In [1]:
```
from dotenv import load_dotenv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Markdown, display

from zenalyze import create_zenalyze_object_with_env_var_and_last5_hist

load_dotenv()
zen = create_zenalyze_object_with_env_var_and_last5_hist(globals(), './../data/raw/')

buddy = lambda query: display(Markdown(zen.buddy(query)))
```

```
Zenalyze(Data(customers), Data(orders), Data(order_products), Data(products))
```

In [2]:
```
zen.do('check the number of customers and cities present.')
```

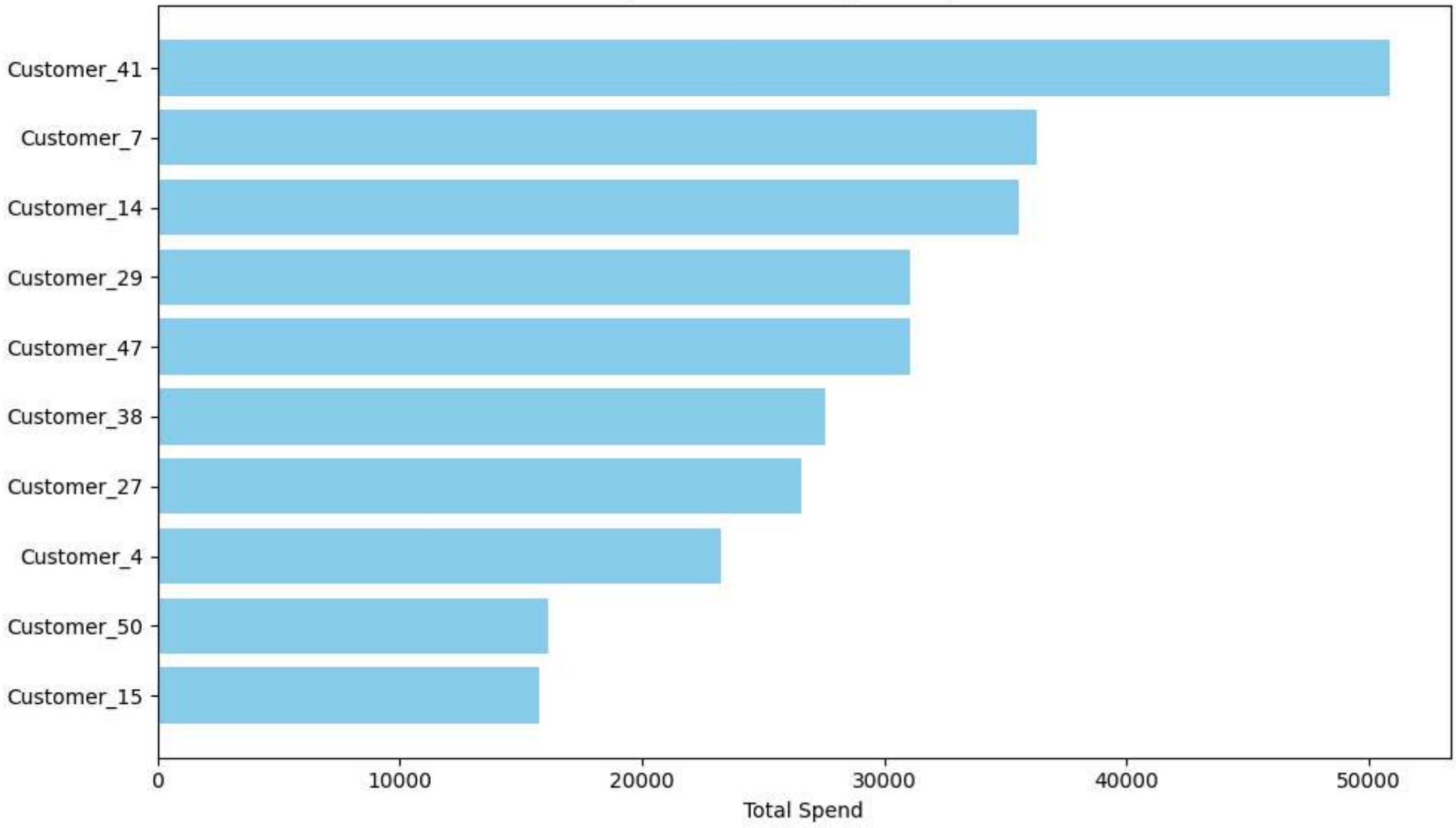
```
# total number of customers (rows)
total_customers = len(customers)

# number of distinct cities
distinct_cities = customers['city'].nunique()

print(f"Total customers: {total_customers}")
print(f"Distinct cities: {distinct_cities}")

# show a sample of the customers table
print("\nSample customers data:")
print(customers.head())
```

```
Total customers: 50
Distinct cities: 5

Sample customers data:
   customer_id customer_name  age    city   join_date
0          101    Customer_1   60   Delhi  2022-01-01
1          102    Customer_2   25  Mumbai  2022-01-16
2          103    Customer_3   34   Delhi  2022-01-31
3          104    Customer_4   38  Mumbai  2022-02-15
4          105    Customer_5   43   Delhi  2022-03-02
```

In [3]:
```
zen.do('give me a table showing each customer with their products ordered, quantity of each product ordered and total cost associated.')
```

```
# Join customers → orders → order_products → products
cust_orders = customers.merge(orders, on='customer_id', how='inner')
cust_orders_products = cust_orders.merge(order_products, on='order_id', how='inner')
full_df = cust_orders_products.merge(products, on='product_id', how='inner')

# Compute cost per line item
full_df['total_cost'] = full_df['quantity'] * full_df['price']

# Aggregate per customer & product
customer_product_summary = (
    full_df
    .groupby(['customer_id', 'customer_name', 'product_id', 'product_name'], as_index=False)
    .agg(quantity=('quantity', 'sum'),
         total_cost=('total_cost', 'sum'))
)

# Show result sample
print("Customer-Product summary (first 10 rows):")
print(customer_product_summary.head(10))
```

```
# Optional: visualize top 10 customers by total spend
top_customers = (
    full_df
    .groupby(['customer_id', 'customer_name'], as_index=False)
    .agg(total_spent=('total_cost', 'sum'))
    .sort_values('total_spent', ascending=False)
    .head(10)
)

plt.figure(figsize=(10,6))
plt.barh(top_customers['customer_name'], top_customers['total_spent'], color='skyblue')
plt.xlabel('Total Spend')
plt.title('Top 10 Customers by Total Spend')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
Customer-Product summary (first 10 rows):
   customer_id customer_name  product_id product_name  quantity  total_cost
0          102    Customer_2         324   Product_24         5        8705
1          103    Customer_3         309    Product_9         4        6860
2          103    Customer_3         325   Product_25         1        1342
3          103    Customer_3         342   Product_42         1        4446
4          104    Customer_4         326   Product_26         4       12388
5          104    Customer_4         328   Product_28         3       10881
6          105    Customer_5         313   Product_13         2        3930
7          105    Customer_5         329   Product_29         3        6171
8          105    Customer_5         344   Product_44         2        5208
9          107    Customer_7         301    Product_1         1        2959
```




```
In [4]: buddy('What we did till now?')
```

Summary of Work Completed

Task	Approach	Key Output
1. **Customer & City counts**	• Calculated `len(customers)` for total customers. • Used `customers['city'].nunique()` for distinct cities.	• Printed total customers and distinct city count. • Displayed a sample of the `customers` table.
2. **Customer-Product Order Summary**	• Joined `customers → orders → order_products → products`. • Created `total_cost = quantity * price`. • Grouped by `customer_id, customer_name, product_id, product_name` to sum `quantity` and `total_cost`.	• Printed first 10 rows of the aggregated `customer_product_summary`.
3. **Optional Insight**	• Calculated total spend per customer and plotted the top 10 customers.	• Bar chart of top 10 customers by total spend.

All operations were performed using pandas DataFrames, and the results were printed (and plotted where applicable).

```
In [5]: buddy('What are all the derived tables and fields created?')
```

Derived tables created

Derived table	Purpose / source tables
`cust_orders`	`customers` joined with `orders` (inner on `customer_id`)
`cust_orders_products`	`cust_orders` joined with `order_products` (inner on `order_id`)
`full_df`	`cust_orders_products` joined with `products` (inner on `product_id`)
`customer_product_summary`	Aggregated view of `full_df` grouped by customer & product
`top_customers`	Aggregated view of `full_df` grouped by customer (total spend)

Derived fields created

Field	Where it appears	Calculation
`total_cost`	In `full_df` (line-item cost)	`quantity * price`
`quantity`	In `customer_product_summary` (aggregated)	`sum(quantity)`
`total_cost`	In `customer_product_summary` (aggregated)	`sum(total_cost)`
`total_spent`	In `top_customers`	`sum(total_cost)`

These are all the intermediate tables and new columns that the code generates.

```
In [6]: zen.do('from the above table, only show me the customer name, product name and cost columns.')
```

