Step 1: load Prepared data

In [40]:

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

In [41]:

```
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12,6)

# load the cleaned data
df = pd.read_csv("Cleaned_fitness_classes_data.csv")


print(df.shape)
print(df.head())
print(df.info())
```

```
(3271, 14)
  Site_ID               Class_Name   End_Date Start_Time  Capacity  Booked  \
0     HXP  20-20-20  2.45pm-3.45pm   8-Apr-18   14:45:00        25      12   
1     HXP  20-20-20  2.45pm-3.45pm  15-Apr-18   14:45:00        25      15   
2     HXP  20-20-20  2.45pm-3.45pm  22-Apr-18   14:45:00        25      14   
3     HXP  20-20-20  2.45pm-3.45pm  29-Apr-18   14:45:00        25       9   
4     HXP  20-20-20  2.45pm-3.45pm   6-May-18   14:45:00        25       7   

   Price_INR End_DateTime Weekday  Hour  Capacity_Utilization  Revenue  \
0      499.0   2018-04-08  Sunday    14                  48.0   5988.0   
1      499.0   2018-04-15  Sunday    14                  60.0   7485.0   
2      499.0   2018-04-22  Sunday    14                  56.0   6986.0   
3      499.0   2018-04-29  Sunday    14                  36.0   4491.0   
4      499.0   2018-05-06  Sunday    14                  28.0   3493.0   

   MaxBookes Dataset  
0         25   Train  
1         25   Train  
2         25   Train  
3         25   Train  
4         25   Train  
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3271 entries, 0 to 3270
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Site_ID               3271 non-null   object 
 1   Class_Name            3271 non-null   object 
 2   End_Date              3271 non-null   object 
 3   Start_Time            3271 non-null   object 
 4   Capacity              3271 non-null   int64  
 5   Booked                3271 non-null   int64  
 6   Price_INR             3271 non-null   float64
 7   End_DateTime          3271 non-null   object 
 8   Weekday               3271 non-null   object 
 9   Hour                  3271 non-null   int64  
 10  Capacity_Utilization  3271 non-null   float64
 11  Revenue               3271 non-null   float64
 12  MaxBookes             3271 non-null   int64  
 13  Dataset               3271 non-null   object 
dtypes: float64(3), int64(4), object(7)
memory usage: 357.9+ KB
None
```

Data Summary

The dataset has 3,271 entries and 14 columns, including booking, price, capacity, and time details. There is no missing data, so analysis can proceed without data imputation.

Step 2: Descriptive Analysis

In [42]:

```
#Basics Statistics
print(df.describe())

#Distribution of bookings
print("Mean booked:", df["Booked"].mean())
print("Median booked:", df["Booked"].median())
print("Standard deviation:", df["Booked"].std())
```

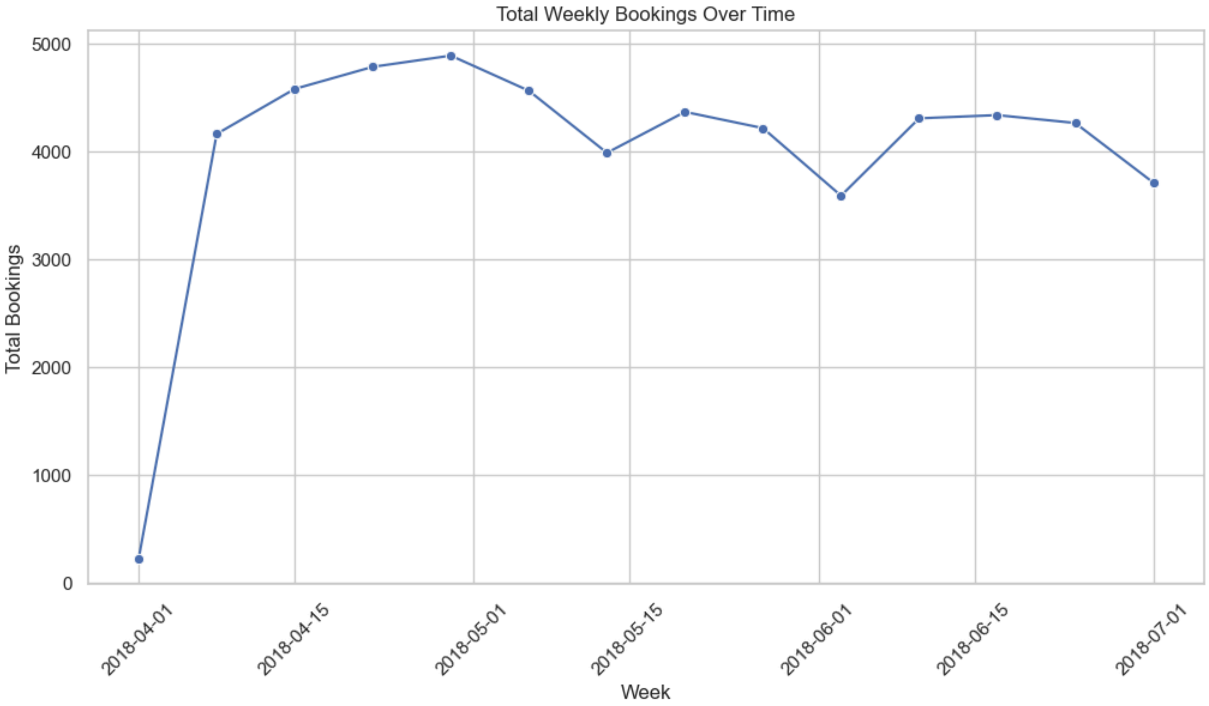
```
          Capacity       Booked    Price_INR         Hour  \
count  3271.000000  3271.000000  3271.000000  3271.000000   
mean     32.224396    17.125955  1852.714460    13.370223   
std      15.121398     9.515352   792.636673     3.920957   
min       2.000000     1.000000   499.000000     6.000000   
25%      24.000000    10.000000  1299.000000    10.000000   
50%      30.000000    16.000000  1499.000000    12.000000   
75%      35.000000    24.000000  2499.000000    17.000000   
max      70.000000    64.000000  3999.000000    20.000000   

       Capacity_Utilization        Revenue   MaxBookes  
count           3271.000000    3271.000000  3271.000000  
mean              57.232947   30288.468664    32.224396  
std               31.906499   20363.598835    15.121398  
min                1.670000     999.000000     2.000000  
25%               34.290000   14990.000000    24.000000  
50%               55.560000   26982.000000    30.000000  
75%               80.000000   41979.000000    35.000000  
max              516.670000  134955.000000    70.000000  
Mean booked: 17.125955365331702
Median booked: 16.0
Standard deviation: 9.515352256194983
```

Step 3: Demand Over Time

In [43]:

```
# Convert End_Date to datetime (if not already)
df["End_Date"] = pd.to_datetime(df["End_Date"])

# Group by week
weekly_bookings = df.groupby(pd.Grouper(key="End_Date", freq="W"))["Booked"].sum().

# Plot
plt.figure(figsize=(12,6))
sns.lineplot(data=weekly_bookings, x="End_Date", y="Booked", marker="o")
plt.title("Total Weekly Bookings Over Time")
plt.xlabel("Week")
plt.ylabel("Total Bookings")
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\2772184835.py:2: UserWarning: Cou
ld not infer format, so each element will be parsed individually, falling back to `d
ateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df["End_Date"] = pd.to_datetime(df["End_Date"])
```

