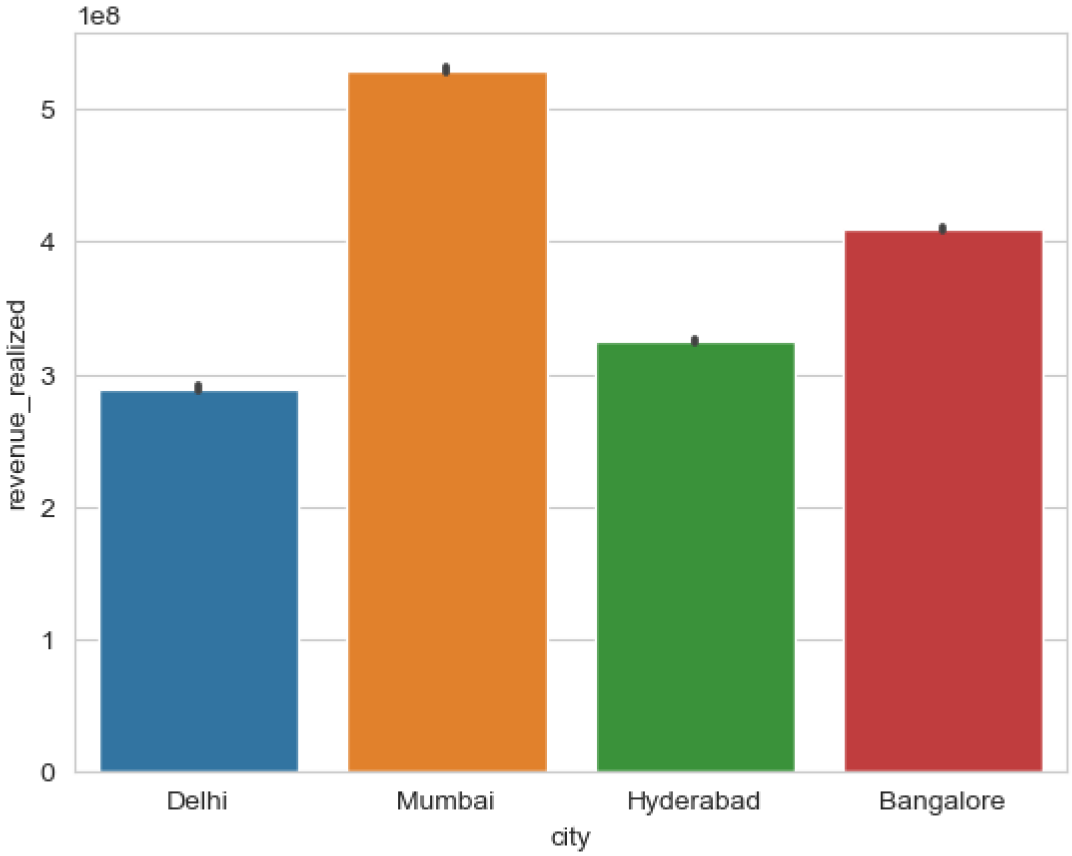
step-01

In [42]:
```
df_booking_all = pd.merge(left = df_bookings, right=df_hotels, left_on = 'property_
```

In [43]:
```
((df_booking_all.groupby('city')['revenue_realized'].sum())/1e6).round(2)
```

Out[43]:
```
city
Bangalore    409.69
Delhi        289.47
Hyderabad    325.23
Mumbai       529.36
Name: revenue_realized, dtype: float64
```

In [44]:
```
ax=sns.barplot(data= df_booking_all,x='city',y='revenue_realized',estimator='sum')
plt.show()
```



Conclusion

1. Delhi has seen highest occupancy rate as well as least revenue realized i.e. Number of cancellation for delhi city is least

2. Mumbai has seen such a high realisation amount.i.e. most number of cancelled bookings were done in mumbai

---

Q.8 Calculate the revenue MoM(Month over Month)

Step 01 - Checked the data type of date column of df_booking_all and df_date. Both are object data type for date column.

Step 02 - We need to convert them into datetime datatype using pd.to_datetime() function

Step 03- Merged the two data frames

Step 04- Group by 'mmm yy' columns and calculate the revenue for each month.

