AtliQ Hotels - Data Analysis

```
[103]: import pandas as pd
```

```
[104]: import matplotlib.pyplot as plt
```

➤ 1. Data Import and Data Exploration

Datasets

We have 5 csv files

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

Reading fact_bookings_data in a datagrame

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

Exploring bookings data

```
[5]: df_bookings.head()
```

[5]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_catego
0	May012216558RT11	16558	27-04-22	1/5/2022	2/5/2022	-3.0	R
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	R
2	May012216558RT13	16558	28-04-22	1/5/2022	4/5/2022	2.0	R
3	May012216558RT14	16558	28-04-22	1/5/2022	2/5/2022	-2.0	R
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	4.0	R

```
[6]: df_bookings.shape
```

```
[6]: (134590, 12)
```

```
[7]: df_bookings.room_category.unique()
```

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

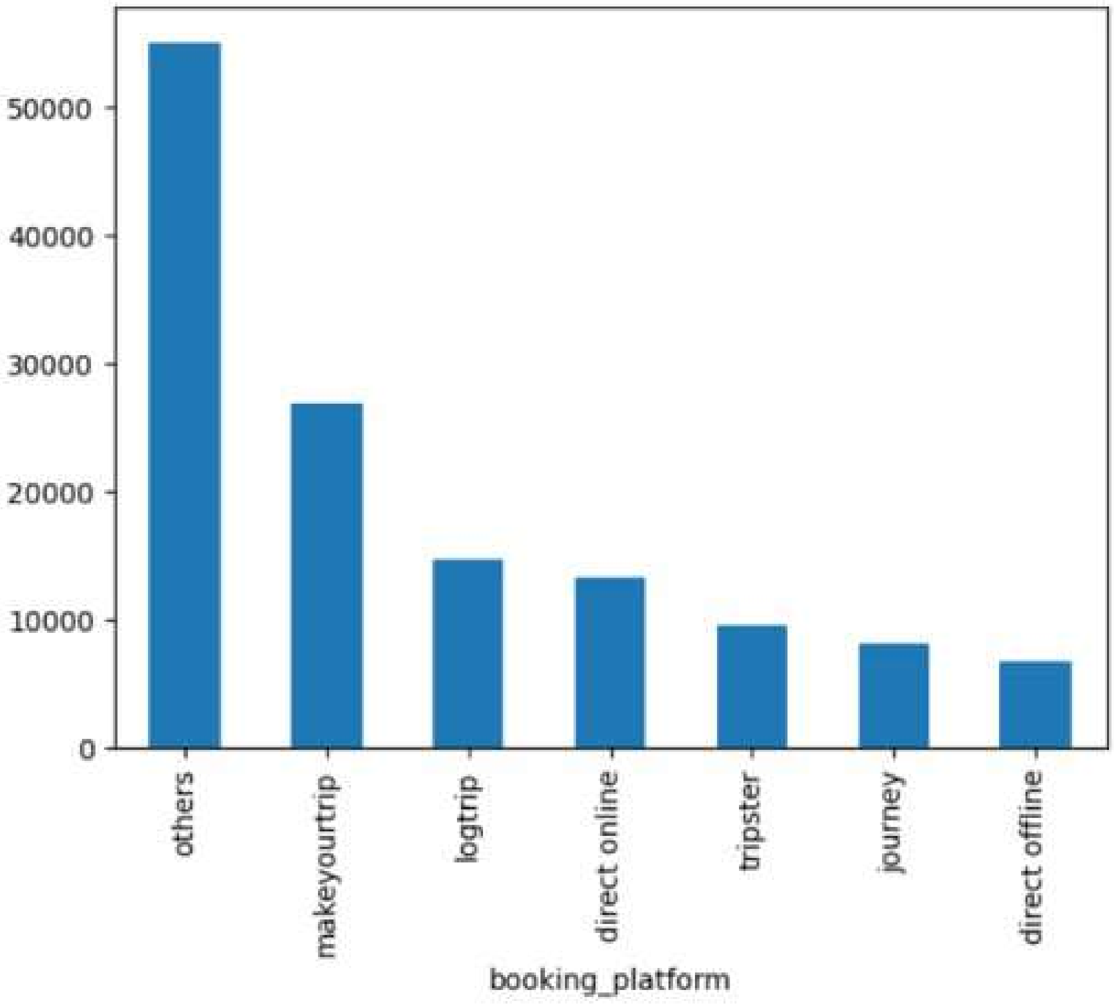
```
[8]: df_bookings.booking_platform.unique()
```

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

```
[9]: df_bookings.booking_platform.value_counts()
```

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

```
[97]: df_bookings.booking_platform.value_counts().plot(kind="bar")
      plt.show()
```



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

[11]:

