



AtliQ Hotels 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 datagrame

```
In [2]: df_bookings = pd.read_csv('datasets/fact_bookings.csv')
```

Explore bookings data

```
In [3]: df_bookings.head()
```

```
Out[3]:
```

	booking_id	property_id	booking_date	check_in_date	checkout_date
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 [4]: df_bookings.shape
```

```
Out[4]: (134590, 12)
```

```
In [5]: df_bookings.room_category.unique()
```

```
Out[5]: array(['RT1', 'RT2', 'RT3', 'RT4'], dtype=object)
```

```
In [6]: df_bookings.booking_platform.unique()
```

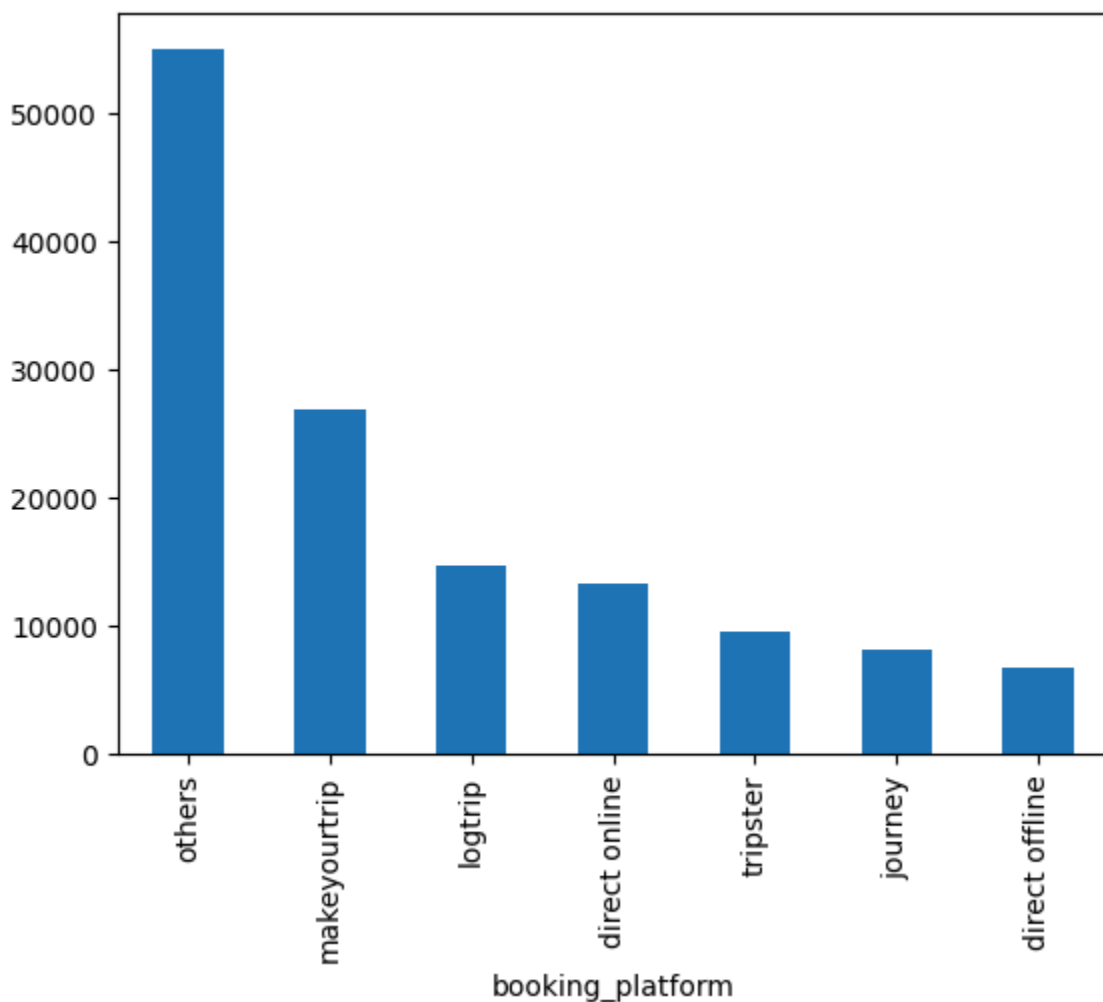
```
Out[6]: array(['direct online', 'others', 'logtrip', 'tripster', 'makeyourtrip',  
              'journey', 'direct offline'], dtype=object)
```