Step 05- Create Visual

In [45]:
```
df_date['date'] = pd.to_datetime(df_date['date'],errors='coerce')
```

In [46]:
```
df_date.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 92 entries, 0 to 91
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      92 non-null     datetime64[ns]
 1   mmm yy    92 non-null     object
 2   week no   92 non-null     object
 3   day_type  92 non-null     object
dtypes: datetime64[ns](1), object(3)
memory usage: 3.0+ KB
```

In [47]:
```
df_booking_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 129814 entries, 0 to 129813
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   booking_id         129814 non-null  object
 1   property_id        129814 non-null  int64
 2   booking_date       129814 non-null  object
 3   check_in_date      129814 non-null  object
 4   checkout_date      129814 non-null  object
 5   no_guests          129814 non-null  int64
 6   room_category      129814 non-null  object
 7   booking_platform   129814 non-null  object
 8   ratings_given      54203 non-null   float64
 9   booking_status     129814 non-null  object
 10  revenue_generated  129814 non-null  int64
 11  revenue_realized   129814 non-null  int64
 12  property_name      129814 non-null  object
 13  category           129814 non-null  object
 14  city               129814 non-null  object
dtypes: float64(1), int64(4), object(10)
memory usage: 15.8+ MB
```

In [48]:
```
df_booking_all['check_in_date'] = pd.to_datetime(df_booking_all['check_in_date'],er
```

In [49]:
```
df_booking_all = pd.merge(left=df_booking_all,right=df_date,left_on='check_in_date'
```

In [50]:
```
((df_booking_all.groupby('mmm yy')['revenue_generated'].sum())/1e6).round(2)
```

Out[50]:
```
mmm yy
Jul 22    616.74
Jun 22    598.47
May 22    628.59
Name: revenue_generated, dtype: float64
```

Conclusion

Revenue generated for all three months are almost same .

In [ ]:

In [ ]:

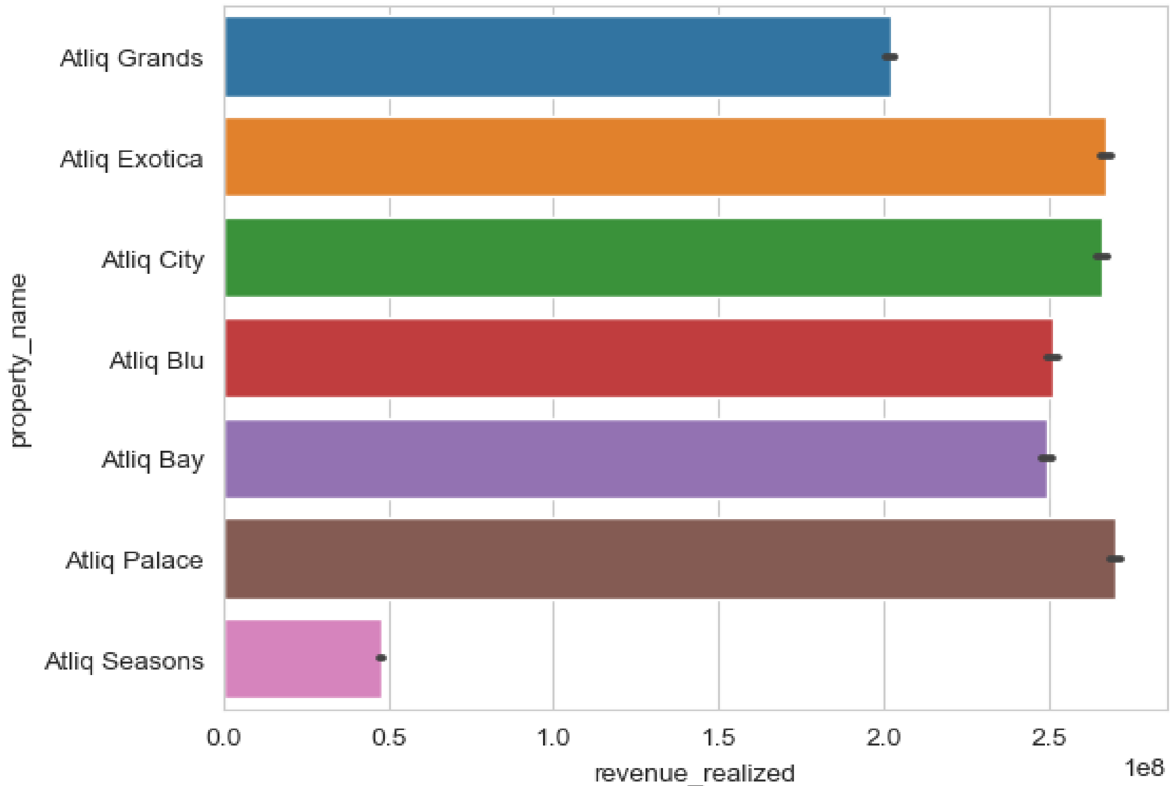
Q.9 Calculate the Revenue realized per hotel

In [51]:
```
rev_real=((df_booking_all.groupby('property_name')['revenue_realized'].sum())/1e6).
rev_real
```

Out[51]:

	property_name	revenue_realized
0	Atliq Bay	249.35
1	Atliq Blu	251.00
2	Atliq City	265.95
3	Atliq Exotica	267.25
4	Atliq Grands	201.98
5	Atliq Palace	270.20
6	Atliq Seasons	48.02

In [52]:
```
sns.barplot(data=df_booking_all,y='property_name',x='revenue_realized',estimator='s
plt.show()
```



Conclusion

1. 'The Palace type of Hotels of Atliq industries has seen highest amount of revenue genrated from cancellation followed by Atliq Exotica and Atliq City

2. AtliQ Seasons has demonstrated exceptional resilience in the face of cancellations, boasting the lowest cancellation rate at 48.02million, significantly surpassing other hotel types. This notable achievement can be attributed to its competitive pricing strategy and strategically advantageous locations, contributing to a more appealing value proposition for customer.

