SALES & PROFIT ANALYSIS

NAME : Abir Ghosh

EMAIL : abirghosh033@gmail.com

PHONE : +919748248459

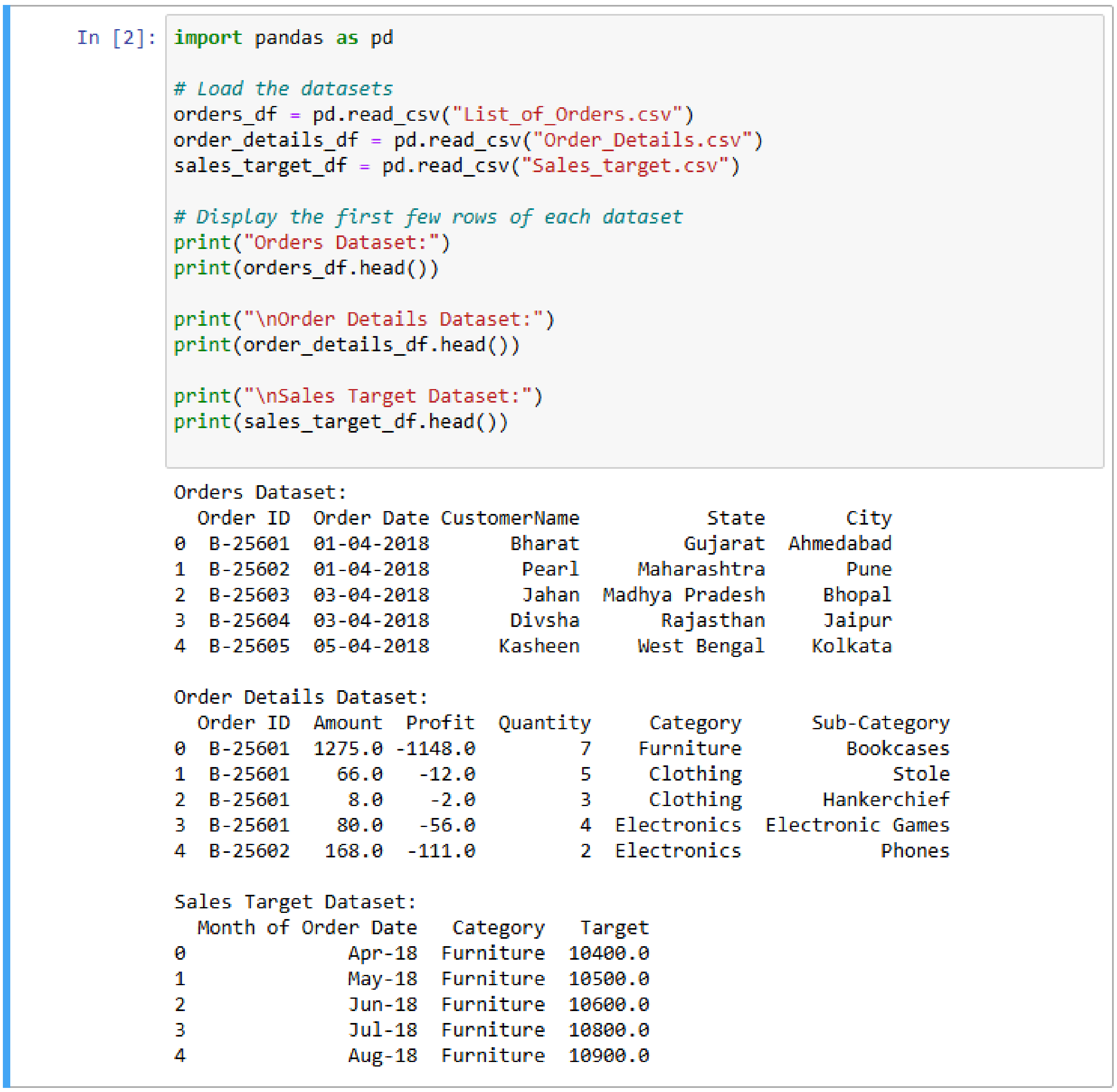
Drive Link for all the files :
https://drive.google.com/drive/folders/1dikuzHY9P_ymfxjiOmvj4J8pvGJU22S2?usp=sharing

Github Link of the Project :
https://github.com/AGTech27/Sales_Analaysis_JAR__BA_Assignment.git