	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

Reading rest of the files

[12]:
```
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')
```

[13]:
```
df_hotels.shape
```

[13]:
```
(25, 4)
```

[14]:
```
df_hotels.head(3)
```

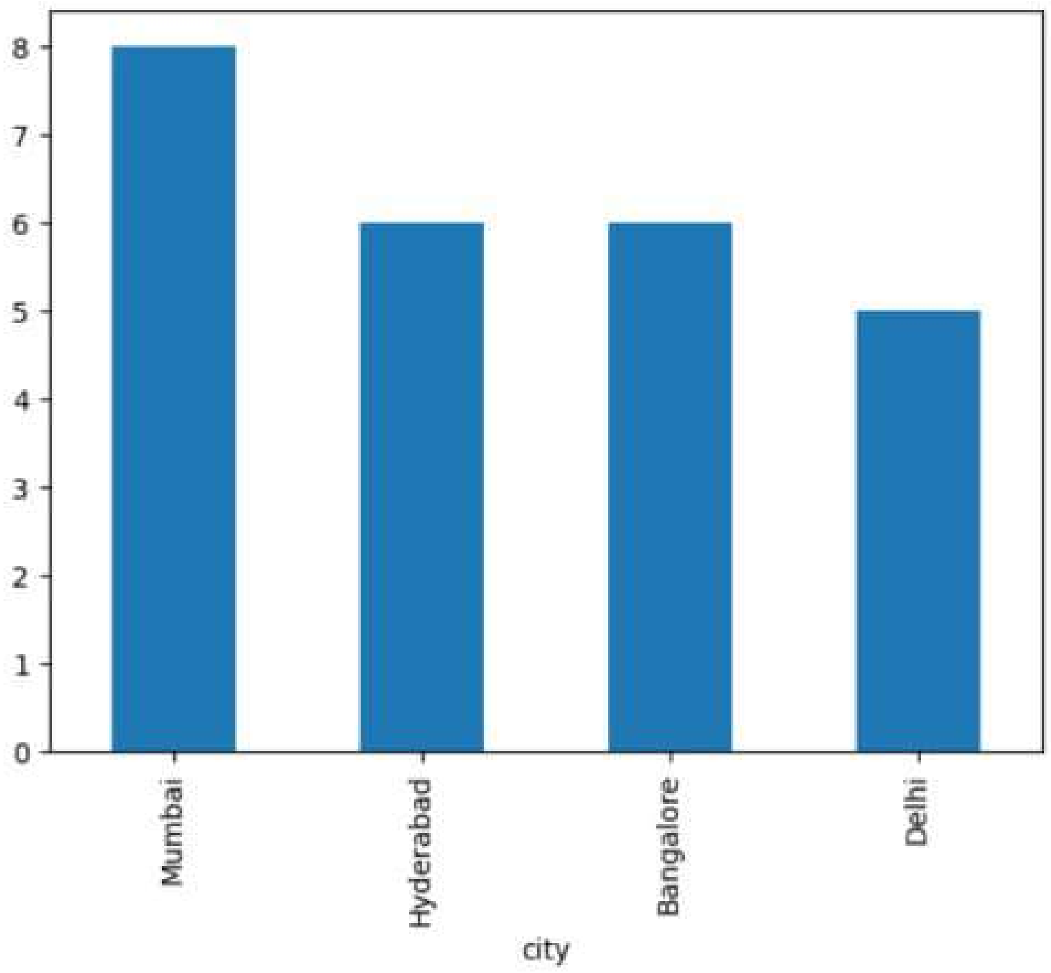
[14]:

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

[15]:
```
df_hotels.category.value_counts()
```

[15]:
```
category
Luxury      16
Business     9
Name: count, dtype: int64
```

[98]:
```
df_hotels.city.value_counts().plot(kind="bar")
plt.show()
```



Exploring aggregate bookings

```
[17]: df_agg_bookings.head(5)
```

[17]:

	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

```
[18]: df_agg_bookings.shape
```

```
[18]: (9200, 5)
```

Finding unique property ids in aggregate bookings dataset

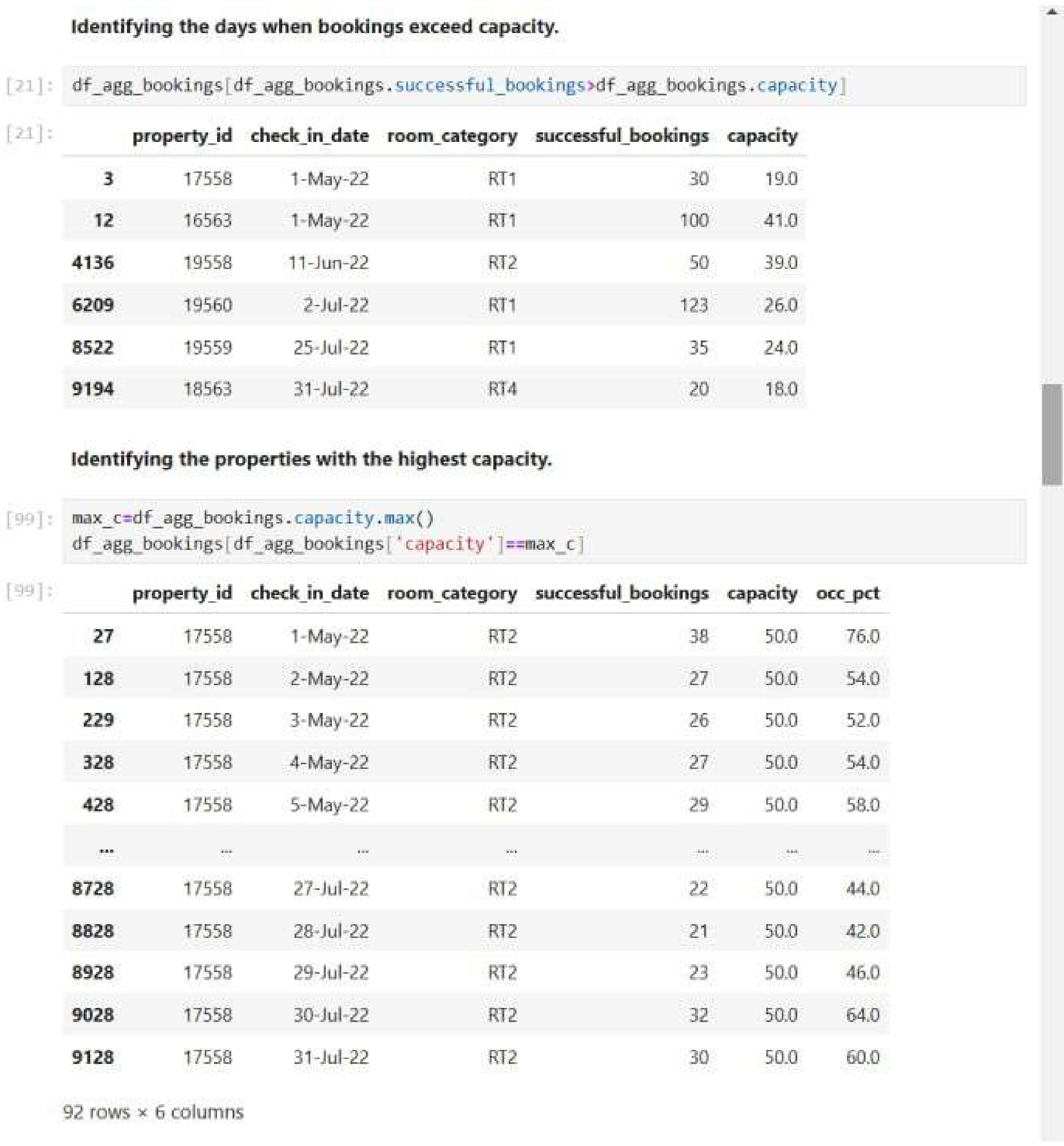
```
[19]: df_agg_bookings.property_id.unique()
```

```
[19]: 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)
```