```
In [7]: df_bookings.booking_platform.value_counts()
```

```
Out[7]: booking_platform  
others          55066  
makeyourtrip    26898  
logtrip         14756  
direct online   13379  
tripster        9630  
journey         8106  
direct offline  6755  
Name: count, dtype: int64
```

```
In [8]: df_bookings.booking_platform.value_counts().plot(kind="bar")
```

```
Out[8]: <Axes: xlabel='booking_platform'>
```



```
In [9]: df_bookings.describe()
```

```
Out[9]:
```

	property_id	no_guests	ratings_given	revenue_generated	revenue
count	134590.000000	134587.000000	56683.000000	1.345900e+05	1345
mean	18061.113493	2.036170	3.619004	1.537805e+04	126
std	1093.055847	1.034885	1.235009	9.303604e+04	69
min	16558.000000	-17.000000	1.000000	6.500000e+03	26
25%	17558.000000	1.000000	3.000000	9.900000e+03	76
50%	17564.000000	2.000000	4.000000	1.350000e+04	117
75%	18563.000000	2.000000	5.000000	1.800000e+04	153
max	19563.000000	6.000000	5.000000	2.856000e+07	452

Read rest of the files

```
In [10]: 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')
```

```
In [11]: df_hotels.shape
```

```
Out[11]: (25, 4)
```

```
In [12]: df_hotels.head(3)
```

```
Out[12]:
```

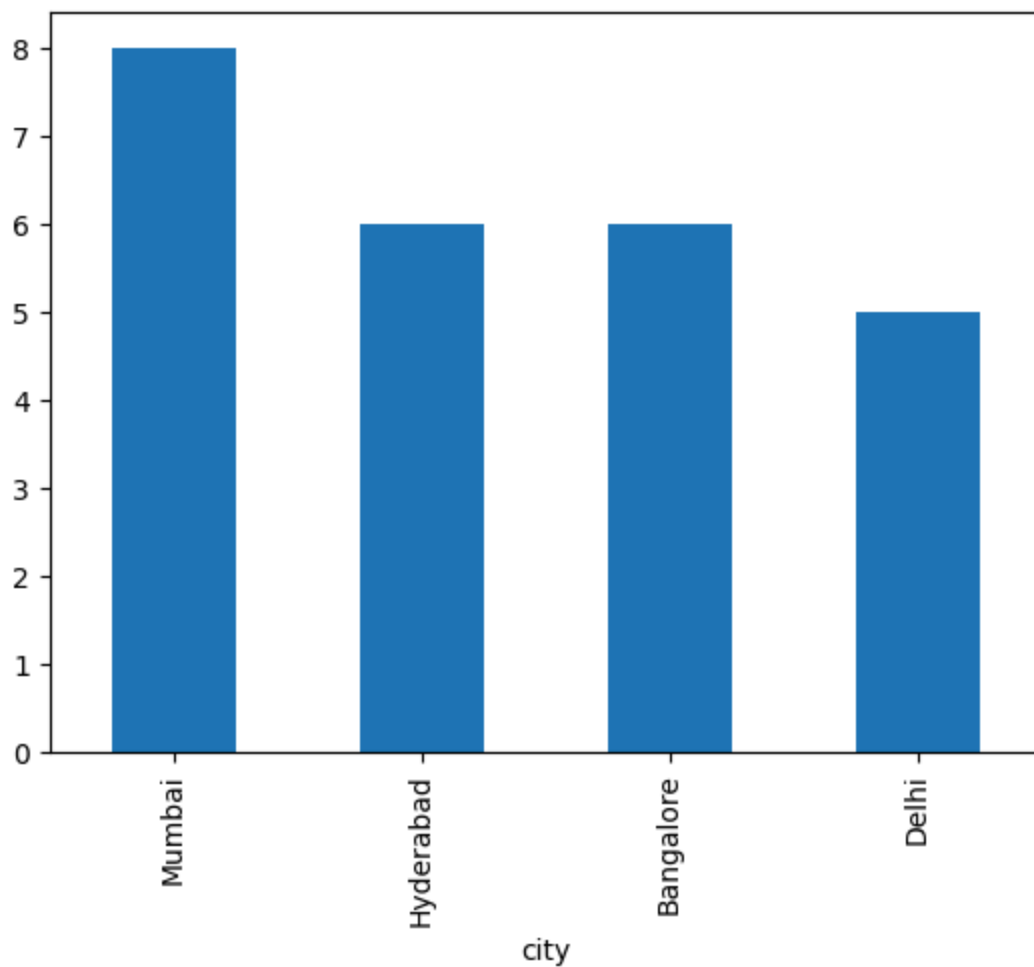
	property_id	property_name	category	city
0	16558	Atliq Grands	Luxury	Delhi
1	16559	Atliq Exotica	Luxury	Mumbai
2	16560	Atliq City	Business	Delhi