QUESTION 1

Part 1: Sales and Profitability Analysis

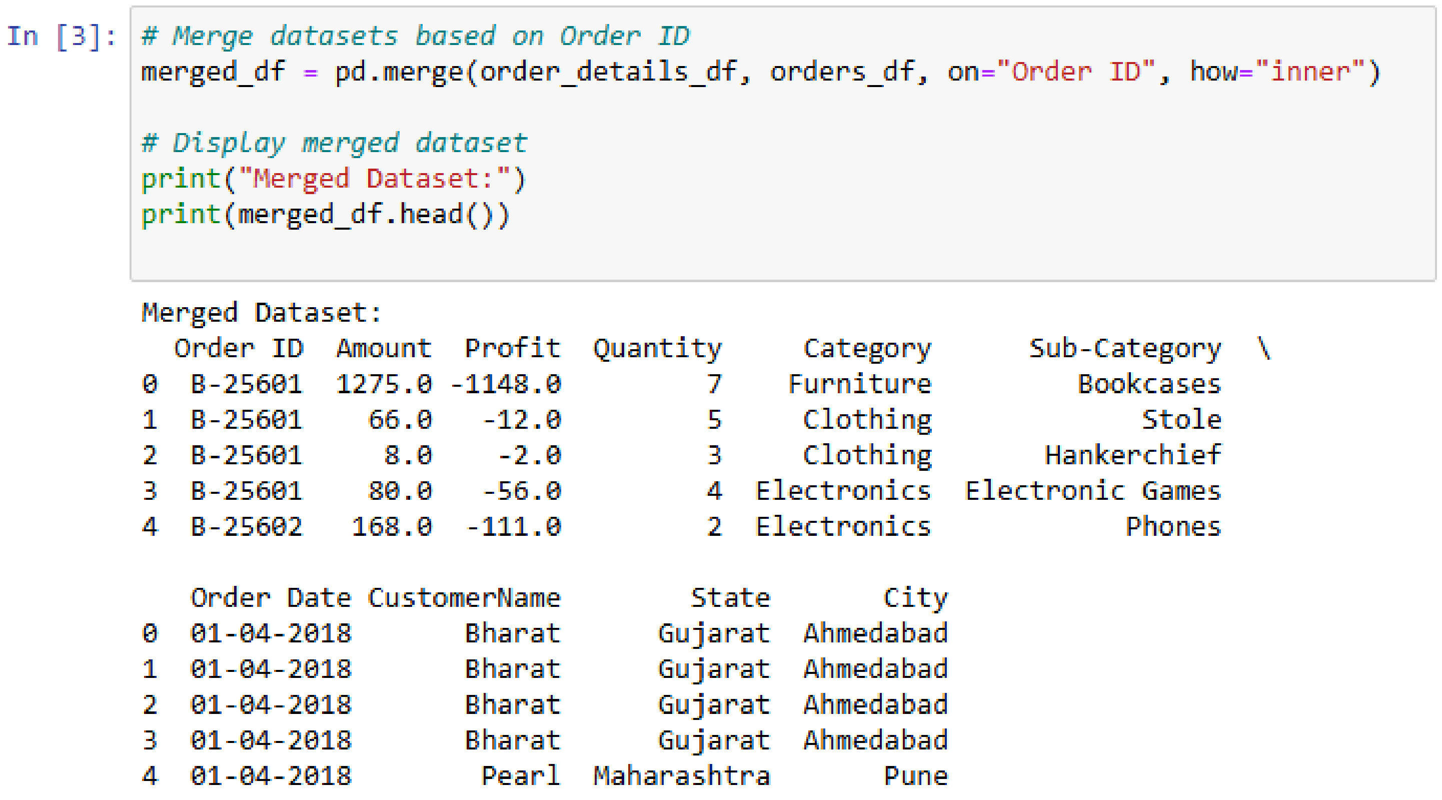
Step 1: Import Libraries and Load Data



In [2]:
```python
import pandas as pd

# Load the datasets
orders_df = pd.read_csv("List_of_Orders.csv")
order_details_df = pd.read_csv("Order_Details.csv")
sales_target_df = pd.read_csv("Sales_target.csv")

# Display the first few rows of each dataset
print("Orders Dataset:")
print(orders_df.head())

print("\nOrder Details Dataset:")
print(order_details_df.head())

print("\nSales Target Dataset:")
print(sales_target_df.head())
```

```
Orders Dataset:
  Order ID  Order Date CustomerName           State       City
0  B-25601  01-04-2018       Bharat         Gujarat  Ahmedabad
1  B-25602  01-04-2018        Pearl     Maharashtra       Pune
2  B-25603  03-04-2018        Jahan  Madhya Pradesh     Bhopal
3  B-25604  03-04-2018       Divsha       Rajasthan     Jaipur
4  B-25605  05-04-2018      Kasheen     West Bengal    Kolkata

Order Details Dataset:
  Order ID  Amount  Profit  Quantity     Category      Sub-Category
0  B-25601  1275.0 -1148.0         7    Furniture         Bookcases
1  B-25601    66.0   -12.0         5     Clothing             Stole
2  B-25601     8.0    -2.0         3     Clothing        Hankerchief
3  B-25601    80.0   -56.0         4  Electronics  Electronic Games
4  B-25602   168.0  -111.0         2  Electronics            Phones

Sales Target Dataset:
  Month of Order Date   Category   Target
0              Apr-18  Furniture  10400.0
1              May-18  Furniture  10500.0
2              Jun-18  Furniture  10600.0
3              Jul-18  Furniture  10800.0
4              Aug-18  Furniture  10900.0
```

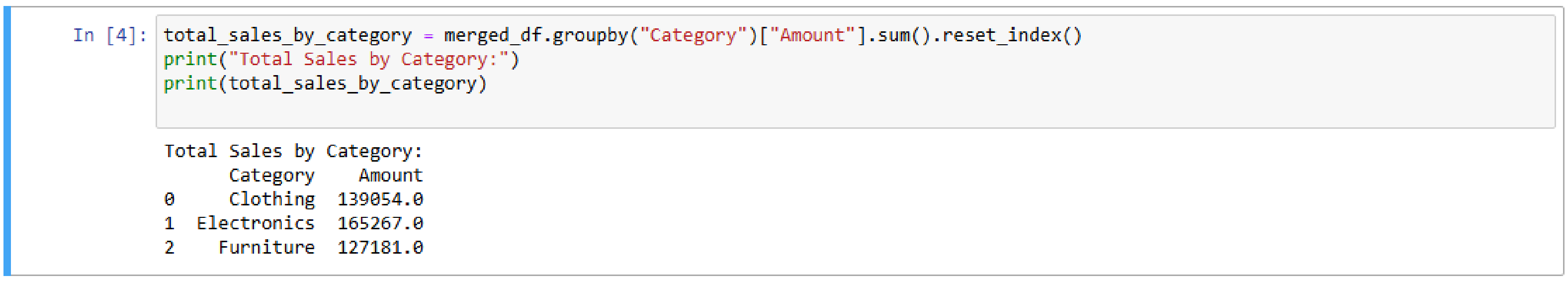
Step 2: Merge Orders and Order Details Datasets



```
In [3]: # Merge datasets based on Order ID
        merged_df = pd.merge(order_details_df, orders_df, on="Order ID", how="inner")

        # Display merged dataset
        print("Merged Dataset:")
        print(merged_df.head())

        Merged Dataset:
          Order ID  Amount  Profit  Quantity     Category      Sub-Category  \
        0  B-25601  1275.0 -1148.0         7    Furniture         Bookcases
        1  B-25601    66.0   -12.0         5     Clothing             Stole
        2  B-25601     8.0    -2.0         3     Clothing       Hankerchief
        3  B-25601    80.0   -56.0         4  Electronics  Electronic Games
        4  B-25602   168.0  -111.0         2  Electronics            Phones

           Order Date CustomerName        State       City
        0  01-04-2018       Bharat      Gujarat  Ahmedabad
        1  01-04-2018       Bharat      Gujarat  Ahmedabad
        2  01-04-2018       Bharat      Gujarat  Ahmedabad
        3  01-04-2018       Bharat      Gujarat  Ahmedabad
        4  01-04-2018        Pearl  Maharashtra       Pune
```

Step 3: Sales and Profitability Analysis

1. **Calculate total sales per category**



```
In [4]: total_sales_by_category = merged_df.groupby("Category")["Amount"].sum().reset_index()
        print("Total Sales by Category:")
        print(total_sales_by_category)

        Total Sales by Category:
              Category    Amount
        0     Clothing  139054.0
        1  Electronics  165267.0
        2    Furniture  127181.0
```

Total Sales by Category:

- **Clothing: ₹139,054**
- **Electronics: ₹165,267**
- **Furniture: ₹127,181**

2. Calculate average profit per order

```
In [5]: avg_profit_per_order = merged_df.groupby("Category")["Profit"].mean().reset_index()
        print("Average Profit Per Order by Category:")
        print(avg_profit_per_order)
```

```
Average Profit Per Order by Category:
      Category     Profit
0     Clothing  11.762908
1  Electronics  34.071429
2    Furniture   9.456790
```

Average Profit per Order:

- **Clothing: ₹11.76**
- **Electronics: ₹34.07**
- **Furniture: ₹9.46**

3. Calculate total profit margin (Profit as a % of Amount)

```
In [6]: profit_margin_by_category = merged_df.groupby("Category").apply(
            lambda x: (x["Profit"].sum() / x["Amount"].sum()) * 100 if x["Amount"].sum() != 0 else 0
        ).reset_index(name="Profit Margin (%)")

        print("Profit Margin by Category:")
        print(profit_margin_by_category)
```

```
Profit Margin by Category:
      Category  Profit Margin (%)
0     Clothing           8.027817
1  Electronics           6.349725
2    Furniture           1.806874
```

Total Profit Margin (Profit as % of Sales):

- **Clothing: 8.03% (Highest)**
- **Electronics: 6.35%**
- **Furniture: 1.81% (Lowest)**

4. Identify top-performing and underperforming categories

```
In [7]: # Combine all calculations into a single dataframe
        category_performance_df = total_sales_by_category.merge(avg_profit_per_order, on="Category")
        category_performance_df = category_performance_df.merge(profit_margin_by_category, on="Category")

        # Sort to find top and underperforming categories
        top_category = category_performance_df.sort_values(by="Profit Margin (%)", ascending=False).head(1)
        underperforming_category = category_performance_df.sort_values(by="Profit Margin (%)").head(1)

        print("Category Performance Summary:")
        print(category_performance_df)

        print("\nTop Performing Category:")
        print(top_category)

        print("\nUnderperforming Category:")
        print(underperforming_category)
```

```
Category Performance Summary:
      Category    Amount     Profit  Profit Margin (%)
0     Clothing  139054.0  11.762908           8.027817
1  Electronics  165267.0  34.071429           6.349725
2    Furniture  127181.0   9.456790           1.806874

Top Performing Category:
   Category    Amount     Profit  Profit Margin (%)
0  Clothing  139054.0  11.762908           8.027817

Underperforming Category:
    Category    Amount   Profit  Profit Margin (%)
2  Furniture  127181.0  9.45679           1.806874
```

Top-Performing Category

- **Clothing** is the best-performing category, with the highest profit margin (**8.03%**).
- Possible reasons:
    - High demand and consistent sales.
    - Better profit margins due to pricing strategies.

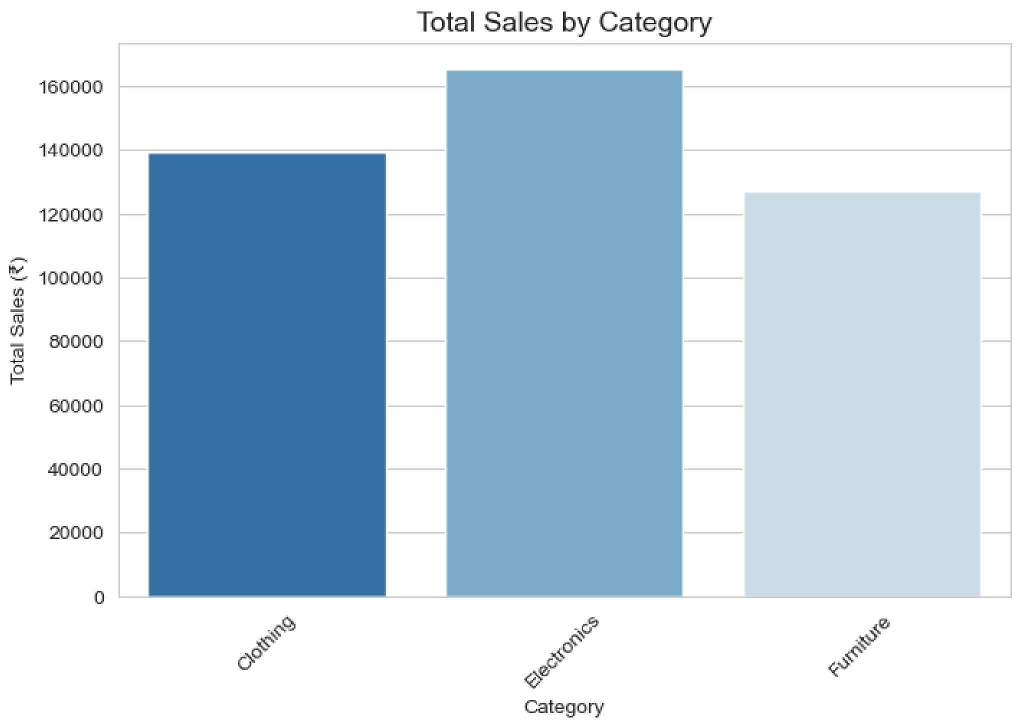
Underperforming Category

- **Furniture** is the lowest-performing category, with a profit margin of just **1.81%**.
- Possible reasons:
    - High cost of production or logistics.
    - Heavy discounts or lower sales volume.

VISUALIZATION

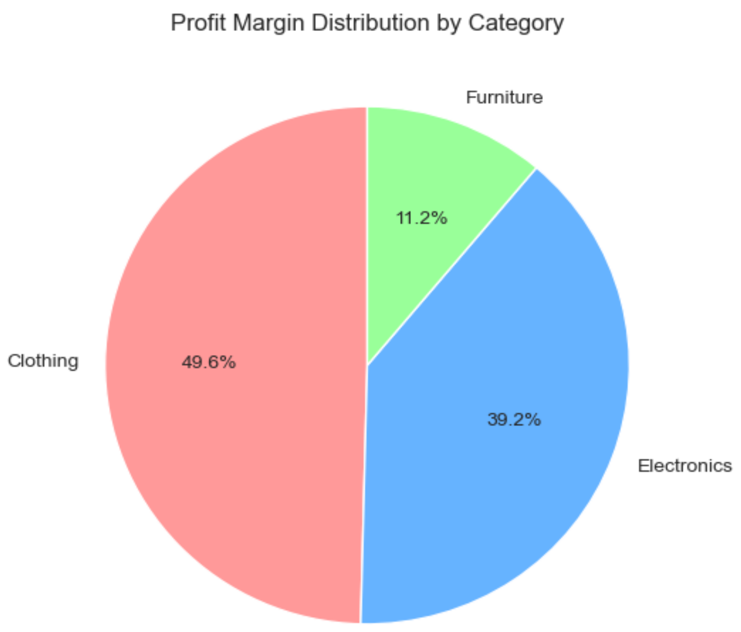
Total Sales by Category:

```
In [15]: import matplotlib.pyplot as plt
         import seaborn as sns

         # Set plot style
         sns.set_style("whitegrid")

         # Bar plot for total sales by category
         plt.figure(figsize=(8, 5))
         sns.barplot(x="Category", y="Amount", data=total_sales_by_category, palette="Blues_r")

         plt.title("Total Sales by Category", fontsize=14)
         plt.xlabel("Category")
         plt.ylabel("Total Sales (₹)")
         plt.xticks(rotation=45)
         plt.show()
```



Profit Margin Distribution by Category:

In [16]:
```
# Pie chart for profit margin by category
plt.figure(figsize=(6, 6))
plt.pie(profit_margin_by_category["Profit Margin (%)"], labels=profit_margin_by_category["Category"],
        autopct="%1.1f%%", colors=["#ff9999", "#66b3ff", "#99ff99"], startangle=90)

plt.title("Profit Margin Distribution by Category")
plt.show()
```



Part 2: Target Achievement Analysis

Step 1: Percentage change in target sales for the Furniture category month-over-month

1. Convert Month to Date Format

```
In [8]: sales_target_df["Month of Order Date"] = pd.to_datetime(sales_target_df["Month of Order Date"], format="%b-%y")
```

2. Filter for Furniture Category

```
In [9]: furniture_target_df = sales_target_df[sales_target_df["Category"] == "Furniture"].sort_values("Month of Order Date")
```

3. Calculate Month-over-Month % Change

```
In [10]: furniture_target_df["Target Change (%)"] = furniture_target_df["Target"].pct_change() * 100
         print("Month-over-Month Target Change for Furniture:")
         print(furniture_target_df)
```

```
Month-over-Month Target Change for Furniture:
   Month of Order Date   Category   Target  Target Change (%)
0           2018-04-01  Furniture  10400.0                NaN
1           2018-05-01  Furniture  10500.0           0.961538
2           2018-06-01  Furniture  10600.0           0.952381
3           2018-07-01  Furniture  10800.0           1.886792
4           2018-08-01  Furniture  10900.0           0.925926
5           2018-09-01  Furniture  11000.0           0.917431
6           2018-10-01  Furniture  11100.0           0.909091
7           2018-11-01  Furniture  11300.0           1.801802
8           2018-12-01  Furniture  11400.0           0.884956
9           2019-01-01  Furniture  11500.0           0.877193
10          2019-02-01  Furniture  11600.0           0.869565
11          2019-03-01  Furniture  11800.0           1.724138
```

Step 2: Identify Significant Fluctuations

```
In [11]: significant_fluctuations = furniture_target_df[abs(furniture_target_df["Target Change (%)"]) > 5]
         print("Months with Significant Target Fluctuations:")
         print(significant_fluctuations)
```

```
Months with Significant Target Fluctuations:
Empty DataFrame
Columns: [Month of Order Date, Category, Target, Target Change (%)]
Index: []
```

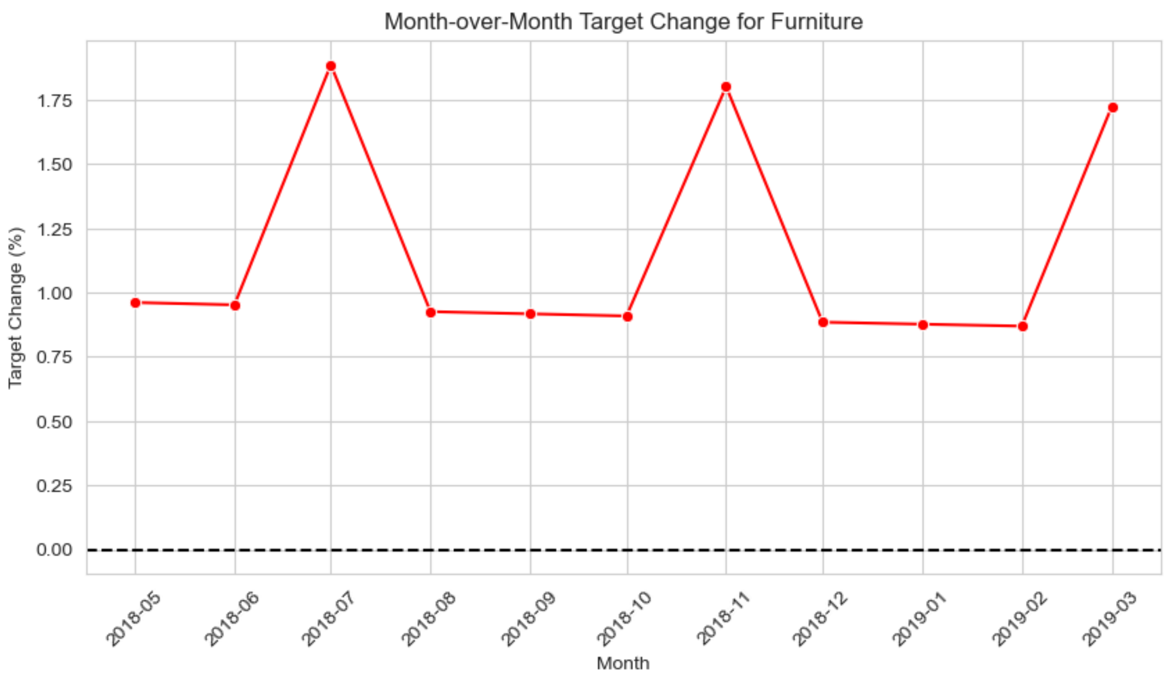
- **Month-over-Month Target Change (%):**
  - The target sales for Furniture increased gradually, with small percentage increases ranging from **0.87% to 1.88%**.
  - No significant fluctuations (above ±5%) were observed in the dataset.

- **Insights:**
  - The sales target was **incremented in a steady manner**, indicating **predictable demand trends**.
  - No major spikes or drops suggest that **sales forecasting was stable**.
- **Strategy Recommendations:**
  - To further align targets with actual performance:
    - **Analyze seasonal demand trends** to set dynamic targets.
    - **Introduce promotions during slow months** to boost actual sales.
    - **Review pricing strategies** to improve profit margins.