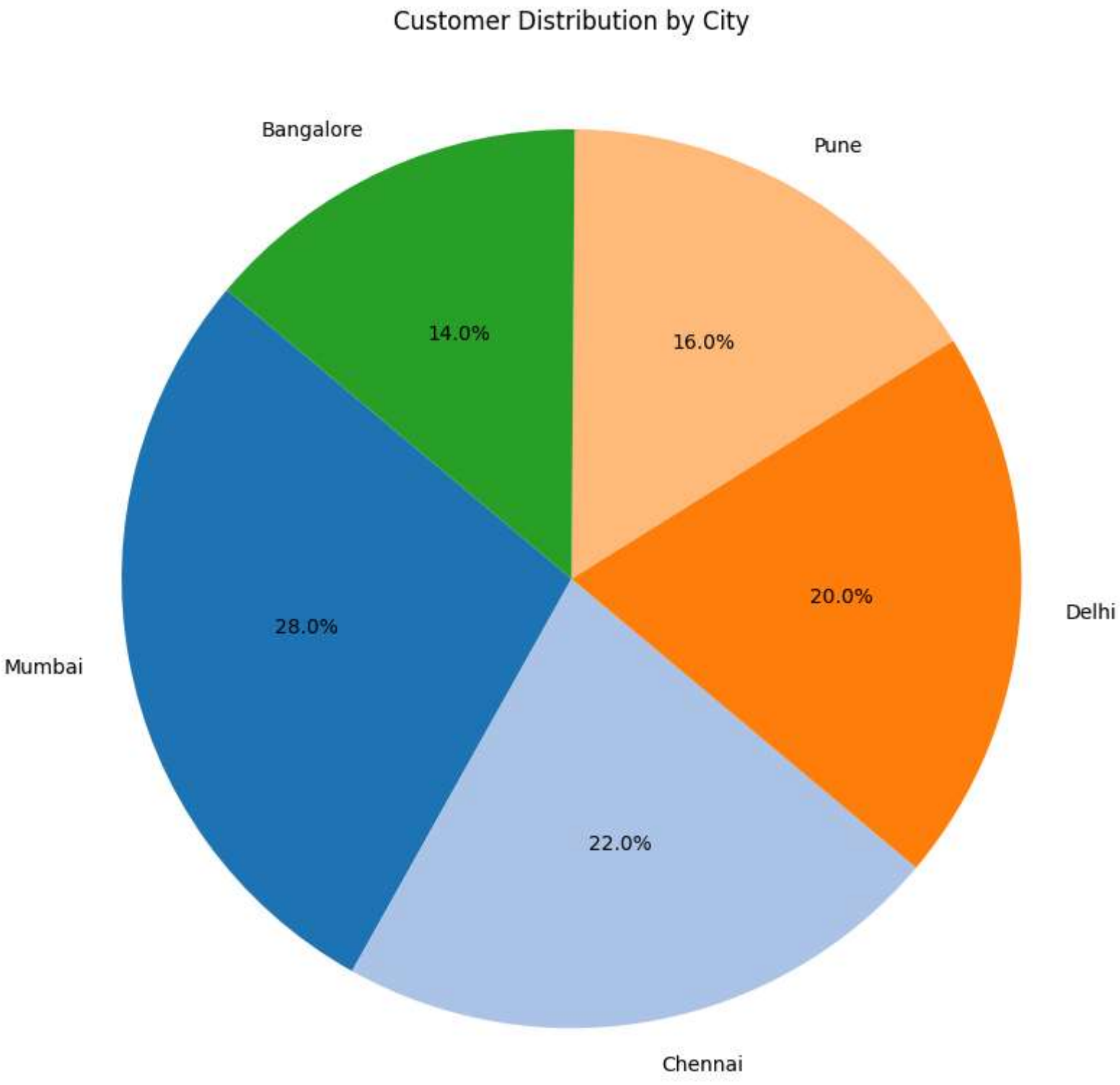
```
selected_columns = customer_product_summary[['customer_name', 'product_name', 'total_cost']]

print("Customer Name | Product Name | Total Cost")
print(selected_columns.head())
```

```
Customer Name | Product Name | Total Cost
  customer_name product_name  total_cost
0    Customer_2   Product_24        8705
1    Customer_3    Product_9        6860
2    Customer_3   Product_25        1342
3    Customer_3   Product_42        4446
4    Customer_4   Product_26       12388
```

In [7]:
```
zen.do('a pie chart plot shwoing the cities and how many customers belog to each.')
```

```
# Count customers per city
city_counts = customers['city'].value_counts()

print("Customers per city:")
print(city_counts)

# Pie chart of city distribution
plt.figure(figsize=(8, 8))
plt.pie(
    city_counts,
    labels=city_counts.index,
    autopct='%1.1f%%',
    startangle=140,
    colors=plt.cm.tab20.colors
)
plt.title('Customer Distribution by City')
plt.axis('equal')  # Ensure pie is drawn as a circle
plt.tight_layout()
plt.show()
```

```
Customers per city:
city
Mumbai       14
Chennai      11
Delhi        10
Pune          8
Bangalore     7
Name: count, dtype: int64
```



In [8]:
```
buddy('what charts we made till now?')
```

Charts created so far

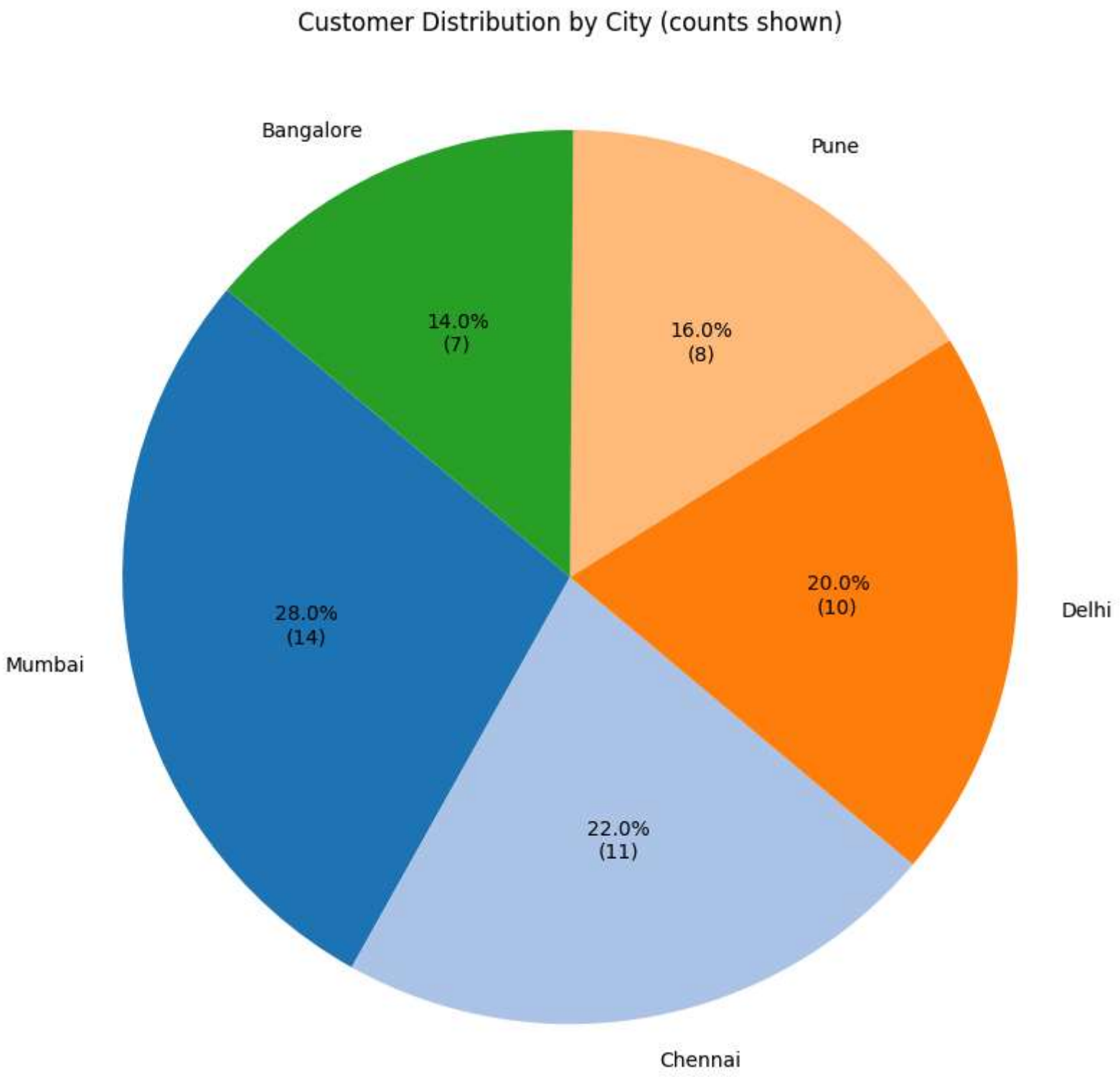
Chart	Purpose
Horizontal bar chart – Top 10 customers by total spend (customer name vs. total cost)	Highlights the highest-spending customers
Pie chart – Distribution of customers across cities	Shows the share of customers per city

These are the only visualizations produced up to this point.

In [9]:
```
zen.do('show me the counts too in the chart.')
```

```
# Count customers per city (already computed as city_counts)
print("Customers per city:")
print(city_counts)

# Pie chart showing both percentage and absolute count for each city
def autopct_with_counts(pct):
    total = city_counts.sum()
    count = int(round(pct * total / 100.0))
    return f'{pct:.1f}%\n({count})'

plt.figure(figsize=(8, 8))
plt.pie(
    city_counts,
    labels=city_counts.index,
    autopct=autopct_with_counts,
    startangle=140,
    colors=plt.cm.tab20.colors
)
plt.title('Customer Distribution by City (counts shown)')
plt.axis('equal')  # Ensure pie is drawn as a circle
plt.tight_layout()
plt.show()
```

```
Customers per city:
city
Mumbai       14
Chennai      11
Delhi        10
Pune          8
Bangalore     7
Name: count, dtype: int64
```



