AtliQ Hotels Data Analysis Project

AtliQ Hotels, a luxury hotel chain in India with locations in Mumbai, Delhi, Hyderabad, and Bangalore, is experiencing a decline in business. To address this issue, they have provided a dataset covering three months from May 2022 to July 2022 for analysis, along with separate data for August 2022.

This notebook aims to analyze the data and deliver insights based on the findings.

- Data Import and Data Exploration
- Data Cleaning
- Data Transformation
- Insights Generation

Importing Necessary Libraries.

In [8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Data Import and Data Exploration

Datasets

We have 5 csv file

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

Load the bookings data into a dataframe.

In [9]:
```
df_bookings = pd.read_csv("datasets/fact_bookings.csv")
df_bookings.head(4)
```

Out[9]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_cate
0	May012216558RT11	16558	27-04-22	1/5/2022	2/5/2022	-3.0	
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	
2	May012216558RT13	16558	28-04-22	1/5/2022	4/5/2022	2.0	
3	May012216558RT14	16558	28-04-22	1/5/2022	2/5/2022	-2.0	

Determining the total count of rows and columns using the `Shape()` function.

In [10]:
```
df_bookings.shape
```

Out[10]:
```
(134590, 12)
```

Lists the distinct room categories found in the bookings data using the `Unique()` function.

In [11]:
```
df_bookings.room_category.unique()
```

Out[11]:
```
array(['RT1', 'RT2', 'RT3', 'RT4'], dtype=object)
```

In [12]:
```
df_bookings.booking_platform.unique()
```

Out[12]:
```
array(['direct online', 'others', 'logtrip', 'tripster', 'makeyourtrip',
       'journey', 'direct offline'], dtype=object)
```

Counts the number of bookings per platform in the bookings dataset using the `Value_counts()` function.

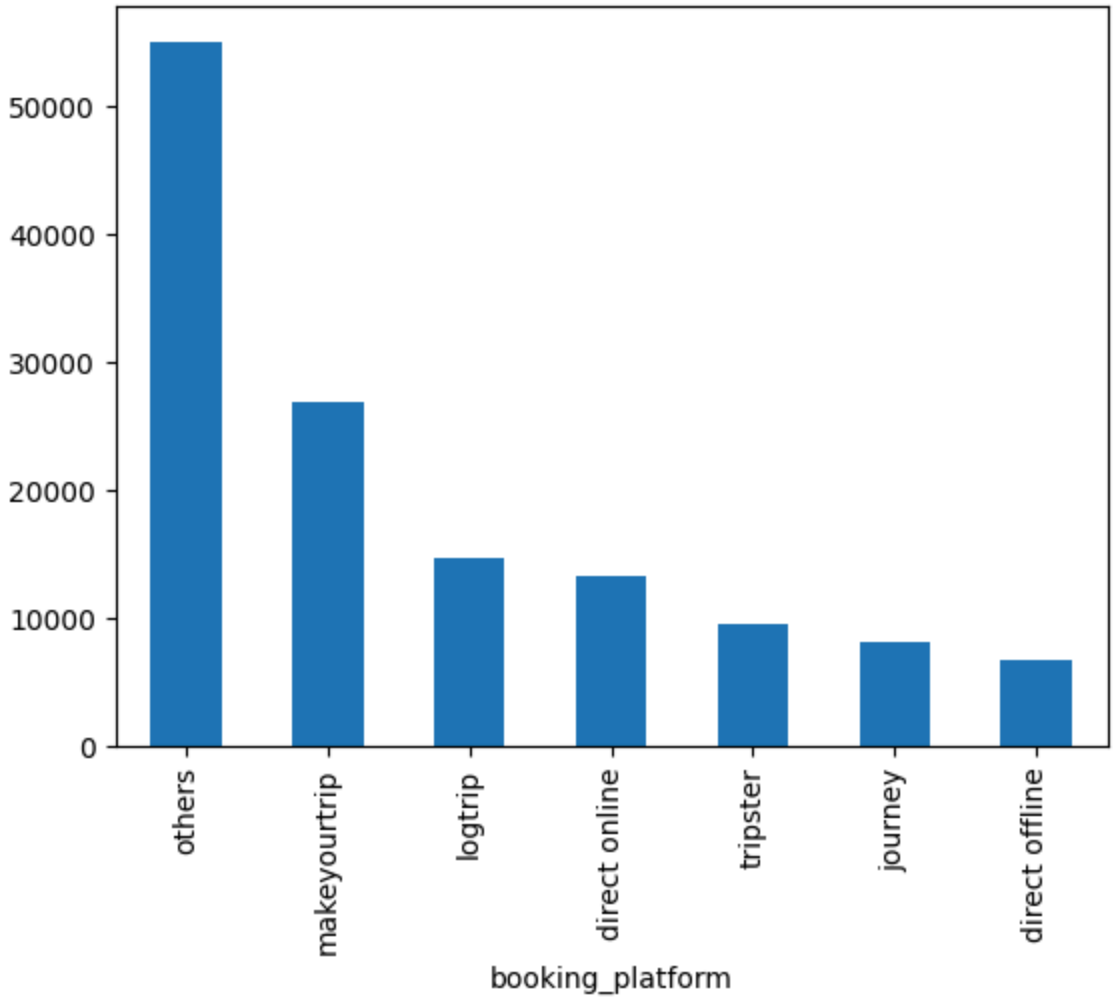
In [13]:
```
df_bookings.booking_platform.value_counts()
```

Out[13]:
```
booking_platform
others            55066
makeyourtrip      26898
logtrip           14756
direct online     13379
tripster           9630
journey            8106
direct offline     6755
Name: count, dtype: int64
```

Generates a bar chart showing the distribution of booking platforms in the dataset using the `Plot()` function.

In [14]:
```
df_bookings.booking_platform.value_counts().plot(kind="bar")
```

Out[14]:
```
<Axes: xlabel='booking_platform'>
```



Provides a summary of descriptive statistics for the bookings dataframe using the `Describe()` *function.*

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

Out[16]:

	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 [17]:
```
df_bookings.revenue_generated.min(),df_bookings.revenue_generated.max()
```

Out[17]:
```
(np.int64(6500), np.int64(28560000))
```

Read the remaining files.

In [19]:
```
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")
df_agg_bookings = pd.read_csv("datasets/fact_aggregated_bookings.csv")
```

Provides the number of rows and columns in the hotels dataframe using the `Shape()` function.

In [20]:
```
df_hotels.shape
```

Out[20]:
```
(25, 4)
```

Displays the first few rows of the hotels dataframe using the `Head()` function.

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

Out[21]:

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

Provides a count of each property_category in the hotels dataframe using the `Value_counts()` function.

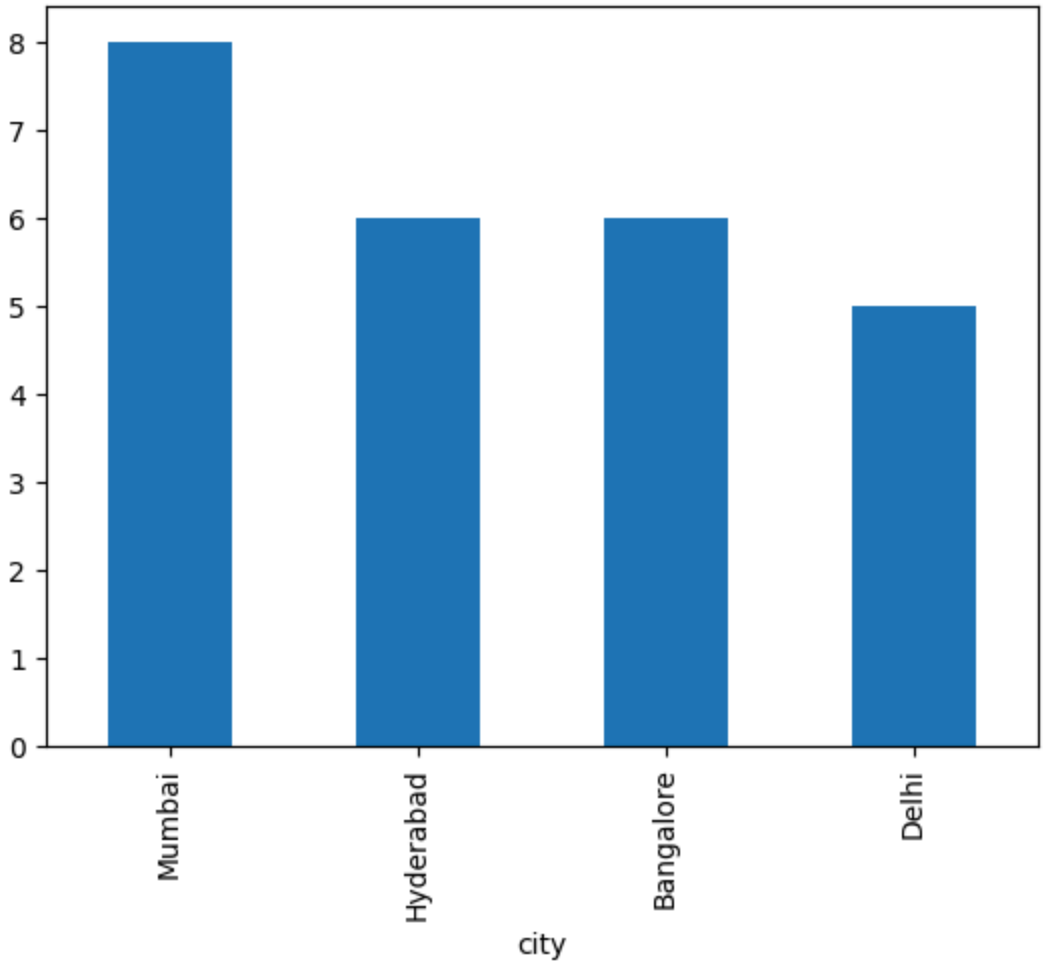
In [22]:
```
df_hotels.category.value_counts()
```

Out[22]:
```
category
Luxury      16
Business     9
Name: count, dtype: int64
```

Plot the number of hotels per city with a bar chart using the `Plot()` function.

In [23]:
```
df_hotels.city.value_counts().plot(kind="bar")
```

Out[23]:
```
<Axes: xlabel='city'>
```



Data Cleaning

Provides a summary of descriptive statistics for the bookings dataframe using the `Describe()` function.

In [24]:
```
df_bookings.describe()
```

Out[24]:

	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

(1) Clean invalid `no_guests`.

Filters bookings where the `number of guests` is less than or equal to zero.

In [25]:
```
df_bookings[df_bookings.no_guests<=0]
```

Out[25]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	roc
0	May012216558RT11	16558	27-04-22	1/5/2022	2/5/2022	-3.0	
3	May012216558RT14	16558	28-04-22	1/5/2022	2/5/2022	-2.0	
17924	May122218559RT44	18559	12/5/2022	12/5/2022	14-05-22	-10.0	
18020	May122218561RT22	18561	8/5/2022	12/5/2022	14-05-22	-12.0	
18119	May122218562RT311	18562	5/5/2022	12/5/2022	17-05-22	-6.0	
18121	May122218562RT313	18562	10/5/2022	12/5/2022	17-05-22	-4.0	
56715	Jun082218562RT12	18562	5/6/2022	8/6/2022	13-06-22	-17.0	
119765	Jul202219560RT220	19560	19-07-22	20-07-22	22-07-22	-1.0	
134586	Jul312217564RT47	17564	30-07-22	31-07-22	1/8/2022	-4.0	

- The rows above indicate data errors.
- Given that less than 0.5% of the total data is invalid guest data, we can disregard them when generating insights.

Filters the bookings dataframe to include entries with more than zero guests.

In [26]:
```
df_bookings = df_bookings[df_bookings.no_guests>0]
```

In [27]:
```
df_bookings.shape
```

Out[27]:
```
(134578, 12)
```

(2) Outlier removal in `revenue generated`.

Calculates the `minimum and maximum` revenue generated in the bookings dataframe.

In [28]:
```
df_bookings.revenue_generated.min(),df_bookings.revenue_generated.max()
```

Out[28]:
```
(np.int64(6500), np.int64(28560000))
```

Calculates and displays the `mean and standard deviation` of revenue generated in the bookings dataframe.

In [29]:
```
avg, std = df_bookings.revenue_generated.mean(),df_bookings.revenue_generated.std()
```

In [30]:
```
avg, std
```

Out[30]:
```
(np.float64(15378.036937686695), np.float64(93040.1549314641))
```

*Calculates the `Upper Limit` using the formula: Higher_limit = avg + 3std.**

In [31]:
```
higher_limit = avg + 3*std
higher_limit
```

Out[31]:
```
np.float64(294498.50173207896)
```

*Calculates the `Lower Limit` using the formula: Lower_limit = avg - 3std.**

In [32]:
```
lower_limit = avg - 3*std
lower_limit
```

Out[32]:
```
np.float64(-263742.4278567056)
```

Filters bookings where revenue generated exceeds a specified higher limit.

In [33]:
```
df_bookings[df_bookings.revenue_generated>higher_limit]
```

Out[33]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	roo
2	May012216558RT13	16558	28-04-22	1/5/2022	4/5/2022	2.0	
111	May012216559RT32	16559	29-04-22	1/5/2022	2/5/2022	6.0	
315	May012216562RT22	16562	28-04-22	1/5/2022	4/5/2022	2.0	
562	May012217559RT118	17559	26-04-22	1/5/2022	2/5/2022	2.0	
129176	Jul282216562RT26	16562	21-07-22	28-07-22	29-07-22	2.0	

- We identified five outliers in the `revenue_generated` column that can be ignored.

Filters the bookings dataframe to include only rows where revenue generated is less than or equal to a specified higher limit, and then displays the shape of the filtered dataframe.

In [34]:
```
df_bookings = df_bookings[df_bookings.revenue_generated<higher_limit]
df_bookings
```

Out[34]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	roor
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	4.0	
5	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	2.0	
6	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	2.0	
7	May012216558RT18	16558	26-04-22	1/5/2022	3/5/2022	2.0	
...	...	...	...	...	...	...	
134584	Jul312217564RT45	17564	30-07-22	31-07-22	1/8/2022	2.0	
134585	Jul312217564RT46	17564	29-07-22	31-07-22	3/8/2022	1.0	
134587	Jul312217564RT48	17564	30-07-22	31-07-22	2/8/2022	1.0	
134588	Jul312217564RT49	17564	29-07-22	31-07-22	1/8/2022	2.0	
134589	Jul312217564RT410	17564	31-07-22	31-07-22	1/8/2022	2.0	

134573 rows × 12 columns

In [35]:
```
df_bookings.shape
```

Out[35]:
```
(134573, 12)
```

Removing outliers in `revenue_realized`.

Generates summary statistics for the `revenue_realized` in the bookings dataframe.

In [36]:
```
df_bookings.revenue_realized.describe()
```

Out[36]:
```
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 [37]:
```
higher_limit =  df_bookings.revenue_realized.mean() + 3*df_bookings.revenue_realized.std()
higher_limit
```

Out[37]:
```
np.float64(33479.358661845814)
```

Displays bookings where revenue realized exceeds the `higher limit.`

In [38]:
```
df_bookings[df_bookings.revenue_realized>higher_limit]
```

Out[38]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	roc
137	May012216559RT41	16559	27-04-22	1/5/2022	7/5/2022	4.0	
139	May012216559RT43	16559	1/5/2022	1/5/2022	2/5/2022	6.0	
143	May012216559RT47	16559	28-04-22	1/5/2022	3/5/2022	3.0	
149	May012216559RT413	16559	24-04-22	1/5/2022	7/5/2022	5.0	
222	May012216560RT45	16560	30-04-22	1/5/2022	3/5/2022	5.0	
...	...	...	...	...	...	...	
134328	Jul312219560RT49	19560	31-07-22	31-07-22	2/8/2022	6.0	
134331	Jul312219560RT412	19560	31-07-22	31-07-22	1/8/2022	6.0	
134467	Jul312219562RT45	19562	28-07-22	31-07-22	1/8/2022	6.0	
134474	Jul312219562RT412	19562	25-07-22	31-07-22	6/8/2022	5.0	
134581	Jul312217564RT42	17564	31-07-22	31-07-22	1/8/2022	4.0	

1299 rows × 12 columns

In [39]:
```
df_rooms
```

Out[39]:

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

Describes the statistical summary of revenue_realized for room category `'RT4'` in the bookings dataframe.

In [40]:
```
df_bookings[df_bookings.room_category=="RT4"].revenue_realized.describe()
```

Out[40]:
```
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
```

Shows the count of `missing values` in each column of the bookings dataframe.

In [43]:
```
df_bookings.isnull().sum()
```

Out[43]:
```
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 dataframe contains a total of **134,573** values, with **77,897** rows having null ratings. Due to the significant number of null ratings, we shouldn't filter or replace them with median or mean values. Because not every customer provides a rating, it is logical that our `ratings_given` column contains null values.

In [ ]:

Data Transformation

In [44]:
```
df_agg_bookings.head()
```

Out[44]:

	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

Create a new column to indicate the `occupancy_percentage`.

In [45]:
```
df_agg_bookings["occ_pct"] = df_agg_bookings["successful_bookings"]/df_agg_bookings["capacity
```

In [46]:
```
df_agg_bookings.head()
```

Out[46]:

	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

Format the column `occupancy_percentage.`

In [47]:

```
df_agg_bookings["occ_pct"] = df_agg_bookings["occ_pct"].apply(lambda x: round(x*100,2))
df_agg_bookings.head(4)
```

Out[47]:

	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
3	17558	1-May-22	RT1	30	19.0	157.89

Insights Generation

Q-1) *What is an average occupancy rate in each of the room categories?*

Calculates the average occupancy percentage for each room category, rounded to two decimal places.

In [48]:

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

Out[48]:

```
room_category
RT1    58.22
RT2    58.04
RT3    58.03
RT4    59.30
Name: occ_pct, dtype: float64
```

In [49]:

```
df_rooms
```

Out[49]:

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

Joins the aggregated bookings data with room details, then previews the first few rows of the combined dataframe.

In [50]:
```
df = pd.merge(df_agg_bookings, df_rooms, left_on="room_category", right_on="room_id")
df.head(4)
```

Out[50]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_id	room_
0	16559	1-May-22	RT1	25	30.0	83.33	RT1	Star
1	19562	1-May-22	RT1	28	30.0	93.33	RT1	Star
2	19563	1-May-22	RT1	23	30.0	76.67	RT1	Star
3	17558	1-May-22	RT1	30	19.0	157.89	RT1	Star

Deletes the `room_id` column from the dataframe `df`.

In [67]:
```
df.drop("room_id",axis=1, inplace=True)
df.head(4)
```

Out[67]:

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

Calculates the average occupancy percentage for each `room_class`, rounded to two decimal places.`

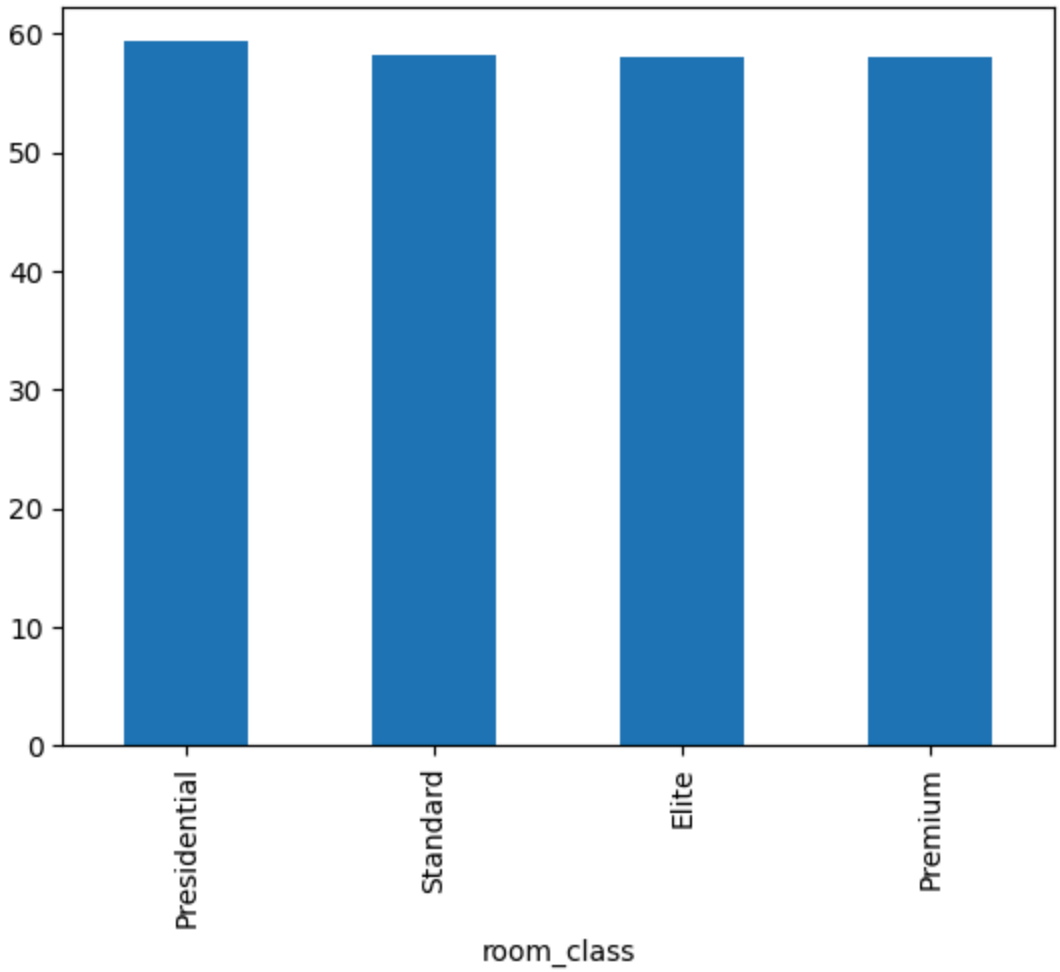
In [51]:
```
df.groupby("room_class")["occ_pct"].mean().round(2)
```

Out[51]:
```
room_class
Elite           58.04
Premium         58.03
Presidential    59.30
Standard        58.22
Name: occ_pct, dtype: float64
```

Plots a bar chart of the average occupancy percentage, for each `room class`.

In [52]:
```
df.groupby('room_class')['occ_pct'].mean().round(2).sort_values(ascending=False).plot(kind='b
```

Out[52]: <Axes: xlabel='room_class'>



Q-2) *Print average occupancy rate per city.*

In [69]:
```
df_hotels.head(3)
```

Out[69]:

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

Joins df and df_hotels on `property_id` and displays the first few rows.

In [54]:
```
df = pd.merge(df, df_hotels, on="property_id")
df.head(3)
```

Out[54]:

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

Calculates the average occupancy percentage for each `city`.

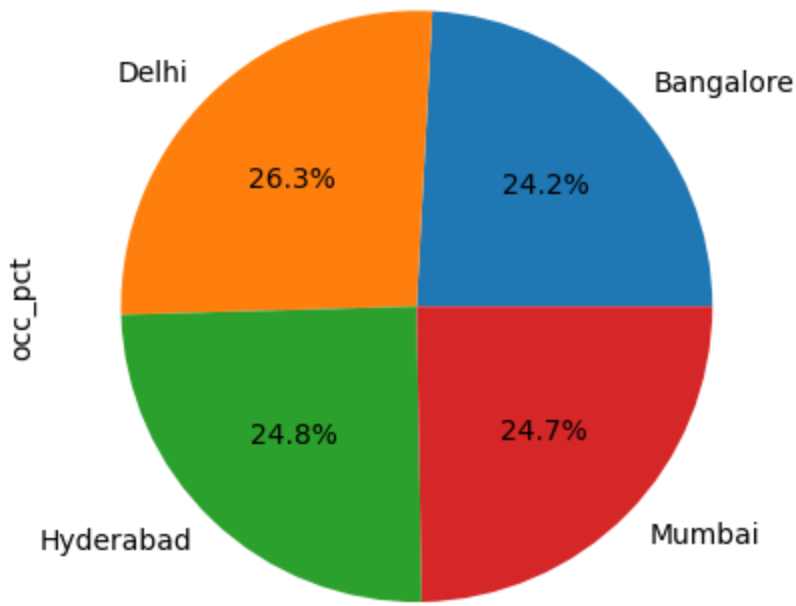
In [55]:
```
df.groupby('city')['occ_pct'].mean()
```

Out[55]:
```
city
Bangalore    56.594207
Delhi        61.606467
Hyderabad    58.144651
Mumbai       57.936305
Name: occ_pct, dtype: float64
```

Plots a pie chart showing the Average occupancy percentage for each `city`.

In [56]:
```
df.groupby('city')['occ_pct'].mean().plot(kind='pie', autopct='%1.1f%%')
```

Out[56]:
```
<Axes: ylabel='occ_pct'>
```



Q-3) *When was the occupancy better? Weekday or Weekend?*

In [61]:
```
df.head(3)
```

Out[61]:

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

In [62]:
```
df_date
```

Out[62]:

	date	mmm yy	week no	day_type
0	2022-05-01	May 22	W 19	weekend
1	2022-05-02	May 22	W 19	weekeday
2	2022-05-03	May 22	W 19	weekeday
3	2022-05-04	May 22	W 19	weekeday
4	2022-05-05	May 22	W 19	weekeday
...	...	...	...	...
87	2022-07-27	Jul 22	W 31	weekeday
88	2022-07-28	Jul 22	W 31	weekeday
89	2022-07-29	Jul 22	W 31	weekeday
90	2022-07-30	Jul 22	W 31	weekend
91	2022-07-31	Jul 22	W 32	weekend

92 rows × 4 columns

In [63]:
```
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
```

- As we can see above, the date column's data type is object, so we need to convert this to datetime data type.

Converts the `date` column to datetime format in the df_date dataframe.

In [64]:
```
df_date['date'] = pd.to_datetime(df_date['date'], format='%d-%b-%y')
```

In [69]:
```
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
```

- Now, as we can see above, the data type of the `date` column has been successfully converted to datetime.

In [70]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9200 entries, 0 to 9199
Data columns (total 11 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             9198 non-null   float64
 5   occ_pct              9198 non-null   float64
 6   room_id              9200 non-null   object
 7   room_class           9200 non-null   object
 8   property_name        9200 non-null   object
 9   category             9200 non-null   object
 10  city                 9200 non-null   object
dtypes: float64(2), int64(2), object(7)
memory usage: 790.8+ KB
```

Converts the `check_in_date` column to datetime format in the df dataframe.

In [71]:
```
df['check_in_date'] = pd.to_datetime(df['check_in_date'], format='%d-%b-%y')
```

In [72]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9200 entries, 0 to 9199
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   property_id          9200 non-null   int64
 1   check_in_date        9200 non-null   datetime64[ns]
 2   room_category        9200 non-null   object
 3   successful_bookings  9200 non-null   int64
 4   capacity             9198 non-null   float64
 5   occ_pct              9198 non-null   float64
 6   room_id              9200 non-null   object
 7   room_class           9200 non-null   object
 8   property_name        9200 non-null   object
 9   category             9200 non-null   object
 10  city                 9200 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(2), object(6)
memory usage: 790.8+ KB
```

- Now, as we can see above, the data type of the `check_in_date` column has been successfully converted to datetime.

Joins df with df_date on the `check_in_date` and date columns.

In [73]:
```
df = pd.merge(df, df_date, left_on="check_in_date", right_on="date")
df.head(4)
```

Out[73]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_id	room_
0	16559	2022-05-01	RT1	25	30.0	83.33	RT1	Star
1	19562	2022-05-01	RT1	28	30.0	93.33	RT1	Star
2	19563	2022-05-01	RT1	23	30.0	76.67	RT1	Star
3	17558	2022-05-01	RT1	30	19.0	157.89	RT1	Star

Calculates the mean occupancy percentage by `day_type`, rounded to two decimal places.

In [74]:
```
df.groupby("day_type")["occ_pct"].mean().round(2)
```

Out[74]:
```
day_type
weekday    51.82
weekend    74.24
Name: occ_pct, dtype: float64
```

Q-4) *In the month of June, what is the occupancy for different cities?*

Filters the dataframe to include data only for `June 2022`.

In [75]:
```
df["mmm yy"].unique()
```

Out[75]:
```
array(['May 22', 'Jun 22', 'Jul 22'], dtype=object)
```

In [76]:
```
df_june_22 = df[df["mmm yy"]=='Jun 22']
df_june_22
```

Out[76]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_id	ro
3100	16559	2022-06-01	RT1	14	30.0	46.67	RT1	
3101	18560	2022-06-01	RT1	18	30.0	60.00	RT1	
3102	19562	2022-06-01	RT1	18	30.0	60.00	RT1	
3103	19563	2022-06-01	RT1	14	30.0	46.67	RT1	
3104	17558	2022-06-01	RT1	8	19.0	42.11	RT1	
...	...	...	...	...	...	...	...	
6095	17562	2022-06-30	RT4	3	6.0	50.00	RT4	Pr
6096	19563	2022-06-30	RT4	3	6.0	50.00	RT4	Pr
6097	16560	2022-06-30	RT4	3	7.0	42.86	RT4	Pr
6098	19558	2022-06-30	RT4	3	7.0	42.86	RT4	Pr
6099	17561	2022-06-30	RT4	3	4.0	75.00	RT4	Pr

3000 rows × 15 columns

Calculates and sorts the average occupancy percentages by `city` in descending order.

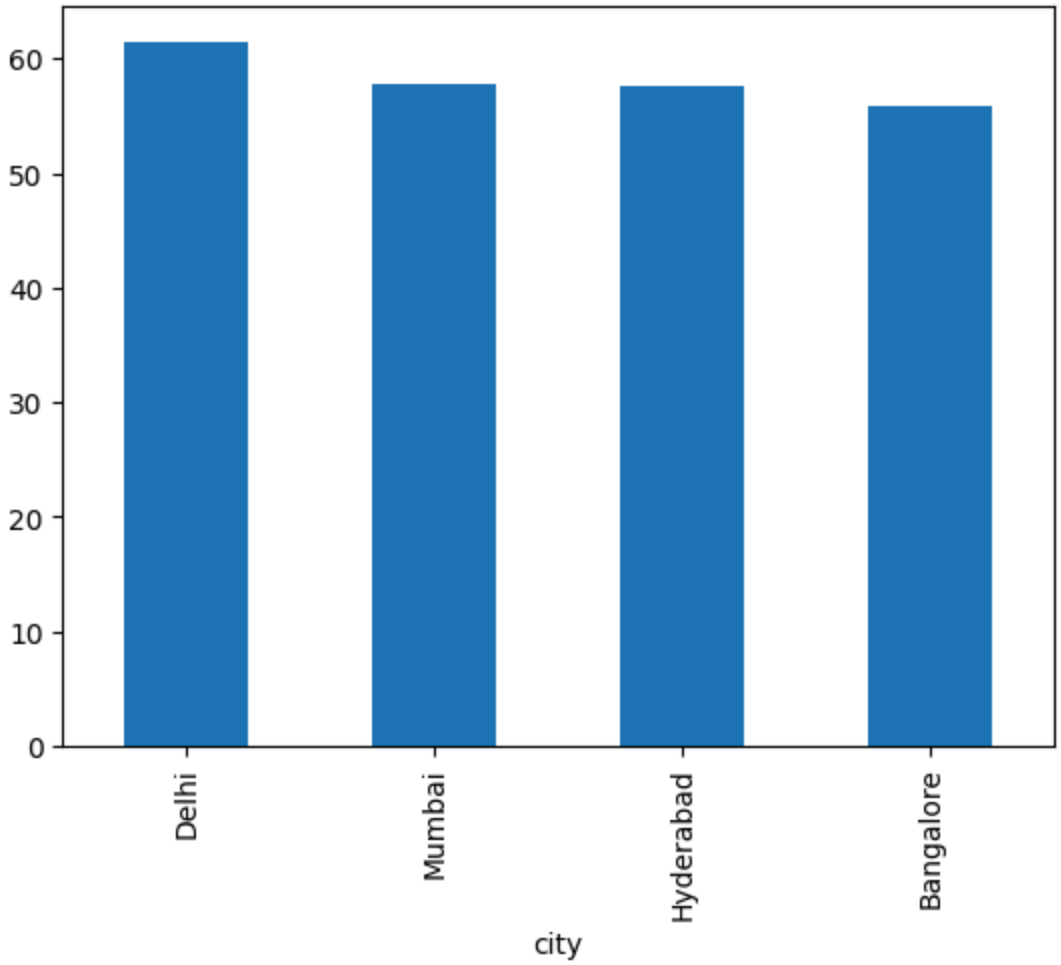
In [77]:
```
df_june_22.groupby("city")["occ_pct"].mean().round(2).sort_values(ascending=False)
```

Out[77]:
```
city
Delhi        61.46
Mumbai       57.79
Hyderabad    57.69
Bangalore    55.95
Name: occ_pct, dtype: float64
```

Generates a bar plot of the average occupancy percentage by city, sorted in descending order, using June 2022 data.

In [78]:
```
df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=False).plot(kind=
```

Out[78]:
```
<Axes: xlabel='city'>
```



Q-5) *We have received new data for the month of* `August` *Append that to the existing data.*

In [79]:
```
df_august = pd.read_csv("datasets/new_data_august.csv")
df_august.head(3)
```

Out[79]:

	property_id	property_name	category	city	room_category	room_class	check_in_date	mmm yy
0	16559	Atliq Exotica	Luxury	Mumbai	RT1	Standard	01-Aug-22	Aug-22
1	19562	Atliq Bay	Luxury	Bangalore	RT1	Standard	01-Aug-22	Aug-22
2	19563	Atliq Palace	Business	Bangalore	RT1	Standard	01-Aug-22	Aug-22

In [80]:
```
df_august.columns
```

Out[80]:
```
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 [81]:
```
df.columns
```

Out[81]:
```
Index(['property_id', 'check_in_date', 'room_category', 'successful_bookings',
       'capacity', 'occ_pct', 'room_id', 'room_class', 'property_name',
       'category', 'city', 'date', 'mmm yy', 'week no', 'day_type'],
      dtype='object')
```

In [82]:
```
df_august.shape
```

Out[82]:
```
(7, 13)
```

In [83]:
```
df.shape
```

Out[83]:
```
(9200, 15)
```

Concatenates 'df' and 'df_august' into `latest_df`, resetting the index to maintain continuity.

In [84]:
```
latest_df = pd.concat([df, df_august], ignore_index=True, axis=0)
latest_df.tail(10)
```

Out[84]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_id	ro
9197	17558	2022-07-31 00:00:00	RT4	3	6.0	50.0	RT4	Pr
9198	19563	2022-07-31 00:00:00	RT4	3	6.0	50.0	RT4	Pr
9199	17561	2022-07-31 00:00:00	RT4	3	4.0	75.0	RT4	Pr
9200	16559	01-Aug-22	RT1	30	30.0	NaN	NaN	
9201	19562	01-Aug-22	RT1	21	30.0	NaN	NaN	
9202	19563	01-Aug-22	RT1	23	30.0	NaN	NaN	
9203	19558	01-Aug-22	RT1	30	40.0	NaN	NaN	
9204	19560	01-Aug-22	RT1	20	26.0	NaN	NaN	
9205	17561	01-Aug-22	RT1	18	26.0	NaN	NaN	
9206	17564	01-Aug-22	RT1	10	16.0	NaN	NaN	

In [85]:
```
latest_df.shape
```

Out[85]:
```
(9207, 16)
```

Q-6) *Print revenue realized per city.*

In [86]:
```
df_bookings.head(4)
```

Out[86]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_cate
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	4.0	
5	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	2.0	
6	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	2.0	

In [87]:
```
df_hotels.head(4)
```

Out[87]:

	property_id	property_name	category	city
0	16558	Atliq Grands	Luxury	Delhi
1	16559	Atliq Exotica	Luxury	Mumbai
2	16560	Atliq City	Business	Delhi
3	16561	Atliq Blu	Luxury	Delhi

Merges the df_bookings and df_hotels dataframes on `property_id` to create `df_bookings_all.`

In [88]:
```
df_bookings_all = pd.merge(df_bookings, df_hotels, on="property_id")
df_bookings_all.head(4)
```

Out[88]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_cate
0	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	
1	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	4.0	
2	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	2.0	
3	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	2.0	

Calculates and sorts the total `revenue_realized` by `city` in descending order from the df_bookings_all dataset.

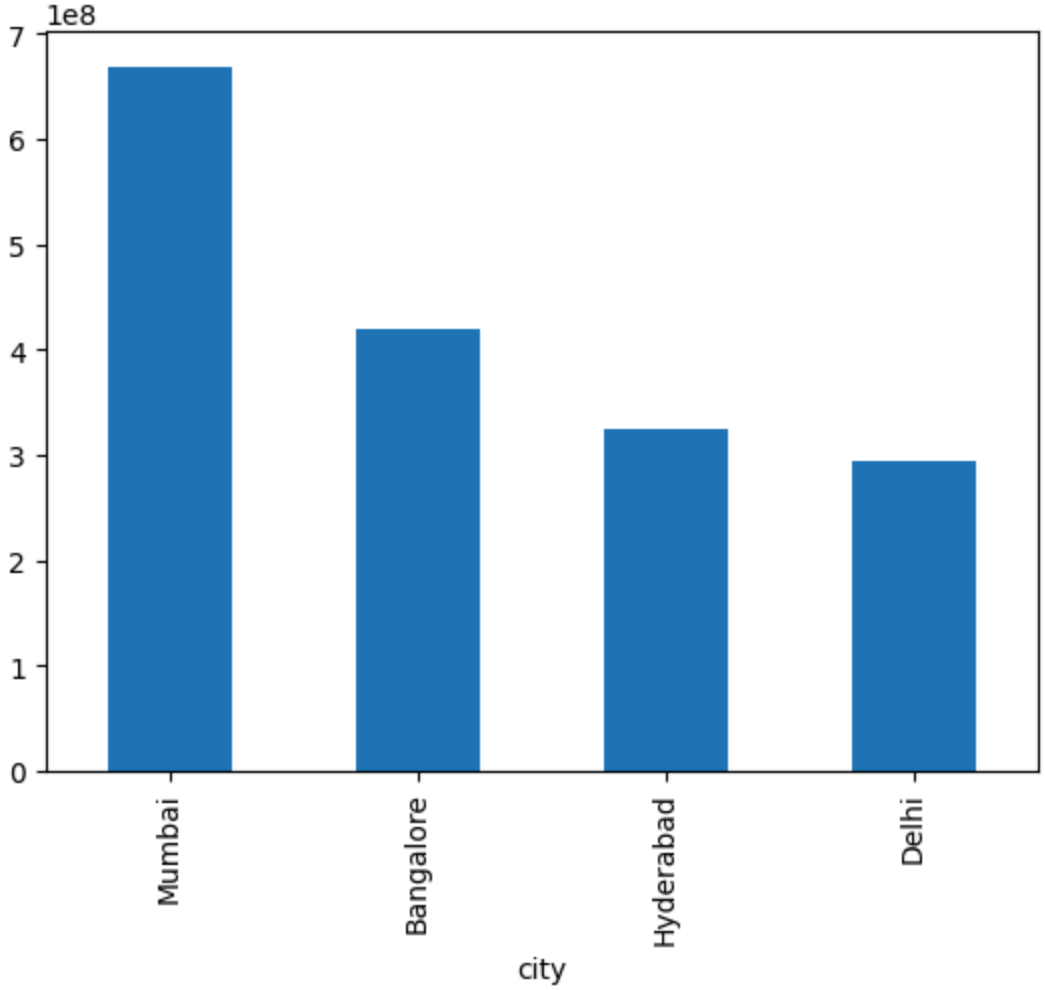
In [89]:
```
df_bookings_all.groupby("city")["revenue_realized"].sum().round(2).sort_values(ascending=Fals
```

Out[89]:
```
city
Mumbai       668569251
Bangalore    420383550
Hyderabad    325179310
Delhi        294404488
Name: revenue_realized, dtype: int64
```

Plots a bar chart showing the total revenue_realized by `city.`

In [90]:
```
df_bookings_all.groupby('city')['revenue_realized'].sum().sort_values(ascending=False).plot(k
```

Out[90]: <Axes: xlabel='city'>



Q-7) *Print month by month revenue.*

In [91]:
```
df_bookings_all.head(4)
```

Out[91]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_cate
0	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	
1	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	4.0	
2	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	2.0	
3	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	2.0	

In [92]:
```
df_date["mmm yy"].unique()
```

Out[92]: array(['May 22', 'Jun 22', 'Jul 22'], dtype=object)

In [93]:
```
df_date.head(4)
```

Out[93]:

	date	mmm yy	week no	day_type
0	2022-05-01	May 22	W 19	weekend
1	2022-05-02	May 22	W 19	weekeday
2	2022-05-03	May 22	W 19	weekeday
3	2022-05-04	May 22	W 19	weekeday

In [96]:
```
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 [97]:
```
df_bookings_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134573 entries, 0 to 134572
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   booking_id         134573 non-null  object
 1   property_id        134573 non-null  int64
 2   booking_date       134573 non-null  object
 3   check_in_date      134573 non-null  object
 4   checkout_date      134573 non-null  object
 5   no_guests          134573 non-null  float64
 6   room_category      134573 non-null  object
 7   booking_platform   134573 non-null  object
 8   ratings_given      56676 non-null   float64
 9   booking_status     134573 non-null  object
 10  revenue_generated  134573 non-null  int64
 11  revenue_realized   134573 non-null  int64
 12  property_name      134573 non-null  object
 13  category           134573 non-null  object
 14  city               134573 non-null  object
dtypes: float64(2), int64(3), object(10)
memory usage: 15.4+ MB
```

- As we can see above, the check_in_date column's data type is an `object`, so we need to convert this to a `datetime` data type.

Converts the `check_in_date` column in df_bookings_all to `datetime` format, handling errors by coercing invalid dates.

In [98]:
```
df_bookings_all["check_in_date"] = pd.to_datetime(df_bookings_all["check_in_date"], errors='c
```

In [99]:
```
df_bookings_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134573 entries, 0 to 134572
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   booking_id         134573 non-null  object
 1   property_id        134573 non-null  int64
 2   booking_date       134573 non-null  object
 3   check_in_date      55790 non-null   datetime64[ns]
 4   checkout_date      134573 non-null  object
 5   no_guests          134573 non-null  float64
 6   room_category      134573 non-null  object
 7   booking_platform   134573 non-null  object
 8   ratings_given      56676 non-null   float64
 9   booking_status     134573 non-null  object
 10  revenue_generated  134573 non-null  int64
 11  revenue_realized   134573 non-null  int64
 12  property_name      134573 non-null  object
 13  category           134573 non-null  object
 14  city               134573 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(3), object(9)
memory usage: 15.4+ MB
```

- Now, as we can see above, the data type of the `check_in_date` column has been successfully converted to datetime.

Joins the bookings dataframe df_bookings_all with the date dataframe df_date using the `check_in_date` and `date` columns.

In [100…

```
df_bookings_all = pd.merge(df_bookings_all, df_date, left_on='check_in_date', right_on='date'
df_bookings_all.head()
```

Out[100…

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_cate
0	May052216558RT11	16558	15-04-22	2022-05-05	7/5/2022	3.0	
1	May052216558RT12	16558	30-04-22	2022-05-05	7/5/2022	2.0	
2	May052216558RT13	16558	1/5/2022	2022-05-05	6/5/2022	3.0	
3	May052216558RT14	16558	3/5/2022	2022-05-05	6/5/2022	2.0	
4	May052216558RT15	16558	30-04-22	2022-05-05	10/5/2022	4.0	

Calculates the aggregate `revenue_realized` for `each month`.

In [101…

```
df_bookings_all.groupby(['mmm yy'])['revenue_realized'].sum()
```

Out[101…

```
mmm yy
Jul 22    60278496
Jun 22    52903014
May 22    60961428
Name: revenue_realized, dtype: int64
```

Q-8) *Print revenue realized per hotel type.*

In [102…

```
df_bookings_all['property_name'].unique()
```

Out[102…

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

Displays the total `revenue_realized` for each `property`, sorted in descending order.

In [103…

```
df_bookings_all.groupby('property_name')['revenue_realized'].sum().sort_values(ascending=Fals
```

Out[103…

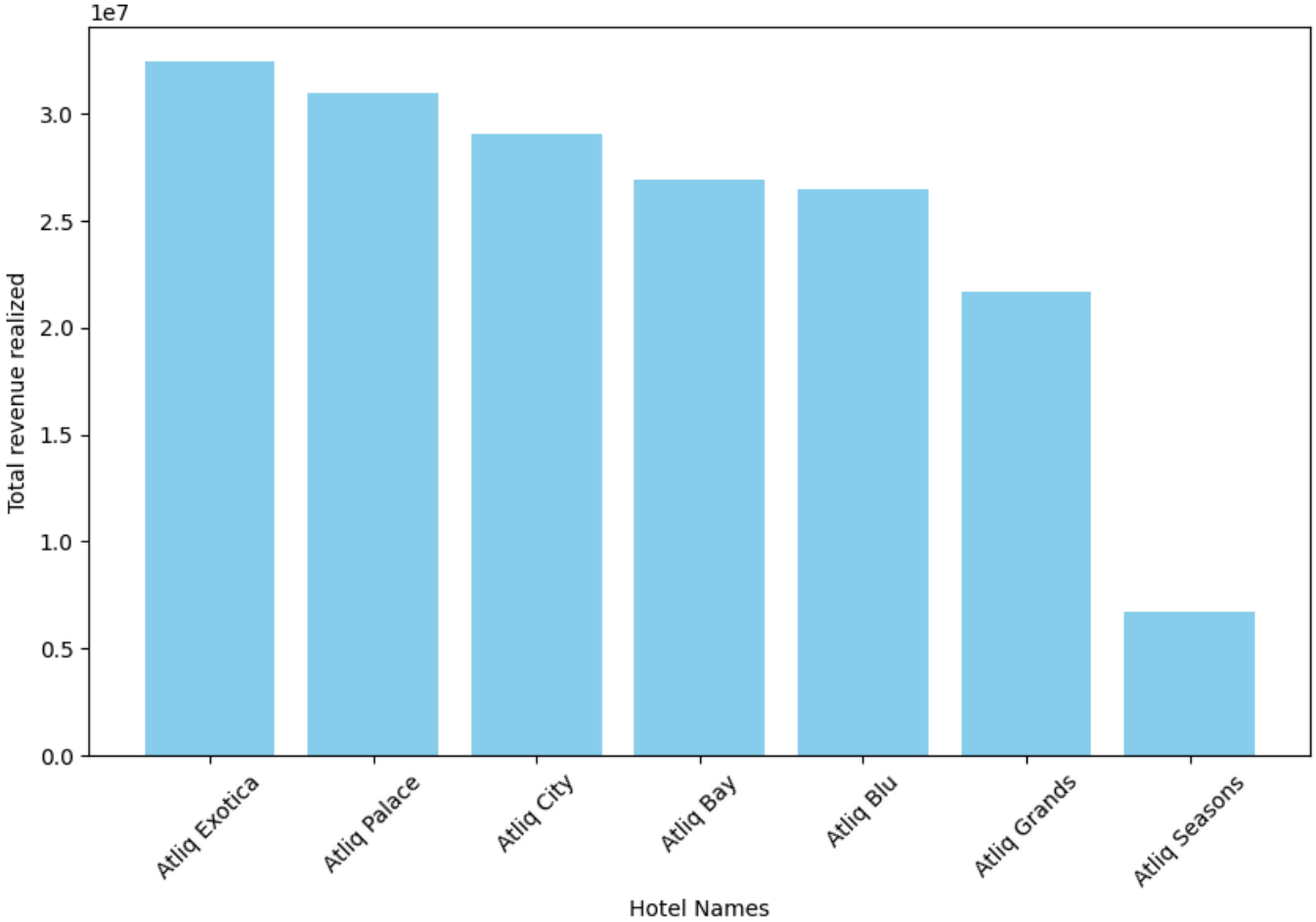
```
property_name
Atliq Exotica    32436799
Atliq Palace     30945855
Atliq City       29047727
Atliq Bay        26936115
Atliq Blu        26459751
Atliq Grands     21644446
Atliq Seasons     6672245
Name: revenue_realized, dtype: int64
```

Plots the total revenue_realized by property name in a bar chart.

In [104…

```
hotels = ['Atliq Exotica', 'Atliq Palace', 'Atliq City', 'Atliq Bay', 'Atliq Blu', 'Atliq Gra
revenue_realized = [32436799, 30945855, 29047727, 26936115, 26459751, 21644446, 6672245]

# Creating the bar chart
plt.figure(figsize=(10, 6))
plt.bar(hotels, revenue_realized, color='skyblue')
plt.xlabel('Hotel Names')
plt.ylabel('Total revenue realized')
plt.xticks(rotation=45)
plt.show()
```



Q-9) *Print average rating per city.*

In [105…
```
df_bookings_all.head()
```

Out[105…

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_cate
0	May052216558RT11	16558	15-04-22	2022-05-05	7/5/2022	3.0	
1	May052216558RT12	16558	30-04-22	2022-05-05	7/5/2022	2.0	
2	May052216558RT13	16558	1/5/2022	2022-05-05	6/5/2022	3.0	
3	May052216558RT14	16558	3/5/2022	2022-05-05	6/5/2022	2.0	
4	May052216558RT15	16558	30-04-22	2022-05-05	10/5/2022	4.0	

Calculates the Average ratings by `city`, rounds to two decimal places, and ranks them from highest to lowest.

In [106…
```
df_bookings_all.groupby('city')['ratings_given'].mean().round(2).sort_values(ascending=False)
```

Out[106…

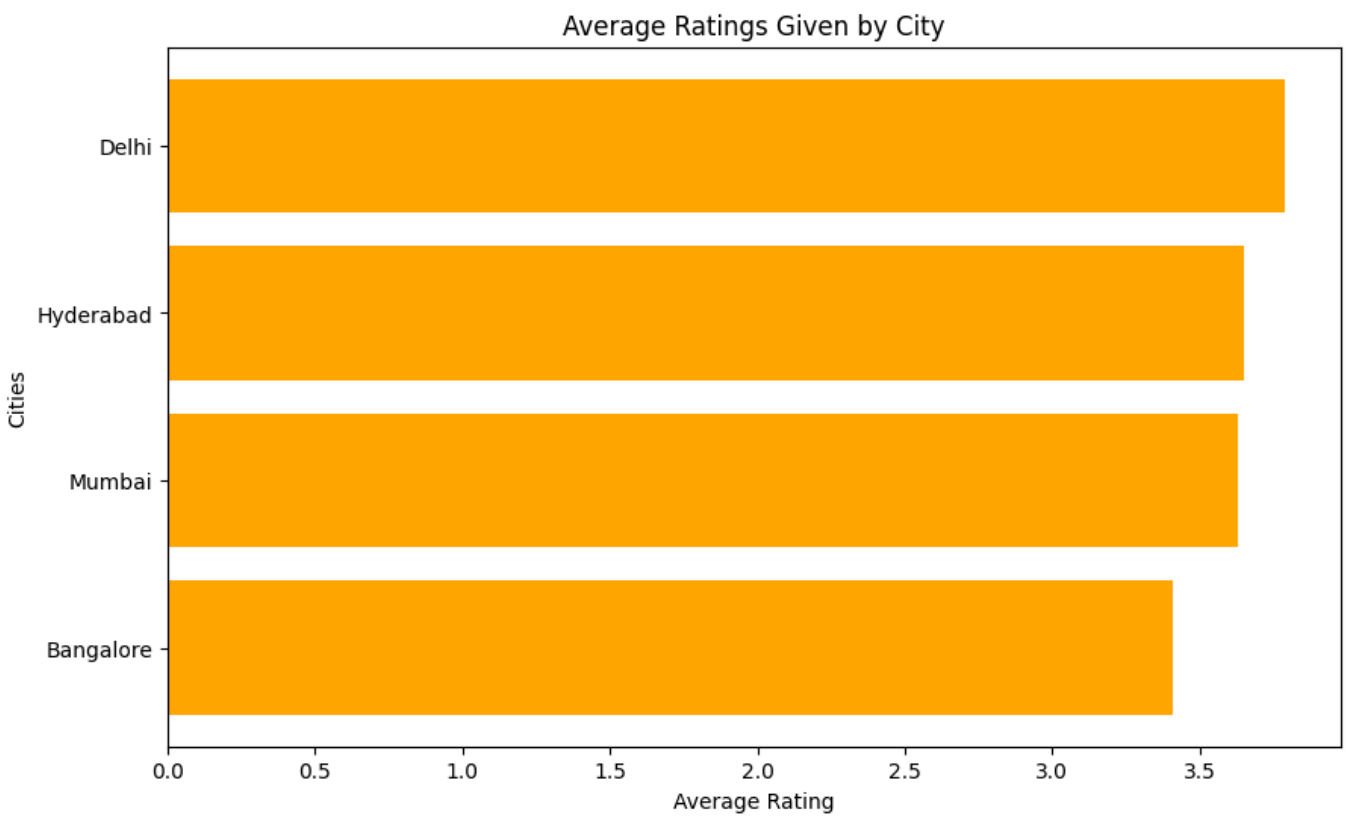
```
city
Delhi        3.79
Hyderabad    3.65
Mumbai       3.63
Bangalore    3.41
Name: ratings_given, dtype: float64
```

Plots the `Average ratings` given by `Cities` in a horizontal bar chart.

In [111…

```python
# Data
cities = ['Delhi', 'Hyderabad', 'Mumbai', 'Bangalore']
ratings = [3.79, 3.65, 3.63, 3.41]

# Creating the horizontal bar chart
plt.figure(figsize=(10, 6))
plt.barh(cities, ratings, color='orange')
plt.xlabel('Average Rating')
plt.ylabel('Cities')
plt.title('Average Ratings Given by City')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest rating at the top
plt.show()
```



Q-10) *Print a pie chart of revenue realized per booking platform.*

In [108…

```python
df_bookings_all.groupby('booking_platform')['revenue_realized'].sum().sort_values(ascending=F
```

Out[108…

```
booking_platform
others            72310965
makeyourtrip      34034257
logtrip           18605339
direct online     17488976
tripster          11959078
journey           10757858
direct offline     8986465
Name: revenue_realized, dtype: int64
```

Visualizes the total `revenue realized` by each `booking platform` in a pie chart.

In [109…

```
# Data
booking_platforms = ['direct offline', 'direct online', 'journey', 'logtrip', 'makeyourtrip',
revenue_realized = [8986465, 17488976, 10757858, 18605339, 34034257, 72310965, 11959078]

# Sorting the data in ascending order
sorted_indices = np.argsort(revenue_realized)
sorted_booking_platforms = np.array(booking_platforms)[sorted_indices]
sorted_revenue_realized = np.array(revenue_realized)[sorted_indices]

# Creating the pie chart
plt.figure(figsize=(10, 6))
plt.pie(sorted_revenue_realized, labels=sorted_booking_platforms, autopct='%1.1f%%', startang
plt.ylabel('revenue_realized')
plt.title('Revenue Realized by Booking Platform')
plt.axis('equal')
plt.show()
```