VISUALIZATION

Month-over-month Target Change for Furniture:

```
In [17]: plt.figure(figsize=(10, 5))
         sns.lineplot(data=furniture_target_df, x="Month of Order Date", y="Target Change (%)", marker="o", color="red")

         plt.axhline(0, color="black", linestyle="dashed")  # Reference line at 0%
         plt.title("Month-over-Month Target Change for Furniture")
         plt.xlabel("Month")
         plt.ylabel("Target Change (%)")
         plt.xticks(rotation=45)
         plt.show()
```



Part 3: Regional Performance Insights

1. Find Top 5 States with Highest Orders

```
In [12]: top_states = orders_df["State"].value_counts().head(5).reset_index()
         top_states.columns = ["State", "Order Count"]
         print("Top 5 States by Order Count:")
         print(top_states)
```

```
Top 5 States by Order Count:
            State  Order Count
0  Madhya Pradesh          101
1     Maharashtra           90
2       Rajasthan           32
3         Gujarat           27
4          Punjab           25
```

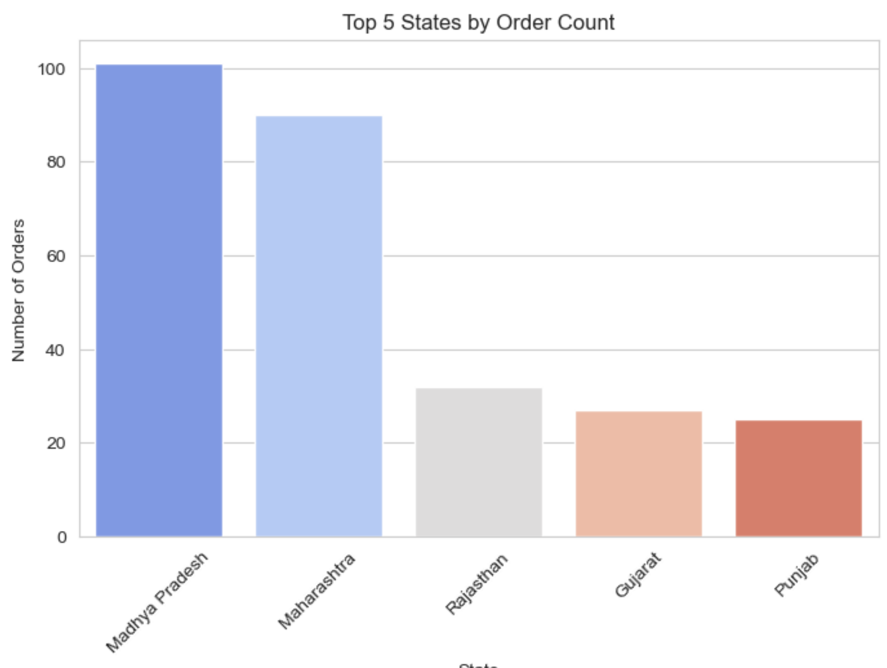
Top 5 States by Order Count:

- **Madhya Pradesh: 101 orders, ₹105,140 total sales, ₹16.33 avg profit**
- **Maharashtra: 90 orders, ₹95,348 total sales, ₹21.30 avg profit**
- **Rajasthan: 32 orders, ₹21,149 total sales, ₹16.99 avg profit**
- **Gujarat: 27 orders, ₹21,058 total sales, ₹5.34 avg profit**
- **Punjab: 25 orders, ₹16,786 total sales, -₹10.15 avg profit (Loss)**

VISUALIZATION

Top 5 States by Order Count:

```
In [18]: plt.figure(figsize=(8, 5))
         sns.barplot(x="State", y="Order Count", data=top_states, palette="coolwarm")

         plt.title("Top 5 States by Order Count")
         plt.xlabel("State")
         plt.ylabel("Number of Orders")
         plt.xticks(rotation=45)
         plt.show()
```



2. Calculate Total Sales and Average Profit per State

```
In [13]: state_sales_profit = merged_df.groupby("State").agg({"Amount": "sum", "Profit": "mean"}).reset_index()
```

3. Merge Data and Find Disparities

```
In [14]: top_states_performance = pd.merge(top_states, state_sales_profit, on="State")
         print("Top 5 States Performance Summary:")
         print(top_states_performance)

         # Identify state with highest and lowest sales
         highest_sales_state = top_states_performance.sort_values(by="Amount", ascending=False).head(1)
         lowest_sales_state = top_states_performance.sort_values(by="Amount").head(1)

         print("\nState with Highest Sales:")
         print(highest_sales_state)

         print("\nState with Lowest Sales:")
         print(lowest_sales_state)
```

```
Top 5 States Performance Summary:
            State  Order Count    Amount     Profit
0  Madhya Pradesh          101  105140.0  16.326471
1     Maharashtra           90   95348.0  21.296552
2       Rajasthan           32   21149.0  16.986486
3         Gujarat           27   21058.0   5.344828
4          Punjab           25   16786.0 -10.150000

State with Highest Sales:
            State  Order Count    Amount     Profit
0  Madhya Pradesh          101  105140.0  16.326471

State with Lowest Sales:
    State  Order Count   Amount  Profit
4  Punjab           25  16786.0  -10.15
```

State with Highest Sales:

- **Madhya Pradesh had the highest sales of ₹105,140 with 101 orders.**

State with Lowest Sales:

- **Punjab had the lowest sales of ₹16,786 and an average profit of -₹10.15, indicating losses.**

<u>Regional Disparities & Recommendations:</u>

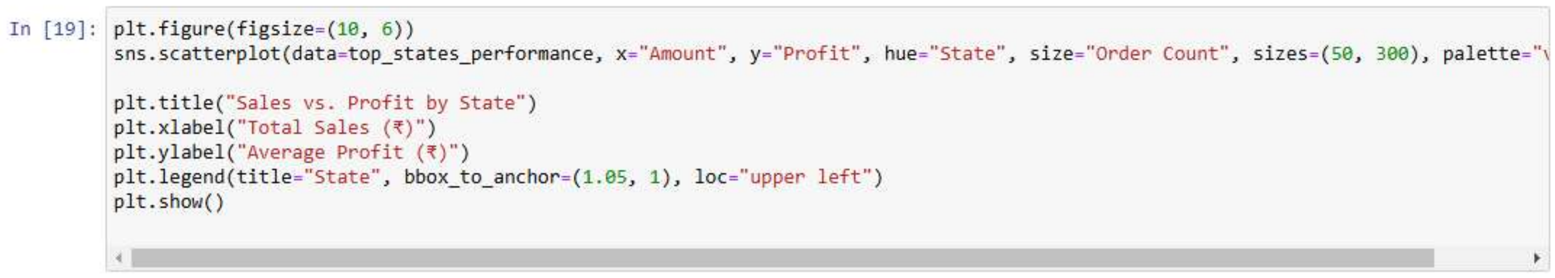
- **Punjab is underperforming** with negative profitability. Possible reasons:
  - High costs or low sales volume.
  - Product mismatch with market demand.
  - Higher return rates or discounts.

Actionable Steps:

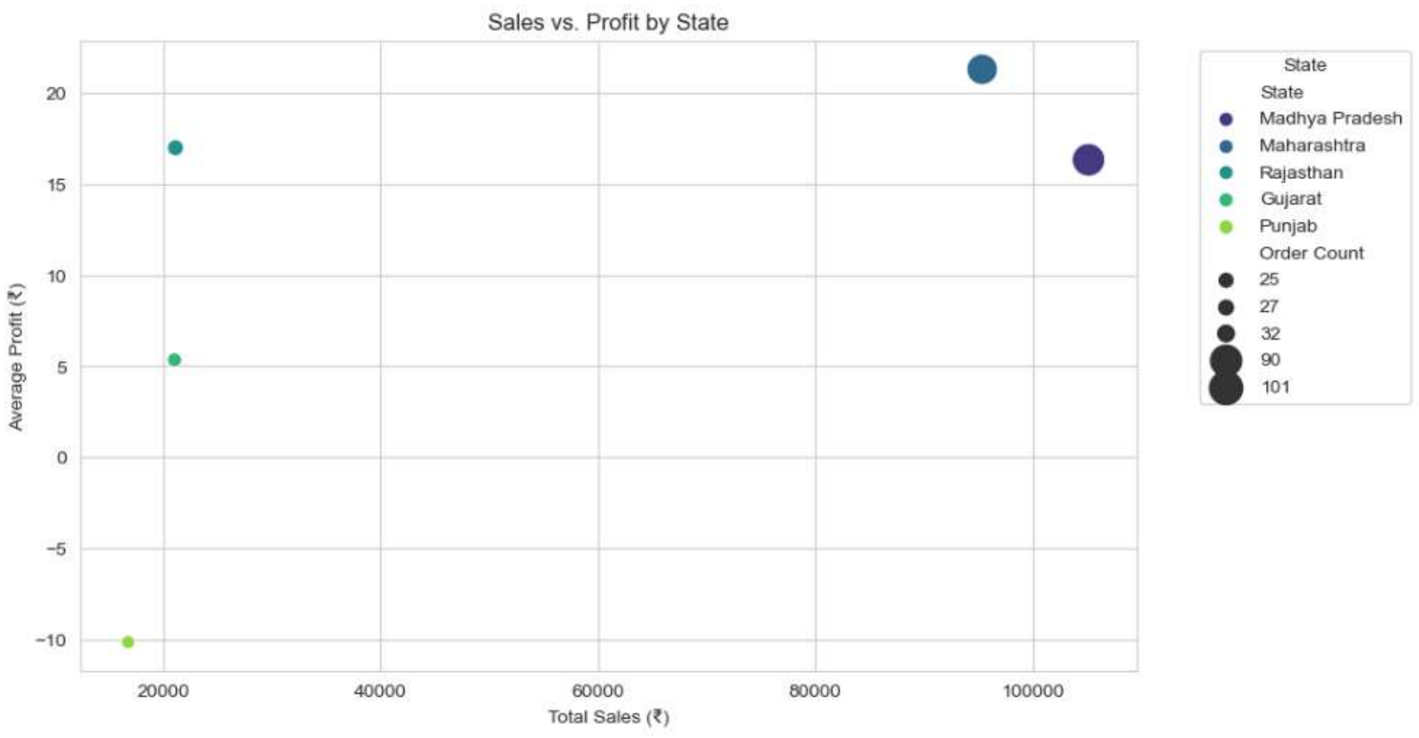
- **Focus on Maharashtra and Madhya Pradesh** (high order count and profitability).
- **Improve marketing & pricing strategies in Punjab** to recover losses.
- **Expand high-performing product categories in top states** to maximize profitability.

<u>VISUALIZATION</u>

Sales vs Profit by State:

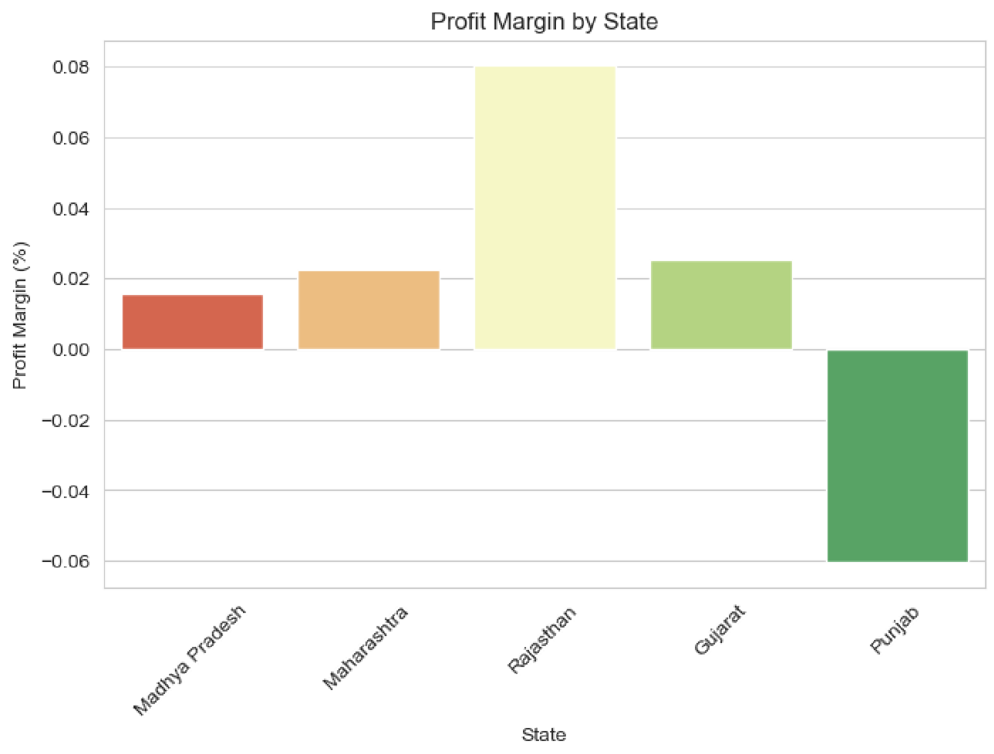


```
In [19]: plt.figure(figsize=(10, 6))
         sns.scatterplot(data=top_states_performance, x="Amount", y="Profit", hue="State", size="Order Count", sizes=(50, 300), palette="

         plt.title("Sales vs. Profit by State")
         plt.xlabel("Total Sales (₹)")
         plt.ylabel("Average Profit (₹)")
         plt.legend(title="State", bbox_to_anchor=(1.05, 1), loc="upper left")
         plt.show()
```



Profit Margin by State:

```
In [20]: # Calculate profit margin for each state
top_states_performance["Profit Margin (%)"] = (top_states_performance["Profit"] / top_states_performance["Amount"]) * 100

plt.figure(figsize=(8, 5))
sns.barplot(x="State", y="Profit Margin (%)", data=top_states_performance, palette="RdYlGn")

plt.title("Profit Margin by State")
plt.xlabel("State")
plt.ylabel("Profit Margin (%)")
plt.xticks(rotation=45)
plt.show()
```



Regional Disparities in Sales & Profitability (City-Level Analysis):

Identify the Top 5 Cities with the Highest Order Count:

In [27]:
```python
# Count the number of orders per city
top_cities = orders_df["City"].value_counts().head(5).reset_index()
top_cities.columns = ["City", "Order Count"]

# Display the top 5 cities
print("Top 5 Cities by Order Count:")
print(top_cities)
```

```
Top 5 Cities by Order Count:
         City  Order Count
0      Indore           76
1      Mumbai           68
2  Chandigarh           30
3       Delhi           25
4      Bhopal           22
```

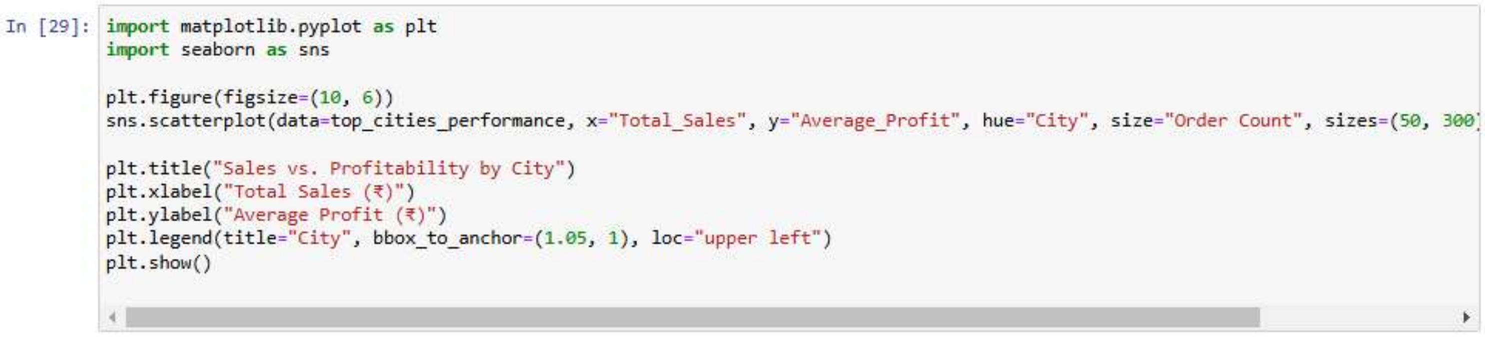
Merge with Order Details & Compute Sales & Profitability:

In [28]:
```python
# Filter data for only the top 5 cities
top_cities_list = top_cities["City"].tolist()
top_cities_data = merged_df[merged_df["City"].isin(top_cities_list)]

# Group by city and calculate total sales and average profit
city_sales_profit = top_cities_data.groupby("City").agg(
    Total_Sales=("Amount", "sum"),
    Average_Profit=("Profit", "mean")
).reset_index()

# Merge with top cities order count data
top_cities_performance = pd.merge(top_cities, city_sales_profit, on="City")

# Display the final results
print("Top 5 Cities Performance Summary:")
print(top_cities_performance)
```

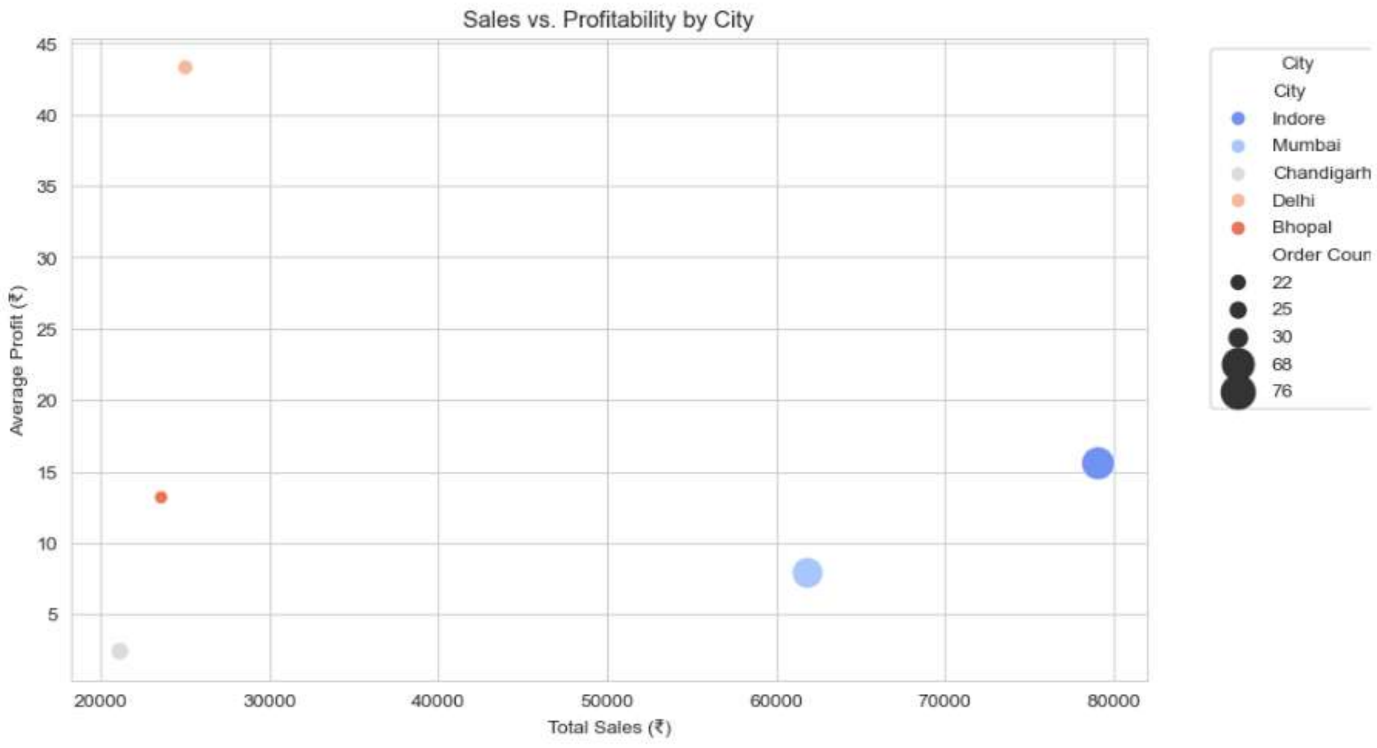
```
Top 5 Cities Performance Summary:
         City  Order Count  Total_Sales  Average_Profit
0      Indore           76      79069.0       15.576779
1      Mumbai           68      61867.0        7.908213
2  Chandigarh           30      21142.0        2.422535
3       Delhi           25      25019.0       43.308642
4      Bhopal           22      23583.0       13.196970
```

VISUALIZATION

Sales vs. Profitability by City (Scatter Plot)

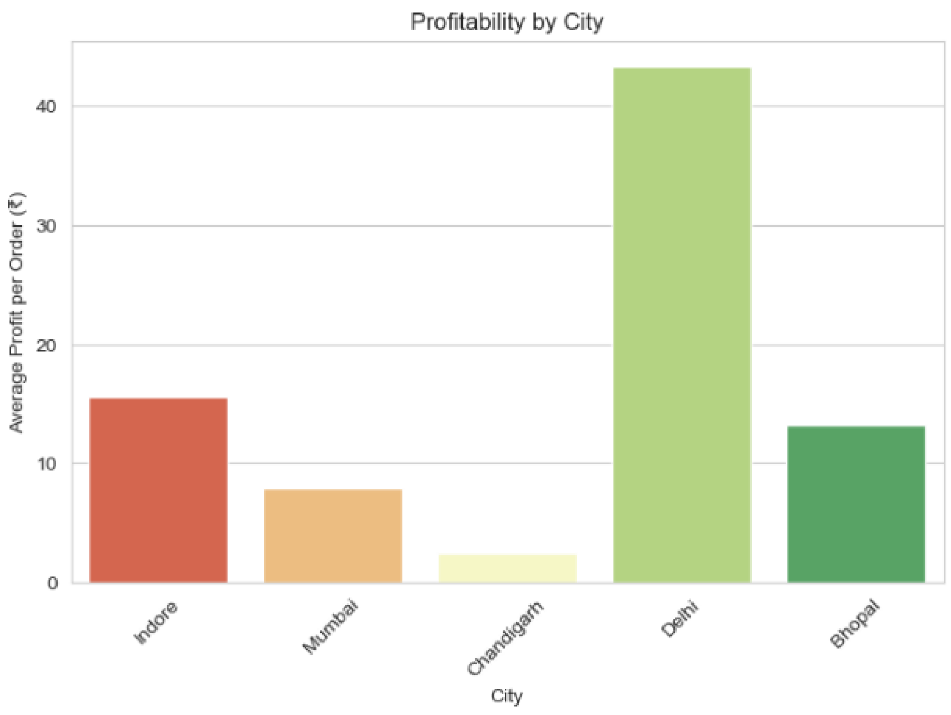


```
In [29]: import matplotlib.pyplot as plt
         import seaborn as sns

         plt.figure(figsize=(10, 6))
         sns.scatterplot(data=top_cities_performance, x="Total_Sales", y="Average_Profit", hue="City", size="Order Count", sizes=(50, 300)

         plt.title("Sales vs. Profitability by City")
         plt.xlabel("Total Sales (₹)")
         plt.ylabel("Average Profit (₹)")
         plt.legend(title="City", bbox_to_anchor=(1.05, 1), loc="upper left")
         plt.show()
```



Profit Margin by City (Bar Chart)

```
In [30]: plt.figure(figsize=(8, 5))
         sns.barplot(x="City", y="Average_Profit", data=top_cities_performance, palette="RdYlGn")

         plt.title("Profitability by City")
         plt.xlabel("City")
         plt.ylabel("Average Profit per Order (₹)")
         plt.xticks(rotation=45)
         plt.show()
```



Action Plan:

- *Focus on Chandigarh: Reduce discounts, improve product mix.*
- *Optimize Mumbai & Bhopal: Increase average profit per order.*
- *Maintain Delhi & Indore: Keep strategies consistent.*
- *Future Expansion: Look at secondary cities with potential growth.*

QUESTION 2

Five Strengths of the Jar App:

1. ***Automated Micro-Savings: Jar turns spare change into digital gold by automatically rounding up consumers' online transactions to the closest ten. Consistent savings are encouraged by this smooth process, which doesn't require user participation.***
2. ***User-Friendly layout: Even people with little financial literacy can use the app thanks to its safe and easy-to-use layout. Its features' simplicity and ease of use have been commended by users.***
3. ***Gamified Experience: Jar uses gamification components to make saving fun. Users can 'spin a jar' to get incentives after every investment, for example, which increases user engagement and offers positive reinforcement.***
4. ***Real-Time Gold Price Updates: By giving users access to the most recent gold prices, the app enables users to make well-informed investing decisions. This openness helps users track the value of their savings and fosters trust.***
5. ***Flexible Investment Options: Jar encourages inclusivity by letting users choose daily savings goals based on their financial comfort levels. Furthermore, consumers can sell or withdraw their gold assets whenever they choose because there is no lock-in period.***

Five Things That Could Be Better:

1. ***Charges and Withdrawal Process: A number of users have complained about disparities between the prices at which gold is bought and sold, which has resulted in unforeseen losses when making withdrawals. These worries might be allayed by improving transaction charge transparency and guaranteeing competitive pricing.***

2. ***Customer service responsiveness: According to user feedback, it might be difficult for consumers to get in touch with customer service, particularly when they have problems with withdrawals and account management. User satisfaction would increase if the support system was strengthened and prompt responses were guaranteed.***

3. ***Efficiency of the KYC Process: Some customers have had trouble finishing the Know Your Customer (KYC) process, which has prevented them from taking advantage of all the capabilities offered by the app. The onboarding experience might be improved by streamlining this procedure and offering precise instructions.***

4. Options for Deleting Accounts: Many customers have complained that it is difficult to remove their accounts. Addressing privacy concerns and meeting user expectations for control over their personal data would be achieved by implementing a simple feature that allows users to delete their accounts.

5. Pricing and Fee Transparency: Customers have complained about unstated costs that reduce their savings and erode their trust. More transparency and trust would be promoted by giving users a comprehensive explanation of all relevant fees and making sure they are informed before completing transactions.

QUESTION 3

Opportunity:

Merchant Payments and UPI Cashback Rewards.

Introduce a UPI-based payment mechanism within the app, allowing users to conduct direct transactions.

Offer cashback benefits in the form of digital gold for purchases made from linked merchants.

How Jar Can Leverage Its Strengths:

1. ***Automate cashback rewards for consumers in real-time, with no effort.***
2. ***Prioritize user experience by ensuring speedy, secure, and integrated transactions within the app.***
3. ***Build trust by partnering with brands and offering valuable benefits.***

Micro-Loans and Credit Building Services.

Provide consumers with small-ticket, short-term loans depending on their savings history and spending habits.

Introduce a credit-building program in which consumers can gradually improve their credit scores by making responsible payments on tiny, automatic credit installments.

How Jar Can Leverage Its Strengths:

1. ***Automate loan repayment from users' savings balances.***
2. ***User Experience: Offer paperless loan applications with rapid approvals.***
3. ***Ensure loan terms are straightforward with no hidden cost or predatory interest rate.***