Calculating the total number of bookings for each property ID

```
[20]: df_agg_bookings.groupby('property_id')['successful_bookings'].sum()
```

```
[20]: 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
```



Identifying the days when bookings exceed capacity.

[21]:
```
df_agg_bookings[df_agg_bookings.successful_bookings>df_agg_bookings.capacity]
```

[21]:

	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

Identifying the properties with the highest capacity.

[99]:
```
max_c=df_agg_bookings.capacity.max()
df_agg_bookings[df_agg_bookings['capacity']==max_c]
```

[99]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
27	17558	1-May-22	RT2	38	50.0	76.0
128	17558	2-May-22	RT2	27	50.0	54.0
229	17558	3-May-22	RT2	26	50.0	52.0
328	17558	4-May-22	RT2	27	50.0	54.0
428	17558	5-May-22	RT2	29	50.0	58.0
...	...	...	...	...	...	...
8728	17558	27-Jul-22	RT2	22	50.0	44.0
8828	17558	28-Jul-22	RT2	21	50.0	42.0
8928	17558	29-Jul-22	RT2	23	50.0	46.0
9028	17558	30-Jul-22	RT2	32	50.0	64.0
9128	17558	31-Jul-22	RT2	30	50.0	60.0

92 rows × 6 columns

➤ 2. Data Cleaning

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

[23]:

	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 guests

```
[24]: df_bookings[df_bookings.no_guests<=0]
```

[24]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room
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	

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

```
[25]: df_bookings = df_bookings[df_bookings.no_guests>0]
```

```
[26]: df_bookings.shape
```

```
[26]: (134578, 12)
```

(2) Outlier removal in revenue generated

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

```
[27]: (6500, 28560000)
```

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

```
[28]: (15378.036937686695, 13500.0)
```

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

```
[30]: higher_limit = avg + 3*std
      higher_limit
```

```
[30]: 294498.50173207896
```

```
[31]: lower_limit = avg - 3*std
      lower_limit
```

```
[31]: -263742.4278567056
```

```
[32]: df_bookings[df_bookings.revenue_generated<=0]
```

[32]:

booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_category	bool

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

[33]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room
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	

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

```
[34]: (134573, 12)
```

```
[35]: df_bookings.revenue_realized.describe()
```

```
[35]: 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
```

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

```
[36]: 33479.358661845814
```

[37]:
```
df_bookings[df_bookings.revenue_realized>higher_limit]
```

[37]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room
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

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

[38]:
```
df_bookings[df_bookings.room_category=="RT4"].revenue_realized.describe()
```

[38]:
```
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
```

[39]:
```
# mean + 3*standard deviation
23439+3*9048
```

[39]:
```
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

[40]:
```
df_bookings[df_bookings.booking_id=="May012216558RT213"]
```

[40]:

booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_category	bool

```
[40]: df_bookings[df_bookings.booking_id=="May012216558RT213"]
```

[40]:

booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_category	bool

```
[41]: df_bookings.isnull().sum()
```

```
[41]: 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 aggregate bookings finding columns that have null values. Filling these null values with Appropriate value

```
[42]: df_agg_bookings.isnull().sum()
```

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

```
[101]: # Display rows where a specific column ('column_name') has null values
       df_agg_bookings[df_agg_bookings['capacity'].isnull()]
```

[101]:

property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct

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

```
[44]: 25.0
```

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

```
[46]: df_agg_bookings.loc[[8,14]]
```

[46]:

	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 identifying records that have successful_bookings value greater than capacity. Filtering those records

```
[47]: df_agg_bookings[df_agg_bookings.successful_bookings>df_agg_bookings.capacity]
```

[47]:

	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

```
[48]: df_agg_bookings=df_agg_bookings[df_agg_bookings.successful_bookings<df_agg_bookings.capacity]
```

➤ 3. Data Transformation

Creating occupancy percentage column

```
[49]: df_agg_bookings.head(3)
```

[49]:

	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

```
[50]: df_agg_bookings['occ_pct'] = df_agg_bookings.apply(lambda row: row['successful_bookings']/row['
```

```
[51]: 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)
```

[51]:

	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

Converting it to a percentage value

[52]:
```
df_agg_bookings['occ_pct'] = df_agg_bookings['occ_pct'].apply(lambda x: round(x*100, 2))
df_agg_bookings.head(3)
```

[52]:

	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

[53]:
```
df_bookings.head()
```

[53]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_catego
1	May012216558RT12	16558	30-04-22	1/5/2022	2/5/2022	2.0	R
4	May012216558RT15	16558	27-04-22	1/5/2022	2/5/2022	4.0	R
5	May012216558RT16	16558	1/5/2022	1/5/2022	3/5/2022	2.0	R
6	May012216558RT17	16558	28-04-22	1/5/2022	6/5/2022	2.0	R
7	May012216558RT18	16558	26-04-22	1/5/2022	3/5/2022	2.0	R

[54]:
```
df_agg_bookings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9082 entries, 0 to 9199
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   property_id          9082 non-null   int64
 1   check_in_date        9082 non-null   object
 2   room_category        9082 non-null   object
 3   successful_bookings  9082 non-null   int64
 4   capacity             9082 non-null   float64
 5   occ_pct              9082 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 496.7+ KB
```

➤ 4. Insights Generation

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

[55]:

```
df_agg_bookings.head(3)
```

[55]:

	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

[56]:

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

[56]:

```
room_category
RT1    57.779310
RT2    57.752486
RT3    57.604256
RT4    58.017915
Name: occ_pct, dtype: float64
```

It is hard to understand RT1, RT2 etc. So Printing room categories such as Standard, Premium, Elite etc along with average occupancy percentage

[57]:

```
df = pd.merge(df_agg_bookings, df_rooms, left_on="room_category", right_on="room_id")
df.head(4)
```

[57]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_id	room_cl
0	16559	1-May-22	RT1	25	30.0	83.33	RT1	Stand
1	19562	1-May-22	RT1	28	30.0	93.33	RT1	Stand
2	19563	1-May-22	RT1	23	30.0	76.67	RT1	Stand
3	16558	1-May-22	RT1	18	19.0	94.74	RT1	Stand

[58]:

```
df.drop("room_id",axis=1, inplace=True)
df.head(4)
```

[58]:

	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	16558	1-May-22	RT1	18	19.0	94.74	Standard

[59]:

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

[59]:

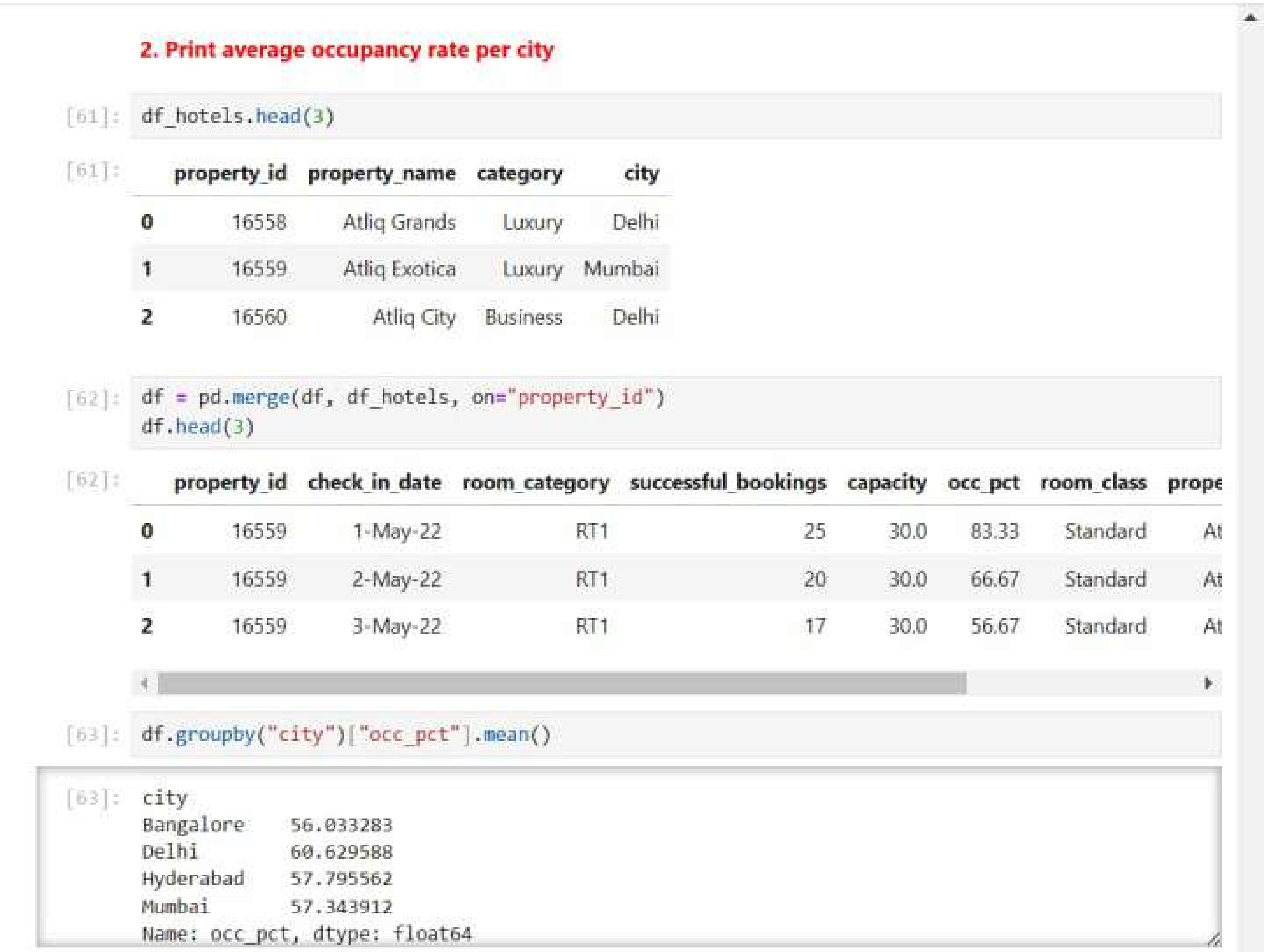
```
room_class
Elite           57.752486
Premium         57.604256
Presidential    58.017915
Standard        57.779310
Name: occ_pct, dtype: float64
```

[60]:

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

[60]:

```
57.77931004366812
```



2. Print average occupancy rate per city

```
[61]: df_hotels.head(3)
```

[61]:

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

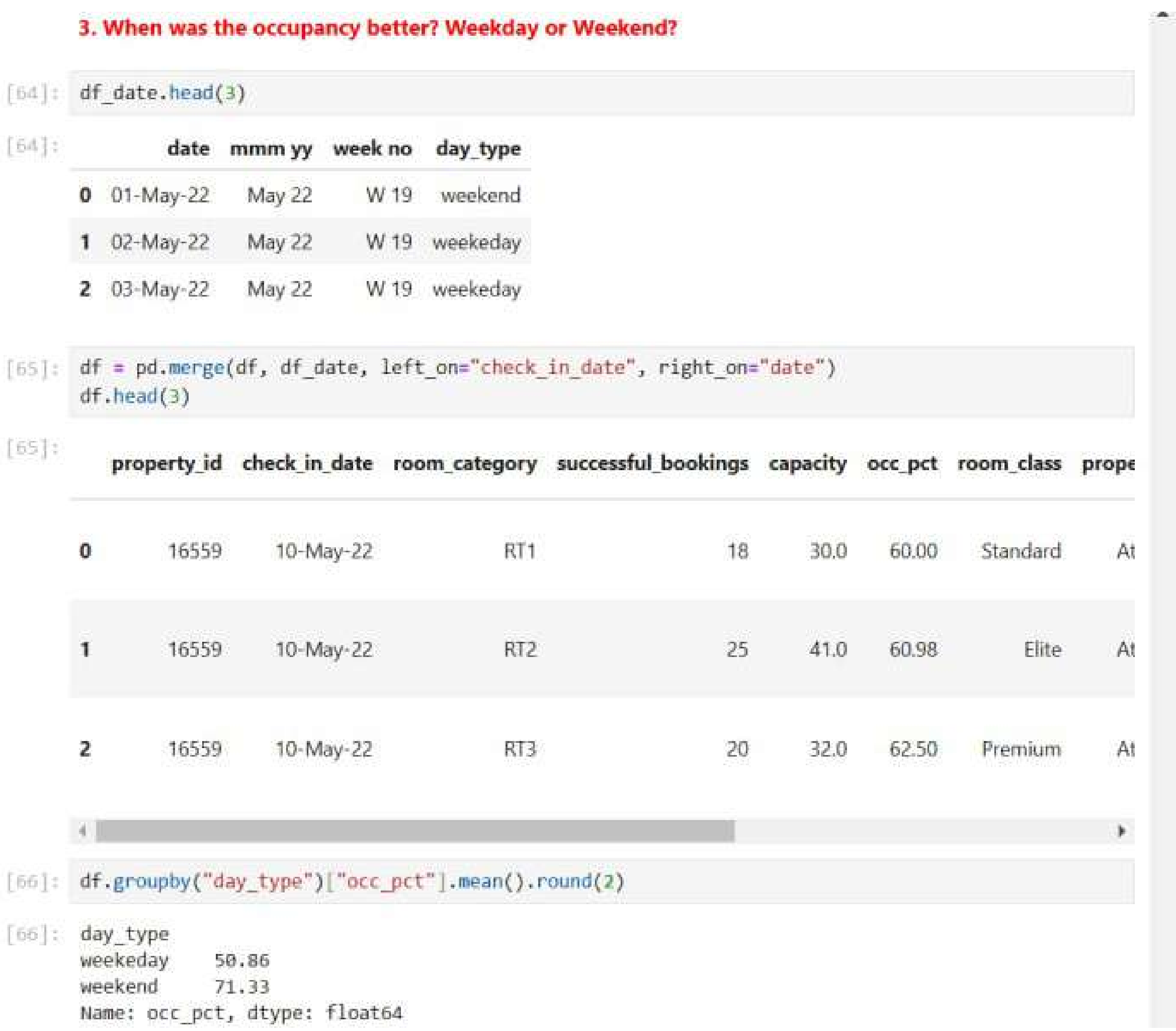
```
[62]: df = pd.merge(df, df_hotels, on="property_id")
      df.head(3)
```

