Hospitality Domain Data Analysis

In [287…
```
import pandas as pd
```

1. Data Import and Data Exploration

Datasets

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

1.1 Read bookings data in a dataframe

In [288…
```
df_bookings = pd.read_csv('datasets/fact_bookings.csv')
```

1.2 Explore bookings data

In [289…
```
df_bookings.head(2)
```

Out[289…

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_guests
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

1.3 Room Category Unique Records

In [290…

```
df_bookings.room_category.unique()
```

Out[290…

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

1.4 Booking Platform Unique Records

In [291…

```
df_bookings.booking_platform.unique()
```

Out[291…

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

1.5 Booking Platform Wise Count

In [292…

```
df_bookings.booking_platform.value_counts()
```

Out[292…

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

1.6 Describe Table df_bookings

In [293…

```
df_bookings.describe()
```

Out[293…

	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

1.7 Read rest of the files

In [294…

```
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')
```

1.8 Explore aggregate bookings

In [295…

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

Out[295…

	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

1.9 Unique property ids in aggregate bookings dataset

In [296…

```
df_agg_bookings.property_id.nunique()
```

Out[296…

```
25
```

1.10 Total bookings per property_id

In [297…

```
property_id_counts = df_agg_bookings.property_id.value_counts()
print(property_id_counts)
```

```
property_id
16559    368
17559    368
17564    368
19561    368
19559    368
18563    368
18562    368
18561    368
18559    368
18558    368
17563    368
17562    368
16563    368
19562    368
16562    368
16561    368
16560    368
17561    368
19560    368
19558    368
17560    368
16558    368
17558    368
19563    368
18560    368
Name: count, dtype: int64
```

1.11 Days on which bookings are greater than capacity

In [298…

```
exceeds_capacity_df =df_agg_bookings[df_agg_bookings['successful_bookings'] > df_ag
exceeds_capacity_df
```

Out[298…

	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

1.12 Properties that have highest capacity

In [299…

```
highest_capacity_properties = df_agg_bookings[df_agg_bookings['capacity'] == df_agg
highest_capacity_properties.head(2)
```

Out[299…

	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

2. Data Cleaning

2.1 Describe Table df_bookings

In [300…

```
df_bookings.describe()
```

Out[300…

	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

2.2 Clean invalid guests

In [301…

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

Out[301…

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

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

df_bookings = df_bookings[df_bookings.no_guests>0]

2.3 Outlier removal in revenue generated

In [302…

```
df_bookings.revenue_generated.min(), df_bookings.revenue_generated.max()
```

Out[302…

```
(6500, 28560000)
```

2.4 Calculate Mean & Median

In [303…

```
df_bookings.revenue_generated.mean(), df_bookings.revenue_generated.median()
```

Out[303…

```
(15378.05412734973, 13500.0)
```

2.5 Average & Standard Deviation

In [304…

```
avg, std = df_bookings['revenue_generated'].mean(), df_bookings['revenue_generatec

higher_limit = avg + 3 * std
lower_limit = avg - 3 * std

print(f"Higher limit: {higher_limit}\nLower limit: {lower_limit}")
```

```
Higher limit: 294486.17014021333
Lower limit: -263730.06188551383
```

df_bookings[df_bookings.revenue_realized>higher_limit]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

2.6 Category=RT4

In [305…

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

Out[305…

```
count    16073.000000
mean     23440.103652
std       9048.865206
min       7600.000000
25%      19000.000000
50%      26600.000000
75%      32300.000000
max      45220.000000
Name: revenue_realized, dtype: float64
```

2.7 Mean + 3*standard deviation

In [306…

```
23439+3*9048
```

Out[306…

```
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

2.8 Booking_ID=="May012216558RT213

In [307…

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

Out[307…

	booking_id	property_id	booking_date	check_in_date	checkout_date	no_gue
30	May012216558RT213	16558	29-04-22	1/5/2022	2/5/2022	N

2.9 Null Values Column Wise

In [308…

```
df_bookings.isnull().sum()
```

Out[308…

```
booking_id             0
property_id            0
booking_date           0
check_in_date          0
checkout_date          0
no_guests              3
room_category          0
booking_platform       0
ratings_given      77907
booking_status         0
revenue_generated      0
revenue_realized       0
dtype: int64
```

2.10 Identify Null Values and Replace it with statistical values

In [309…