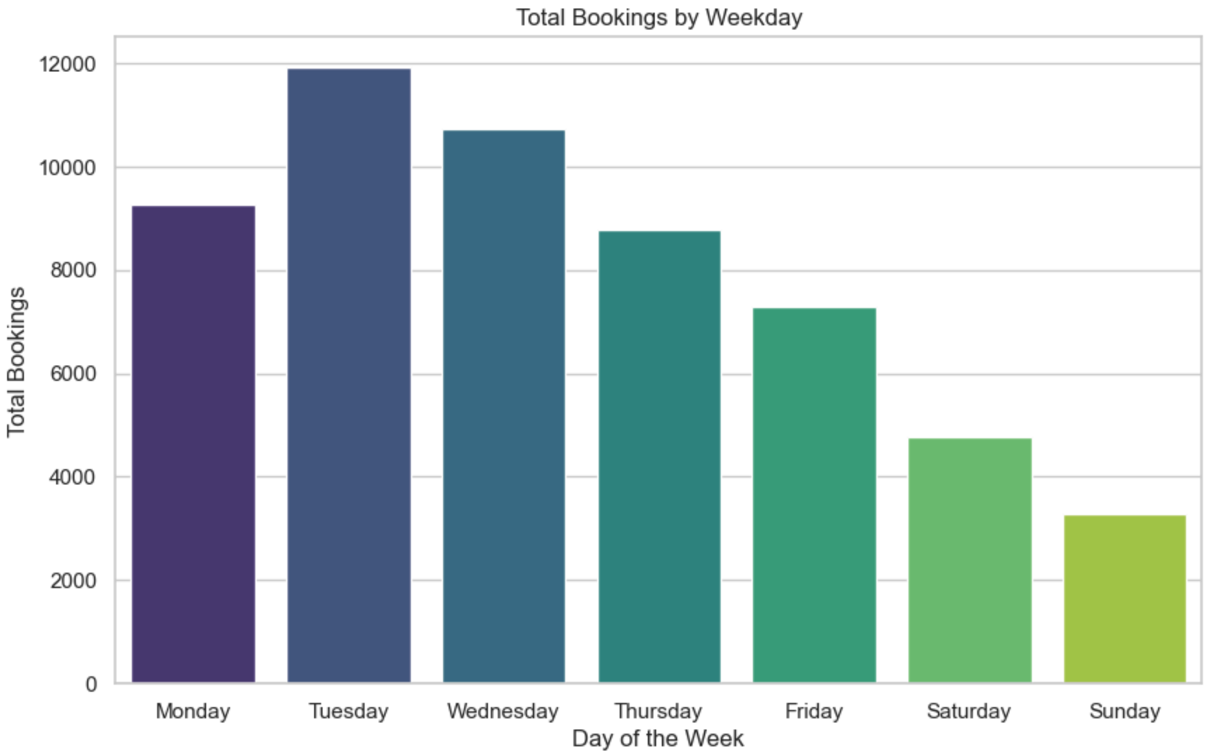


Step 4: Weekday Analysis

In [44]:

```python
# Group by weekday
weekday_bookings = df.groupby("Weekday")["Booked"].sum().reindex(
    ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
).reset_index()

# Plot
plt.figure(figsize=(10,6))
sns.barplot(data=weekday_bookings, x="Weekday", y="Booked", palette="viridis")
plt.title("Total Bookings by Weekday")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bookings")
plt.show()
```

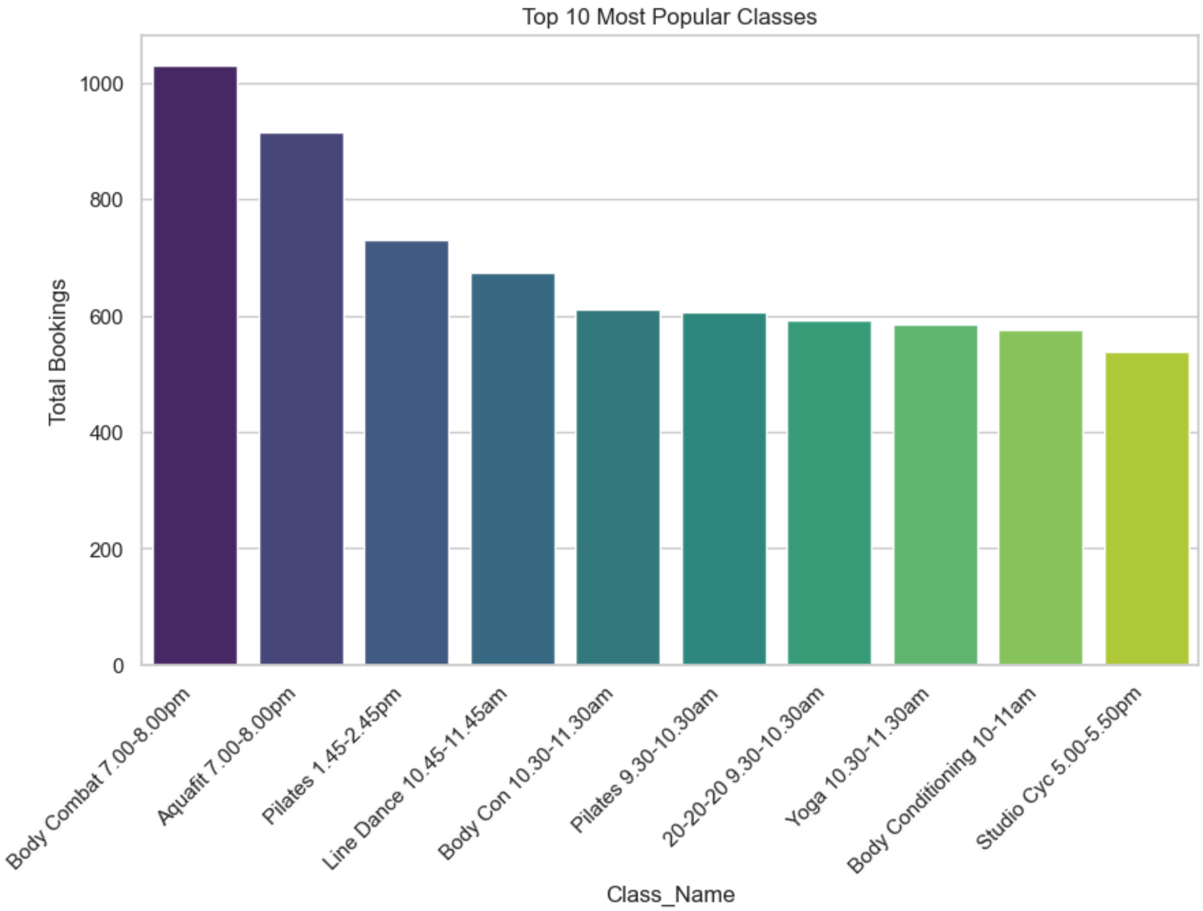
```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\2644578095.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=weekday_bookings, x="Weekday", y="Booked", palette="viridis")
```



Step 6: Class Popularity Analysis

```
In [45]:
# Top 10 classes by total bookings
top_classes = class_bookings.head(10)

plt.figure(figsize=(10,6))
sns.barplot(data=top_classes, x="Class_Name", y="Booked", palette="viridis")
plt.title("Top 10 Most Popular Classes")
plt.xticks(rotation=45, ha="right")
plt.ylabel("Total Bookings")
plt.show()
```