```
In [13]: df_hotels.category.value_counts()
```

```
Out[13]: category
Luxury      16
Business     9
Name: count, dtype: int64
```

```
In [14]: df_hotels.city.value_counts().plot(kind="bar")
```

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



Exercise: Explore aggregate bookings

```
In [15]: df_agg_bookings.head(3)
```

```
Out[15]:
```

	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

Exercise-1. Find out unique property ids in aggregate bookings dataset

```
In [16]: df_agg_bookings.property_id.unique()
```

```
Out[16]: 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])
```

Exercise-2. Find out total bookings per property_id

```
In [17]: df_agg_bookings.groupby("property_id")["successful_bookings"].sum()
```

```
Out[17]: 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
```

Exercise-3. Find out days on which bookings are greater than capacity

```
In [18]: df_agg_bookings[df_agg_bookings.successful_bookings>df_agg_bookings.capacity]
```

```
Out[18]:
```

	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

Exercise-4. Find out properties that have highest capacity

```
In [19]: df_agg_bookings.capacity.max()
```

Out[19]: np.float64(50.0)

```
In [20]: df_agg_bookings[df_agg_bookings.capacity==df_agg_bookings.capacity.max()]
```

Out[20]:

	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 [21]: df_bookings.describe()
```

Out[21]:

	property_id	no_guests	ratings_given	revenue_generated	revenue
count	134590.000000	134587.000000	56683.000000	1.345900e+05	1345
mean	18061.113493	2.036170	3.619004	1.537805e+04	126
std	1093.055847	1.034885	1.235009	9.303604e+04	69
min	16558.000000	-17.000000	1.000000	6.500000e+03	26
25%	17558.000000	1.000000	3.000000	9.900000e+03	76
50%	17564.000000	2.000000	4.000000	1.350000e+04	117
75%	18563.000000	2.000000	5.000000	1.800000e+04	153
max	19563.000000	6.000000	5.000000	2.856000e+07	452

(1) Clean invalid guests

```
In [22]: df_bookings[df_bookings.no_guests<=0]
```

```
Out[22]:
```

	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	14/5/2022
18020	May122218561RT22	18561	8/5/2022	12/5/2022	14/5/2022
18119	May122218562RT311	18562	5/5/2022	12/5/2022	17/5/2022
18121	May122218562RT313	18562	10/5/2022	12/5/2022	17/5/2022
56715	Jun082218562RT12	18562	5/6/2022	8/6/2022	13/6/2022
119765	Jul202219560RT220	19560	19-07-22	20-07-22	22/07-22
134586	Jul312217564RT47	17564	30-07-22	31-07-22	1/8/2022

As you can see above, number of guests having less than zero value represents data error. We can ignore these records.

```
In [23]: df_bookings = df_bookings[df_bookings.no_guests>0]
```

```
In [24]: df_bookings.shape
```

```
Out[24]: (134578, 12)
```

(2) Outlier removal in revenue generated

```
In [25]: df_bookings.revenue_generated.min(), df_bookings.revenue_generated.max()
```

```
Out[25]: (np.int64(6500), np.int64(28560000))
```

```
In [26]: df_bookings.revenue_generated.mean(), df_bookings.revenue_generated.median()
```

```
Out[26]: (np.float64(15378.036937686695), np.float64(13500.0))
```

```
In [27]: avg, std = df_bookings.revenue_generated.mean(), df_bookings.revenue_generated.std()
```

```
In [28]: higher_limit = avg + 3*std  
higher_limit
```

```
Out[28]: np.float64(294498.50173207896)
```

```
In [29]: lower_limit = avg - 3*std  
lower_limit
```

```
Out[29]: np.float64(-263742.4278567056)
```

```
In [30]: df_bookings[df_bookings.revenue_generated<=0]
```

```
Out[30]:
```

booking_id	property_id	booking_date	check_in_date	checkout_date	no_gues
------------	-------------	--------------	---------------	---------------	---------

```
In [31]: df_bookings[df_bookings.revenue_generated>higher_limit]
```

```
Out[31]:
```