[62]:

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

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

```
[63]: city
      Bangalore    56.033283
      Delhi        60.629588
      Hyderabad    57.795562
      Mumbai       57.343912
      Name: occ_pct, dtype: float64
```



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

[64]:
```
df_date.head(3)
```

[64]:

	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

[65]:
```
df = pd.merge(df, df_date, left_on="check_in_date", right_on="date")
df.head(3)
```

[65]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_class	prope
0	16559	10-May-22	RT1	18	30.0	60.00	Standard	At
1	16559	10-May-22	RT2	25	41.0	60.98	Elite	At
2	16559	10-May-22	RT3	20	32.0	62.50	Premium	At

[66]:
```
df.groupby("day_type")["occ_pct"].mean().round(2)
```

[66]:
```
day_type
weekeday    50.86
weekend     71.33
Name: occ_pct, dtype: float64
```

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

[67]:
```
df_june_22 = df[df["mmm yy"]=="Jun 22"]
df_june_22.head(4)
```

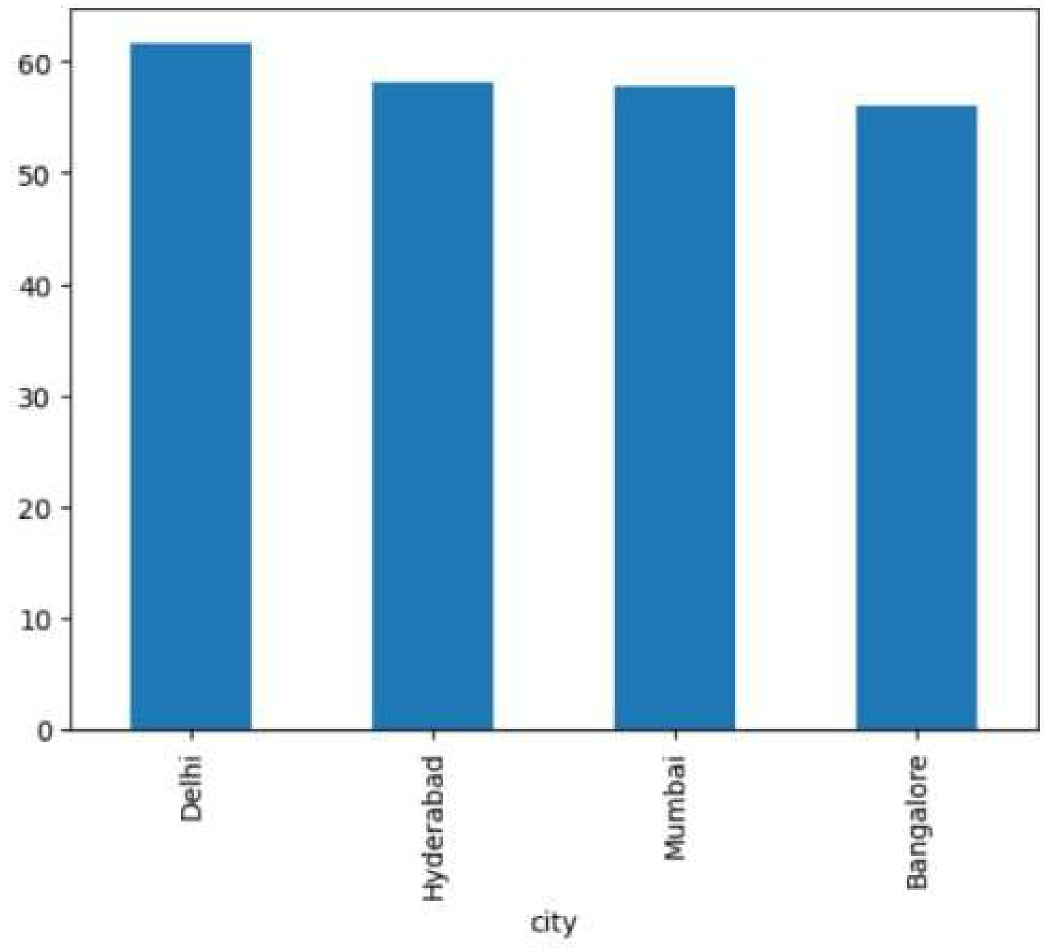
[67]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_class	pr
2177	16559	10-Jun-22	RT1	20	30.0	66.67	Standard	
2178	16559	10-Jun-22	RT2	26	41.0	63.41	Elite	
2179	16559	10-Jun-22	RT3	20	32.0	62.50	Premium	
2180	16559	10-Jun-22	RT4	11	18.0	61.11	Presidential	

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

[68]:
```
city
Delhi        61.65
Hyderabad    58.21
Mumbai       57.82
Bangalore    56.00
Name: occ_pct, dtype: float64
```

[102]:
```
df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=False).plot(kind="
plt.show()
```



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

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

[70]:

	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

```
[71]: df_august.columns
```

```
[71]: 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')
```

```
[72]: df.columns
```

```
[72]: 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')
```

```
[73]: df_august.shape
```

```
[73]: (7, 13)
```

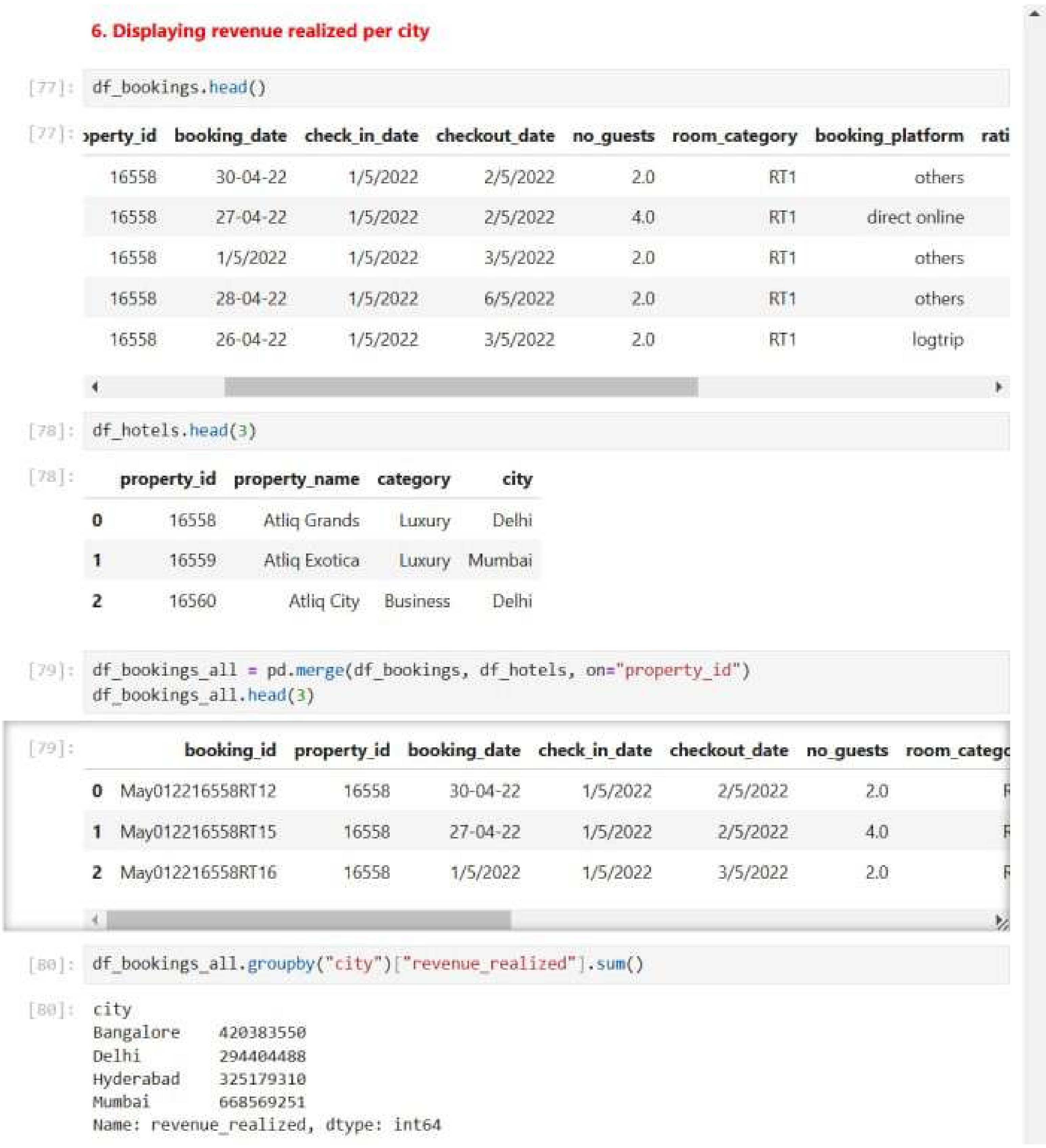
```
[74]: df.shape
```

```
[74]: (6428, 14)
```

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

[75]:

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	room_class	pr
6425	16563	31-Jul-22	RT2	32	38.0	84.21	Elite	
6426	16563	31-Jul-22	RT3	14	20.0	70.00	Premium	
6427	16563	31-Jul-22	RT4	13	18.0	72.22	Presidential	
6428	16559	01-Aug-22	RT1	30	30.0	NaN	Standard	
6429	19562	01-Aug-22	RT1	21	30.0	NaN	Standard	
6430	19563	01-Aug-22	RT1	23	30.0	NaN	Standard	
6431	19558	01-Aug-22	RT1	30	40.0	NaN	Standard	



6. Displaying revenue realized per city

[77]:
```
df_bookings.head()
```

[77]:

perty_id	booking_date	check_in_date	checkout_date	no_guests	room_category	booking_platform	rati
16558	30-04-22	1/5/2022	2/5/2022	2.0	RT1	others	
16558	27-04-22	1/5/2022	2/5/2022	4.0	RT1	direct online	
16558	1/5/2022	1/5/2022	3/5/2022	2.0	RT1	others	
16558	28-04-22	1/5/2022	6/5/2022	2.0	RT1	others	
16558	26-04-22	1/5/2022	3/5/2022	2.0	RT1	logtrip	

[78]:
```
df_hotels.head(3)
```

[78]:

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

[79]:
```
df_bookings_all = pd.merge(df_bookings, df_hotels, on="property_id")
df_bookings_all.head(3)
```

[79]:

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

[80]:
```
df_bookings_all.groupby("city")["revenue_realized"].sum()
```

[80]:
```
city
Bangalore    420383550
Delhi        294404488
Hyderabad    325179310
Mumbai       668569251
Name: revenue_realized, dtype: int64
```

7. Print month by month revenue

[81]:
```
df_date.head(3)
```

[81]:

	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

[82]:
```
df_date["mmm yy"].unique()
```

[82]:
```
array(['May 22', 'Jun 22', 'Jul 22'], dtype=object)
```

[83]:
```
df_bookings_all.head(3)
```

[83]:

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

[84]:
```
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
```

[85]:
```
df_date["date"] = pd.to_datetime(df_date["date"])
df_date.head(3)
```

```
C:\Users\balag\AppData\Local\Temp\ipykernel_19068\173964601.py:1: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to `dateutil`. To ensure par
sing is consistent and as-expected, please specify a format.
  df_date["date"] = pd.to_datetime(df_date["date"])
```

[85]:

	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

[86]:
```
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
```

[87]:
```
df_bookings_all["check_in_date"] = pd.to_datetime(df_bookings_all["check_in_date"],dayfirst=Tru
df_bookings_all.head(4)
```

[87]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_catego
0	May012216558RT12	16558	30-04-22	2022-05-01	2/5/2022	2.0	R
1	May012216558RT15	16558	27-04-22	2022-05-01	2/5/2022	4.0	R
2	May012216558RT16	16558	1/5/2022	2022-05-01	3/5/2022	2.0	R
3	May012216558RT17	16558	28-04-22	2022-05-01	6/5/2022	2.0	R

[88]:
```
df_bookings_all = pd.merge(df_bookings_all, df_date, left_on="check_in_date", right_on="date")
df_bookings_all.head(3)
```

[88]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_catego
0	May012216558RT12	16558	30-04-22	2022-05-01	2/5/2022	2.0	R
1	May012216558RT15	16558	27-04-22	2022-05-01	2/5/2022	4.0	R
2	May012216558RT16	16558	1/5/2022	2022-05-01	3/5/2022	2.0	R

[89]:
```
df_bookings_all.groupby("mmm yy")["revenue_realized"].sum()
```

[89]:
```
mmm yy
Jul 22    243180932
Jun 22    229637640
May 22    234353183
Name: revenue_realized, dtype: int64
```

8. Print revenue realized per hotel type

```
[90]: df_bookings_all.head()
```

[90]:

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests	room_catego
0	May012216558RT12	16558	30-04-22	2022-05-01	2/5/2022	2.0	F
1	May012216558RT15	16558	27-04-22	2022-05-01	2/5/2022	4.0	F
2	May012216558RT16	16558	1/5/2022	2022-05-01	3/5/2022	2.0	F
3	May012216558RT17	16558	28-04-22	2022-05-01	6/5/2022	2.0	F
4	May012216558RT18	16558	26-04-22	2022-05-01	3/5/2022	2.0	F

```
[91]: df_bookings_all.groupby('category')['revenue_realized'].sum()
```

```
[91]: category
      Business    270682149
      Luxury      436489606
      Name: revenue_realized, dtype: int64
```

9. Print average rating per city

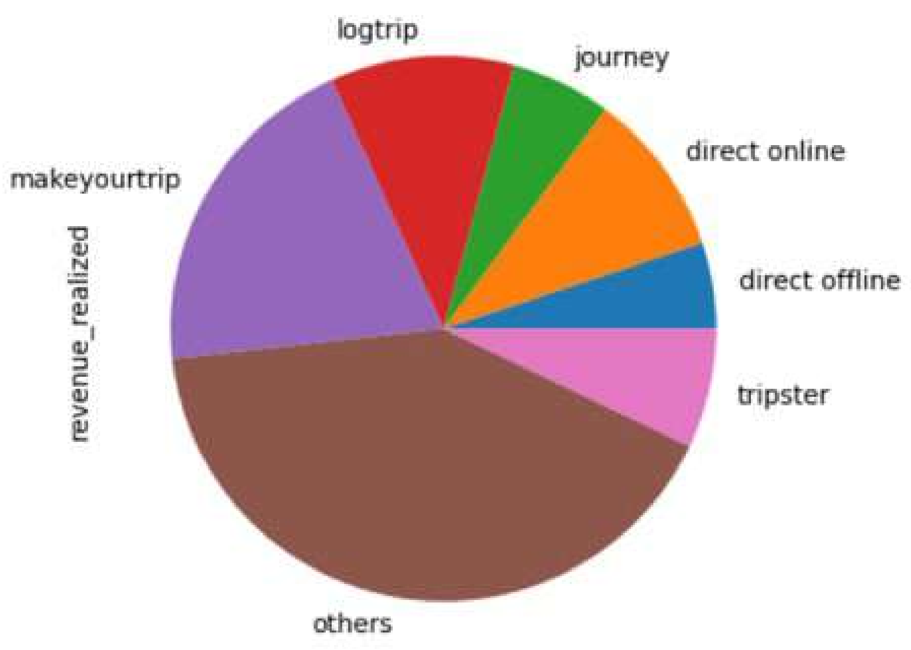
```
[92]: # write your code here
      df_bookings_all.groupby('city')['ratings_given'].mean()
```

```
[92]: city
      Bangalore    3.414599
      Delhi        3.788105
      Hyderabad    3.653903
      Mumbai       3.655835
      Name: ratings_given, dtype: float64
```

10. Print a pie chart of revenue realized per booking platform

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

```
[94]: <Axes: ylabel='revenue_realized'>
```


logtrip
journey
direct online
makeyourtrip
revenue_realized
direct offline
tripster
others