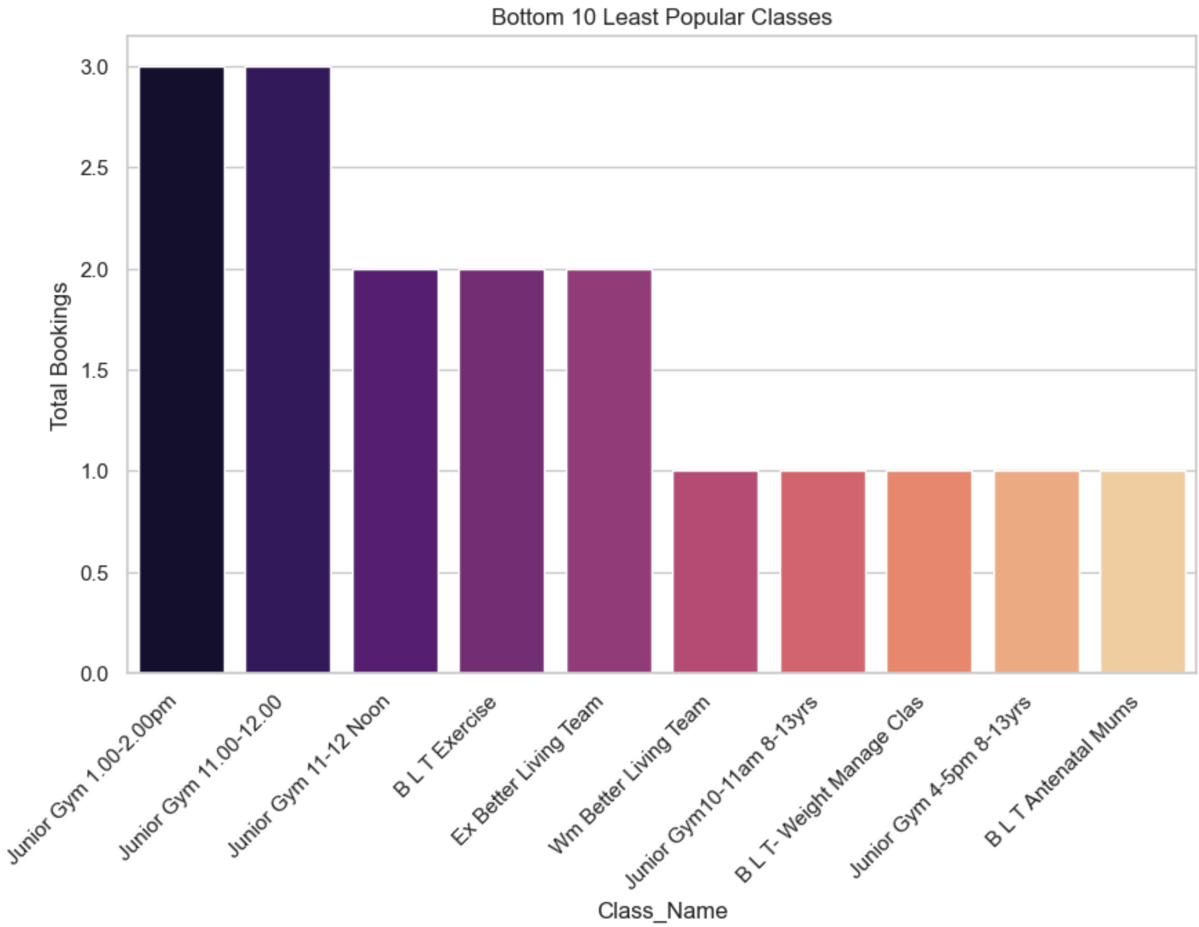
```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\2955321013.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=top_classes, x="Class_Name", y="Booked", palette="viridis")
```



In [46]:

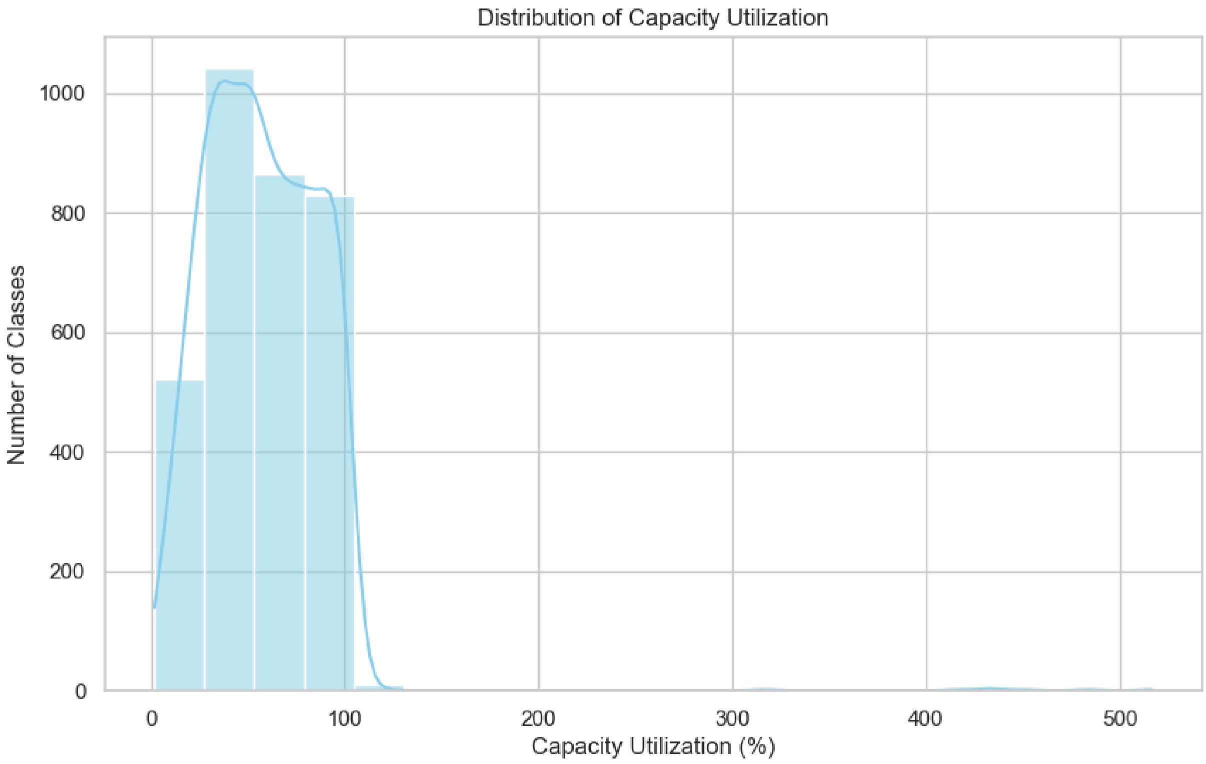
```python
# Bottom 10 classes by total bookings
bottom_classes = class_bookings.tail(10)

plt.figure(figsize=(10,6))
sns.barplot(data=bottom_classes, x="Class_Name", y="Booked", palette="magma")
plt.title("Bottom 10 Least Popular Classes")
plt.xticks(rotation=45, ha="right")
plt.ylabel("Total Bookings")
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\1715027743.py:5: FutureWarning: 

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=bottom_classes, x="Class_Name", y="Booked", palette="magma")
```



Step7: Capacity Utilization Distribution

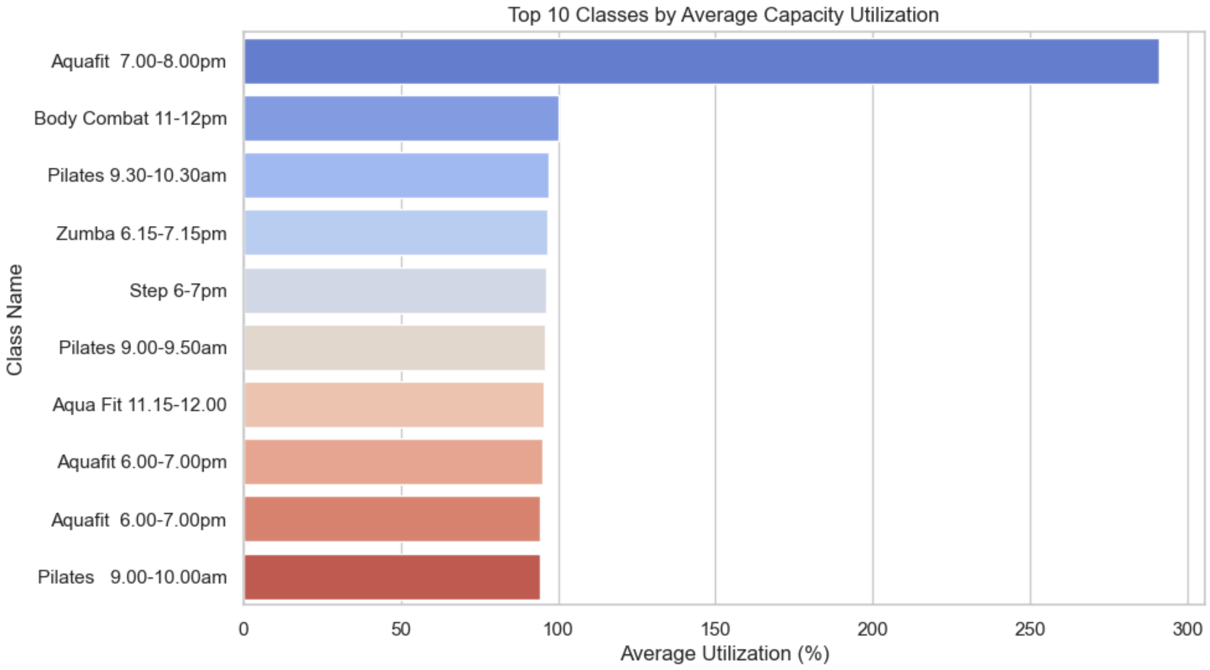
In [47]:
```
plt.figure(figsize=(10,6))
sns.histplot(df["Capacity_Utilization"], bins=20, kde=True, color="skyblue")
plt.title("Distribution of Capacity Utilization")
plt.xlabel("Capacity Utilization (%)")
plt.ylabel("Number of Classes")
plt.show()
```



In [48]:

```
class_utilization = df.groupby("Class_Name")["Capacity_Utilization"].mean().sort_va

plt.figure(figsize=(10,6))
sns.barplot(x=class_utilization.values, y=class_utilization.index, palette="coolwar
plt.title("Top 10 Classes by Average Capacity Utilization")
plt.xlabel("Average Utilization (%)")
plt.ylabel("Class Name")
plt.show()
```

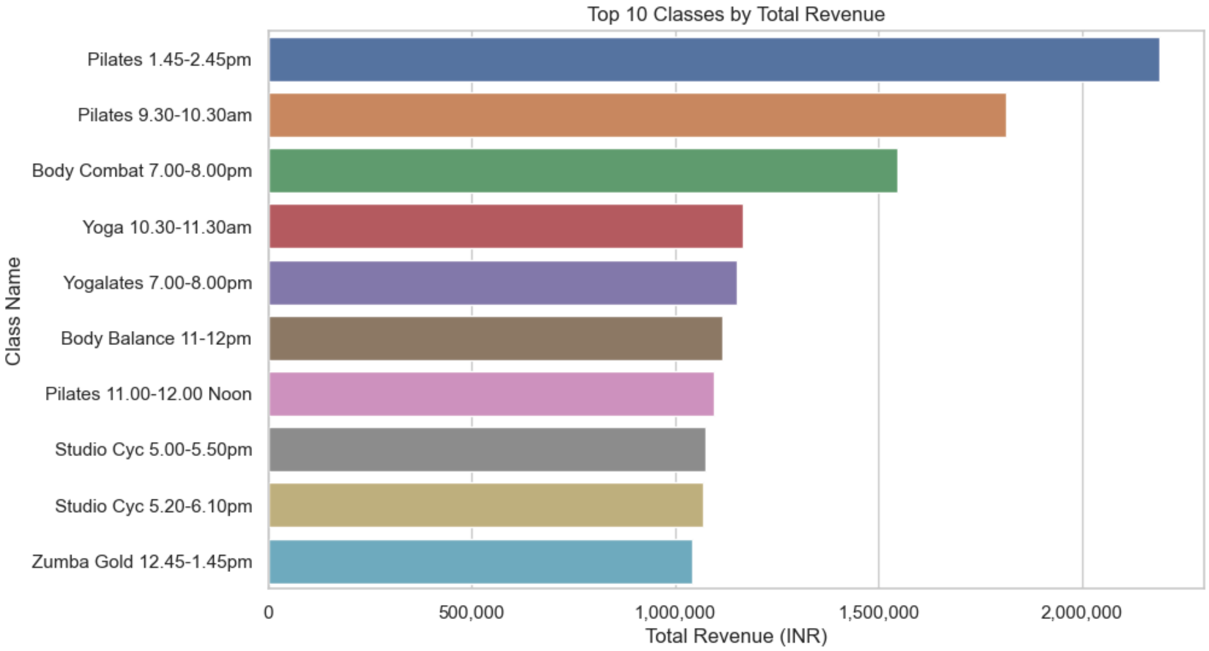
```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\4193308850.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=class_utilization.values, y=class_utilization.index, palette="coolwa
rm")
```



Step 8: Revenue Analysis

In [49]:

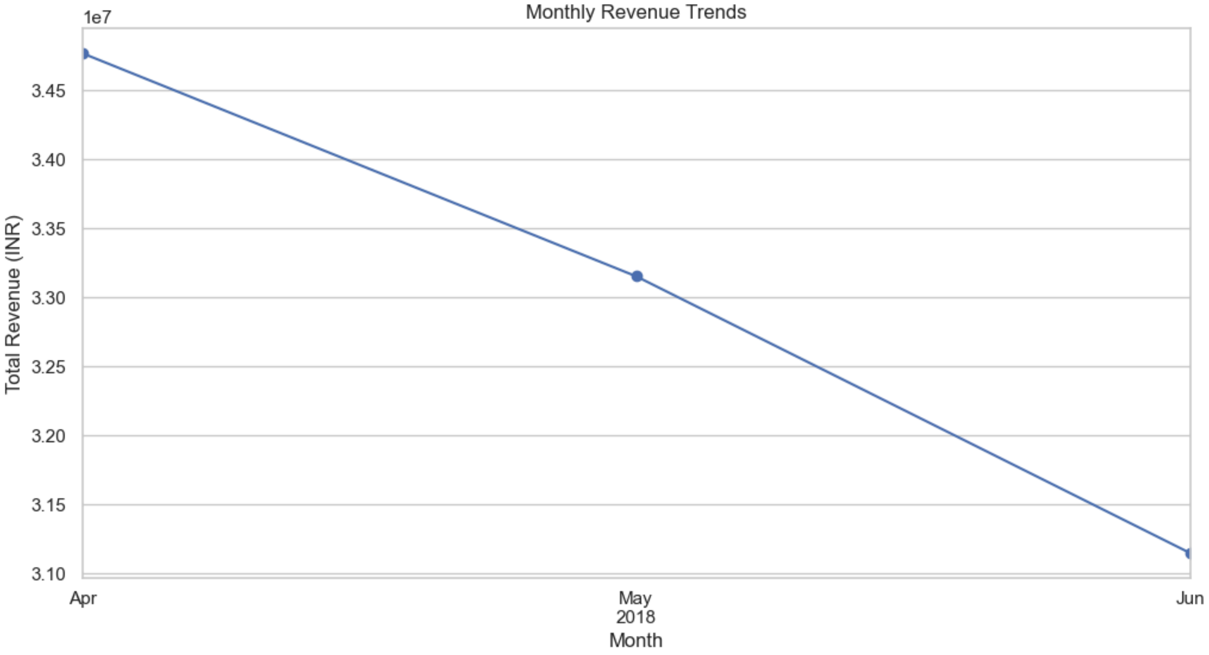
```
#Revenue by Class (Top 10)
import matplotlib.ticker as mtick
class_revenue = df.groupby("Class_Name")["Revenue"].sum().sort_values(ascending = F
# print(class_revenue)
plt.figure(figsize=(10, 6))
sns.barplot( x = class_revenue.values, y = class_revenue.index, hue = class_revenue
plt.title("Top 10 Classes by Total Revenue")
plt.xlabel("Total Revenue (INR)")
plt.ylabel("Class Name")
plt.gca().xaxis.set_major_formatter(mtick.StrMethodFormatter('{x:,.0f}'))
plt.show()
```



Step 9: Revenue Trends Over Time

In [50]:

```
monthly_revenue = df.groupby(df["End_Date"].dt.to_period("M"))["Revenue"].sum()

plt.figure(figsize=(12,6))
monthly_revenue.plot(marker="o")
plt.title("Monthly Revenue Trends")
plt.xlabel("Month")
plt.ylabel("Total Revenue (INR)")
plt.xticks(rotation=45)
plt.show()
```



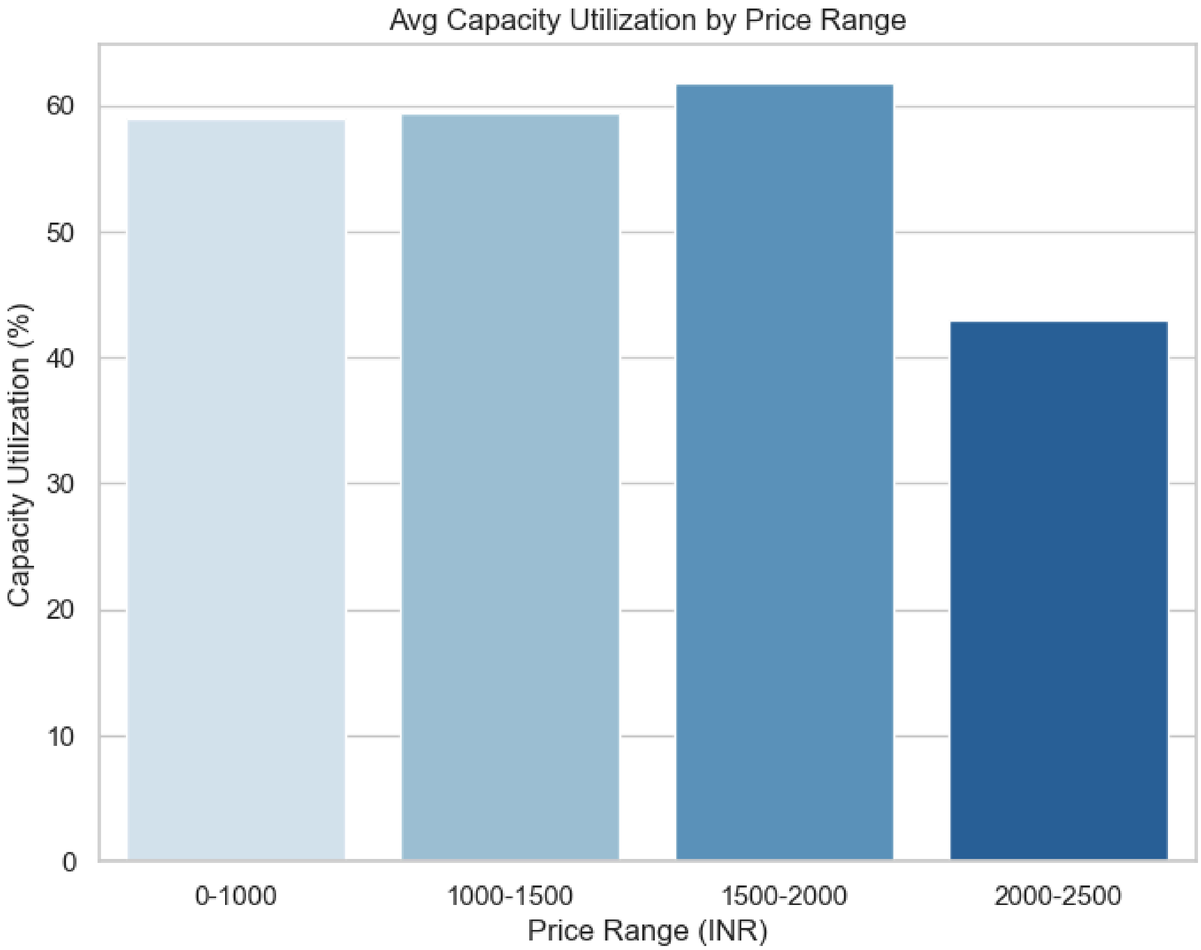
Step 10: Price vs Utilization Relationship

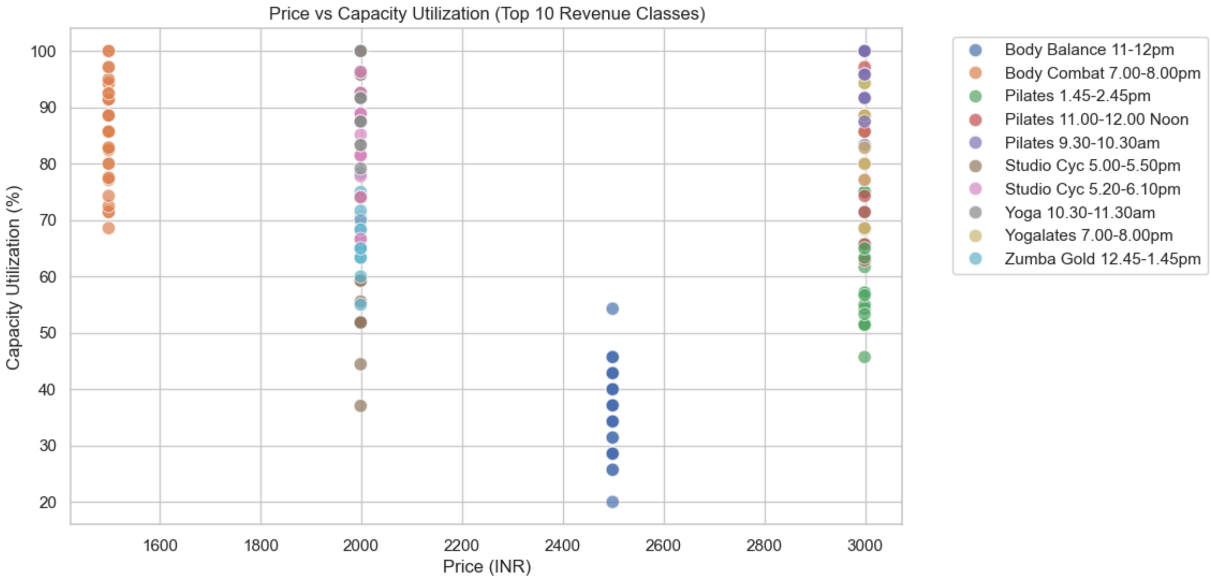
In [51]:

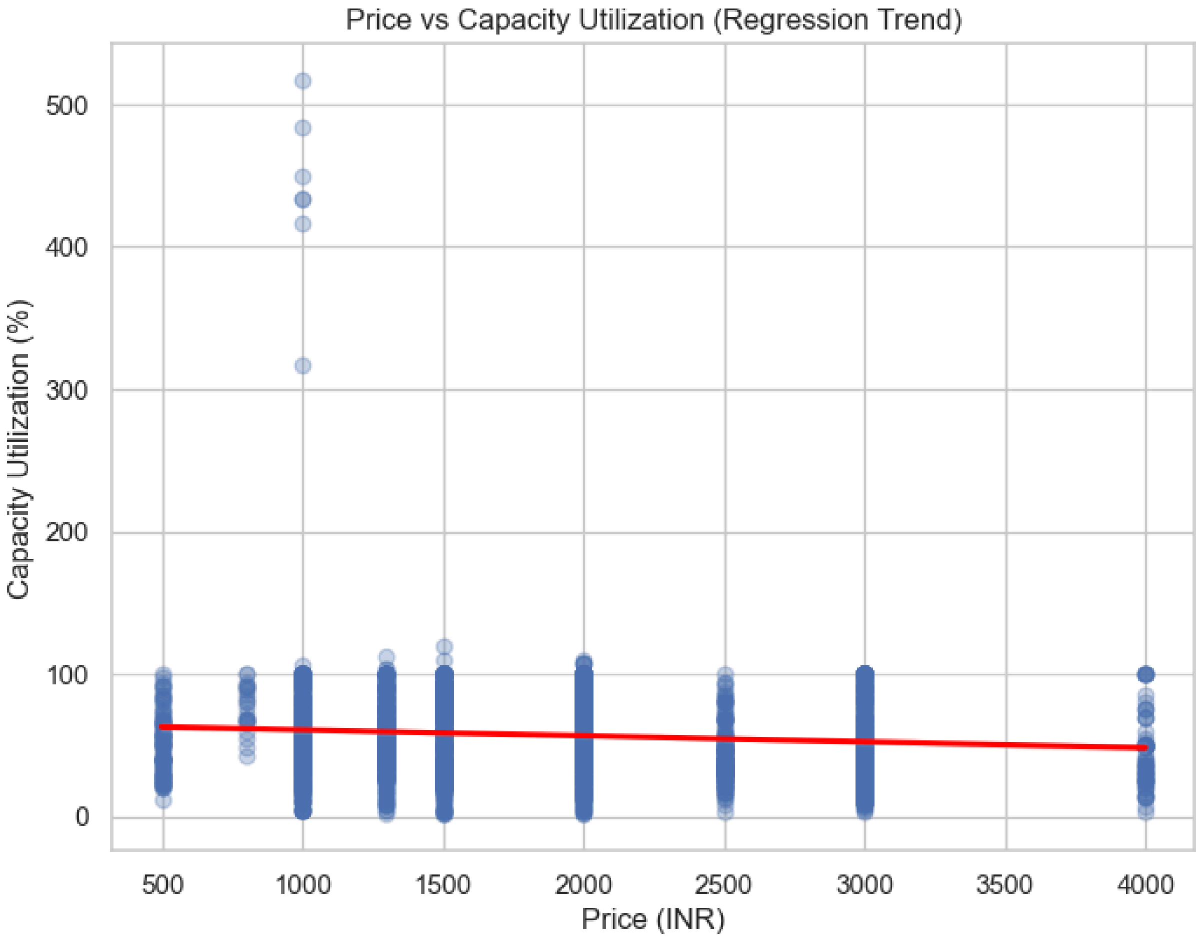
```
df["Price_Range"] = pd.cut(
    df["Price_INR"],
    bins=[0, 1000, 1500, 2000, 2500],
    labels=["0-1000", "1000-1500", "1500-2000", "2000-2500"]
)

price_util = df.groupby("Price_Range")["Capacity_Utilization"].mean().reset_index()

plt.figure(figsize=(8,6))
sns.barplot(data=price_util, x="Price_Range", y="Capacity_Utilization", palette="Bl
plt.title("Avg Capacity Utilization by Price Range")
plt.ylabel("Capacity Utilization (%)")
plt.xlabel("Price Range (INR)")
plt.show()


# ------------------------------------------------------------
# 2. Top 10 Classes by Revenue (Price vs Utilization)
# ------------------------------------------------------------
top_classes = (
    df.groupby("Class_Name")["Revenue"]
    .sum()
    .sort_values(ascending=False)
    .head(10)
    .index
)

df_top = df[df["Class_Name"].isin(top_classes)]

plt.figure(figsize=(10,6))
sns.scatterplot(
    data=df_top,
    x="Price_INR",
    y="Capacity_Utilization",
    hue="Class_Name",
    alpha=0.7,
    s=80
)
plt.title("Price vs Capacity Utilization (Top 10 Revenue Classes)")
plt.xlabel("Price (INR)")
plt.ylabel("Capacity Utilization (%)")
plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
plt.show()


# ------------------------------------------------------------
# 3. Regression (General Trend Across All Classes)
# ------------------------------------------------------------
plt.figure(figsize=(8,6))
sns.regplot(
    data=df,
```

```python
    x="Price_INR",
    y="Capacity_Utilization",
    scatter_kws={"alpha":0.3, "s":40},
    line_kws={"color":"red"}
)
plt.title("Price vs Capacity Utilization (Regression Trend)")
plt.xlabel("Price (INR)")
plt.ylabel("Capacity Utilization (%)")
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\2457167122.py:7: FutureWarning: T
he default of observed=False is deprecated and will be changed to True in a future v
ersion of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  price_util = df.groupby("Price_Range")["Capacity_Utilization"].mean().reset_index
()
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\2457167122.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=price_util, x="Price_Range", y="Capacity_Utilization", palette="B
lues")
```







Overall Story from these graphs:

Low to mid-priced classes (₹0–1500) are most efficient at filling capacity.

Premium classes (₹2000+) earn revenue but don't fill as well → so they rely on fewer people paying higher prices.

For future pricing strategy → keep a mix:

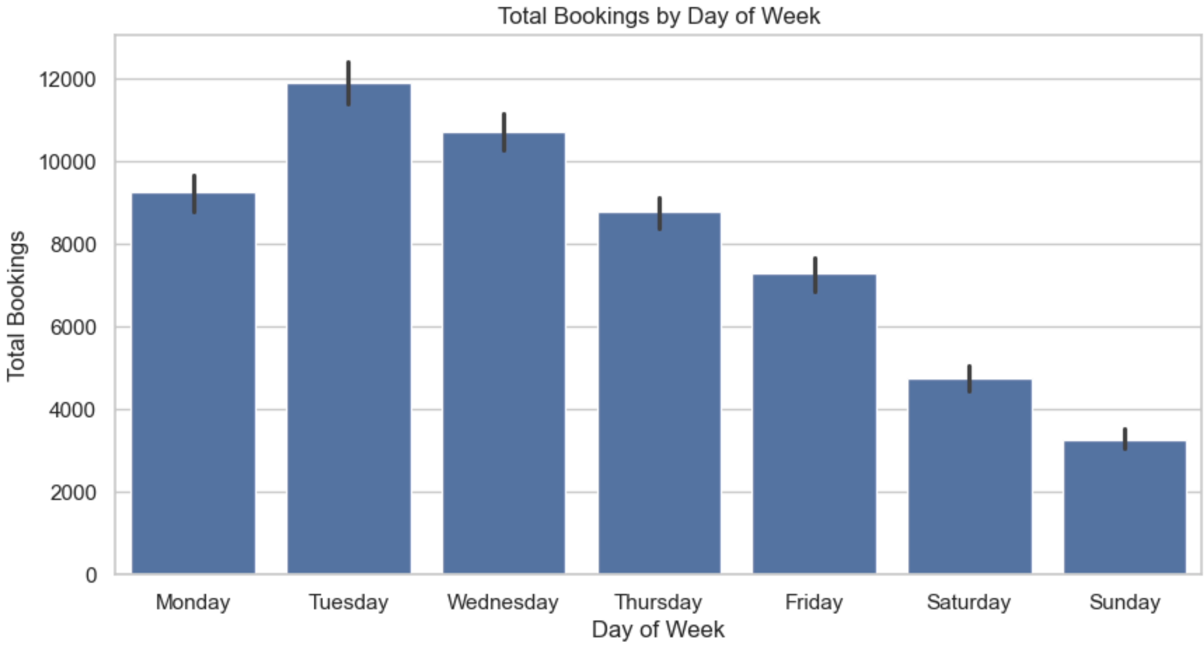
Affordable classes to keep utilization high.

A few premium classes for revenue, but don’t overprice most classes.

Step 11: Day-of-Week Analysis

In [52]:

```
plt.figure(figsize=(10,5))
sns.barplot(data=df, x="Weekday", y="Booked", estimator="sum",
            order=["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","S
plt.title("Total Bookings by Day of Week")
plt.xlabel("Day of Week")
plt.ylabel("Total Bookings")
plt.show()
```

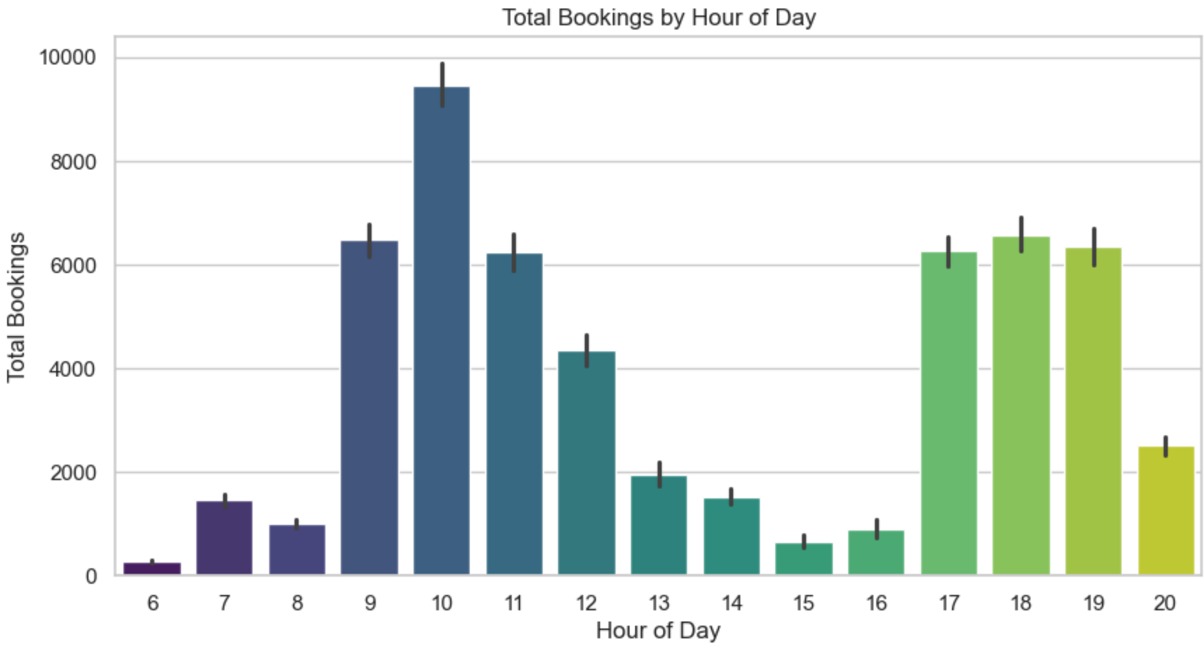


Step 12: Time-of-Day Analysis

In [53]:

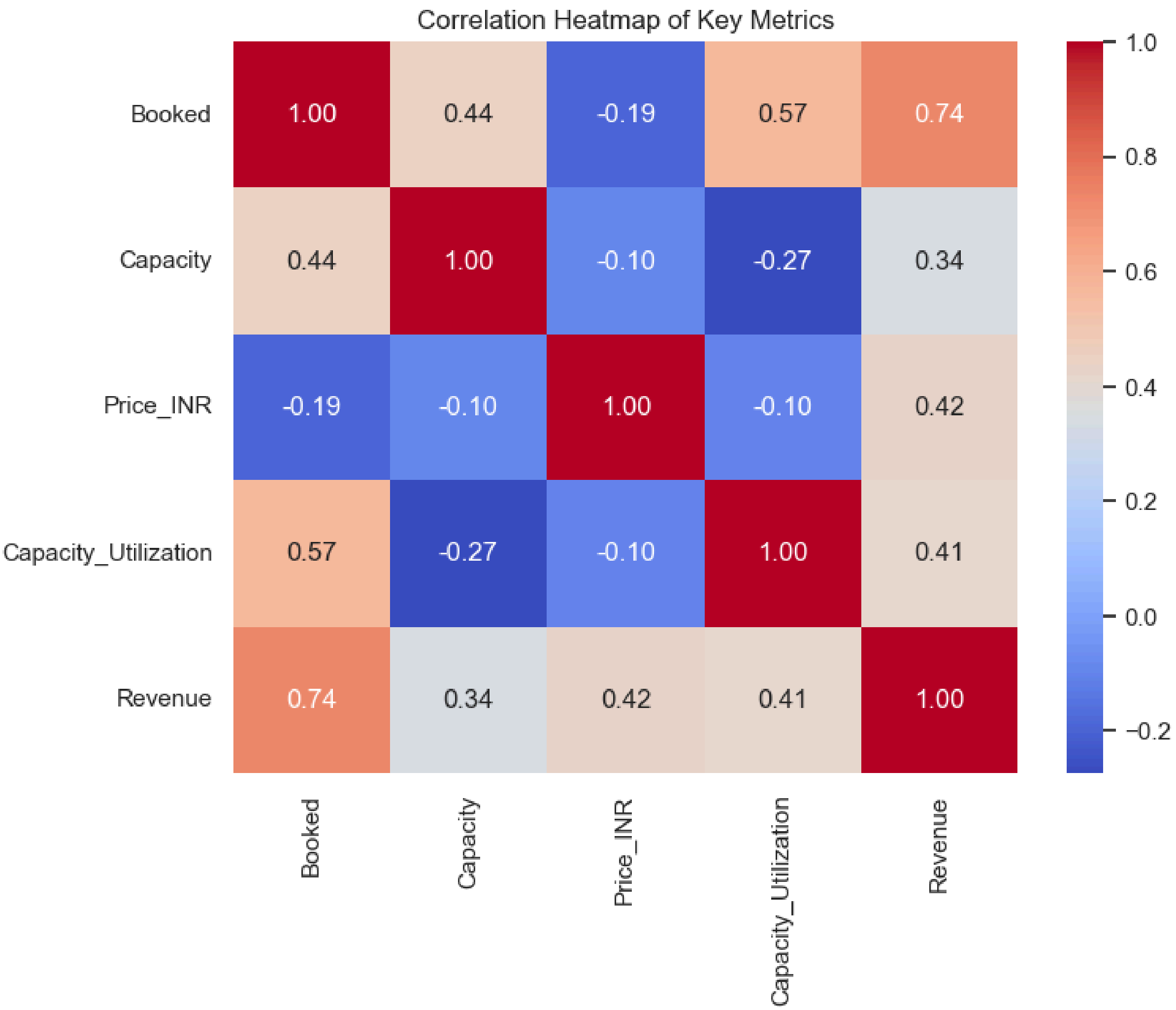
```
plt.figure(figsize=(10,5))
sns.barplot(data=df, x="Hour", y="Booked", estimator="sum", palette="viridis")
plt.title("Total Bookings by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Total Bookings")
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_12312\3019291248.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=df, x="Hour", y="Booked", estimator="sum", palette="viridis")
```



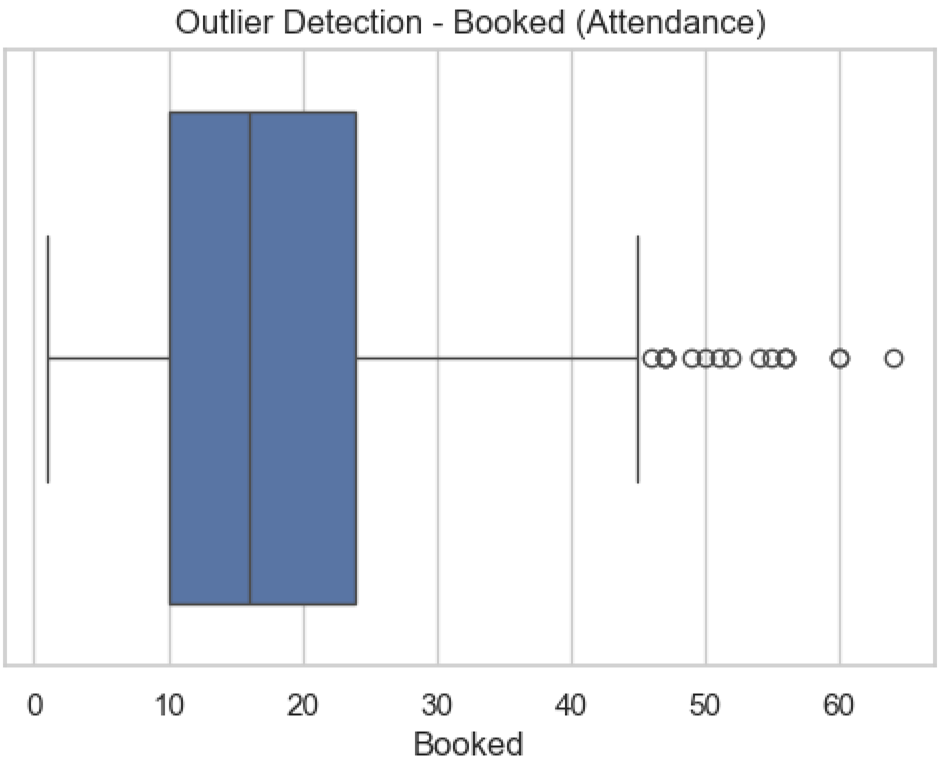
Step 13: Correlation Heatmap

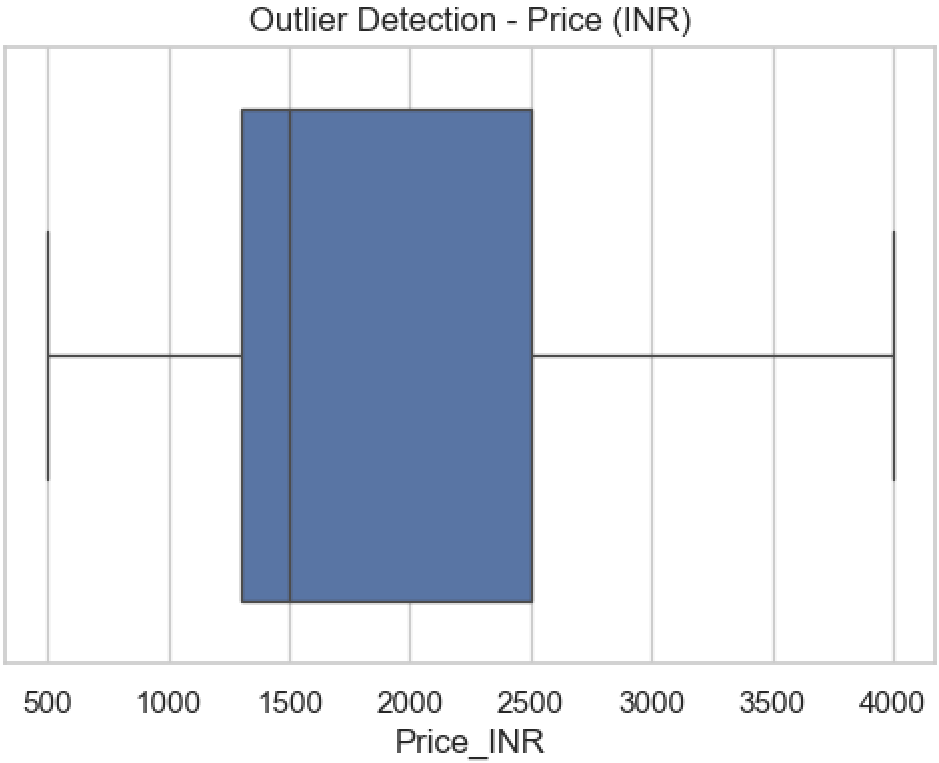
In [54]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df[["Booked","Capacity","Price_INR","Capacity_Utilization","Revenue"]].
            annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap of Key Metrics")
plt.show()
```



Step 14: Outlier Detection (Attendance & Price)

In [56]:
```
# Outliers in Booked (attendance)
plt.figure(figsize=(6,4))
sns.boxplot(x=df['Booked'])
plt.title("Outlier Detection - Booked (Attendance)")
plt.show()

# Outliers in Price
plt.figure(figsize=(6,4))
sns.boxplot(x=df['Price_INR'])
plt.title("Outlier Detection - Price (INR)")
plt.show()
```

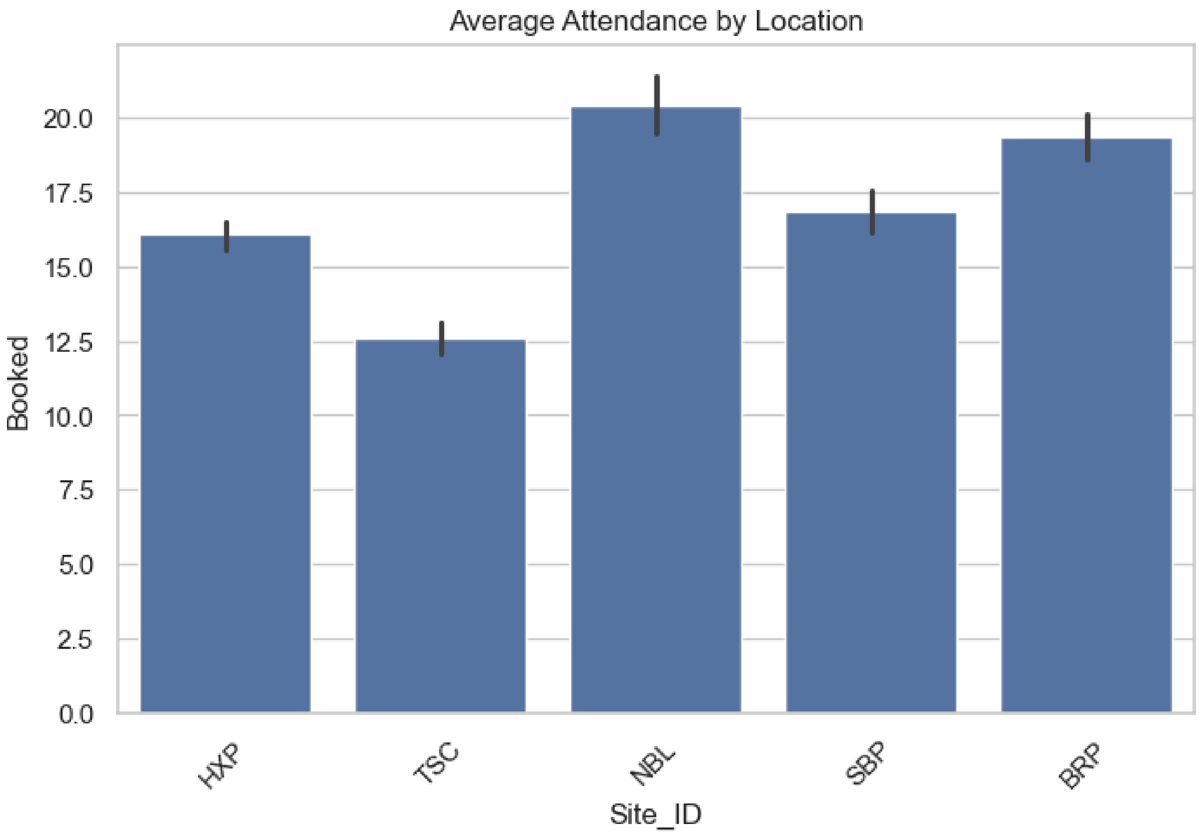




Step 15: Attendance by Location

In [57]:

```
plt.figure(figsize=(8,5))
sns.barplot(x="Site_ID", y="Booked", data=df, estimator="mean")
plt.title("Average Attendance by Location")
plt.xticks(rotation=45)
plt.show()
```



Key Insights from EDA

1. Overall Demand & Descriptive Stats

   · Dataset includes 3,271 entries with 14 columns (bookings, price, utilization, revenue, etc.).

   · Average bookings per class show high variance - some classes are consistently full, others underutilized.

2. Demand Over Time

   . Weekly booking trends reveal seasonal demand fluctuations.

   . Strong peaks during certain months and weeks, suggesting higher seasonal interest in fitness (likely New Year or pre-summer).

3. Weekday Analysis

   · Bookings are highest on Mondays and midweek (Tuesday/Wednesday).

   · Demand dips on Fridays and partially on weekends, indicating opportunity for weekend promotions.

4. Class Popularity

· Yoga, Spin, HIIT consistently among top booked classes.

. Least popular classes show very low bookings - candidates for price discounts or removal from schedule.

5. Capacity Utilization

. Many classes run below 50% utilization - wasted capacity.

· Top classes achieve >80% utilization, mostly in premium times (mornings, evenings).

. Indicates that pricing strategy should aim to shift demand to underutilized slots.

6. Revenue Analysis

· Top revenue comes from popular, high-priced classes (e.g., Spin, Yoga).

· Some less popular classes still generate revenue due to premium pricing, but their utilization is weak.

· Suggests revenue is disproportionately dependent on a few class types.

7. Revenue Trends Over Time

. Monthly revenue shows growth trend, but fluctuations highlight sensitivity to seasonality.

. Dynamic pricing could stabilize revenue by smoothing out dips.

8. Price vs Utilization Relationship

. Low to mid-priced classes (₹0-1500) have the highest utilization rates.

· Premium classes (₹2000+) earn revenue but attract fewer participants.

· Clear case for a mixed pricing model:

```
• Affordable classes - drive volume/utilization.


• Premium classes - maintain exclusivity and margin.
```

9. Day-of-Week Demand

· Confirms earlier weekday trend: demand peaks midweek, dips on Fridays/weekends.

· Suggests weekday premiums and weekend discounts as part of pricing rules.

In [ ]:

```
In [ ]:
```