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

```
In [32]: df_bookings = df_bookings[df_bookings.revenue_generated<=higher_limit]  
df_bookings.shape
```

```
Out[32]: (134573, 12)
```

```
In [33]: df_bookings.revenue_realized.describe()
```

```
Out[33]: 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 [34]: higher_limit = df_bookings.revenue_realized.mean() + 3*df_bookings.revenue_realized.std()  
higher_limit
```

```
Out[34]: np.float64(33479.358661845814)
```

```
In [35]: df_bookings[df_bookings.revenue_realized>higher_limit]
```



```
Out[35]:
```

	booking_id	property_id	booking_date	check_in_date	checkout
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 × 6 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 [36]: df_bookings[df_bookings.room_category=="RT4"].revenue_realized.describe()
```

```
Out[36]: 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 [37]: # mean + 3*standard deviation
23439+3*9048
```

```
Out[37]: 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 [38]: df_bookings[df_bookings.booking_id=="May012216558RT213"]
```

```
Out[38]: booking_id property_id booking_date check_in_date checkout_date no_gues
```

```
In [39]: df_bookings.isnull().sum()
```

```
Out[39]: 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
```

Total values in our dataframe is 134576. Out of that 77899 rows has null rating. Since there are many rows with null rating, we should not filter these values. Also we should not replace this rating with a median or mean rating etc

```
In [ ]:
```

Exercise-1. In aggregate bookings find columns that have null values. Fill these null values with whatever you think is the appropriate substitute (possible ways is to use mean or median)

```
In [40]: df_agg_bookings.isnull().sum()
```

```
Out[40]: property_id          0
check_in_date          0
room_category          0
successful_bookings      0
capacity                 2
dtype: int64
```

```
In [41]: df_agg_bookings[df_agg_bookings.capacity.isna()]
```

```
Out[41]:
```

	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 [42]: df_agg_bookings.capacity.median()
```

```
Out[42]: np.float64(25.0)
```

```
In [43]: df_agg_bookings.capacity.fillna(df_agg_bookings.capacity.median(),inplace=True)
```

C:\Users\BITTU DAS\AppData\Local\Temp\ipykernel_16520\2127972865.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the original object.

```
df_agg_bookings.capacity.fillna(df_agg_bookings.capacity.median(),inplace=True)
```

```
In [44]: df_agg_bookings.loc[[8,15]]
```

```
Out[44]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity
8	17561	1-May-22	RT1	22	25.0
15	17563	1-May-22	RT1	21	25.0

Exercise-2. In aggregate bookings find out records that have successful_bookings value greater than capacity. Filter those records

```
In [45]: df_agg_bookings[df_agg_bookings.successful_bookings>df_agg_bookings.capacity]
```

```
Out[45]:
```

	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

```
In [46]: df_agg_bookings.shape
```

```
Out[46]: (9200, 5)
```

```
In [47]: df_agg_bookings = df_agg_bookings[df_agg_bookings.successful_bookings<=df_agg_bookings.capacity]
```

```
Out[47]: (9194, 5)
```

In []:

==> 3. Data Transformation

Create occupancy percentage column

In [48]: `df_agg_bookings.head(3)`

Out[48]:

	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 [49]: `df_agg_bookings['occ_pct'] = df_agg_bookings.apply(lambda row: row['successful_bookings']/row['capacity'], axis=1)`

You can use following approach to get rid of SettingWithCopyWarning

In [50]: `new_col = df_agg_bookings.apply(lambda row: row['successful_bookings']/row['capacity'], axis=1)
df_agg_bookings = df_agg_bookings.assign(occ_pct=new_col.values)
df_agg_bookings.head(3)`

Out[50]:

	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

Convert it to a percentage value

In [51]: `df_agg_bookings['occ_pct'] = df_agg_bookings['occ_pct'].apply(lambda x: round(x*100, 2))
df_agg_bookings.head(3)`

Out[51]:

	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 [52]: `df_bookings.head()`

