Exploratory Data Analysis (EDA) – Descriptive Analysis

Geographic Sales Analysis

Which city achieved the highest total sales, and what was the total sales amount for that city?

1. Overview

This analysis explores geographic sales performance across multiple cities using exploratory data analysis (EDA). By aggregating total sales figures city-wise, we aim to uncover which city contributed the highest to the overall revenue. Visualizing this information not only provides insights into regional performance but also supports strategic decisions for resource allocation, marketing investment, and operational optimization.

2. Goal

- Identify the city with the highest total sales.
- Quantify the total sales amount associated with each city.
- Visualize sales distribution across cities using an intuitive bar chart.
- Support business strategy through location-based performance insights.

3. Business Challenge

- Sales performance is uneven across regions, and decision-makers lack clarity on high-performing cities.
- Without clear geographic insights, marketing and inventory investments may be misaligned.
- Limited visibility into revenue contribution by location may affect expansion or scaling plans.

4. Methodology

- Load and preprocess the dataset to ensure accurate and complete data.
- Group sales data by city and compute total sales for each location.
- Visualize the data using a bar chart to highlight the top-performing city.
- Format the chart for readability and presentation (currency formatting, labels, and grid).

- Derive insights from the chart and apply them to inform business decisions.

Import necessary libraries

In [9]:
```
import pandas as pd
import os
import glob
```

Combine the sales data from all months into a single consolidated CSV file

In [11]:
```
folder_path = r"C:\Monthly_Sales"

# Retrieve all CSV files from the folder using glob
all_files = glob.glob(os.path.join(folder_path, "*.csv"))

# All CSV files combined as one DataFrame
all_data = pd.concat([pd.read_csv(file) for file in all_files], ignore_index=True)

# Merged DataFrame saved into a new CSV
output_file = os.path.join(folder_path, "all_data.csv")
all_data.to_csv(output_file, index=False)

print("All files integrated into:", output_file)
```

```
All files integrated into: C:\Monthly_Sales\all_data.csv
```

Load the updated DataFrame

In [13]:
```
# Skip Blank Rows if present in the dataset

df = pd.read_csv(r'C:\Monthly_Sales\all_data.csv', skip_blank_lines=True)
df.head()
```

Out[13]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address
0	175667	iPhone	1	700.0	04/24/24 19:12	135 Meadow St, Boston, MA 02215
1	175668	AA Batteries (4-pack)	1	5.84	04/20/24 13:45	592 4th St, San Francisco, CA 94016
2	175669	AA Batteries (4-pack)	1	5.84	04/28/24 09:17	632 Park St, Dallas, TX 75001
3	175670	AA Batteries (4-pack)	2	5.84	04/23/24 14:06	131 Pine St, San Francisco, CA 94016
4	175671	Samsung Odyssey Monitor	1	409.99	04/23/24 12:13	836 Forest St, Boston, MA 02215

In [14]:
```
df.shape
```

Out[14]:
```
(8281490, 6)
```

Data Cleaning Process

Thoroughly clean and standardize the data to eliminate errors, ensure consistency, and build a solid foundation for meaningful insights.

Find and remove rows with NaN values

In [17]:
```
df.isna().sum()
```

Out[17]:
```
Order ID            21504
Product Name        21504
Units Purchased     21506
Unit Price          21506
Order Date          21507
Delivery Address    21508
dtype: int64
```

In [18]:
```
# If Nan value is present in Order ID and Unit Purchased, it will be impossible to
# Therefore, drop Nan values in Order ID and Units Purchased.

df.dropna(subset=['Order ID', 'Units Purchased'], inplace=True)
```

In [19]:
```
# Check if Nan value is present

df.isna().sum()
```

Out[19]:
```
Order ID            0
Product Name        0
Units Purchased     0
Unit Price          0
Order Date          1
Delivery Address    2
dtype: int64
```

In [20]:
```
# Further check if any NaN values or blank rows are present

blank_rows_na = df[df.isnull().any(axis=1)]
blank_rows_na
```

Out[20]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address
2195228	Charging Cable	1	14.95	05/24/24 07:04	852 Hickory St, San Francisco, CA 94016	NaN
3001506	150766	iPhone	1	7	NaN	NaN

Find and remove rows with duplicate values

In [22]:
```
# Find duplicate values

df.duplicated()
```

Out[22]:
```
0          False
1          False
2          False
3          False
4          False
           ...
8281485     True
8281486     True
8281487     True
8281488     True
8281489     True
Length: 8259984, dtype: bool
```

In [23]:
```
# Remove duplicated values

df.drop_duplicates(inplace = True)
```

In [24]:
```
# Check again for duplicated values

df.duplicated()
```

Out[24]:
```
0          False
1          False
2          False
3          False
4          False
           ...
172530     False
2195228    False
3001506    False
6370083    False
6403571    False
Length: 171546, dtype: bool
```

Verify and fix incorrect data types in the dataset

In [26]:
```
# check for data types

df.dtypes
```

Out[26]:
```
Order ID            object
Product Name        object
Units Purchased     object
Unit Price          object
Order Date          object
Delivery Address    object
dtype: object
```

Fix incorrect data types

In [28]:
```
df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%y %H:%M', errors

df['Units Purchased'] = pd. to_numeric(df['Units Purchased'], errors='coerce')

df['Unit Price'] = pd. to_numeric(df['Unit Price'], errors='coerce')
```

In [29]:
```
# Verify the presence of NaN values remaining in the columns as a result of using e

df.isna().sum()
```

Out[29]:
```
Order ID            0
Product Name        0
Units Purchased     1
Unit Price          2
Order Date          3
Delivery Address    2
dtype: int64
```

In [30]:
```
df = df.dropna()
```

Change the data type to optimize memory usage (Optional)

In [32]:
```
df['Order ID'] = pd.to_numeric(df['Order ID'], downcast='integer')
df['Product Name'] = df['Product Name'].astype('category')
df['Units Purchased'] = df['Units Purchased']. astype('int8')
df['Unit Price'] = pd.to_numeric(df['Unit Price'], downcast='float')
df['Delivery Address'] = df['Delivery Address'].astype('category')
```

Expand the dataset with supplementary columns

Add month column

In [35]:
```
df['Month'] = df['Order Date'].dt.month
df
```

Out[35]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4
...	...	...	...	...	...	...	...
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8

171543 rows × 7 columns

In [36]:
```
df['Month Name'] = df['Order Date'].dt.strftime('%B')
df
```

Out[36]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	
...	...	...	...	...	...	...	...	
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	Septer
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	Septer
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	Septer
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	Oct

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	Au

171543 rows × 8 columns

Add week day column

In [38]:
```
df['Day of Week'] = df['Order Date'].dt.strftime('%a')
df
```

Out[38]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	
...	...	...	...	...	...	...	...	
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	Septer
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	Septer
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	Septer
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	Oct

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	Au

171543 rows × 9 columns

Add hour column

In [40]:
```
df['Hour'] = df['Order Date'].dt.hour
df
```

Out[40]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	
...	...	...	...	...	...	...	...	
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	Septer
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	Septer
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	Septer
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	Oct

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	Au

171543 rows × 10 columns

Add city column

In [42]:
```
def city(address):
    return address.split(",")[1].strip(" ")

def state_abbrev(address):
    return address.split(",")[2].split(" ")[1]

df['City'] = df['Delivery Address'].apply(lambda x: f"{city(x)}  ({state_abbrev(x)}
df.head()
```

Out[42]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name	Day of Week
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	April	Wed
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	April	Sat
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	April	Sun
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	April	Tue
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	April	Tue

Organize Data by Order Date Chronologically and Reindex

In [44]:
```
df = df.sort_values(by = 'Order Date')
df
```

Out[44]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name
78282	160155	Alienware Monitor	1	400.989990	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	January
68761	151041	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	January
64303	146765	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	January
63092	145617	Amana Washing Machine	1	600.000000	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	January
74502	156535	iPhone	1	700.000000	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	January
...	...	...	...	...	...	...	...	...
44457	297748	iPhone	1	700.000000	2025-01-01 02:37:00	258 Forest St, Los Angeles, CA 90001	1	January
30663	284606	Bose SoundSport Headphones	1	99.989998	2025-01-01 02:50:00	211 Johnson St, Boston, MA 02215	1	January
49246	302330	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:03:00	665 6th St, San Francisco, CA 94016	1	January
30770	284711	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:19:00	250 8th St, San Francisco, CA 94016	1	January

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name
50619	303626	USB-C Charging Cable	3	11.950000	2025-01-01 04:43:00	651 Lakeview St, Dallas, TX 75001	1	January

171543 rows × 11 columns

In [45]:
```
df = df.reset_index(drop=True)
df
```

Out[45]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Nam
0	160155	Alienware Monitor	1	400.989990	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	Januar
1	151041	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	Januar
2	146765	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	Januar
3	145617	Amana Washing Machine	1	600.000000	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	Januar
4	156535	iPhone	1	700.000000	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	Januar
...	...	...	...	...	...	...	...	.
171538	297748	iPhone	1	700.000000	2025-01-01 02:37:00	258 Forest St, Los Angeles, CA 90001	1	Januar
171539	284606	Bose SoundSport Headphones	1	99.989998	2025-01-01 02:50:00	211 Johnson St, Boston, MA 02215	1	Januar
171540	302330	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:03:00	665 6th St, San Francisco, CA 94016	1	Januar
171541	284711	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:19:00	250 8th St, San Francisco, CA 94016	1	Januar

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Mont Nam
171542	303626	USB-C Charging Cable	3	11.950000	2025-01-01 04:43:00	651 Lakeview St, Dallas, TX 75001	1	Januar

171543 rows × 11 columns

Add Total Sales column

In [47]:
```
df['Total Sales'] = df['Units Purchased'] * df['Unit Price']
df.head()
```

Out[47]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name	Day of Week
0	160155	Alienware Monitor	1	400.98999	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	January	Mon
1	151041	AAA Batteries (4-pack)	1	4.99000	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	January	Mon
2	146765	AAA Batteries (4-pack)	1	4.99000	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	January	Mon
3	145617	Amana Washing Machine	1	600.00000	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	January	Mon
4	156535	iPhone	1	700.00000	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	January	Mon

Format Unit Price and Total Sales to 2 decimal places

In [49]:
```
df['Unit Price'] = df['Unit Price'].apply(lambda x: "%.2f" % x)
```

In [50]:
```
df['Total Sales'] = df['Total Sales'].apply(lambda x: "%.2f" % x)
df.head()
```

Out[50]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name	Day of Week	Hc
0	160155	Alienware Monitor	1	400.99	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	January	Mon	
1	151041	AAA Batteries (4-pack)	1	4.99	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	January	Mon	
2	146765	AAA Batteries (4-pack)	1	4.99	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	January	Mon	
3	145617	Amana Washing Machine	1	600.00	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	January	Mon	
4	156535	iPhone	1	700.00	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	January	Mon	

Format Unit Price and Total Sales to numeric

In [52]:
```
df['Unit Price'] = pd.to_numeric(df['Unit Price'])
df['Total Sales'] = pd.to_numeric(df['Total Sales'])
```

Replace 'Boston (\rA)' with 'Boston (MA)'

In [97]:
```
import re

df['City'] = df['City'].str.replace(r'Boston\s+\(\rA\)', 'Boston  (MA)', regex=True
```

Total Sales for each City

In [99]:
```
City_sales = df.groupby('City', observed=False)['Total Sales'].sum()
City_sales
```

Out[99]:
```
City
Atlanta  (GA)          3145310.00
Austin  (TX)           2178728.99
Boston  (MA)           4166657.99
Dallas  (TX)           3193629.21
Los Angeles  (CA)      6343771.43
New York City  (NY)    5200781.23
Portland  (ME)          541531.52
Portland  (OR)         2053173.81
San Francisco  (CA)    9391881.93
Seattle  (WA)          3114138.92
Name: Total Sales, dtype: float64
```