Q.10 What is the average rating by city

In [53]:
```
df_booking_all.groupby('city')['ratings_given'].mean()
```

Out[53]:
```
city
Bangalore    3.404000
Delhi        3.778084
Hyderabad    3.661132
Mumbai       3.646522
Name: ratings_given, dtype: float64
```

Conclusion

1.The Average ratings are almost same for all the cities.

2.None of the ratings are greater than 4

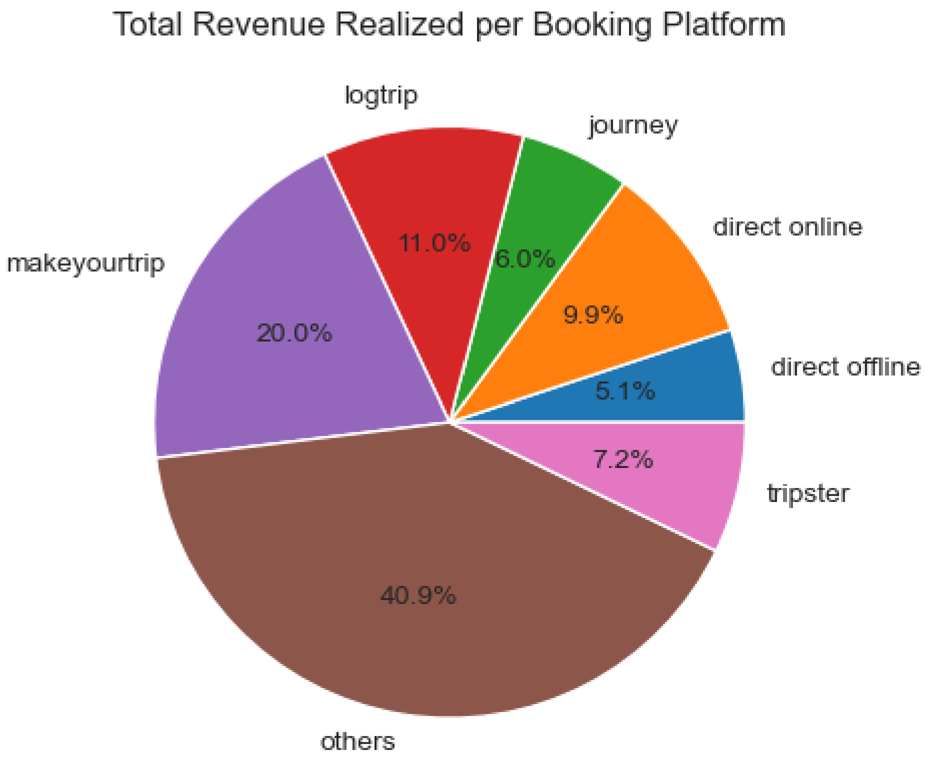
3.To elevate hotel ratings, prioritize exceptional customer service, maintain impeccable cleanliness, offer personalized experiences, and provide engaging amenities, while fostering a positive online presence and promptly addressing guest feedback. Consistency in delivering high-quality service across all aspects is key to building a favorable reputation.

Q.11 Calculate the revenue realized per booking platform

In [62]:
```
rev_real_bflat=((df_booking_all.groupby('booking_platform')['revenue_realized'].sum
rev_real_bflat
```

Out[62]:
```
booking_platform
direct offline     78.64
direct online     153.94
journey            93.57
logtrip           170.39
makeyourtrip      310.37
others            635.56
tripster          111.28
Name: revenue_realized, dtype: float64
```

In [69]:
```
plt.pie(x=rev_real_bflat,autopct='%0.1f%%',labels=rev_real_bflat.index)
plt.title('Total Revenue Realized per Booking Platform')
plt.show()
```



Conclusion

1. Majority of the bookings (40.9%) are from 'others' type of transaction

2. Investigate 'others' transactions to understand their nature, enhance data collection processes for accurate categorization, and communicate with customers to clarify choices. Implement system updates, regular audits, and staff training to ensure ongoing accuracy in booking data.

```
In [ ]:
```