```
Out[52]:
```

	booking_id	property_id	booking_date	check_in_date	checkout_date
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 [53]: df_agg_bookings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9194 entries, 0 to 9199
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype
---  -
0   property_id            9194 non-null   int64
1   check_in_date          9194 non-null   object
2   room_category          9194 non-null   object
3   successful_bookings     9194 non-null   int64
4   capacity               9194 non-null   float64
5   occ_pct               9194 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 502.8+ KB
```

There are various types of data transformations that you may have to perform based on the need. Few examples of data transformations are,

1. Creating new columns
2. Normalization
3. Merging data
4. Aggregation

==> 4. Insights Generation

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

```
In [54]: df_agg_bookings.head(3)
```

```
Out[54]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity	oc
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 [55]: df_agg_bookings.groupby("room_category")["occ_pct"].mean()
```

```
Out[55]: room_category
RT1      57.889643
RT2      58.009756
RT3      58.028213
RT4      59.277925
Name: occ_pct, dtype: float64
```

I don't understand RT1, RT2 etc. Print room categories such as Standard, Premium, Elite etc along with average occupancy percentage

```
In [56]: df = pd.merge(df_agg_bookings, df_rooms, left_on="room_category", right_on="room_category")
df.head(4)
```

```
Out[56]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
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	16558	1-May-22	RT1	18	19.0	

```
In [57]: df.drop("room_id",axis=1, inplace=True)
df.head(4)
```

```
Out[57]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
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	16558	1-May-22	RT1	18	19.0	

```
In [58]: df.groupby("room_class")["occ_pct"].mean()
```

```
Out[58]: room_class
Elite      58.009756
Premium    58.028213
Presidential 59.277925
Standard   57.889643
Name: occ_pct, dtype: float64
```

```
In [59]: df[df.room_class=="Standard"].occ_pct.mean()
```

```
Out[59]: np.float64(57.88964285714285)
```

2. Print average occupancy rate per city

```
In [60]: df_hotels.head(3)
```

```
Out[60]:
```

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

```
Out[61]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity	oc
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 [62]: df.groupby("city")["occ_pct"].mean()
```

```
Out[62]: city
Bangalore    56.332376
Delhi        61.507341
Hyderabad    58.120652
Mumbai       57.909181
Name: occ_pct, dtype: float64
```

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

```
In [63]: df_date.head(3)
```

```
Out[63]:
```

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

```
Out[64]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity	oc
0	19563	10-May-22	RT3	15	29.0	
1	18560	10-May-22	RT1	19	30.0	
2	19562	10-May-22	RT1	18	30.0	

```
In [65]: df.groupby("day_type")["occ_pct"].mean().round(2)
```

```
Out[65]: day_type
weekday    50.88
weekend    72.34
Name: occ_pct, dtype: float64
```

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

```
In [66]: df_june_22 = df[df["mmm yy"]=="Jun 22"]
df_june_22.head(4)
```

```
Out[66]:
```

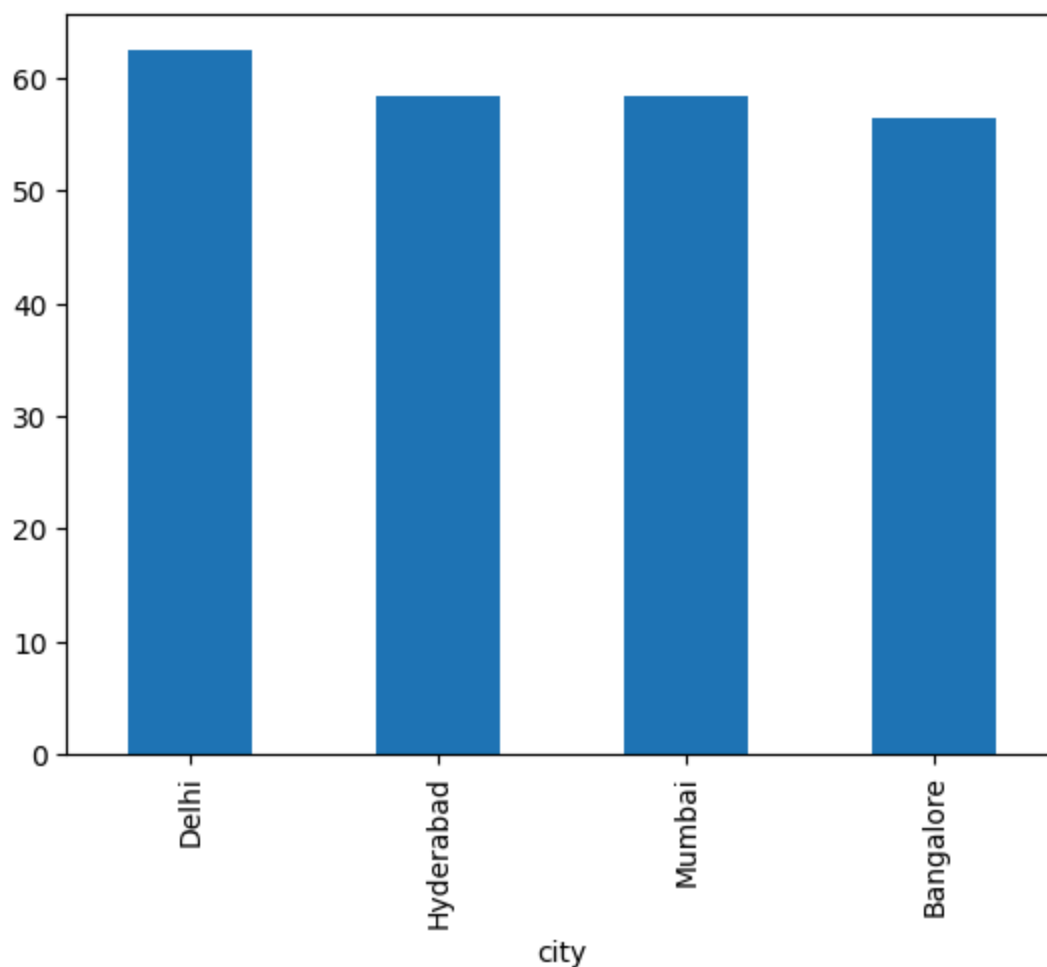
	property_id	check_in_date	room_category	successful_bookings	capacity
2200	16559	10-Jun-22	RT1	20	30.0
2201	19562	10-Jun-22	RT1	19	30.0
2202	19563	10-Jun-22	RT1	17	30.0
2203	17558	10-Jun-22	RT1	9	19.0

```
In [67]: df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=False)
```

```
Out[67]: city
Delhi      62.47
Hyderabad  58.46
Mumbai     58.38
Bangalore  56.44
Name: occ_pct, dtype: float64
```

```
In [68]: df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=False)
```

```
Out[68]: <Axes: xlabel='city'>
```

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

```
In [69]: df_august = pd.read_csv("datasets/new_data_august.csv")
df_august.head(3)
```

```
Out[69]:
```

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

```
In [70]: df_august.columns
```

```
Out[70]: 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 [71]: df.columns
```

```
Out[71]: 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 [72]: df_august.shape
```

```
Out[72]: (7, 13)
```

```
In [73]: df.shape
```

```
Out[73]: (6497, 14)
```

```
In [74]: latest_df = pd.concat([df, df_august], ignore_index = True, axis = 0)  
latest_df.tail(10)
```

```
Out[74]:
```

	property_id	check_in_date	room_category	successful_bookings	capacity
6494	17558	31-Jul-22	RT4	3	6.0
6495	19563	31-Jul-22	RT4	3	6.0
6496	17561	31-Jul-22	RT4	3	4.0
6497	16559	01-Aug-22	RT1	30	30.0
6498	19562	01-Aug-22	RT1	21	30.0
6499	19563	01-Aug-22	RT1	23	30.0
6500	19558	01-Aug-22	RT1	30	40.0
6501	19560	01-Aug-22	RT1	20	26.0
6502	17561	01-Aug-22	RT1	18	26.0
6503	17564	01-Aug-22	RT1	10	16.0

```
In [75]: latest_df.shape
```

```
Out[75]: (6504, 15)
```

6. Print revenue realized per city

```
In [76]: df_bookings.head()
```

```
Out[76]:
```

	booking_id	property_id	booking_date	check_in_date	checkout_date
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 [77]: df_hotels.head(3)
```

```
Out[77]:
```

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

```
Out[78]:
```

	booking_id	property_id	booking_date	check_in_date	checkout_date
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 [79]: df_bookings_all.groupby("city")["revenue_realized"].sum()
```

```
Out[79]: city
Bangalore    420383550
Delhi        294404488
Hyderabad    325179310
Mumbai       668569251
Name: revenue_realized, dtype: int64
```

7. Print month by month revenue

```
In [80]: df_date.head(3)
```

```
Out[80]:
```

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

```
Out[81]: array(['May 22', 'Jun 22', 'Jul 22'], dtype=object)
```

```
In [82]: df_bookings_all.head(3)
```

```
Out[82]:
```

	booking_id	property_id	booking_date	check_in_date	checkout_date
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 [83]: 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    object  
1   mmm yy      92 non-null    object  
2   week no     92 non-null    object  
3   day_type    92 non-null    object  
dtypes: object(4)  
memory usage: 3.0+ KB
```

```
In [84]: df_date["date"] = pd.to_datetime(df_date["date"], format="%d-%b-%y")  
df_date.head(3)
```

```
Out[84]:
```

	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

```
In [85]: 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

```

```

In [86]: df_bookings_all["check_in_date"] = pd.to_datetime(
          df_bookings_all["check_in_date"].str.strip(),
          format=None,           # let pandas infer
          dayfirst=True,
          errors="coerce"       # bad values -> NaT
        )

df_bookings_all.head(4)

```

```

Out[86]:

```

	booking_id	property_id	booking_date	check_in_date	checkout_date
0	May012216558RT12	16558	30-04-22	2022-05-01	2/5/2022
1	May012216558RT15	16558	27-04-22	2022-05-01	2/5/2022
2	May012216558RT16	16558	1/5/2022	2022-05-01	3/5/2022
3	May012216558RT17	16558	28-04-22	2022-05-01	6/5/2022

```

In [87]: df_bookings_all = pd.merge(df_bookings_all, df_date, left_on="check_in_date",
          df_bookings_all.head(3)

```

Out[87]:

	booking_id	property_id	booking_date	check_in_date	checkout_date
0	May012216558RT12	16558	30-04-22	2022-05-01	2/5/2022
1	May012216558RT15	16558	27-04-22	2022-05-01	2/5/2022
2	May012216558RT16	16558	1/5/2022	2022-05-01	3/5/2022

```
In [88]: df_bookings_all["mmm yy"] = df_bookings_all["check_in_date"].dt.strftime('%b %
df_bookings_all.groupby("mmm yy")["revenue_realized"].sum()
```

```
Out[88]: mmm yy
Jul 22    243180932
Jun 22    229637640
May 22    234353183
Name: revenue_realized, dtype: int64
```

Exercise-1. Print revenue realized per hotel type

```
In [89]: df_bookings_all.property_name.unique()
```

```
Out[89]: array(['Atliq Grands', 'Atliq Exotica', 'Atliq City', 'Atliq Blu',
                'Atliq Bay', 'Atliq Palace', 'Atliq Seasons'], dtype=object)
```

```
In [90]: df_bookings_all.groupby("property_name")["revenue_realized"].sum().round(2).sc
```

```
Out[90]: property_name
Atliq Seasons    26838223
Atliq Grands     87245939
Atliq Bay        107516312
Atliq Blu        108108129
Atliq City       118290783
Atliq Palace     125553143
Atliq Exotica    133619226
Name: revenue_realized, dtype: int64
```

Exercise-2 Print average rating per city

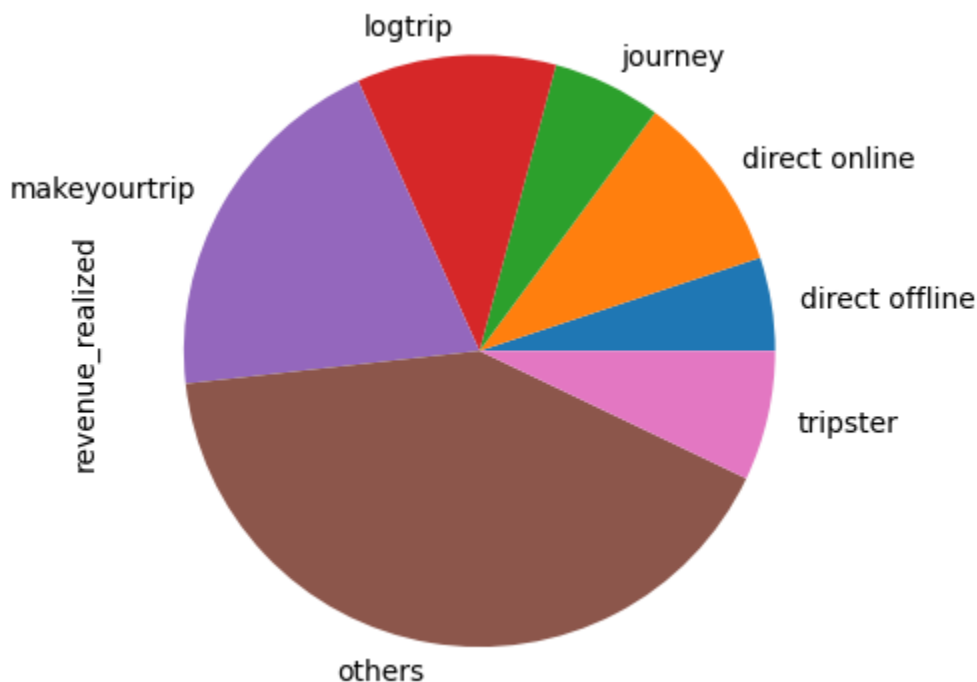
```
In [91]: df_bookings_all.groupby("city")["ratings_given"].mean().round(2)
```

```
Out[91]: city
Bangalore    3.41
Delhi        3.79
Hyderabad    3.65
Mumbai       3.66
Name: ratings_given, dtype: float64
```

Exercise-3 Print a pie chart of revenue realized per booking platform

```
In [92]: df_bookings_all.groupby("booking_platform")["revenue_realized"].sum().plot(kind="pie")
```

```
Out[92]: <Axes: ylabel='revenue_realized'>
```



```
In [93]: df_bookings_all.columns
```

```
Out[93]: Index(['booking_id', 'property_id', 'booking_date', 'check_in_date',  
               'checkout_date', 'no_guests', 'room_category', 'booking_platform',  
               'ratings_given', 'booking_status', 'revenue_generated',  
               'revenue_realized', 'property_name', 'category', 'city', 'date',  
               'mmm yy', 'week no', 'day_type'],  
              dtype='object')
```

```
In [94]: # 1) Making a clean date key in df_agg_bookings from its string check_in_date  
df_agg_bookings["check_in_dt"] = pd.to_datetime(  
    df_agg_bookings["check_in_date"], format="%d-%b-%y", errors="coerce"  
)  
  
# 2) Making a clean date key in df_bookings_all from its datetime check_in_date  
df_bookings_all["check_in_dt"] = pd.to_datetime(  
    df_bookings_all["check_in_date"], errors="coerce"  
)  
  
# 3) Aggregating revenue from bookings on the same key  
rev_agg = (  
    df_bookings_all  
        .groupby(["property_id", "check_in_dt", "room_category"], as_index=False)  
        .agg(revenue_realized=("revenue_realized", "sum"))  
)
```

```

# 4) Merging occupancy (df_agg_bookings) with revenue (rev_agg)
df_occ_room = df_agg_bookings.merge(
    rev_agg,
    left_on=["property_id", "check_in_dt", "room_category"],
    right_on=["property_id", "check_in_dt", "room_category"],
    how="left"
)

# 5) ADR(average daily rate) and RevPAR(revenue per available room)
df_occ_room["adr"] = (
    df_occ_room["revenue_realized"] / df_occ_room["successful_bookings"]
)
df_occ_room["revpar"] = (
    df_occ_room["revenue_realized"] / df_occ_room["capacity"]
)

df_occ_room[
    ["property_id", "check_in_date", "room_category",
     "successful_bookings", "capacity", "occ_pct", "adr", "revpar"]
].head()

```

Out[94]:

	property_id	check_in_date	room_category	successful_bookings	capacity	oc
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	16558	1-May-22	RT1	18	19.0	
4	17560	1-May-22	RT1	28	40.0	

In []: