AtliQ Grands Hospitality Analysis using Python

Presented by Swagata Chandra

CONTENT

- Company Overview
- AtliQ Grands Hotel Chain System
- Problem Statement
- Project Overview
- Solution adopted and Process
- Python Codes
- Insights & Recommendations
- Dashboard Resources Utilized

Company Overview

🏢 **Established Presence:** AtliQ Grands is a leading five-star hotel chain in major Indian cities, known for luxury and top-tier service. Despite 20+ years of excellence, recent market changes and rising competition have challenged its leadership.

🏨 **Hotel Properties:** AtliQ Bay, AtliQ Blu, AtliQ City, AtliQ Exotica, AtliQ Grands, AtliQ Palace, AtliQ Seasons.

🗺️ **City Locations:** Hyderabad, Bangalore, Delhi, Mumbai.

🛏️ **Room Categories/Class:** Business, Luxury, Standard, Elite, Premium, Presidential.

📲 **Booking Platforms:** Direct Offline, Direct Online, Journey, LogTrip, MakeMyTrip, Tripster, Others.

AtliQ Grands Hotel Chain System

CATEGORY

- **LUXURY** — **PROPERTY NAMES**
  1. AtliQ Grands
  2. AtliQ Exotica
  3. AtliQ Blu
  4. AtliQ Bay
- **BUSINESS** — **PROPERTY NAMES**
  1. AtliQ City
  2. AtliQ Palace
  3. AtliQ Seasons

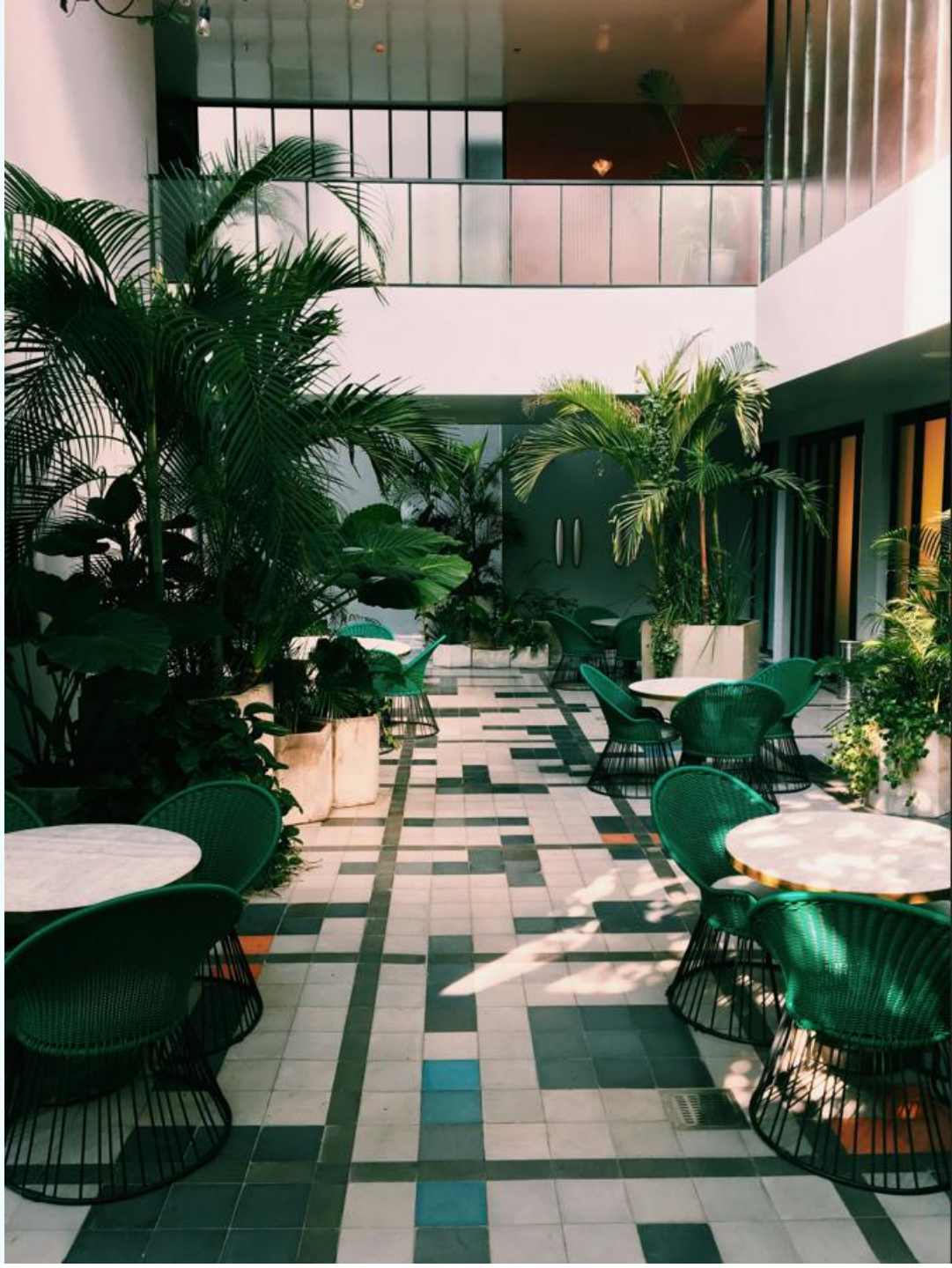
Each Property has the following 4 Types of Room classes:

- Standard
- Elite
- Premium
- Presidential

Problem Statement

- AtliQ Grands is losing market share and revenue due to poor decisions and a lack of data insights, while competitors use analytics to optimize performance. Without a data team, it’s struggling to adapt.

- To address the issue, AtliQ Grands has hired a data analytics firm to extract insights from past data, improve decisions, identify revenue opportunities, and regain market leadership.



Project Overview

Objective:

- Perform EDA on hotel booking and performance data to uncover key trends.

- Deliver insights to optimize revenue, room utilization, and seasonal planning.

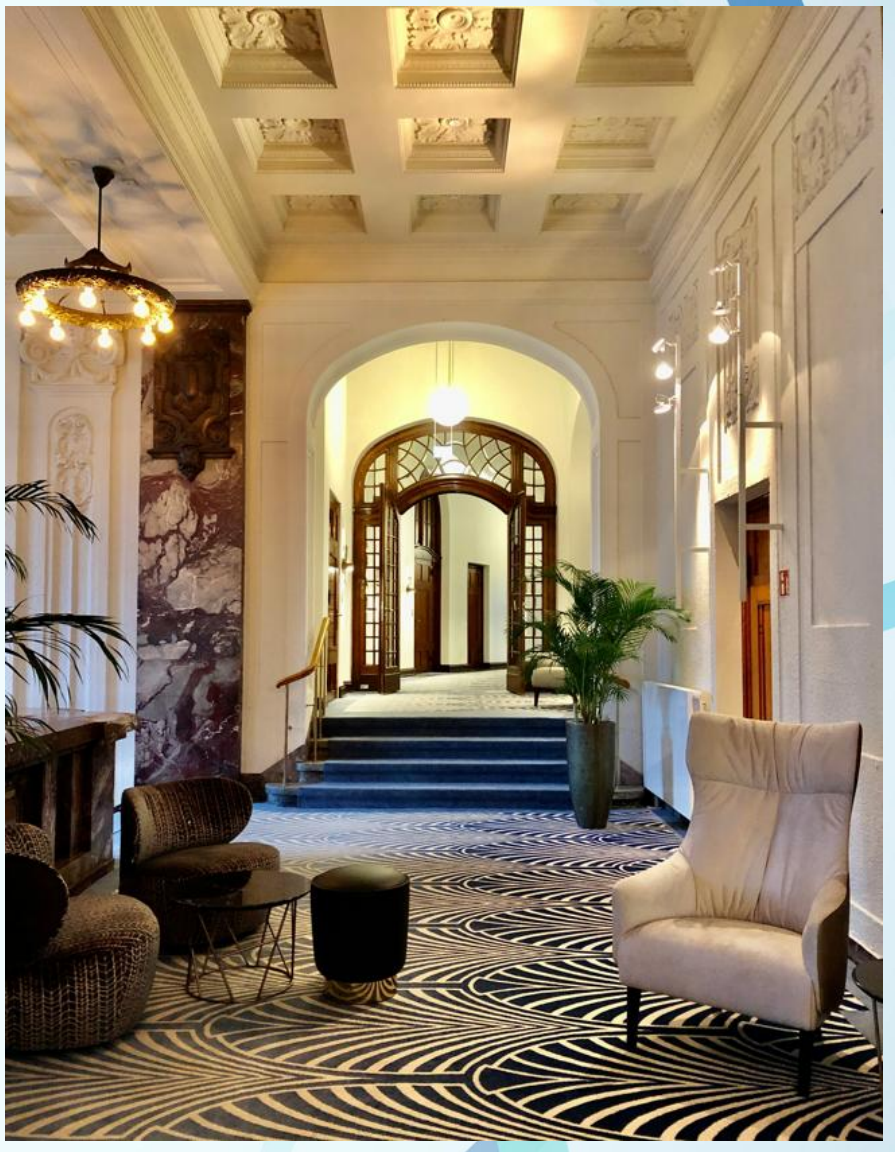
Key Goals:

- Analyse booking, revenue, guest behaviour, and occupancy trends across hotel types and periods.

- Generate insights to support strategic decisions and boost business performance.

Tech Stack:

- Python (Jupyter Notebook) – Core platform for data analysis
- Pandas, NumPy – Data cleaning, transformation, and manipulation.
- Matplotlib, Seaborn – Data visualization and trend analysis.



Solution Approach & Process

1. Data Import & Exploration:

- Loaded multiple datasets into the Jupyter notebook.
- Checked the structure and basic statistics to understand the data.

2. Data Cleaning:

- Filled or removed missing values.
- Removed outlier values.
- Fixed inconsistent formats.

3. Data Transformation:

- Combined related datasets for better analysis.
- Created new columns to add more insights.
- Standardized values to make data consistent.

4. Insight Generation:

- Created charts to show trends by city, room type, and time.
- Analysed revenue and room occupancy to find key patterns.

Key Performance Indicators (KPIs)

- ADR (Average Daily Rate) - Total Rooms Revenue/ No. of Rooms Sold
- DSRN (Daily Sellable Room Nights) = Total Rooms Available to Sell/ No. of Days
- DURN (Daily Utilized Room Nights) = Total Checked out/ No. of Days
- DBRN (Daily Booked Room Nights) = Total Bookings/ No. of Rooms Sold
- Occupancy% = Total Rooms Occupied/ Total Rooms Available
- RevPAR (Revenue Per Available Room) = Total Revenue / Total Rooms Available to Sell
- Realization = DURN/DBRN

Python Codes

AtliQ Hospitality Domain Data Analysis Project

In [1]:
```
import pandas as pd
```

==> 1. Data Import and Data Exploration

Datasets

We have 5 csv file

- dim_date.csv
- dim_hotels.csv
- dim_rooms.csv
- fact_aggregated_bookings
- fact_bookings.csv

Read bookings data in a dataframe

In [4]:
```
# Load dimension tables
df_date = pd.read_csv('datasets/dim_date.csv')
df_hotels = pd.read_csv('datasets/dim_hotels.csv')
df_rooms = pd.read_csv('datasets/dim_rooms.csv')
```

In [5]:
```
# Load fact tables
df_agg_bookings = pd.read_csv('datasets/fact_aggregated_bookings.csv')
df_bookings = pd.read_csv('datasets/fact_bookings.csv')
```

Explore bookings data

In [6]:
```
df_bookings.head()
```

Out[6]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_gue
0	May012216558RT11	16558	27-04-22	1/5/2022	2/5/2022	-
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	
2	May012216558RT13	16558	28-04-22	1/5/2022	4/5/2022	
3	May012216558RT14	16558	28-04-22	1/5/2022	2/5/2022	-
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	

In [7]:
```
df_bookings.shape
```

Out[7]:
```
(134590, 12)
```

In [8]:
```
df_bookings.room_category.unique()
```

Out[8]:
```
array(['RT1', 'RT2', 'RT3', 'RT4'], dtype=object)
```

In [9]:
```
df_bookings.booking_platform.unique()
```

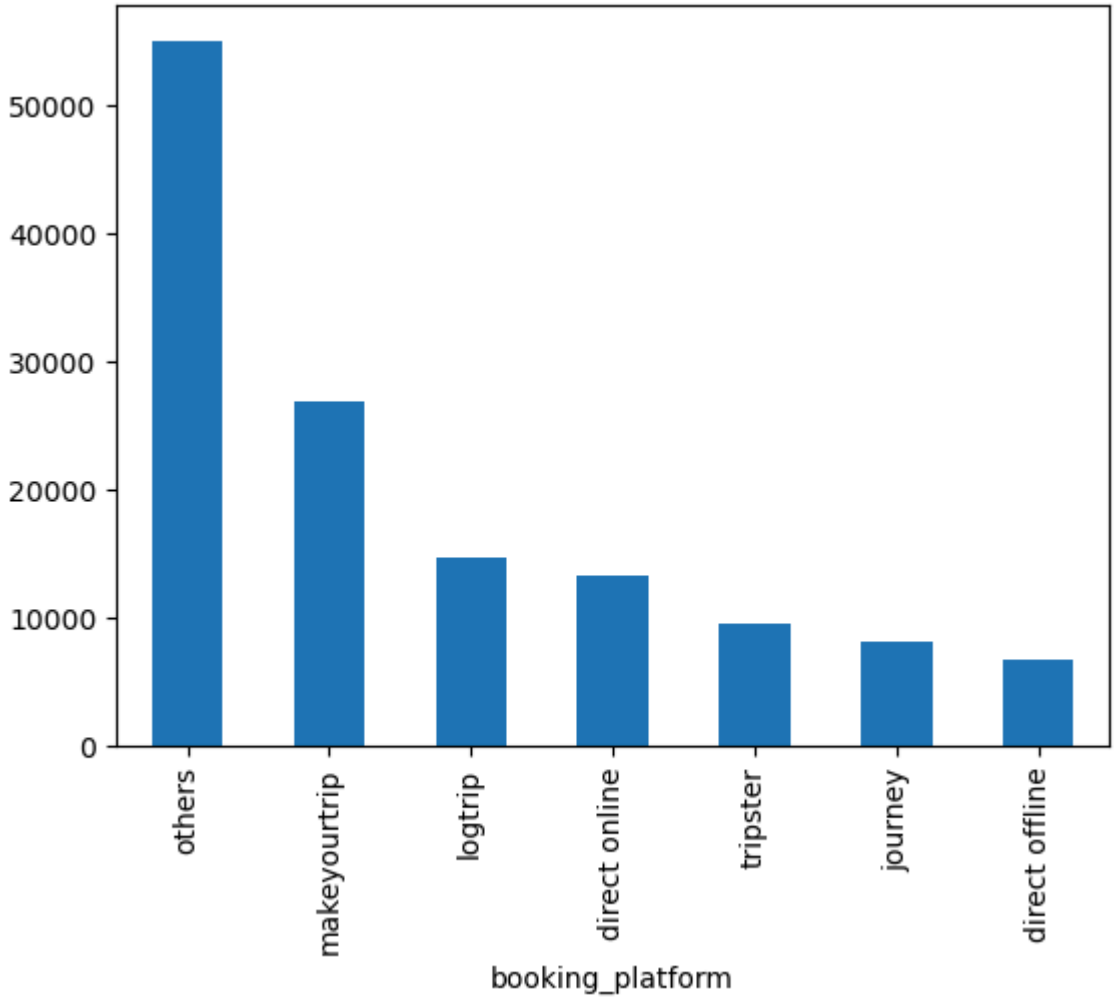
Out[9]:
```
array(['direct online', 'others', 'logtrip', 'tripster', 'makeyourtrip',
       'journey', 'direct offline'], dtype=object)
```

In [10]:
```
df_bookings.booking_platform.value_counts()
```

Out[10]:
```
booking_platform
others            55066
makeyourtrip      26898
logtrip           14756
direct online     13379
tripster           9630
journey            8106
direct offline     6755
Name: count, dtype: int64
```

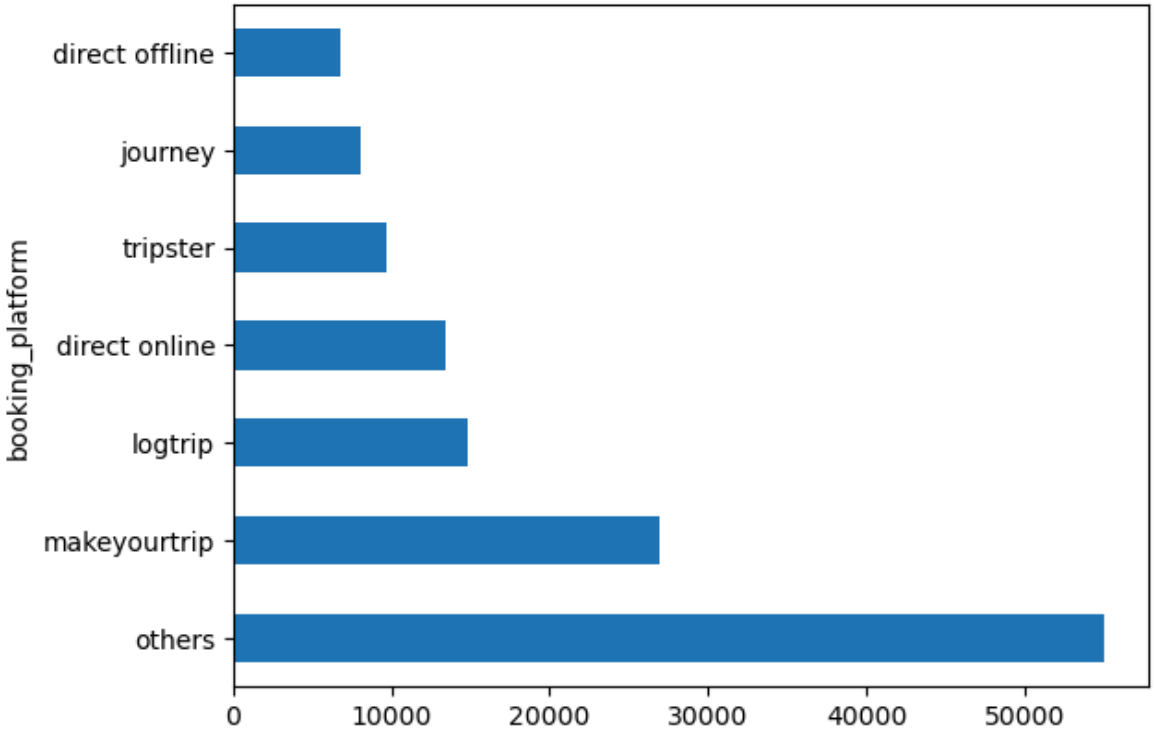
In [11]:
```
df_bookings.booking_platform.value_counts().plot(kind="bar")
```

Out[11]:
```
<Axes: xlabel='booking_platform'>
```



In [13]:
```
df_bookings.booking_platform.value_counts().plot(kind="barh")
```

Out[13]:
```
<Axes: ylabel='booking_platform'>
```



In [14]:
```
df_bookings.describe()
```

Out[14]:

	property_id	no_guests	ratings_given	revenue_generated	revenue_realized
count	134590.000000	134587.000000	56683.000000	1.345900e+05	134590.000000
mean	18061.113493	2.036170	3.619004	1.537805e+04	12696.123256
std	1093.055847	1.034885	1.235009	9.303604e+04	6928.108124
min	16558.000000	-17.000000	1.000000	6.500000e+03	2600.000000
25%	17558.000000	1.000000	3.000000	9.900000e+03	7600.000000
50%	17564.000000	2.000000	4.000000	1.350000e+04	11700.000000
75%	18563.000000	2.000000	5.000000	1.800000e+04	15300.000000
max	19563.000000	6.000000	5.000000	2.856000e+07	45220.000000

In [15]:
```
df_bookings.revenue_generated.min(),df_bookings.revenue_generated.max()
```

Out[15]: (6500, 28560000)

In [16]:
```
df_hotels.shape
```

Out[16]: (25, 4)

In [17]:
```
df_hotels.head(3)
```

Out[17]:

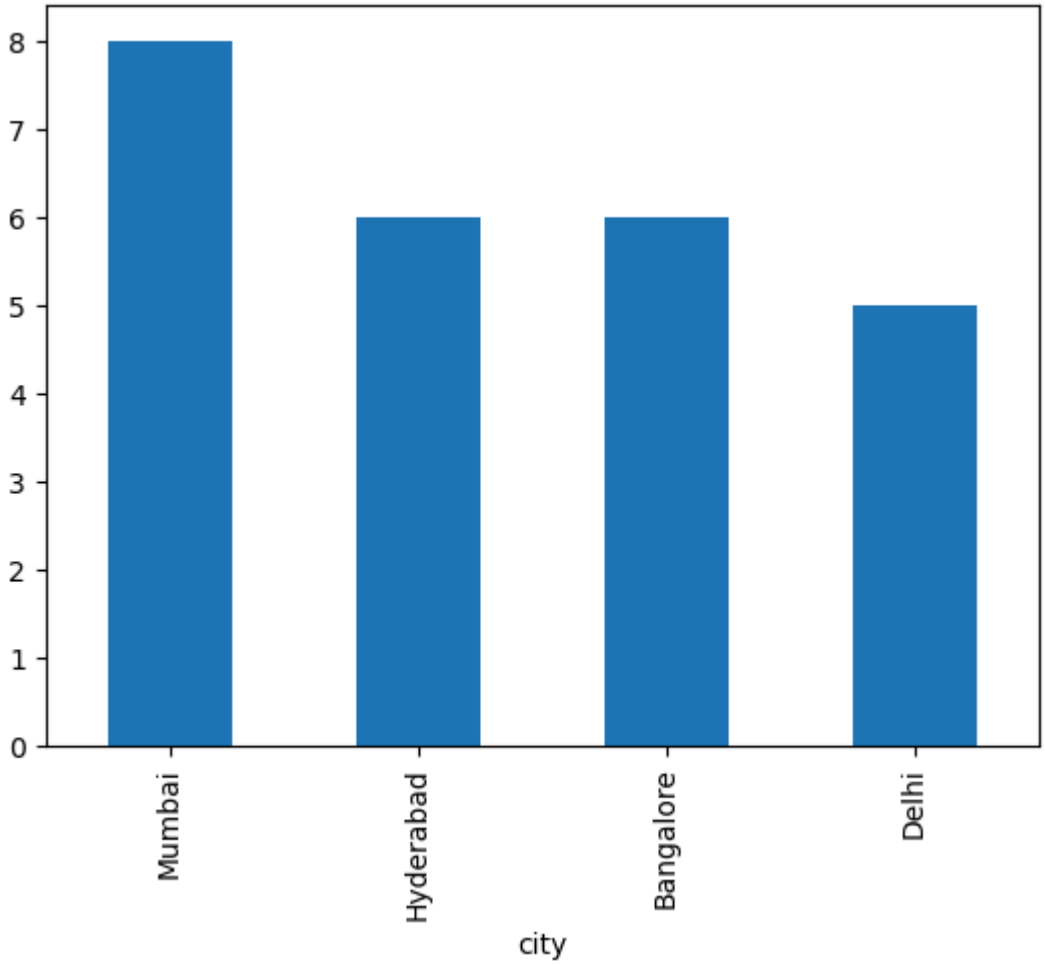
	property_id	property_name	category	city
0	16558	Atliq Grands	Luxury	Delhi
1	16559	Atliq Exotica	Luxury	Mumbai
2	16560	Atliq City	Business	Delhi

In [18]:
```
df_hotels.category.value_counts()
```

Out[18]:
```
category
Luxury      16
Business     9
Name: count, dtype: int64
```

In [17]:
```
df_hotels.city.value_counts().plot(kind="bar")
```

Out[17]:
```
<Axes: xlabel='city'>
```



In [19]:
```
df_hotels.city.value_counts().sort_values()
```

Out[19]:
```
city
Delhi        5
Hyderabad    6
Bangalore    6
Mumbai       8
Name: count, dtype: int64
```

Explore aggregate bookings

In [20]:
```
df = pd.read_csv('datasets/fact_aggregated_bookings.csv')
df
```

Out[20]:

	property_id	check_in_date	room_category	successful_bookings	capacity
0	16559	1-May-22	RT1	25	30.0
1	19562	1-May-22	RT1	28	30.0
2	19563	1-May-22	RT1	23	30.0
3	17558	1-May-22	RT1	30	19.0
4	16558	1-May-22	RT1	18	19.0
...	...	...	...	...	...
9195	16563	31-Jul-22	RT4	13	18.0
9196	16559	31-Jul-22	RT4	13	18.0
9197	17558	31-Jul-22	RT4	3	6.0
9198	19563	31-Jul-22	RT4	3	6.0
9199	17561	31-Jul-22	RT4	3	4.0

9200 rows × 5 columns

In [21]:
```
df_agg_bookings.head(3)
```

Out[21]:

	property_id	check_in_date	room_category	successful_bookings	capacity
0	16559	1-May-22	RT1	25	30.0
1	19562	1-May-22	RT1	28	30.0
2	19563	1-May-22	RT1	23	30.0

Find out unique property ids in aggregate bookings dataset

In [23]:
```
df_agg_bookings.property_id.unique()
```

Out[23]:
```
array([16559, 19562, 19563, 17558, 16558, 17560, 19558, 19560, 17561,
       16560, 16561, 16562, 16563, 17559, 17562, 17563, 18558, 18559,
       18561, 18562, 18563, 19559, 19561, 17564, 18560], dtype=int64)
```

Find out total bookings per property_id

In [24]:
```
df_agg_bookings.groupby("property_id")["successful_bookings"].sum()
```

Out[24]:
```
property_id
16558    3153
16559    7338
16560    4693
16561    4418
16562    4820
16563    7211
17558    5053
17559    6142
17560    6013
17561    5183
17562    3424
17563    6337
17564    3982
18558    4475
18559    5256
18560    6638
18561    6458
18562    7333
18563    4737
19558    4400
19559    4729
19560    6079
19561    5736
19562    5812
19563    5413
Name: successful_bookings, dtype: int64
```

Find out days on which bookings are greater than capacity

In [25]:
```
df_agg_bookings[df_agg_bookings.successful_bookings>df_agg_bookings.capacity]
```

Out[25]:

	property_id	check_in_date	room_category	successful_bookings	capacity
3	17558	1-May-22	RT1	30	19.0
12	16563	1-May-22	RT1	100	41.0
4136	19558	11-Jun-22	RT2	50	39.0
6209	19560	2-Jul-22	RT1	123	26.0
8522	19559	25-Jul-22	RT1	35	24.0
9194	18563	31-Jul-22	RT4	20	18.0

Find out properties that have highest capacity

In [25]:
```
df_agg_bookings.groupby("property_id")["capacity"].max().sort_values()
```

Out[25]:
```
property_id
16558    22.0
16561    24.0
18563    29.0
17562    30.0
19562    30.0
18558    30.0
16560    34.0
17561    36.0
19560    38.0
18562    38.0
17559    39.0
19558    40.0
18561    40.0
18560    40.0
17564    40.0
16563    41.0
19559    41.0
16559    41.0
16562    43.0
17563    44.0
18559    44.0
17560    45.0
19561    45.0
19563    45.0
17558    50.0
Name: capacity, dtype: float64
```

In [27]:
```
df_agg_bookings[df_agg_bookings.capacity==df_agg_bookings.capacity.max()]
```

Out[27]:

	property_id	check_in_date	room_category	successful_bookings	capacity
27	17558	1-May-22	RT2	38	50.0
128	17558	2-May-22	RT2	27	50.0
229	17558	3-May-22	RT2	26	50.0
328	17558	4-May-22	RT2	27	50.0
428	17558	5-May-22	RT2	29	50.0
...	...	...	...	...	...
8728	17558	27-Jul-22	RT2	22	50.0
8828	17558	28-Jul-22	RT2	21	50.0
8928	17558	29-Jul-22	RT2	23	50.0
9028	17558	30-Jul-22	RT2	32	50.0
9128	17558	31-Jul-22	RT2	30	50.0

92 rows × 5 columns

==> 2. Data Cleaning

In [28]:
```
df_bookings.describe()
```

Out[28]:

	property_id	no_guests	ratings_given	revenue_generated	revenue_realized
count	134590.000000	134587.000000	56683.000000	1.345900e+05	134590.000000
mean	18061.113493	2.036170	3.619004	1.537805e+04	12696.123256
std	1093.055847	1.034885	1.235009	9.303604e+04	6928.108124
min	16558.000000	-17.000000	1.000000	6.500000e+03	2600.000000
25%	17558.000000	1.000000	3.000000	9.900000e+03	7600.000000
50%	17564.000000	2.000000	4.000000	1.350000e+04	11700.000000
75%	18563.000000	2.000000	5.000000	1.800000e+04	15300.000000
max	19563.000000	6.000000	5.000000	2.856000e+07	45220.000000

In [29]:
```
df_bookings.shape
```

Out[29]:
```
(134590, 12)
```

(1) Clean invalid guests

In [31]:
```
df=df_bookings[df_bookings.no_guests<=0]
df
```

Out[31]:

	booking_id	property_id	booking_date	check_in_date	checkout_date
0	May012216558RT11	16558	27-04-22	1/5/2022	2/5/2022
3	May012216558RT14	16558	28-04-22	1/5/2022	2/5/2022
17924	May122218559RT44	18559	12/5/2022	12/5/2022	14-05-22
18020	May122218561RT22	18561	8/5/2022	12/5/2022	14-05-22
18119	May122218562RT311	18562	5/5/2022	12/5/2022	17-05-22
18121	May122218562RT313	18562	10/5/2022	12/5/2022	17-05-22
56715	Jun082218562RT12	18562	5/6/2022	8/6/2022	13-06-22
119765	Jul202219560RT220	19560	19-07-22	20-07-22	22-07-22
134586	Jul312217564RT47	17564	30-07-22	31-07-22	1/8/2022

In [ ]:
```
As you can see above, number of guests having less than zero value represents da
```

In [32]:
```
df.shape ## row x column (where guests are in -ve values)
```

Out[32]:
```
(9, 12)
```

In [35]:
```
df_bookings = df_bookings[df_bookings.no_guests>0]
```

In [36]:
```
df_bookings.shape
```

Out[36]:
```
(134578, 12)
```

(2) Outlier removal in revenue generated

In [37]:
```
df_bookings.revenue_generated.min(), df_bookings.revenue_generated.max() ## from
```

Out[37]:
```
(6500, 28560000)
```

In [38]:
```
df_bookings.revenue_generated.mean(), df_bookings.revenue_generated.median()
```

Out[38]:
```
(15378.036937686695, 13500.0)
```

In [41]:
```
avg, std = df_bookings.revenue_generated.mean(), df_bookings.revenue_generated.s
```

In [42]:
```
higher_limit = avg + 3*std
higher_limit ## it will be considered as a +ve outlier
```

Out[42]:
```
294498.50173198653
```

In [43]:
```
lower_limit = avg - 3*std
lower_limit ## it will be considered as a -ve outlier
```

Out[43]:
```
-263742.4278566132
```

In [44]:
```
df_bookings[df_bookings.revenue_generated<=0]
```

Out[44]:

booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	roo

In [45]:
```
df_bookings[df_bookings.revenue_generated>higher_limit] ## these are the outlier
```

Out[45]:

	booking_id	property_id	booking_date	check_in_date	checkout_date
2	May012216558RT13	16558	28-04-22	1/5/2022	4/5/2022
111	May012216559RT32	16559	29-04-22	1/5/2022	2/5/2022
315	May012216562RT22	16562	28-04-22	1/5/2022	4/5/2022
562	May012217559RT118	17559	26-04-22	1/5/2022	2/5/2022
129176	Jul282216562RT26	16562	21-07-22	28-07-22	29-07-22

In [46]:
```
df_bookings[df_bookings.revenue_generated<higher_limit]
```

Out[46]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	r
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	
5	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	
6	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	
7	May012216558RT18	16558	26-04-22	1/5/2022	3/5/2022	
...	...	...	...	...	...	
134584	Jul312217564RT45	17564	30-07-22	31-07-22	1/8/2022	
134585	Jul312217564RT46	17564	29-07-22	31-07-22	3/8/2022	
134587	Jul312217564RT48	17564	30-07-22	31-07-22	2/8/2022	
134588	Jul312217564RT49	17564	29-07-22	31-07-22	1/8/2022	
134589	Jul312217564RT410	17564	31-07-22	31-07-22	1/8/2022	

134573 rows × 12 columns

In [47]:
```
df_bookings = df_bookings[df_bookings.revenue_generated<=higher_limit]
df_bookings.shape ## outliers are removed
```

Out[47]:
```
(134573, 12)
```

In [48]:
```
df_bookings.revenue_realized.describe()
```

Out[48]:
```
count    134573.000000
mean      12695.983585
std        6927.791692
min        2600.000000
25%        7600.000000
50%       11700.000000
75%       15300.000000
max       45220.000000
Name: revenue_realized, dtype: float64
```

In [49]:
```
higher_limit = df_bookings.revenue_realized.mean() + 3*df_bookings.revenue_reali
higher_limit ## this value is an outlier
```

Out[49]:
```
33479.3586618449
```

In [50]:
```
df_bookings[df_bookings.revenue_realized>higher_limit]
```

Out[50]:

	booking_id	property_id	booking_date	check_in_date	checkout_date
137	May012216559RT41	16559	27-04-22	1/5/2022	7/5/2022
139	May012216559RT43	16559	1/5/2022	1/5/2022	2/5/2022
143	May012216559RT47	16559	28-04-22	1/5/2022	3/5/2022
149	May012216559RT413	16559	24-04-22	1/5/2022	7/5/2022
222	May012216560RT45	16560	30-04-22	1/5/2022	3/5/2022
...	...	...	...	...	...
134328	Jul312219560RT49	19560	31-07-22	31-07-22	2/8/2022
134331	Jul312219560RT412	19560	31-07-22	31-07-22	1/8/2022
134467	Jul312219562RT45	19562	28-07-22	31-07-22	1/8/2022
134474	Jul312219562RT412	19562	25-07-22	31-07-22	6/8/2022
134581	Jul312217564RT42	17564	31-07-22	31-07-22	1/8/2022

1299 rows × 12 columns

One observation we can have in above dataframe is that all rooms are RT4 which means presidential suit. Now since RT4 is a luxurious room it is likely their rent will be higher. To make a fair analysis, we need to do data analysis only on RT4 room types

In [51]:
```
df_bookings[df_bookings.room_category=="RT4"]
```

Out[51]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	r
47	May012216558RT41	16558	26-04-22	1/5/2022	3/5/2022	
48	May012216558RT42	16558	27-04-22	1/5/2022	2/5/2022	
49	May012216558RT43	16558	29-04-22	1/5/2022	4/5/2022	
137	May012216559RT41	16559	27-04-22	1/5/2022	7/5/2022	
138	May012216559RT42	16559	11/4/2022	1/5/2022	3/5/2022	
...	...	...	...	...	...	
134584	Jul312217564RT45	17564	30-07-22	31-07-22	1/8/2022	
134585	Jul312217564RT46	17564	29-07-22	31-07-22	3/8/2022	
134587	Jul312217564RT48	17564	30-07-22	31-07-22	2/8/2022	
134588	Jul312217564RT49	17564	29-07-22	31-07-22	1/8/2022	
134589	Jul312217564RT410	17564	31-07-22	31-07-22	1/8/2022	

16071 rows × 12 columns

In [52]:
```
df_bookings[df_bookings.room_category=="RT4"].revenue_realized.describe()
```

Out[52]:
```
count    16071.000000
mean     23439.308444
std       9048.599076
min       7600.000000
25%      19000.000000
50%      26600.000000
75%      32300.000000
max      45220.000000
Name: revenue_realized, dtype: float64
```

In [53]:
```
# mean + 3*standard deviation
23439+3*9048 ## this is the higher limit for RT-4 rooms, beyond this value, it i
```

Out[53]:
```
50583
```

Here higher limit comes to be 50583 and in our dataframe above we can see that max value for revenue realized is 45220. Hence we can conclude that there is no outlier and we don't need to do any data cleaning on this particular column

In [54]:
```
df_bookings[df_bookings.booking_id=="May012216558RT213"]
```

Out[54]:

booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	roo

In [55]:
```
df_bookings.isnull().sum()
```

Out[55]:
```
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        77897
booking_status           0
revenue_generated        0
revenue_realized         0
dtype: int64
```

The total values in our dataframe are 134576. Out of those 77897 rows, 77897 have a null rating. Since there are many rows with null ratings, we should not filter these values. Also, we should not replace this rating with a median or mean rating, etc.

In aggregate bookings, find columns that have null values. Fill these null values with whatever you think is the appropriate substitute (possible ways are to use mean or median)

In [56]:
```
df_agg_bookings.isnull().sum()
```

Out[56]:
```
property_id            0
check_in_date          0
room_category          0
successful_bookings    0
capacity               2
dtype: int64
```

In [57]:
```
df_agg_bookings[df_agg_bookings.capacity.isna()]
```

Out[57]:

	property_id	check_in_date	room_category	successful_bookings	capacity
8	17561	1-May-22	RT1	22	NaN
14	17562	1-May-22	RT1	12	NaN

In [62]:
```
df_agg_bookings.capacity.median()
```

Out[62]:
```
25.0
```

In [63]:
```
df_agg_bookings.capacity.fillna(df_agg_bookings.capacity.median(), inplace=True)
```

In [65]:
```
df_agg_bookings.loc[[8,14]]
```

Out[65]:

	property_id	check_in_date	room_category	successful_bookings	capacity
8	17561	1-May-22	RT1	22	25.0
14	17562	1-May-22	RT1	12	25.0

In aggregate bookings find out records that have successful_bookings value greater than capacity. Filter those records

In [66]:
```
df_agg_bookings[df_agg_bookings.successful_bookings>df_agg_bookings.capacity]
```

Out[66]:

	property_id	check_in_date	room_category	successful_bookings	capacity
3	17558	1-May-22	RT1	30	19.0
12	16563	1-May-22	RT1	100	41.0
4136	19558	11-Jun-22	RT2	50	39.0
6209	19560	2-Jul-22	RT1	123	26.0
8522	19559	25-Jul-22	RT1	35	24.0
9194	18563	31-Jul-22	RT4	20	18.0

==> 3. Data Transformation

Create occupancy percentage column

In [67]:
```
df_agg_bookings.head(3)
```

Out[67]:

	property_id	check_in_date	room_category	successful_bookings	capacity
0	16559	1-May-22	RT1	25	30.0
1	19562	1-May-22	RT1	28	30.0
2	19563	1-May-22	RT1	23	30.0

In [68]:
```
## occupancy% = successful booking/capacity (83% is occupancy%)
25/30
```

Out[68]:
```
0.8333333333333334
```

In [69]:
```
df_agg_bookings['occ_pct'] = df_agg_bookings['successful_bookings']/df_agg_booki
```

In [70]:
```
df_agg_bookings.head()
```

Out[70]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
0	16559	1-May-22	RT1	25	30.0	0.833333
1	19562	1-May-22	RT1	28	30.0	0.933333
2	19563	1-May-22	RT1	23	30.0	0.766667
3	17558	1-May-22	RT1	30	19.0	1.578947
4	16558	1-May-22	RT1	18	19.0	0.947368

In [84]:
```
# Convert it into a percentage value
df_agg_bookings['occ_pct'] = df_agg_bookings['occ_pct'].apply(lambda x: round(x*
df_agg_bookings.head(3)
```

Out[84]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
0	16559	1-May-22	RT1	25	30.0	83.33
1	19562	1-May-22	RT1	28	30.0	93.33
2	19563	1-May-22	RT1	23	30.0	76.67

In [74]:
```
df_agg_bookings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9200 entries, 0 to 9199
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   property_id          9200 non-null   int64
 1   check_in_date        9200 non-null   object
 2   room_category        9200 non-null   object
 3   successful_bookings  9200 non-null   int64
 4   capacity             9200 non-null   float64
 5   occ_pct              9200 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 431.4+ KB
```

==> 4. Insights Generation

---

1. What is an average occupancy rate in each of the room categories?

In [85]:
```
df_agg_bookings.head(3)
```

Out[85]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
0	16559	1-May-22	RT1	25	30.0	83.33
1	19562	1-May-22	RT1	28	30.0	93.33
2	19563	1-May-22	RT1	23	30.0	76.67

In [87]:
```
df_agg_bookings.groupby("room_category")["occ_pct"].mean().round(2)
```

Out[87]:
```
room_category
RT1    58.23
RT2    58.04
RT3    58.03
RT4    59.30
Name: occ_pct, dtype: float64
```

In [88]:
```
df_rooms
```

Out[88]:

	room_id	room_class
0	RT1	Standard
1	RT2	Elite
2	RT3	Premium
3	RT4	Presidential

In [89]:
```
df = pd.merge(df_agg_bookings, df_rooms, left_on="room_category", right_on="room
df.head()
```

Out[89]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	r
0	16559	1-May-22	RT1	25	30.0	83.33	
1	19562	1-May-22	RT1	28	30.0	93.33	
2	19563	1-May-22	RT1	23	30.0	76.67	
3	17558	1-May-22	RT1	30	19.0	157.89	
4	16558	1-May-22	RT1	18	19.0	94.74	

In [90]:
```
df.drop("room_id",axis=1, inplace=True)
df.head() ## drop room_id column/axis=1 means it drops the column not the row/in
```

Out[90]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	r
0	16559	1-May-22	RT1	25	30.0	83.33	
1	19562	1-May-22	RT1	28	30.0	93.33	
2	19563	1-May-22	RT1	23	30.0	76.67	
3	17558	1-May-22	RT1	30	19.0	157.89	
4	16558	1-May-22	RT1	18	19.0	94.74	

In [91]:
```
df.groupby("room_class")["occ_pct"].mean().round(2)
```

Out[91]:
```
room_class
Elite           58.04
Premium         58.03
Presidential    59.30
Standard        58.23
Name: occ_pct, dtype: float64
```

In [92]:
```
df[df.room_class=="Standard"].occ_pct.mean().round(2)
```

Out[92]:
```
58.23
```

2. Print average occupancy rate per city

In [93]:
```
df_hotels.head(3)
```

Out[93]:

	property_id	property_name	category	city
0	16558	Atliq Grands	Luxury	Delhi
1	16559	Atliq Exotica	Luxury	Mumbai
2	16560	Atliq City	Business	Delhi

In [94]:
```
df = pd.merge(df, df_hotels, on="property_id")
df.head(3)
```

Out[94]:

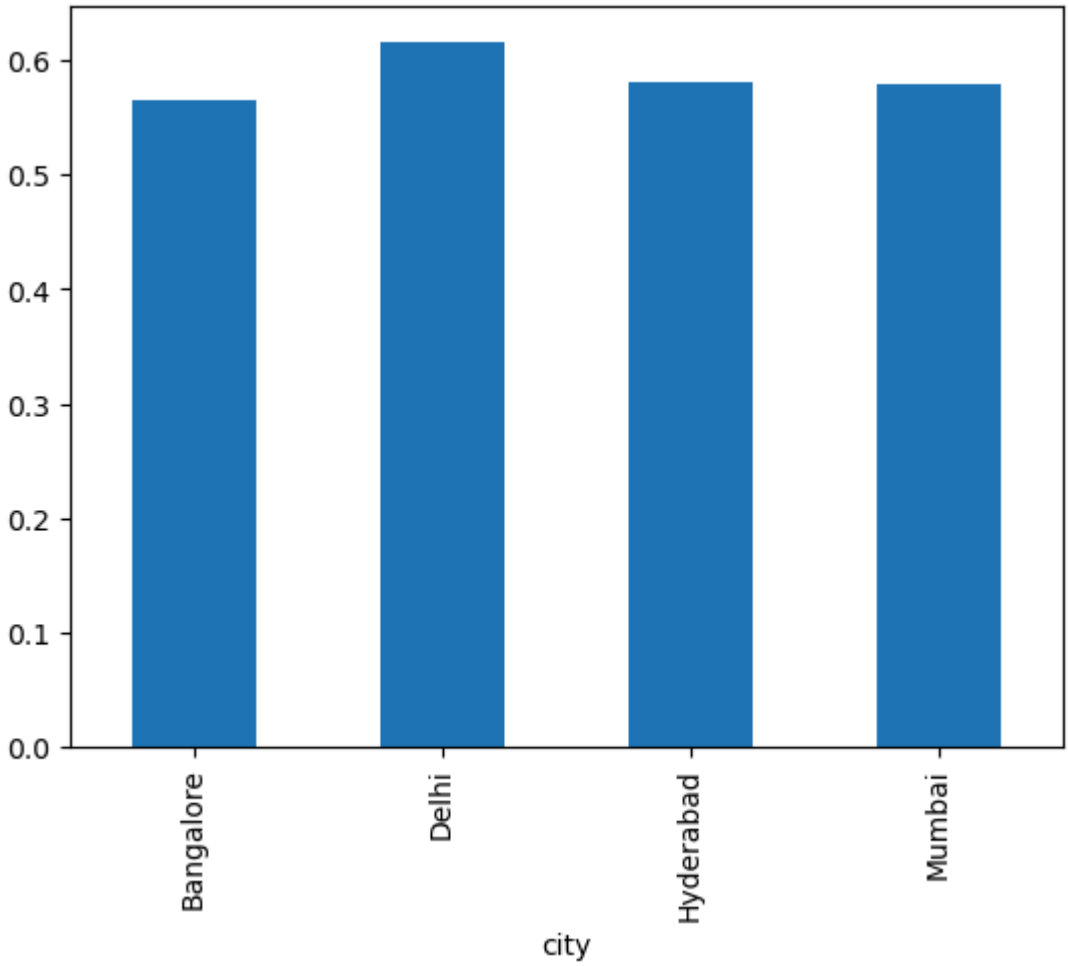
	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	r
0	16559	1-May-22	RT1	25	30.0	83.33	
1	16559	2-May-22	RT1	20	30.0	66.67	
2	16559	3-May-22	RT1	17	30.0	56.67	

In [95]:
```
df.groupby("city")["occ_pct"].mean()
```

Out[95]:
```
city
Bangalore    56.594207
Delhi        61.606467
Hyderabad    58.144651
Mumbai       57.943142
Name: occ_pct, dtype: float64
```

In [75]:
```
df.groupby("city")["occ_pct"].mean().plot(kind="bar")
```

Out[75]:
```
<Axes: xlabel='city'>
```



3. When was the occupancy better? Weekday or Weekend?

In [96]:
```
df_date.head(3)
```

Out[96]:

	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

In [97]:
```
df = pd.merge(df, df_date, left_on="check_in_date", right_on="date")
df.head(3)
```

Out[97]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	r
0	16559	10-May-22	RT1	18	30.0	60.00	
1	16559	10-May-22	RT2	25	41.0	60.98	
2	16559	10-May-22	RT3	20	32.0	62.50	

In [98]:
```
df.groupby("day_type")["occ_pct"].mean().round(2)
```

Out[98]:
```
day_type
weekday    50.90
weekend    72.39
Name: occ_pct, dtype: float64
```

4: In the month of June, what is the occupancy for different cities

In [99]:
```
df_june_22 = df[df["mmm yy"]=="Jun 22"]
df_june_22.head(4)
```

Out[99]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pc
2200	16559	10-Jun-22	RT1	20	30.0	66.67
2201	16559	10-Jun-22	RT2	26	41.0	63.41
2202	16559	10-Jun-22	RT3	20	32.0	62.50
2203	16559	10-Jun-22	RT4	11	18.0	61.11

In [100…
```
df["mmm yy"].unique()
```

Out[100…
```
array(['May 22', 'Jun 22', 'Jul 22'], dtype=object)
```

In [101…
```
df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=Fals
```

Out[101…

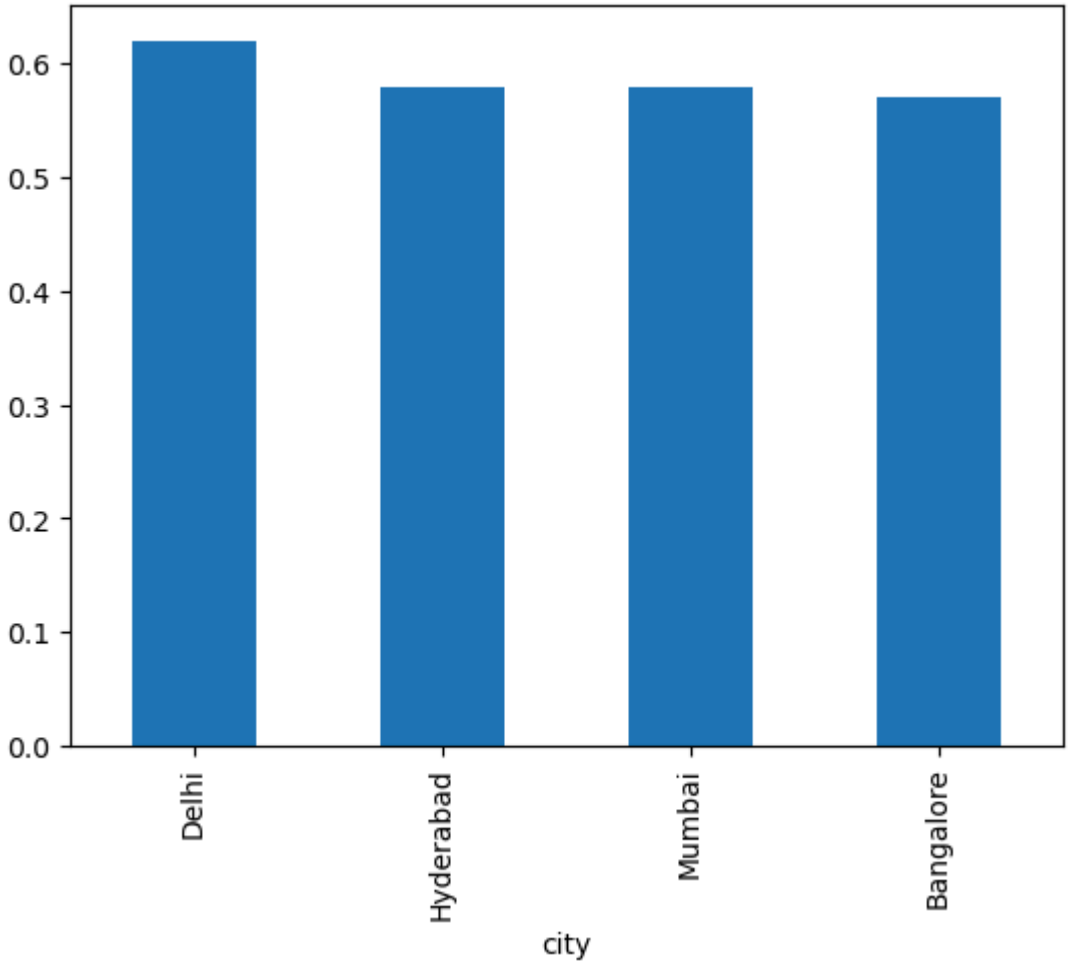
```
city
Delhi        62.47
Hyderabad    58.46
Mumbai       58.38
Bangalore    56.58
Name: occ_pct, dtype: float64
```

In [82]:

```
df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=Fals
```

Out[82]:

```
<Axes: xlabel='city'>
```



5: We got new data for the month of august. Append that to existing data

In [102…

```
df_august = pd.read_csv("datasets/new_data_august.csv")
df_august.head(3)
```

Out[102…

	property_id	property_name	category	city	room_category	room_class	check_
0	16559	Atliq Exotica	Luxury	Mumbai	RT1	Standard	01
1	19562	Atliq Bay	Luxury	Bangalore	RT1	Standard	01
2	19563	Atliq Palace	Business	Bangalore	RT1	Standard	01

In [103…
```
df_august.columns
```

Out[103…
```
Index(['property_id', 'property_name', 'category', 'city', 'room_category',
       'room_class', 'check_in_date', 'mmm yy', 'week no', 'day_type',
       'successful_bookings', 'capacity', 'occ%'],
      dtype='object')
```

In [104…
```
df.columns
```

Out[104…
```
Index(['property_id', 'check_in_date', 'room_category', 'successful_bookings',
       'capacity', 'occ_pct', 'room_class', 'property_name', 'category',
       'city', 'date', 'mmm yy', 'week no', 'day_type'],
      dtype='object')
```

In [105…
```
df_august.shape
```

Out[105…
```
(7, 13)
```

In [106…
```
df.shape
```

Out[106…
```
(6500, 14)
```

In [107…
```
latest_df = pd.concat([df, df_august], ignore_index = True, axis = 0)
latest_df.tail(10)
```

Out[107…

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pc
6497	18560	31-Jul-22	RT2	34	40.0	85.0
6498	18560	31-Jul-22	RT3	17	24.0	70.8
6499	18560	31-Jul-22	RT4	12	15.0	80.0
6500	16559	01-Aug-22	RT1	30	30.0	NaN
6501	19562	01-Aug-22	RT1	21	30.0	NaN
6502	19563	01-Aug-22	RT1	23	30.0	NaN
6503	19558	01-Aug-22	RT1	30	40.0	NaN
6504	19560	01-Aug-22	RT1	20	26.0	NaN
6505	17561	01-Aug-22	RT1	18	26.0	NaN
6506	17564	01-Aug-22	RT1	10	16.0	NaN

In [108…

```
latest_df.shape
```

Out[108…

```
(6507, 15)
```

6. Print revenue realized per city

In [109…

```
df_bookings.head()
```

Out[109…

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_gue
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	
5	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	
6	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	
7	May012216558RT18	16558	26-04-22	1/5/2022	3/5/2022	

In [110…

```
df_hotels.head(3)
```

Out[110…

	property_id	property_name	category	city
0	16558	Atliq Grands	Luxury	Delhi
1	16559	Atliq Exotica	Luxury	Mumbai
2	16560	Atliq City	Business	Delhi

In [111…

```
df_bookings_all = pd.merge(df_bookings, df_hotels, on="property_id")
df_bookings_all.head(3)
```

Out[111…

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_gue
0	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	
1	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	
2	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	

In [112…

```
df_bookings_all.groupby("city")["revenue_realized"].sum()
```

Out[112…

```
city
Bangalore    420383550
Delhi        294404488
Hyderabad    325179310
Mumbai       668569251
Name: revenue_realized, dtype: int64
```

Print revenue realized per hotel type

In [139…

```
df_bookings_all.property_name.unique()
```

Out[139…

```
array(['Atliq Grands', 'Atliq Exotica', 'Atliq City', 'Atliq Blu',
       'Atliq Bay', 'Atliq Palace', 'Atliq Seasons'], dtype=object)
```

In [140…

```
df_bookings_all.groupby("property_name")["revenue_realized"].sum().round(2).sort
```

Out[140…

```
property_name
Atliq Seasons     66086735
Atliq Grands     211462134
Atliq Bay        259996918
Atliq Blu        260851922
Atliq City       285798439
Atliq Palace     304081863
Atliq Exotica    320258588
Name: revenue_realized, dtype: int64
```

Print average rating per city

In [141…

```
df_bookings_all.groupby("city")["ratings_given"].mean().round(2)
```

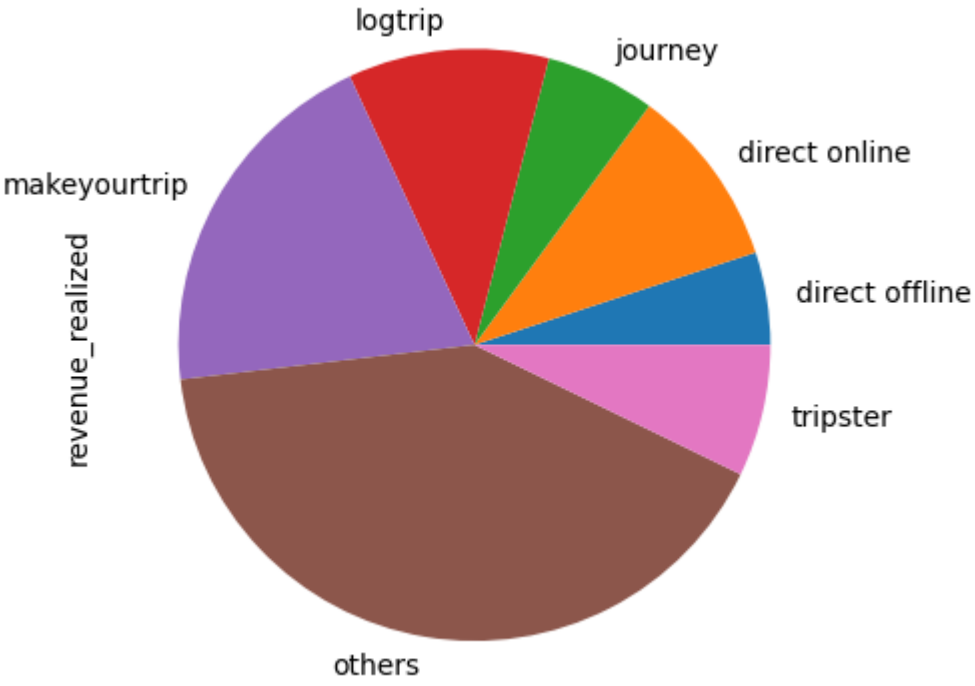
Out[141…

```
city
Bangalore    3.41
Delhi        3.78
Hyderabad    3.66
Mumbai       3.65
Name: ratings_given, dtype: float64
```

Print a pie chart of revenue realized per booking platform

In [142…
```
df_bookings_all.groupby("booking_platform")["revenue_realized"].sum().plot(kind=
```

Out[142…
```
<Axes: ylabel='revenue_realized'>
```



Strategic Insights

- **Mumbai** leads in revenue, contributing **668.6M**, followed by Bangalore, Hyderabad, and Delhi.
- **Luxury** rooms contributed the most (**61.61%),** while Business rooms contributed **38.39%**.
- **Delhi** tops in **occupancy% (60.5%)** and rating (3.8).
- **AtliQ Exotica** earned **320M**, with **57.3% occupancy**, a **3.62 rating**, and **24.37% cancellation**.
- **Weekend occupancy** rose by **7%,** though RevPAR stayed stable.
- **Weekdays** brought in **69.34%** revenue; **weekends 30.66%.**
- **May** was the best month with **581.93M** revenue.
- **Elite rooms** earned the **highest (560M);** Standard rooms the lowest (₹310M).
- Most bookings came from **other sources, 55K** (699M), followed by **MakeYourTrip, 27K** (314M). The lowest number of bookings came from **direct offline, 7K** (86M).

Recommendations

- **Improve Customer Ratings:** Focus on service quality, cleanliness, and food to boost ratings and bookings.

- **Use Dynamic Pricing:** Adjust rates for weekdays/weekends to increase overall revenue.

- **Reduce Cancellation Rates:** Address high cancellations from Others and MakeMyTrip (25%), especially for Elite rooms.

- **Price Based on Occupancy:** Apply dynamic pricing for properties with low occupancy to improve room utilization and revenue.

Dashboard Resources Utilized

Image courtesy:

- Photo by Unsplash
  - Isha Ralhan
  - Arno Senoner
  - Possessed Photography
  - Jon Tyson
- Icon by freepik
- Background by freepik

Thank you for your attention