Plot City Sales

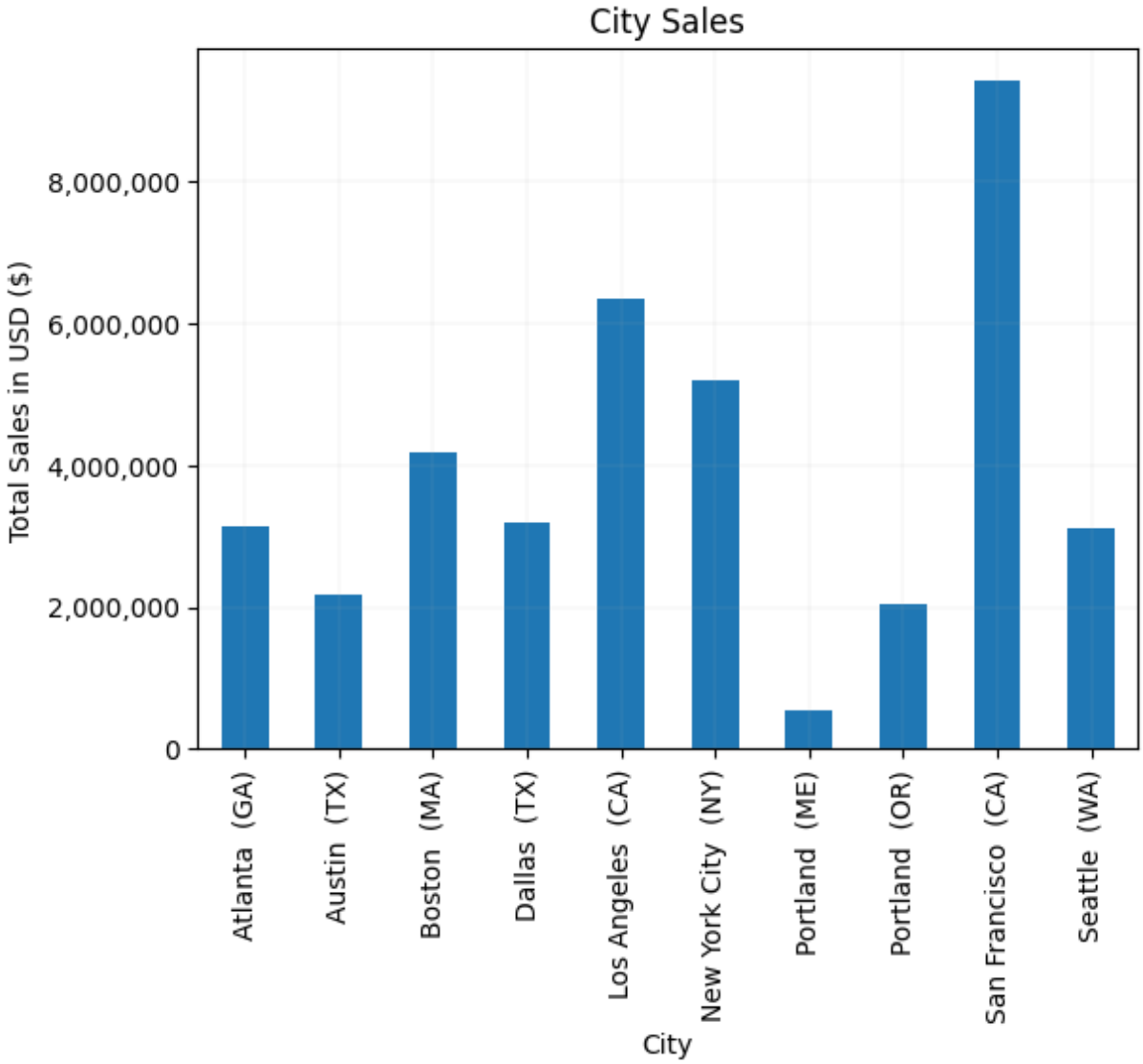
In [103…
```python
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

ax = City_sales.plot(kind='bar', title="City Sales")

ax.set_xlabel('City')
ax.set_ylabel('Total Sales in USD ($)')

ax.get_yaxis().set_major_formatter(plt.FuncFormatter(lambda x, _: f'{int(x):,}'))

plt.grid(linewidth=0.1)
plt.show()
```



Key Insights

1. San Francisco (CA) recorded the highest total sales, pulling in approximately $9.39M, significantly outpacing all other cities.
2. Los Angeles (CA) and New York City (NY) followed with $6.34Mand5.2M$ respectively — making California and New York powerhouses of revenue generation.
3. Portland (ME) posted the lowest total sales at $541K, over 17x lower than San Francisco.

There is a notable performance gap between top-tier and mid/lower-tier cities. Cities like Austin (TX), Portland (OR), and Seattle (WA) are in the mid-range, suggesting room for growth.

Strategic Recommendations

1. Double Down on High-Performing Cities: Focus marketing spend and promotions in San Francisco, LA, and NYC — these cities are already strong, and pushing further can yield compounding gains.
2. Analyze Seasonal Trends: Investigate month-by-month growth patterns to pinpoint peak sales months, then align product launches and campaigns with those windows.
3. Activate Underperforming Markets: Design city-specific campaigns to boost engagement in Portland (ME) and Austin (TX) — consider pricing tweaks, localized offers, or market research.
4. Segment Customers by Region: Perform deeper customer segmentation to uncover regional preferences — what works in SF might flop in Portland.
5. Forecast Sales Using Monthly Trends: Build a monthly forecasting model to proactively manage supply chain, inventory, and budgeting — reducing waste and maximizing responsiveness.