```
missing_values = df_agg_bookings.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
property_id            0
check_in_date          0
room_category          0
successful_bookings    0
capacity               2
dtype: int64
```

2.11 Fill missing values in 'capacity' column with the mean

In [310…

```
df_agg_bookings['capacity'] = df_agg_bookings['capacity'].fillna(df_agg_bookings[
df_agg_bookings.head(4)
```

Out[310…

	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

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

In [311…

```
overbooked_records = df_agg_bookings[df_agg_bookings['successful_bookings'] > df_a

print("\n Records with successful_bookings greater than capacity:\n")
overbooked_records
```

```
Records with successful_bookings greater than capacity:
```

Out[311…

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

3. Data Transformation

2.13 Create occupancy percentage column

In [312…

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

Out[312…

	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

You can use following approach to get rid of SettingWithCopyWarning

2.14 Create Occ_Pct Column Using Function

In [313…

```
df_agg_bookings['occ_pct']=df_agg_bookings.apply(lambda row: round((row['success
df_agg_bookings.head(3)
```

Out[313…

	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

2.15 Display Top 3 Rows

In [314…
```
df_bookings.head(3)
```

Out[314…

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

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

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

In [315…
```
df_agg_bookings.groupby("room_category")["occ_pct"].mean()
```

Out[315…
```
room_category
RT1    58.232091
RT2    58.040278
RT3    58.028213
RT4    59.300461
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

4.2 Join Tables: df_agg_bookings & df_rooms

In [316…
```
df = pd.merge(df_agg_bookings, df_rooms, left_on="room_category", right_on="room
df.head(4)
```

Out[316...

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

4.3 Drop Column room_id

In [317...

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

Out[317...

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

4.4 Group By - Room Class

In [318...

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

Out[318...

```
room_class
Elite           58.040278
Premium         58.028213
Presidential    59.300461
Standard        58.232091
Name: occ_pct, dtype: float64
```

4.5 Print average occupancy rate per city

In [319...

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

Out[319...

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

4.6 Table Join: df & df_hotels

In [320…
```
df = pd.merge(df, df_hotels, on="property_id")
df.head(3)
```

Out[320…

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

4.7 City Wise Occupnacy % Mean

In [321…
```
df.groupby("city")["occ_pct"].mean()
```

Out[321…
```
city
Bangalore    56.594207
Delhi        61.606467
Hyderabad    58.144651
Mumbai       57.942629
Name: occ_pct, dtype: float64
```

4.8 When was the occupancy better? Weekday or Weekend?

In [322…
```
df_date.head(3)
```

Out[322…

	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

4.9 Left Join Table df With df_date

In [323…
```
df = pd.merge(df, df_date, left_on="check_in_date", right_on="date")
df.head(3)
```

Out[323…

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct	r
0	19563	10-May-22	RT3	15	29.0	51.72	
1	18560	10-May-22	RT1	19	30.0	63.33	
2	19562	10-May-22	RT1	18	30.0	60.00	

4.10 Day Type Mean

In [324…

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

Out[324…

```
day_type
weekeday    50.90
weekend     72.39
Name: occ_pct, dtype: float64
```

4.11 Occupancy For Different Cities in June

In [325…

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

Out[325…

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
2200	16559	10-Jun-22	RT1	20	30.0	66.67
2201	19562	10-Jun-22	RT1	19	30.0	63.33
2202	19563	10-Jun-22	RT1	17	30.0	56.67
2203	17558	10-Jun-22	RT1	9	19.0	47.37

4.12 City Wise Occupancy %

In [326…
```
df_june_22.groupby('city')['occ_pct'].mean().round(2).sort_values(ascending=Fals
```

Out[326…
```
city
Delhi        62.47
Hyderabad    58.46
Mumbai       58.38
Bangalore    56.58
Name: occ_pct, dtype: float64
```

4.13 Read CSV File %

In [327…
```
df_august = pd.read_csv("datasets/new_data_august.csv")
df_august.head(3)
```

Out[327…

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

4.14 Append [df & df_august

In [328…
```
latest_df = pd.concat([df, df_august], ignore_index = True, axis = 0)
latest_df.tail(4)
```

Out[328…

	property_id	check_in_date	room_category	successful_bookings	capacity	occ_pct
6503	19558	01-Aug-22	RT1	30	40.0	NaN
6504	19560	01-Aug-22	RT1	20	26.0	NaN
6505	17561	01-Aug-22	RT1	18	26.0	NaN
6506	17564	01-Aug-22	RT1	10	16.0	NaN

4.17 Print revenue realized per city

In [329…

```
df_bookings.head()
```

Out[329…

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

4.18 Merge Tables Bookings and Hotel

In [330…

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

Out[330…

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

4.19 City Wise Revenue

In [331…

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

Out[331…

```
city
Bangalore    420397050
Delhi        294500318
Hyderabad    325232870
Mumbai       668640991
Name: revenue_realized, dtype: int64
```

4.20 Print month by month revenue

In [332…

```
df_date.head(3)
```

Out[332…

	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

4.21 Distinct entries of "mmm yy"

In [333…

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

Out[333…

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

4.22 Display df_bookings_all table

In [334…

```
df_bookings_all.head(3)
```

Out[334…

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

4.23 Column Info

In [337…

```
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
```

4.24 Specify the date format using the format parameter

In [338…

```
df_bookings_all["check_in_date"] = pd.to_datetime(df_bookings_all["check_in_date
df_bookings_all.head(4)
```

Out[338…

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

4.25 Column Info

In [246…

```
df_bookings_all.info()
```

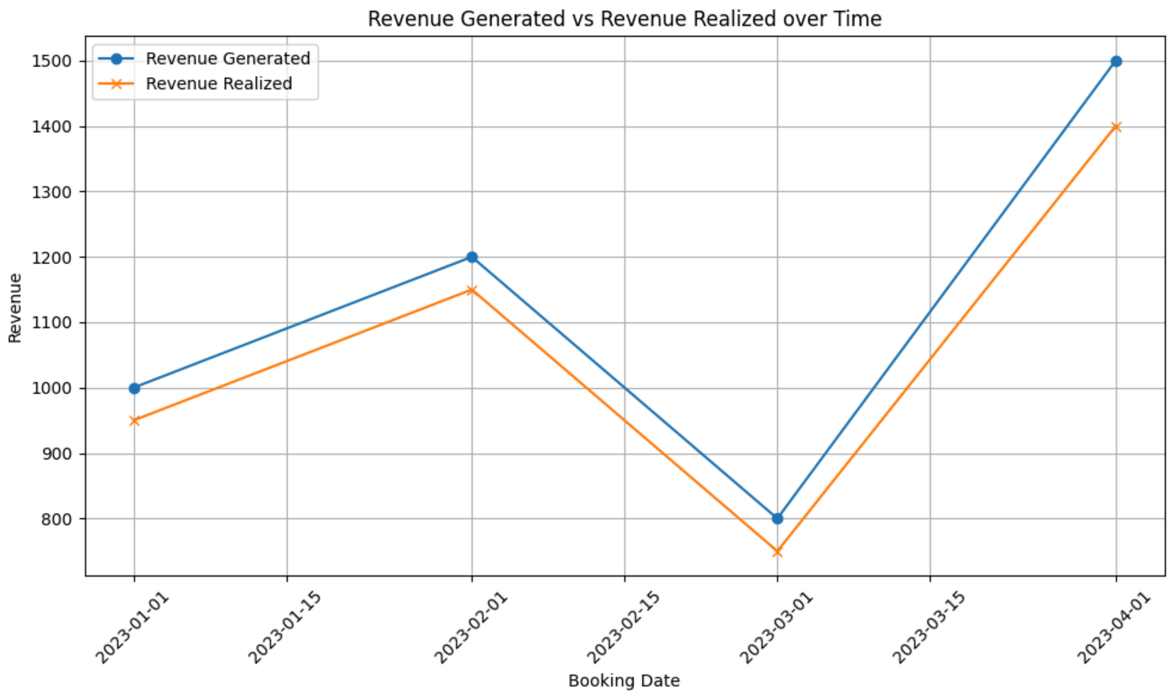
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134590 entries, 0 to 134589
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   booking_id         134590 non-null  object 
 1   property_id        134590 non-null  int64  
 2   booking_date       134590 non-null  object 
 3   check_in_date      134590 non-null  object 
 4   checkout_date      134590 non-null  object 
 5   no_guests          134587 non-null  float64
 6   room_category      134590 non-null  object 
 7   booking_platform   134590 non-null  object 
 8   ratings_given      56683 non-null   float64
 9   booking_status     134590 non-null  object 
 10  revenue_generated  134590 non-null  int64  
 11  revenue_realized   134590 non-null  int64  
 12  property_name      134590 non-null  object 
 13  category           134590 non-null  object 
 14  city               134590 non-null  object 
dtypes: float64(2), int64(3), object(10)
memory usage: 15.4+ MB
```

4.26 Revenue Generated vs Revenue Realized over Time

In [248…

```
import matplotlib.pyplot as plt
import pandas as pd

# Sample DataFrame (replace with your actual df_bookings_all)
data = {
    'sbooking_id': [1, 2, 3, 4],
    'booking_date': ['2023-01-01', '2023-02-01', '2023-03-01', '2023-04-01'],
    'revenue_generated': [1000, 1200, 800, 1500],
    'revenue_realized': [950, 1150, 750, 1400]
}

df_bookings_all = pd.DataFrame(data)
```

```
# Convert booking_date to datetime with format specification
df_bookings_all['booking_date'] = pd.to_datetime(df_bookings_all['booking_date']

# Sort the dataframe by booking_date (optional, but usually a good practice)
df_bookings_all = df_bookings_all.sort_values('booking_date')

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(df_bookings_all['booking_date'], df_bookings_all['revenue_generated'],
plt.plot(df_bookings_all['booking_date'], df_bookings_all['revenue_realized'], l

plt.title('Revenue Generated vs Revenue Realized over Time')
plt.xlabel('Booking Date')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)  # Rotates x-axis labels for better readability if neede

plt.tight_layout()
plt.show()
```

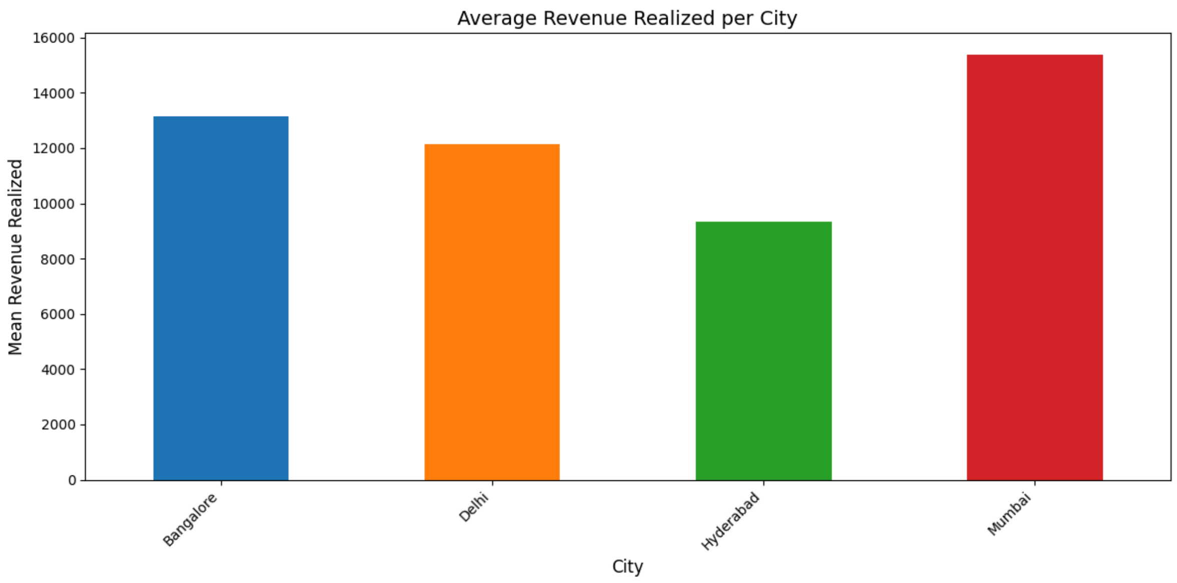


4.27 Average Revenue Realized per City

In [201…

```
import matplotlib.pyplot as plt

# Assuming df_bookings_all is your DataFrame

# Calculate mean revenue realized per city
mean_revenue_per_city = df_bookings_all.groupby("city")["revenue_realized"].mean

# Define custom colors for each city (adjust as needed)
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b
```

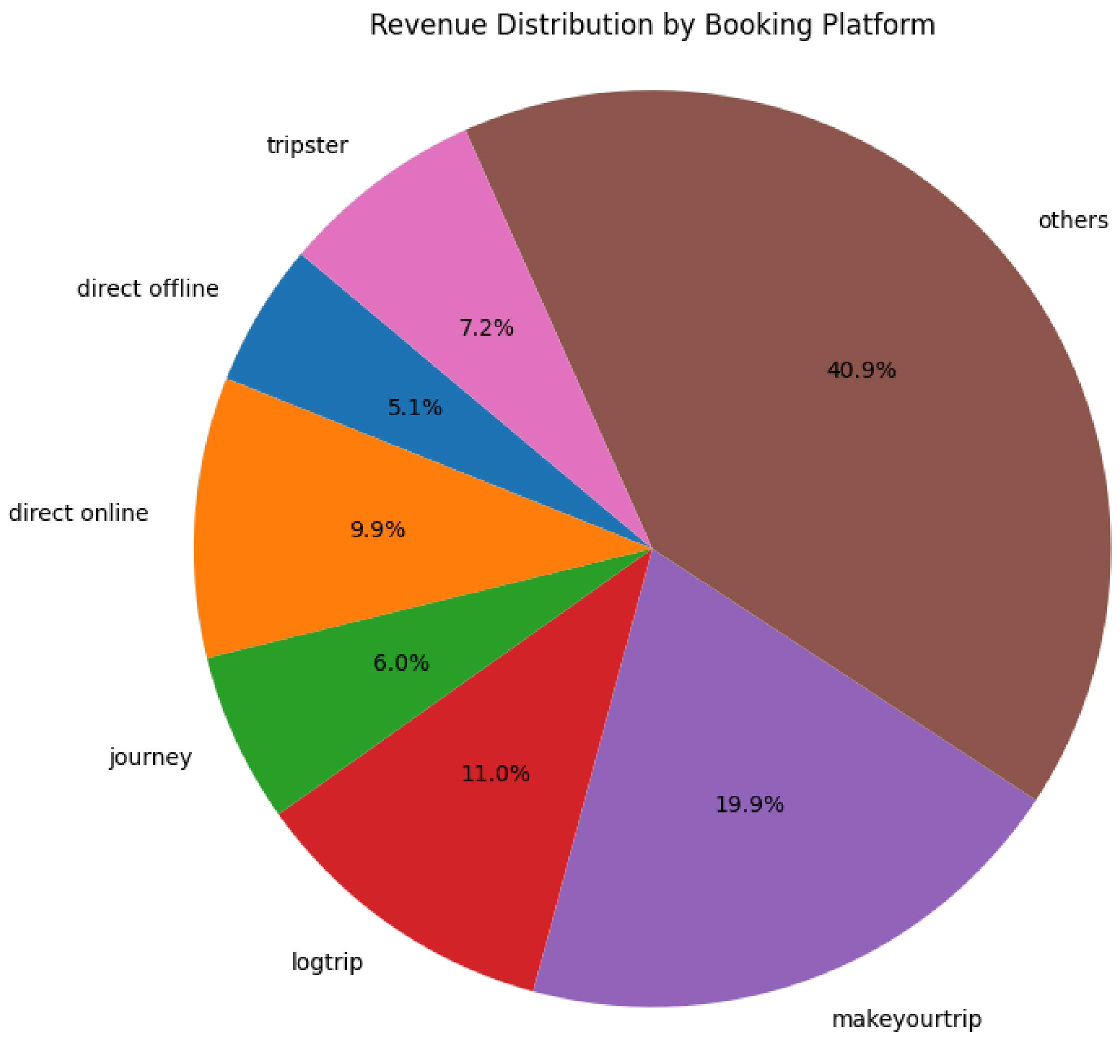
```
# Plotting using pandas built-in plot function
plt.figure(figsize=(12, 6))  # Adjust the figure size

# Plotting the bar chart with custom colors
mean_revenue_per_city.plot(kind="bar", color=custom_colors)

# Customize labels and title
plt.xlabel("City", fontsize=12)
plt.ylabel("Mean Revenue Realized", fontsize=12)
plt.title("Average Revenue Realized per City", fontsize=14)

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45, ha='right')

# Display the plot
plt.tight_layout()  # Ensures labels fit well in the plot area
plt.show()
```



4.28 Revenue Distribution by Booking Platform

In [202…

```
import matplotlib.pyplot as plt

# Grouping and calculating sums
platform_revenue = df_bookings_all.groupby("booking_platform")["revenue_realized

# Plotting the pie chart
plt.figure(figsize=(8, 8))  # Adjust figure size if necessary
plt.pie(platform_revenue, labels=platform_revenue.index, autopct='%1.1f%%', star

# Customizing further
plt.title('Revenue Distribution by Booking Platform')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# Show plot
plt.show()
```

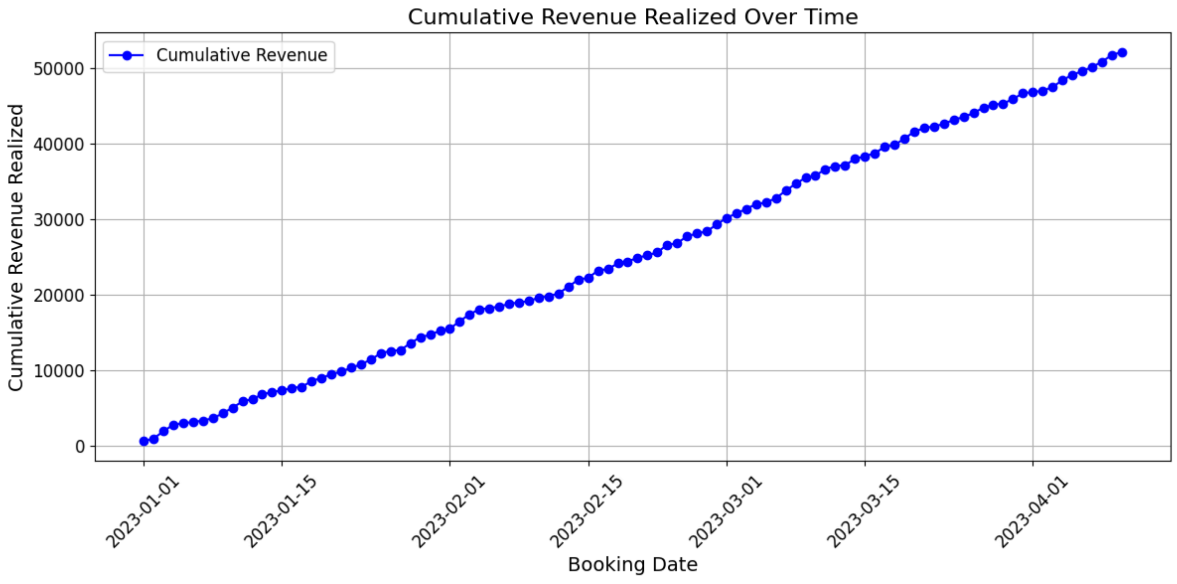


4.29 Cumulative Revenue Realized Over Time

In [203…

```
import pandas as pd
import numpy as np  # Import NumPy for random data generation
import matplotlib.pyplot as plt

# Example DataFrame (replace this with your actual data loading code)
data = {
    'booking_date': pd.date_range(start='2023-01-01', periods=100),
    'revenue_realized': np.random.randint(100, 1000, 100)
}
df_bookings_all = pd.DataFrame(data)

# Convert booking_date to datetime if not already
df_bookings_all['booking_date'] = pd.to_datetime(df_bookings_all['booking_date']

# Sort DataFrame by booking_date
df_bookings_all.sort_values(by='booking_date', inplace=True)

# Plotting
plt.figure(figsize=(12, 6))
```

```
# Calculate cumulative revenue realized on-the-fly
cumulative_revenue = df_bookings_all['revenue_realized'].cumsum()

plt.plot(df_bookings_all['booking_date'], cumulative_revenue, marker='o', linest
plt.title('Cumulative Revenue Realized Over Time', fontsize=16)
plt.xlabel('Booking Date', fontsize=14)
plt.ylabel('Cumulative Revenue Realized', fontsize=14)
plt.xticks(fontsize=12, rotation=45)
plt.yticks(fontsize=12)
plt.grid(True)
plt.legend(loc='upper left', fontsize=12)
plt.tight_layout()
plt.show()
```


Cumulative Revenue Realized Over Time
Cumulative Revenue
Cumulative Revenue Realized
Booking Date