In [10]:
```
buddy("Tell me all the individual tasks we did.")
```

Tasks completed

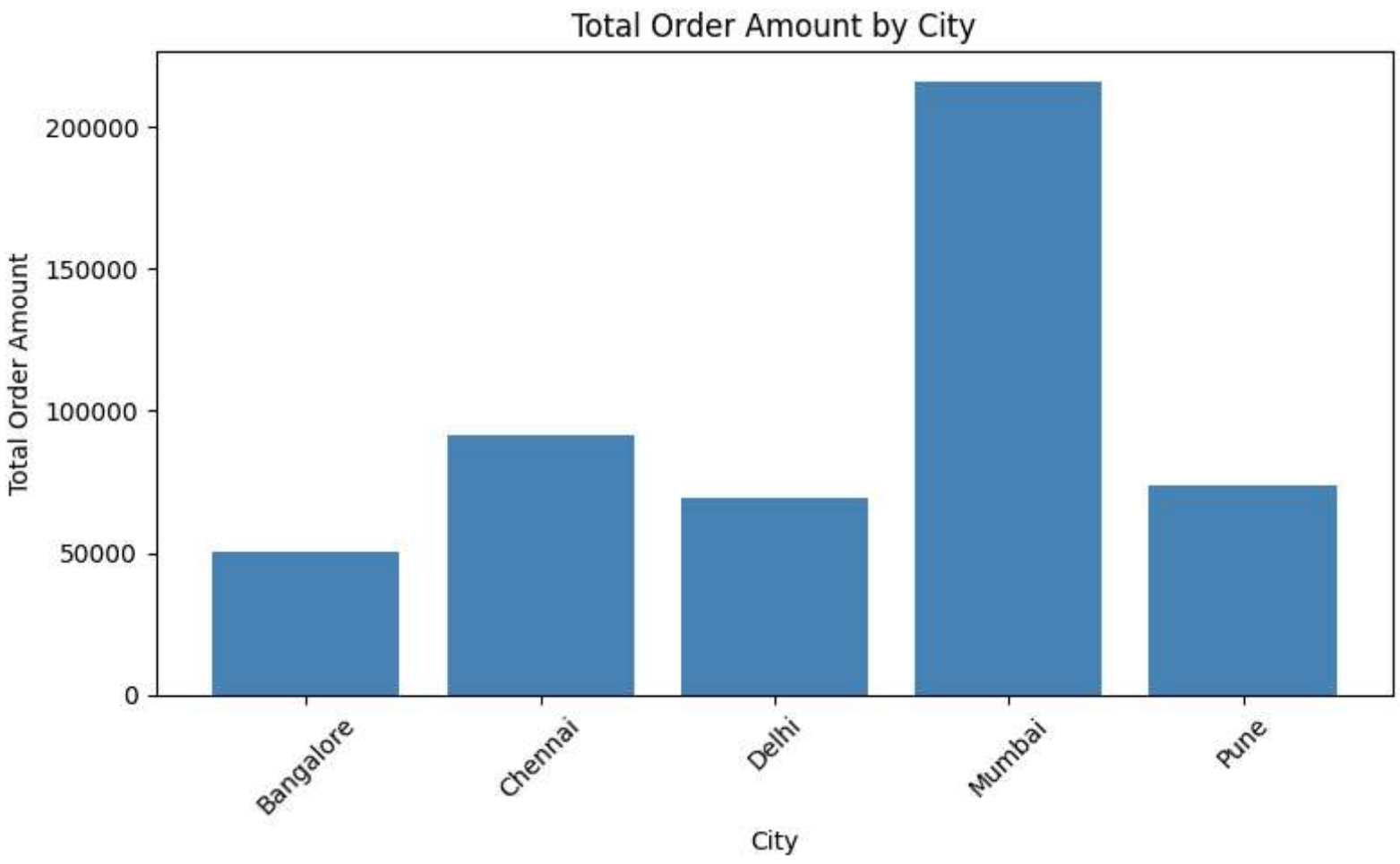
#	Task description
1	Count total customers and distinct cities.
2	Create a customer-product summary table with quantity and total cost per product.
3	Select only the customer name, product name, and total cost columns from that summary.
4	Plot a pie chart of customer distribution by city.
5	Plot the same pie chart but include absolute counts in the labels.

These steps covered data aggregation, filtering, and visualisation of customer and city metrics.

In [11]:
```
zen.do('what is the total amounts from each city?')
```

```
# Join customers with orders to associate each order amount with a city
city_amounts = customers.merge(orders, on='customer_id', how='inner')\
                        .groupby('city', as_index=False)['amount']\
                        .sum()\
                        .rename(columns={'amount': 'total_amount'})

print("Total order amount per city:")
print(city_amounts)

# Bar chart of total amounts by city
plt.figure(figsize=(8, 5))
plt.bar(city_amounts['city'], city_amounts['total_amount'], color='steelblue')
plt.xlabel('City')
plt.ylabel('Total Order Amount')
plt.title('Total Order Amount by City')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Total order amount per city:
        city  total_amount
0  Bangalore         50383
1    Chennai         91618
2      Delhi         69313
3     Mumbai        215624
4       Pune         74008
```



In [15]:
```
zen.do('divide the total cost field for each customer by 100 and show me. There is no customer_total_spend column in total_spend field')
```

```
customer_total_cost = (
    customer_product_summary
    .groupby(['customer_id', 'customer_name'], as_index=False)['total_cost']
    .sum()
)
customer_total_cost['total_cost_div_100'] = customer_total_cost['total_cost'] / 100
print(customer_total_cost.head())
```

```
   customer_id customer_name  total_cost  total_cost_div_100
0          102    Customer_2        8705               87.05
1          103    Customer_3       12648              126.48
2          104    Customer_4       23269              232.69
3          105    Customer_5       15309              153.09
4          107    Customer_7       36272              362.72
```

In [17]:
```
zen.do('from my 3rd query, give me the same table but trim the word "Product_" from the product column.')
```

```
# Trim "Product_" prefix from product_name in the previously selected table
selected_columns_trimmed = selected_columns.copy()
selected_columns_trimmed['product_name'] = selected_columns_trimmed['product_name'].str.replace(r'^Product_', '', regex=True)

# Show result
print(selected_columns_trimmed.head())
```

```
  customer_name product_name  total_cost
0    Customer_2           24        8705
1    Customer_3            9        6860
2    Customer_3           25        1342
3    Customer_3           42        4446
4    Customer_4           26       12388
```

In [18]:
```
zen.do('for each customer, multiply the customer count by 2')
```

```
# Use existing merged DataFrame `cust_orders` (customers + orders)
customer_counts = (
    cust_orders
    .groupby(['customer_id', 'customer_name'], as_index=False)
    .size()
    .rename(columns={'size': 'order_count'})
)

# Multiply the count by 2
customer_counts['order_count_x2'] = customer_counts['order_count'] * 2

# Show result
print(customer_counts.head())
```

```
   customer_id customer_name  order_count  order_count_x2
0          102    Customer_2            2               4
1          103    Customer_3            1               2
2          104    Customer_4            2               4
3          105    Customer_5            1               2
4          107    Customer_7            1               2
```

In [19]:
```
zen.do('give me only the product name and total cost.')
```

```
product_total_cost = (
    customer_product_summary
    .groupby(['product_name'], as_index=False)['total_cost']
    .sum()
)

print(product_total_cost.head())
```

```
  product_name  total_cost
0    Product_1        8877
1   Product_11       17704
2   Product_12        2070
3   Product_13       11790
4   Product_14        2258
```

In [20]:
```
print(zen.get_history)
```

[Q: count total customers and distinct cities] `total_customers` set to `len(customers)`; `distinct_cities` set to `customers['‹city›'].nunique()`; printed both values and a sample of `customers`.

[Q: list each customer with ordered products, quantities, and total cost] `cust_orders` created by merging `customers` with `orders` on `‹customer_id›`; `cust_orders_products` created by merging `cust_orders` with `order_products` on `‹order_id›`; `full_df` created by merging `cust_orders_products` with `products` on `‹product_id›`; added column `‹total_cost›` as `‹quantity› * ‹price›`; `customer_product_summary` aggregated `full_df` by `‹customer_id›`, `‹customer_name›`, `‹product_id›`, `‹product_name›` summing `‹quantity›` and `‹total_cost›`; `top_customers` aggregated `full_df` by `‹customer_id›`, `‹customer_name›` summing `‹total_cost›` as `‹total_spent›`, sorted descending and kept top 10; plotted horizontal bar chart of `top_customers`.

[Q: keep only customer name, product name, and cost] `selected_columns` created from `customer_product_summary` selecting `‹customer_name›`, `‹product_name›`, `‹total_cost›`; printed sample.

[Q: pie chart of customer counts per city] `city_counts` set to `customers['‹city›'].value_counts()`; printed counts; plotted pie chart of `city_counts` with labels and percentages.

[Q: show counts on the pie chart] reused `city_counts`; defined `autopct_with_counts` to display percentage and absolute count; plotted pie chart using this function for autopct.

[Q: total order amount per city] `city_amounts` created by merging `customers` with `orders` on `‹customer_id›`, grouping by `‹city›` and summing `‹amount›`, then renaming to `‹total_amount›`; printed result; plotted bar chart of `‹city›` vs `‹total_amount›`.

[Q: divide each customer's total cost by 100] `customer_total_cost` was created by grouping `customer_product_summary` on ‹customer_id› and ‹customer_name›, summing ‹total_cost›, then a new column ‹total_cost_div_100› was added as ‹total_cost› / 100; the first rows were printed.

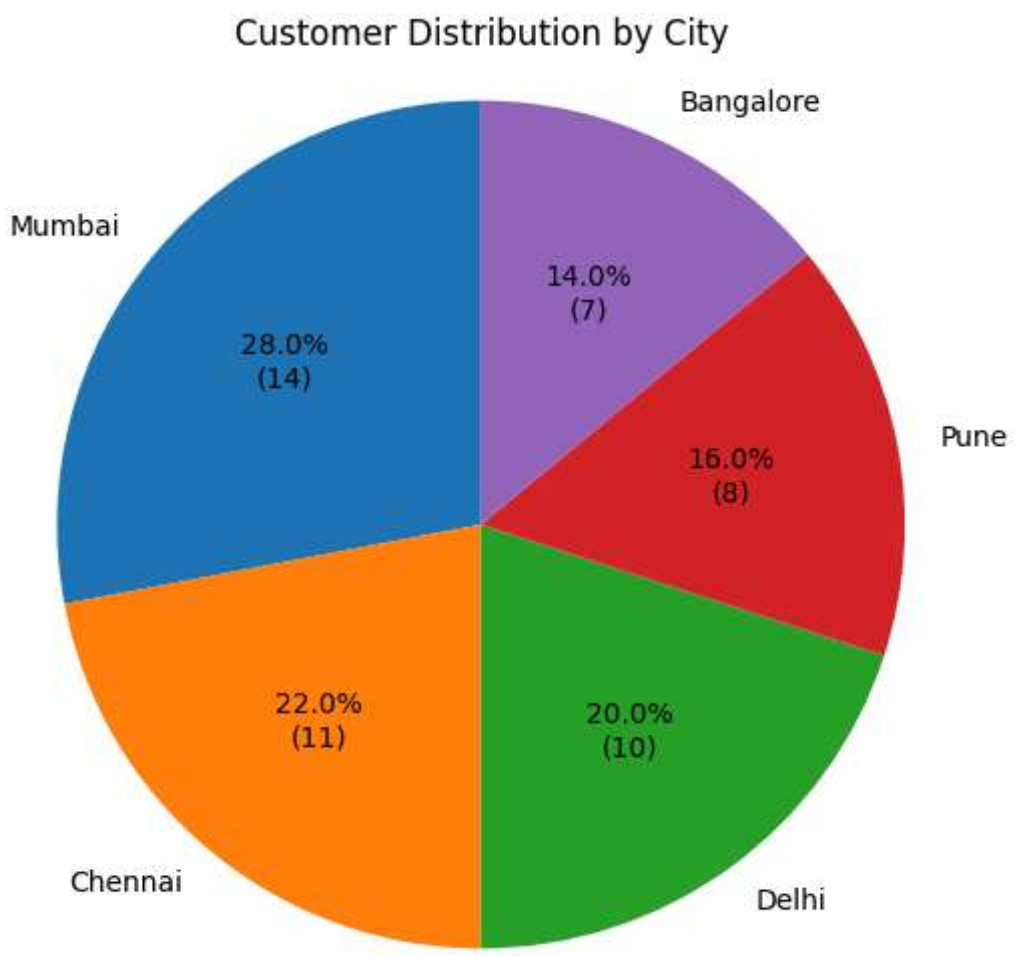
[Q: remove the "Product_" prefix from product names] `selected_columns_trimmed` was made as a copy of `selected_columns` and its ‹product_name› column was transformed with a regex replace to drop the leading "Product_"; the first rows were printed.

[Q: double the order count for each customer] `customer_counts` was created by grouping `cust_orders` on ‹customer_id› and ‹customer_name›, counting rows with .size(), renaming the count column to ‹order_count›, then adding ‹order_count_x2› as ‹order_count› * 2; the first rows were printed.

```
----------------------------
Time 06:36 - User query: give me only the product name and total cost.
Response:
product_total_cost = (
    customer_product_summary
    .groupby(['product_name'], as_index=False)['total_cost']
    .sum()
)

print(product_total_cost.head())
----------------------------
```

In [21]:
```
zen.do('show me the last pie chart again.')
```

```
# Re-plot the last pie chart of customer counts per city using existing variables
plt.figure(figsize=(6,6))
city_counts.plot.pie(
    labels=city_counts.index,
    autopct=autopct_with_counts,   # function defined earlier to show % and count
    startangle=90,
    shadow=False
)
plt.title('Customer Distribution by City')
plt.ylabel('')  # hide y-label for cleaner look
plt.axis('equal')  # ensure pie is a circle
plt.show()
```



In [22]:
```
buddy('Did we yet check the distribution curve of the expenditure for each customer?')
```

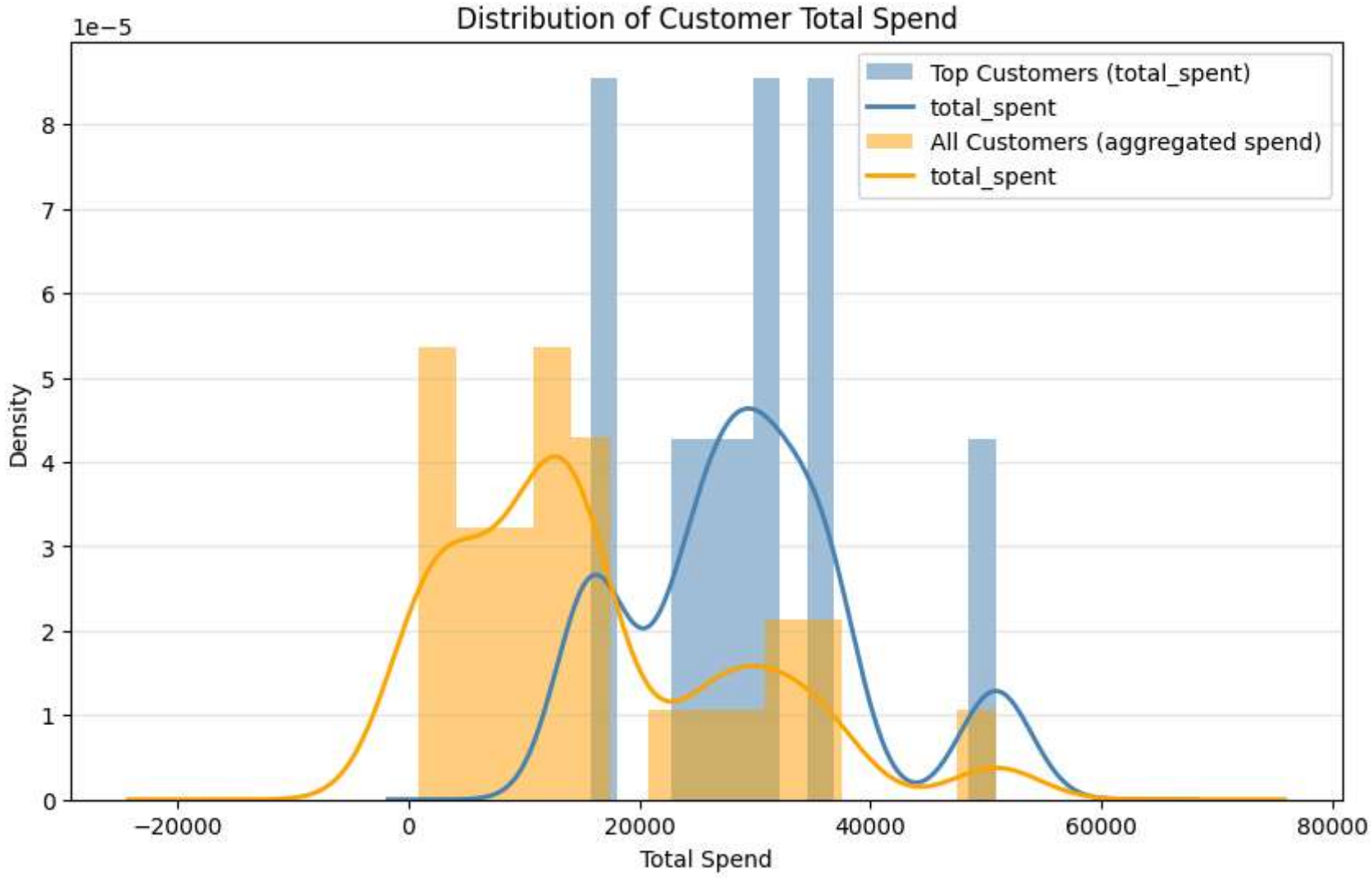
No, we haven't plotted a distribution curve for each customer's expenditure yet.
If you'd like to see it, you could:

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

# total spent per customer
cust_spent = (
    customer_product_summary
    .groupby(['customer_id', 'customer_name'], as_index=False)['total_cost']
    .sum()
)

# distribution plot
plt.figure(figsize=(8, 4))
sns.histplot(cust_spent['total_cost'], kde=True, bins=30)
plt.title('Distribution of Customer Expenditure')
plt.xlabel('Total Spent')
plt.ylabel('Frequency')
plt.show()
```

This will give you a histogram with a KDE overlay showing the expenditure distribution across customers.

In [23]:
```
zen.do('plot a histogram or kernel density estimate for the total_spent column in customer_product_summary and the aggregated customer_total
```

```
# Aggregate total spend per customer from product-level data
customer_total_spend = (
    customer_product_summary
    .groupby(['customer_id', 'customer_name'], as_index=False)['total_cost']
    .sum()
    .rename(columns={'total_cost': 'total_spent'})
)

# Ensure the column exists in the previously created top_customers (already has 'total_spent')
# Plot histogram + KDE for both distributions
plt.figure(figsize=(10, 6))

# Histogram for top_customers total_spent
top_customers['total_spent'].plot.hist(
    bins=15,
    density=True,
    alpha=0.5,
    color='steelblue',
    label='Top Customers (total_spent)'
)

# KDE for top_customers total_spent
top_customers['total_spent'].plot.kde(
    bw_method=0.3,
    color='steelblue',
    linewidth=2
)

# Histogram for aggregated customer_total_spend
customer_total_spend['total_spent'].plot.hist(
    bins=15,
    density=True,
    alpha=0.5,
    color='orange',
    label='All Customers (aggregated spend)'
)

# KDE for aggregated customer_total_spend
customer_total_spend['total_spent'].plot.kde(
    bw_method=0.3,
    color='orange',
    linewidth=2
)

plt.title('Distribution of Customer Total Spend')
plt.xlabel('Total Spend')
plt.ylabel('Density')
plt.legend()
plt.grid(axis='y', alpha=0.3)
plt.show()
```



In [24]:
```
buddy('Did we yet check the distribution curve of the expenditure for each customer?')
```

Yes – we plotted a histogram + KDE for the **total_spent** column (both for the top 10 customers and for all customers aggregated). This shows the distribution of each customer's expenditure.

In [26]:
```
buddy('What derived tables we have?')
```

Derived tables created in the workflow

Table name	Source / operation	Purpose
`cust_orders`	`customers` ⋈ `orders` on `customer_id`	Links customers to their orders
`cust_orders_products`	`cust_orders` ⋈ `order_products` on `order_id`	Adds product line-items to each order
`full_df`	`cust_orders_products` ⋈ `products` on `product_id` + `total_cost = quantity * price`	Full order-level detail with cost
`customer_product_summary`	`full_df` grouped by `customer_id, customer_name, product_id, product_name` (sum `quantity` , `total_cost`)	Per-customer per-product totals
`top_customers`	`full_df` grouped by `customer_id, customer_name` (sum `total_cost` → `total_spent`)	Top 10 spenders
`selected_columns`	`customer_product_summary` selecting `customer_name, product_name, total_cost`	Simplified view
`selected_columns_trimmed`	`selected_columns` with `product_name` regex-cleaned	Clean product names
`customer_total_cost`	`customer_product_summary` grouped by `customer_id, customer_name` (sum `total_cost`) → `total_cost_div_100`	Total spend per customer, scaled
`customer_counts`	`cust_orders` grouped by `customer_id, customer_name` (count rows → `order_count`) → `order_count_x2`	Order counts per customer, doubled
`city_counts`	`customers['city'].value_counts()`	Customer counts per city
`city_amounts`	`customers` ⋈ `orders` on `customer_id` , grouped by `city` (sum `amount` → `total_amount`)	Total order amount per city
`customer_total_spend`	`customer_product_summary` grouped by `customer_id, customer_name` (sum `total_cost` → `total_spent`)	Aggregated spend per customer (used for histograms)

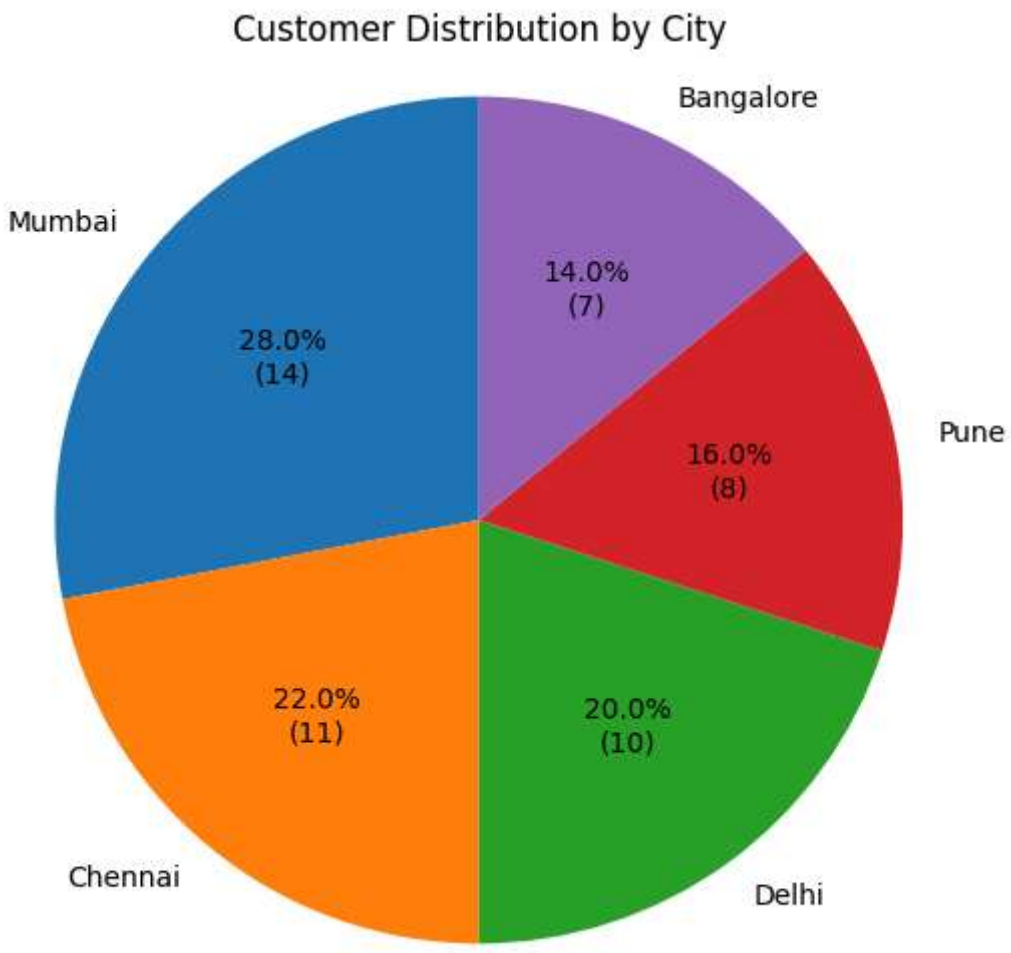
These are the intermediate (derived) tables generated throughout the analysis.

In [27]:
```
zen.do('Which customer has the highest number of orders?')
```

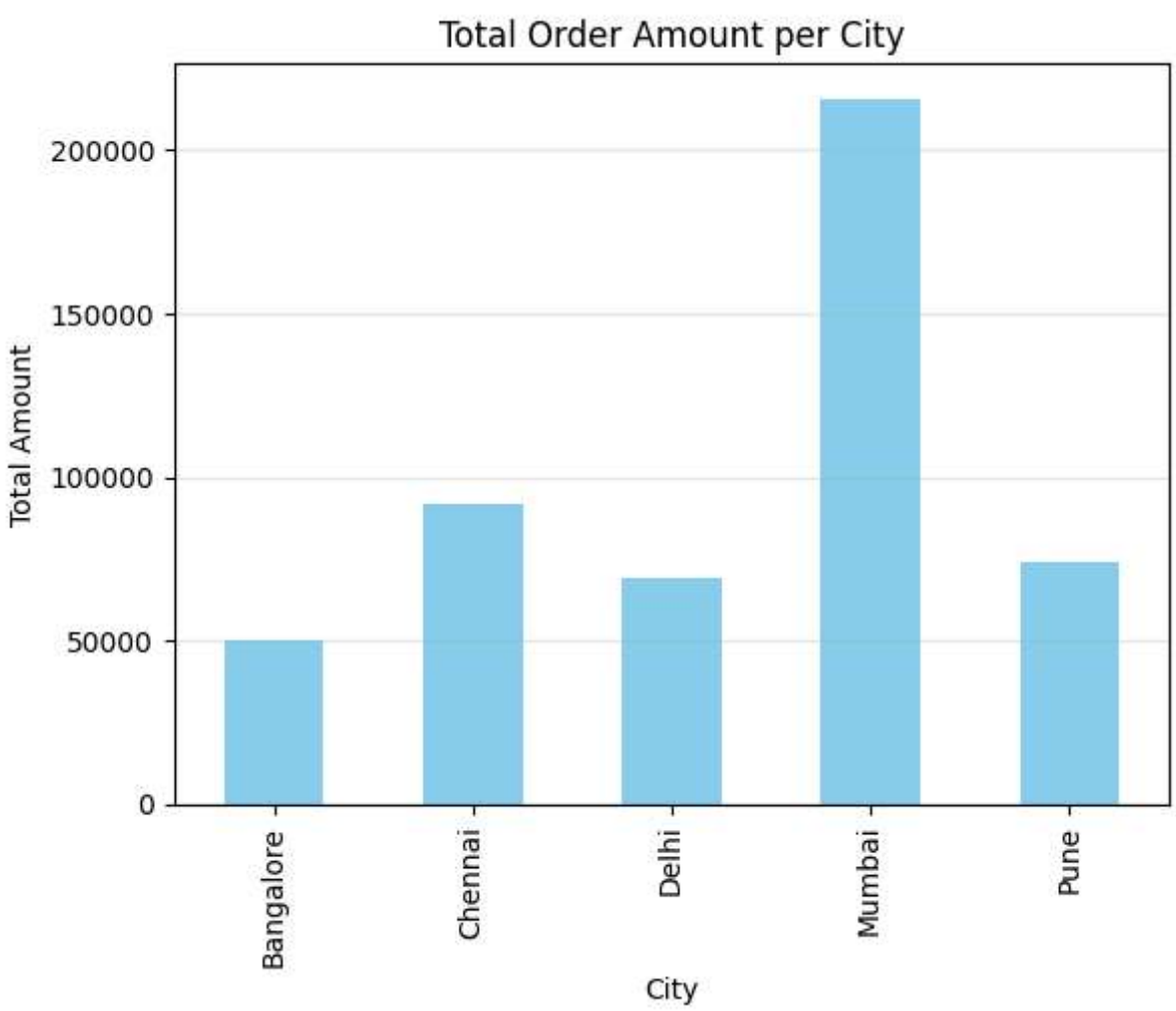
```
# Identify the customer with the highest number of orders using the existing `customer_counts` DataFrame
max_order_row = customer_counts.loc[customer_counts['order_count'].idxmax()]

print("Customer with the highest number of orders:")
print(max_order_row[['customer_id', 'customer_name', 'order_count']])
Customer with the highest number of orders:
customer_id              130
customer_name    Customer_30
order_count                4
Name: 18, dtype: object
```

In [29]:
```
zen.do('Pie chart of customers per city, bar chart of total order amount per city, city-level reporting')
```

```
# Pie chart of customers per city (reuse existing variables)
plt.figure(figsize=(6, 6))
city_counts.plot.pie(
    labels=city_counts.index,
    autopct=autopct_with_counts,
    startangle=90,
    shadow=False
)
plt.title('Customer Distribution by City')
plt.ylabel('')          # hide y-label for cleaner look
plt.axis('equal')       # ensure pie is a circle
plt.show()

# Bar chart of total order amount per city (reuse existing variables)
plt.figure(figsize=(8, 5))
city_amounts.plot.bar(
    x='city',
    y='total_amount',
    legend=False,
    color='skyblue'
)
plt.title('Total Order Amount per City')
plt.xlabel('City')
plt.ylabel('Total Amount')
plt.grid(axis='y', alpha=0.3)
plt.show()

# City-level reporting: combine customer counts and total order amounts
city_report = (
    pd.DataFrame({
        'city': city_counts.index,
        'customer_count': city_counts.values
    })
    .merge(city_amounts, on='city')
    .sort_values('total_amount', ascending=False)
)

print("City-level report (customer count & total order amount):")
print(city_report.head())
```




```
<Figure size 800x500 with 0 Axes>
```




```
City-level report (customer count & total order amount):
        city  customer_count  total_amount
0     Mumbai              14        215624
1    Chennai              11         91618
3       Pune               8         74008
2      Delhi              10         69313
4  Bangalore               7         50383
```

In [30]:
```
zen.do('Show me the doubled order count table')
```

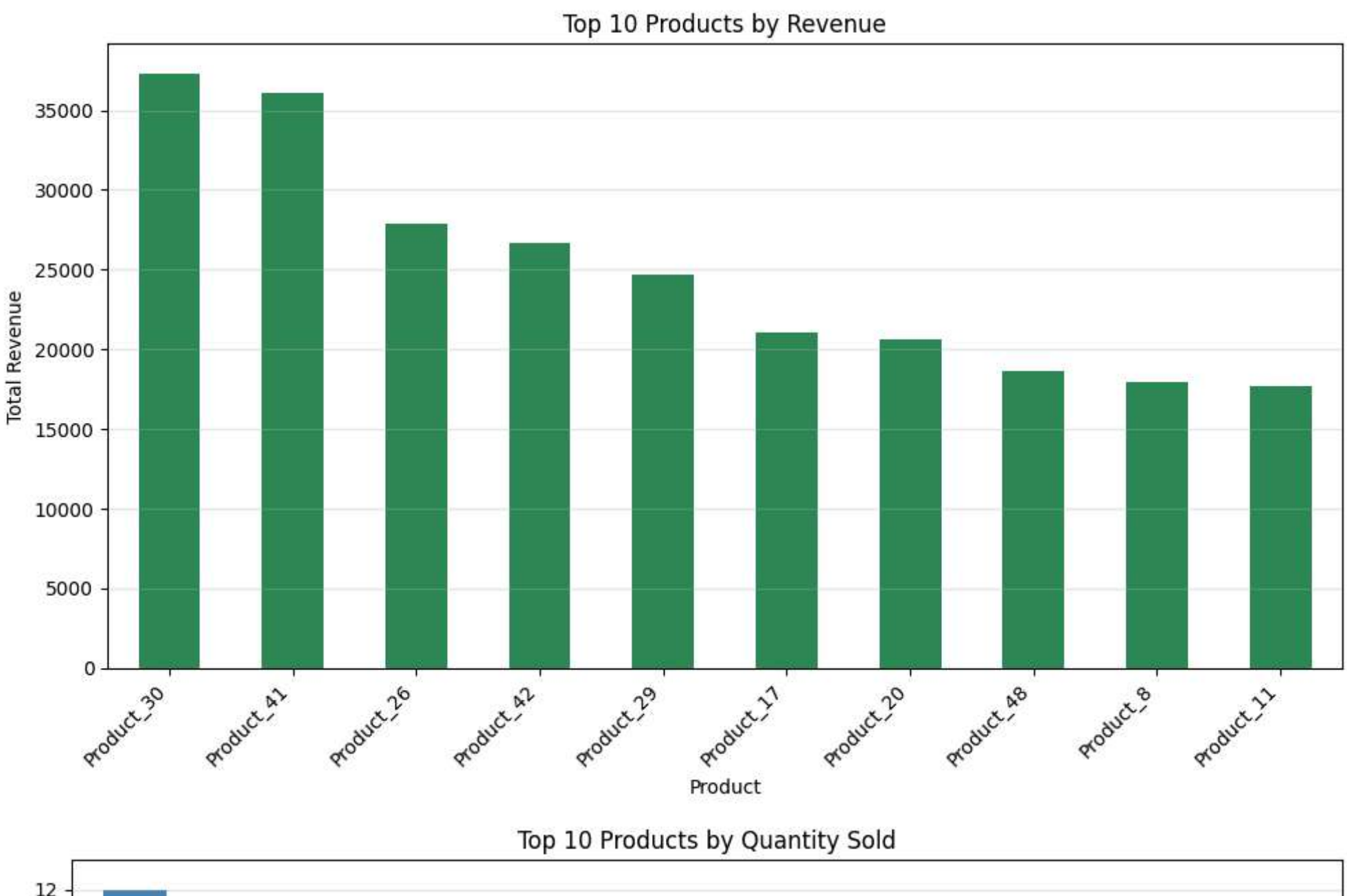
```
print("Doubled order count per customer:")
print(customer_counts[['customer_id', 'customer_name', 'order_count', 'order_count_x2']].head())
Doubled order count per customer:
   customer_id customer_name  order_count  order_count_x2
0          102    Customer_2            2               4
1          103    Customer_3            1               2
2          104    Customer_4            2               4
3          105    Customer_5            1               2
4          107    Customer_7            1               2
```

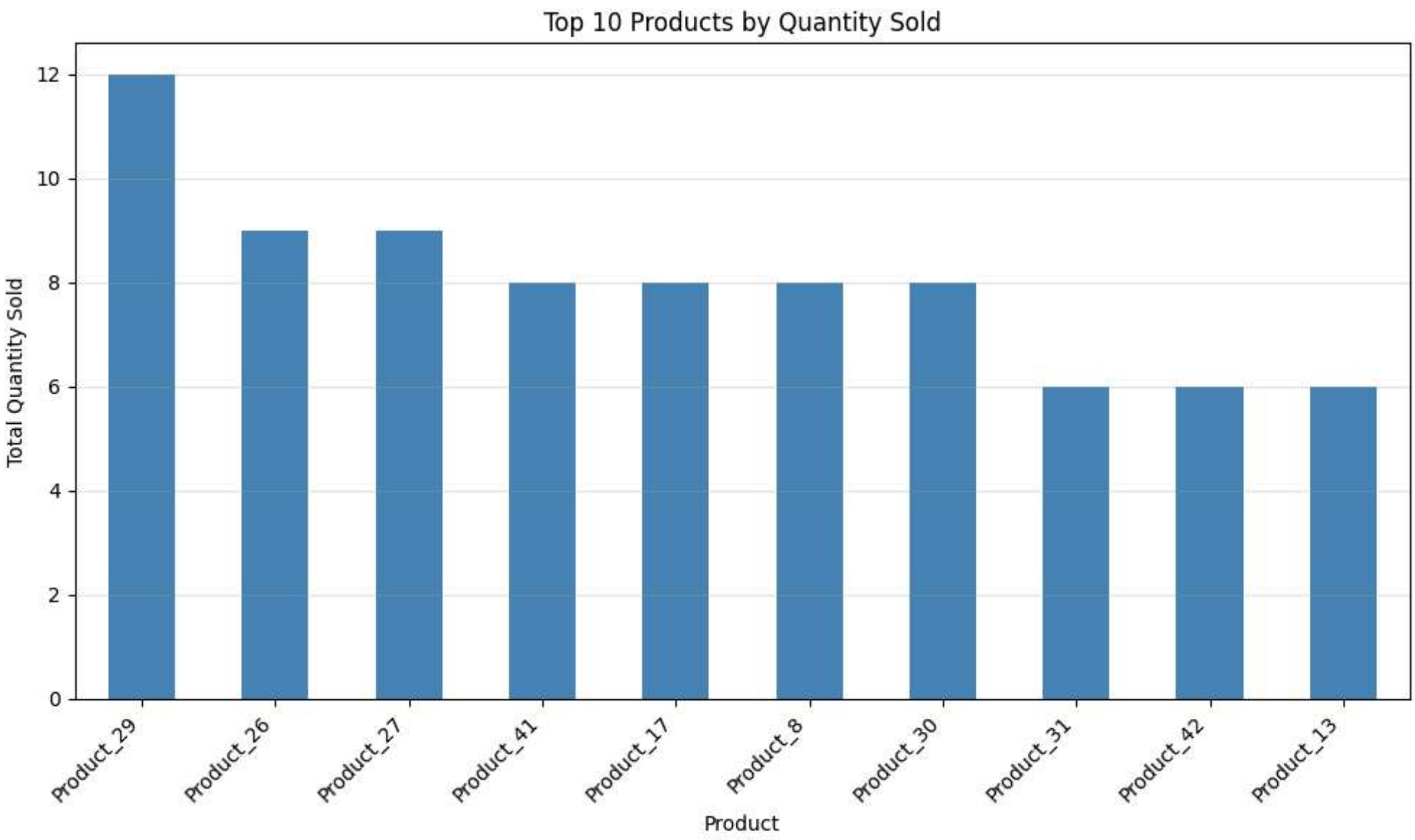
In [32]:
```
zen.do('what is the Product profitability, top-selling product list')
```

```
# Compute total quantity sold per product
product_quantity_total = (
    order_products
    .groupby('product_id', as_index=False)['quantity']
    .sum()
    .rename(columns={'quantity': 'total_quantity'})
)

# Merge quantity with product details to get product_name
product_quantity_total = product_quantity_total.merge(
    products[['product_id', 'product_name']],
    on='product_id',
    how='left'
)

# Merge revenue (product_total_cost) with quantity
# product_total_cost has columns: product_name, total_cost
product_profitability = product_total_cost.merge(
    product_quantity_total[['product_name', 'total_quantity']],
    on='product_name',
    how='left'
).rename(columns={'total_cost': 'total_revenue'})

# Fill any missing quantities (e.g., products never sold) with 0
product_profitability['total_quantity'] = product_profitability['total_quantity'].fillna(0).astype(int)

# Sort by revenue for profitability ranking
product_profitability = product_profitability.sort_values('total_revenue', ascending=False)

print("Product profitability (revenue & quantity sold):")
print(product_profitability.head(10))
```

```
# Top-selling products by quantity sold
top_selling_products = product_profitability.sort_values('total_quantity', ascending=False)

print("\nTop-selling products (by quantity sold):")
print(top_selling_products.head(10))

# Plot top 10 products by revenue
plt.figure(figsize=(10, 6))
product_profitability.head(10).set_index('product_name')['total_revenue'].plot.bar(
    color='seagreen',
    legend=False
)
plt.title('Top 10 Products by Revenue')
plt.xlabel('Product')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

# Plot top 10 products by quantity sold
plt.figure(figsize=(10, 6))
top_selling_products.head(10).set_index('product_name')['total_quantity'].plot.bar(
    color='steelblue',
    legend=False
)
plt.title('Top 10 Products by Quantity Sold')
plt.xlabel('Product')
plt.ylabel('Total Quantity Sold')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```

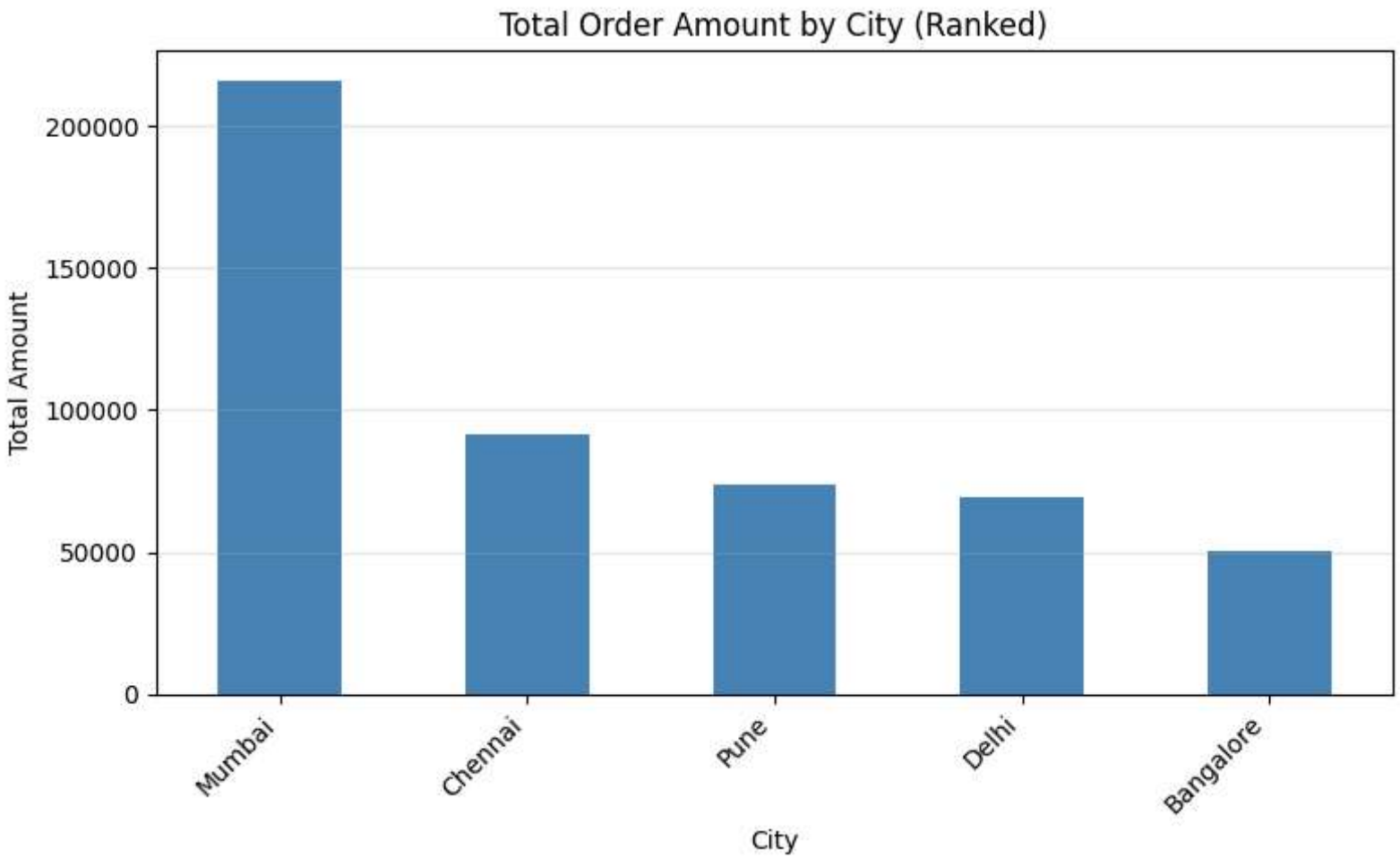
```
Product profitability (revenue & quantity sold):
   product_name  total_revenue  total_quantity
20   Product_30          37304               8
29   Product_41          36104               8
15   Product_26          27873               9
30   Product_42          26676               6
18   Product_29          24684              12
6    Product_17          21088               8
10   Product_20          20635               5
33   Product_48          18620               5
37    Product_8          17992               8
1    Product_11          17704               4

Top-selling products (by quantity sold):
   product_name  total_revenue  total_quantity
18   Product_29          24684              12
15   Product_26          27873               9
16   Product_27           6354               9
29   Product_41          36104               8
6    Product_17          21088               8
37    Product_8          17992               8
20   Product_30          37304               8
21   Product_31           4062               6
30   Product_42          26676               6
3    Product_13          11790               6
```

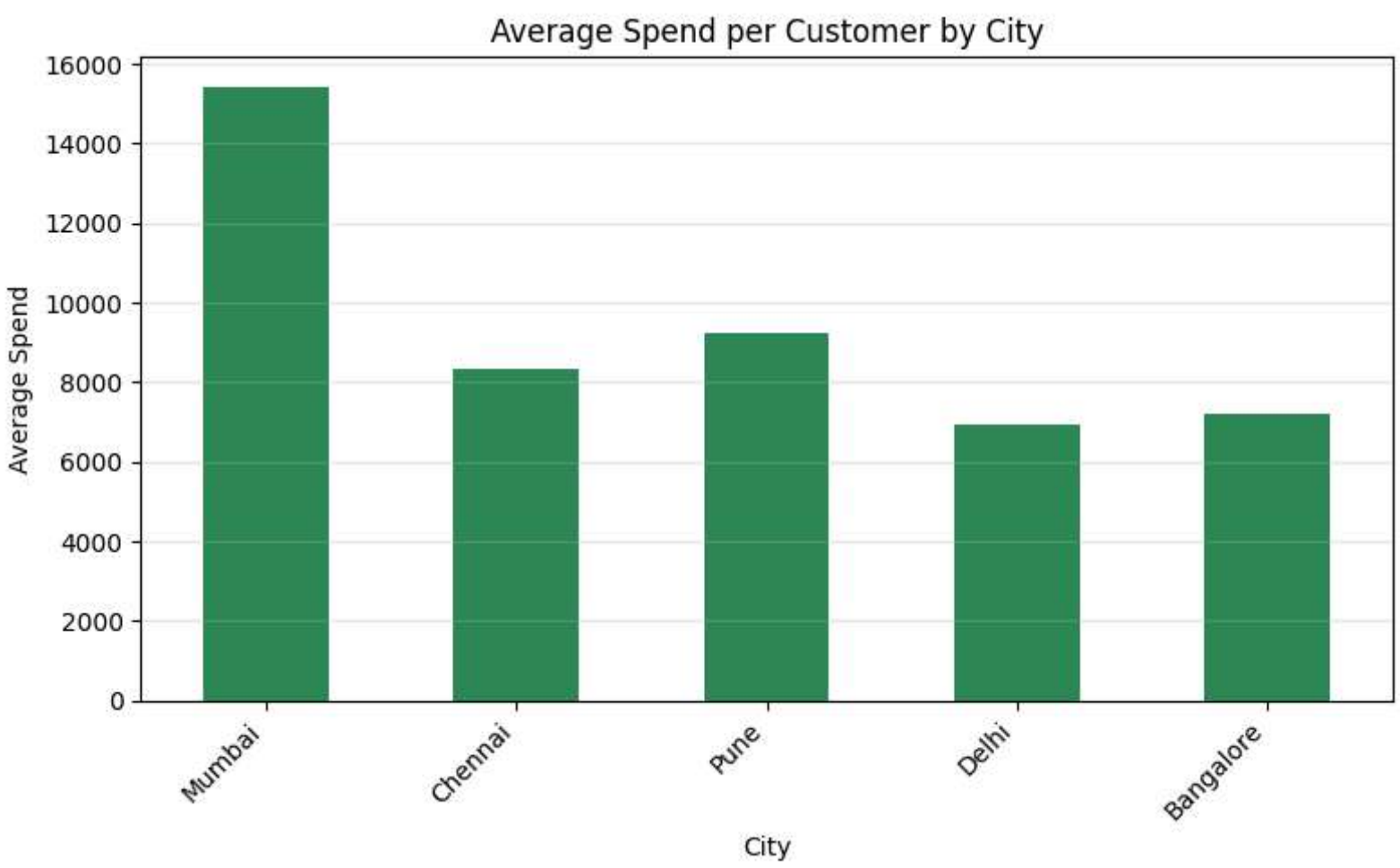




In [33]:
```
zen.do('City-level revenue analysis, ranking cities by spend')
```

```
# City-level revenue analysis: combine customer counts with total order amount, compute average spend, rank cities
city_report = (
    pd.DataFrame({
        'city': city_counts.index,
        'customer_count': city_counts.values
    })
    .merge(city_amounts, on='city')
)

city_report['avg_spend_per_customer'] = city_report['total_amount'] / city_report['customer_count']
city_report = city_report.sort_values('total_amount', ascending=False)

print("City-level revenue report (sorted by total amount):")
print(city_report.head())

# Bar chart of total order amount per city (ranked)
plt.figure(figsize=(8, 5))
city_report.set_index('city')['total_amount'].plot.bar(color='steelblue')
plt.title('Total Order Amount by City (Ranked)')
plt.xlabel('City')
plt.ylabel('Total Amount')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

# Bar chart of average spend per customer per city (ranked)
plt.figure(figsize=(8, 5))
city_report.set_index('city')['avg_spend_per_customer'].plot.bar(color='seagreen')
plt.title('Average Spend per Customer by City')
plt.xlabel('City')
plt.ylabel('Average Spend')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
City-level revenue report (sorted by total amount):
        city  customer_count  total_amount  avg_spend_per_customer
0     Mumbai              14        215624            15401.714286
1    Chennai              11         91618             8328.909091
3       Pune               8         74008             9251.000000
2      Delhi              10         69313             6931.300000
4  Bangalore               7         50383             7197.571429
```





In [34]:
```
buddy('Among all the derived tables, what analysis could potentially use a already derived table and which one except the one in the 3rd que
```

Derived tables that can be reused

Derived table	Analysis that can reuse it
`city_counts`	City-level revenue analysis (pie chart, bar chart, report)
`city_amounts`	City-level revenue analysis (bar chart, report)
`customer_product_summary`	Top-customers, product profitability, total-spend histogram
`cust_orders`	Double-order-count analysis
`product_total_cost`	Product profitability
`product_quantity_total`	Product profitability
`customer_counts`	Customer-with-most-orders, doubled-order-count display

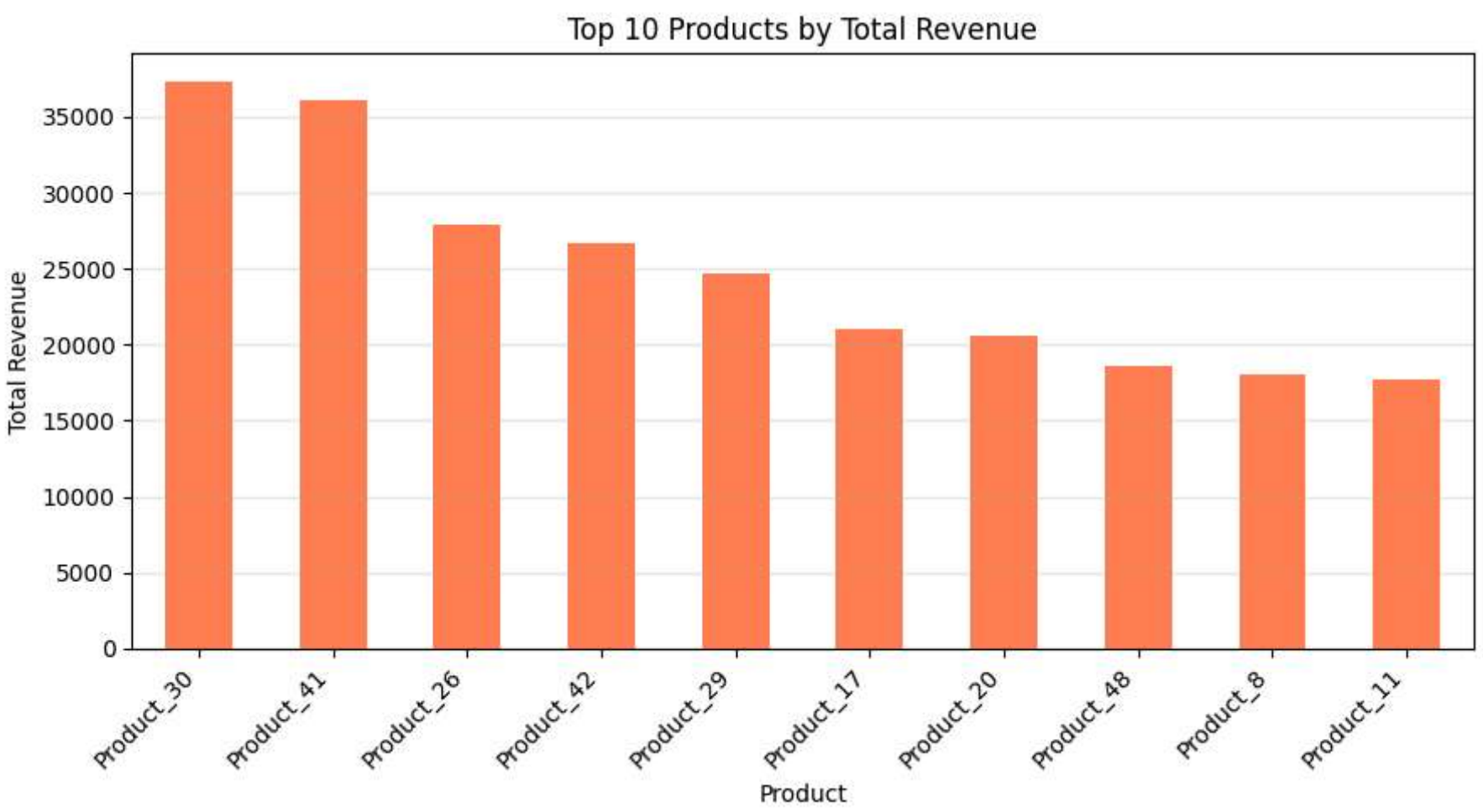
Derived table used only in the 3rd query

- `selected_columns` is created in the 3rd query and is only referenced there (its trimmed copy is a derivative, not a separate reuse).

So, the city-level revenue analysis can reuse existing derived tables, while the 3rd query's `selected_columns` table is not reused elsewhere.

In [35]:
```
zen.do('show aggregated product-level cost view. There is no product_id column.')
```

```
# aggregated product-level cost view (product_name, total_revenue, total_quantity)
agg_product_cost = (
    product_profitability[['product_name', 'total_revenue', 'total_quantity']]
    .sort_values('total_revenue', ascending=False)
)

print("Aggregated product-level cost view (top 10 by revenue):")
print(agg_product_cost.head(10))

# bar chart of top 10 products by total revenue
plt.figure(figsize=(9, 5))
agg_product_cost.head(10).set_index('product_name')['total_revenue'].plot.bar(color='coral')
plt.title('Top 10 Products by Total Revenue')
plt.xlabel('Product')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
Aggregated product-level cost view (top 10 by revenue):
   product_name  total_revenue  total_quantity
20   Product_30          37304               8
29   Product_41          36104               8
15   Product_26          27873               9
30   Product_42          26676               6
18   Product_29          24684              12
6    Product_17          21088               8
10   Product_20          20635               5
33   Product_48          18620               5
37    Product_8          17992               8
1    Product_11          17704               4
```



In [36]:
```
zen.do('show a table with product ids and their total revenue.')
```

```
# Compute total revenue per product_id by merging existing product_total_cost with products
product_revenue_by_id = (
    product_total_cost
    .merge(products[['product_id', 'product_name']], on='product_name')
    .rename(columns={'total_cost': 'total_revenue'})
    [['product_id', 'total_revenue']]
    .sort_values('total_revenue', ascending=False)
)

print("Product IDs with their total revenue (sorted descending):")
print(product_revenue_by_id.head(20))  # show top 20 rows for inspection
```

```
Product IDs with their total revenue (sorted descending):
    product_id  total_revenue
20         330          37304
29         341          36104
15         326          27873
30         342          26676
18         329          24684
6          317          21088
10         320          20635
33         348          18620
37         308          17992
1          311          17704
34         350          17520
28         340          14396
8          319          13296
26         339          12708
3          313          11790
25         337          11679
17         328          10881
32         345          10745
31         344          10416
0          301           8877
```

In [37]:
```
print(zen.get_history)
```

[Q: count total customers and distinct cities] `total_customers` set to `len(customers)`; `distinct_cities` set to `customers['‹city›'].nunique()`; printed both values and a sample of `customers`.

[Q: list each customer with ordered products, quantities, and total cost] `cust_orders` created by merging `customers` with `orders` on `‹customer_id›`; `cust_orders_products` created by merging `cust_orders` with `order_products` on `‹order_id›`; `full_df` created by merging `cust_orders_products` with `products` on `‹product_id›`; added column `‹total_cost›` as `‹quantity› * ‹price›`; `customer_product_summary` aggregated `full_df` by `‹customer_id›`, `‹customer_name›`, `‹product_id›`, `‹product_name›` summing `‹quantity›` and `‹total_cost›`; `top_customers` aggregated `full_df` by `‹customer_id›`, `‹customer_name›` summing `‹total_cost›` as `‹total_spent›`, sorted descending and kept top 10; plotted horizontal bar chart of `top_customers`.

[Q: keep only customer name, product name, and cost] `selected_columns` created from `customer_product_summary` selecting `‹customer_name›`, `‹product_name›`, `‹total_cost›`; printed sample.

[Q: pie chart of customer counts per city] `city_counts` set to `customers['‹city›'].value_counts()`; printed counts; plotted pie chart of `city_counts` with labels and percentages.

[Q: show counts on the pie chart] reused `city_counts`; defined `autopct_with_counts` to display percentage and absolute count; plotted pie chart using this function for autopct.

[Q: total order amount per city] `city_amounts` created by merging `customers` with `orders` on `‹customer_id›`, grouping by `‹city›` and summing `‹amount›`, then renaming to `‹total_amount›`; printed result; plotted bar chart of `‹city›` vs `‹total_amount›`.

[Q: divide each customer's total cost by 100] `customer_total_cost` was created by grouping `customer_product_summary` on ‹customer_id› and ‹customer_name›, summing ‹total_cost›, then a new column ‹total_cost_div_100› was added as ‹total_cost› / 100; the first rows were printed.

[Q: remove the "Product_" prefix from product names] `selected_columns_trimmed` was made as a copy of `selected_columns` and its ‹product_name› column was transformed with a regex replace to drop the leading "Product_"; the first rows were printed.

[Q: double the order count for each customer] `customer_counts` was created by grouping `cust_orders` on ‹customer_id› and ‹customer_name›, counting rows with .size(), renaming the count column to ‹order_count›, then adding ‹order_count_x2› as ‹order_count› * 2; the first rows were printed.

[Q: get product name and total cost] Created `product_total_cost` by grouping `customer_product_summary` on ‹product_name› and summing ‹total_cost›, then printed its head.

[Q: re-plot last pie chart] Used `plt.figure` and called `city_counts.plot.pie` with labels from `city_counts.index`, `autopct_with_counts` for percentages/counts, startangle 90, no shadow, added title “Customer Distribution by City”, hid y-label, set equal axis, and displayed the plot.

[Q: histogram/KDE of total_spent] Built `customer_total_spend` by grouping `customer_product_summary` on ‹customer_id› and ‹customer_name›, summing ‹total_cost› and renaming it to ‹total_spent›. Then plotted overlapping histograms (15 bins, density) and KDEs for `top_customers['total_spent']` (steelblue) and `customer_total_spend['total_spent']` (orange), added title, axis labels, legend, grid, and showed the figure.

[Q: customer with most orders] Selected the row in `customer_counts` where ‹order_count› is maximal, stored it in `max_order_row`, and printed ‹customer_id›, ‹customer_name›, and ‹order_count›.

[Q: Pie chart of customers per city, bar chart of total order amount per city, city-level report] Plotted a pie chart using `city_counts` (labels ‹city›, values ‹customer_count›) and a bar chart using `city_amounts` (x ‹city›, y ‹total_amount›); created `city_report` by merging a DataFrame built from `city_counts` (columns ‹city›, ‹customer_count›) with `city_amounts` on ‹city› and sorting descending by ‹total_amount›.

[Q: Show doubled order count per customer] Printed the first rows of `customer_counts` selecting columns ‹customer_id›, ‹customer_name›, ‹order_count›, ‹order_count_x2›.

[Q: Compute product profitability and list top-selling products] Calculated `product_quantity_total` by grouping `order_products` on ‹product_id› and summing ‹quantity›, renamed to ‹total_quantity›, then merged with `products` to add ‹product_name›; merged `product_total_cost` (renamed ‹total_revenue› from ‹total_cost›) with `product_quantity_total` on ‹product_name› to form `product_profitability`, filled missing ‹total_quantity› with 0, sorted by ‹total_revenue› descending and displayed the top 10; created `top_selling_products` by sorting `product_profitability` on ‹total_quantity› descending and displayed the top 10; plotted bar charts of the top 10 products by ‹total_revenue› and by ‹total_quantity› respectively.

[Q: city-level revenue ranking] Created `city_report` by merging a DataFrame built from `city_counts` (‹city›, ‹customer_count›) with `city_amounts` on ‹city›, added ‹avg_spend_per_customer› as ‹total_amount› / ‹customer_count›, sorted descending by ‹total_amount›, printed the top rows, and plotted bar charts of ‹total_amount› and ‹avg_spend_per_customer› per ‹city›.

[Q: product-level cost aggregation] Created `agg_product_cost` from `product_profitability` selecting ‹product_name›, ‹total_revenue›, ‹total_quantity›, sorted descending by ‹total_revenue›, printed the top 10 rows, and plotted a bar chart of the top 10 ‹total_revenue› values by ‹product_name›.

Time 06:39 - User query: show a table with product ids and their total revenue.
Response:
```
# Compute total revenue per product_id by merging existing product_total_cost with products
product_revenue_by_id = (
    product_total_cost
    .merge(products[['product_id', 'product_name']], on='product_name')
    .rename(columns={'total_cost': 'total_revenue'})
    [['product_id', 'total_revenue']]
    .sort_values('total_revenue', ascending=False)
)

print("Product IDs with their total revenue (sorted descending):")
print(product_revenue_by_id.head(20))  # show top 20 rows for inspection
```

In [ ]: