**Code Explanation:**

**1️⃣ Extract: Load the Dataset**

python

CopyEdit

df = pd.read\_csv("superstore\_sales.csv", encoding="ISO-8859-1")

* This line loads the dataset named "superstore\_sales.csv" into a pandas DataFrame df.
* The encoding="ISO-8859-1" argument is used to ensure proper reading of the file in case of non-UTF-8 characters.

**2️⃣ Transform: Data Cleaning and Processing**

* **Convert Date Columns to Datetime Format:**

python

CopyEdit

df['Order Date'] = pd.to\_datetime(df['Order Date'])

df['Ship Date'] = pd.to\_datetime(df['Ship Date'])

* These two lines convert the Order Date and Ship Date columns to pandas' datetime format for easy date manipulations and analysis.
* **Fill Missing Values:**

python

CopyEdit

df.fillna(0, inplace=True)

* This line replaces any missing (NaN) values in the DataFrame df with 0. The argument inplace=True ensures the operation modifies the original df directly without creating a copy.
* **Explanation of fillna(0, inplace=True)**:
  + fillna(0) replaces all the missing or NaN (Not a Number) values in the DataFrame with 0. This is useful in cases where missing values might interfere with analysis, especially when performing mathematical operations.
  + inplace=True modifies the DataFrame df in place, meaning it doesn’t return a new DataFrame but updates df directly. Without this argument, the result would be returned as a new DataFrame, and you would need to assign it back to df or another variable.
* **Remove Duplicates:**

python

CopyEdit

df.drop\_duplicates(inplace=True)

* This line removes any duplicate rows in the DataFrame df, ensuring that there are no repeated entries in the dataset. The inplace=True argument ensures the changes are made directly to df.

**3️⃣ Load: Save the Cleaned Data**

python

CopyEdit

df.to\_csv("cleaned\_superstore\_sales.csv", index=False)

* This line saves the cleaned DataFrame df into a new CSV file called "cleaned\_superstore\_sales.csv".
* The index=False argument ensures that the DataFrame index is not included in the saved CSV file.

**4️⃣ Data Visualization**

**🎯 Sales Trend Over Time (General Visualization)**

python

CopyEdit

plt.figure(figsize=(10,5))

df.groupby("Order Date")["Sales"].sum().plot()

plt.title("Sales Trend Over Time")

plt.xlabel("Date")

plt.ylabel("Total Sales")

plt.grid()

plt.show()

* **Explanation:**
  + df.groupby("Order Date")["Sales"].sum().plot():
    - Groups the data by Order Date, sums up the Sales for each date, and then plots the data.
  + plt.title("Sales Trend Over Time") adds a title to the plot.
  + plt.xlabel("Date") and plt.ylabel("Total Sales") label the x and y axes, respectively.
  + plt.grid() adds grid lines to the plot for better readability.
  + plt.show() displays the plot.
* **Purpose:** This plot shows how the total sales evolved over time, with each data point representing total sales on a particular date.

**🎯 Sales Trend Over Time (Monthly Aggregation)**

python

CopyEdit

df\_monthly\_sales = df.groupby(pd.Grouper(key="Order Date", freq="M"))["Sales"].sum()

plt.figure(figsize=(12, 6))

plt.plot(df\_monthly\_sales.index, df\_monthly\_sales, marker="o", linestyle="-", color="b", label="Total Sales")

plt.title("Sales Trend Over Time (Monthly Aggregation)", fontsize=14)

plt.xlabel("Date", fontsize=12)

plt.ylabel("Total Sales", fontsize=12)

plt.xticks(rotation=45)

plt.grid(True, linestyle="--", alpha=0.6)

plt.legend()

plt.show()

* **Explanation:**
  + pd.Grouper(key="Order Date", freq="M") groups the data by month, and ["Sales"].sum() calculates the total sales for each month.
  + plt.plot(df\_monthly\_sales.index, df\_monthly\_sales, marker="o", linestyle="-", color="b", label="Total Sales") plots the monthly sales data, with blue line (color="b") and circular markers (marker="o").
  + plt.title() and axis labels add necessary annotations.
  + plt.xticks(rotation=45) rotates the x-axis labels (dates) for better readability.
  + plt.grid(True, linestyle="--", alpha=0.6) adds grid lines with dashed style for better visibility.
  + plt.legend() adds a legend for the plot.
* **Purpose:** This plot illustrates the sales trend over time with monthly aggregation, helping identify trends and seasonal patterns in sales.

**🎯 Sales by Region**

python

CopyEdit

plt.figure(figsize=(8,5))

sns.barplot(x=df["Region"], y=df["Sales"], estimator=sum, palette="viridis")

plt.title("Total Sales by Region")

plt.xlabel("Region")

plt.ylabel("Total Sales")

plt.show()

* **Explanation:**
  + sns.barplot(x=df["Region"], y=df["Sales"], estimator=sum, palette="viridis") creates a bar plot that sums up total sales per region. The color palette "viridis" is used to apply a color gradient to the bars.
  + plt.title(), plt.xlabel(), and plt.ylabel() add title and axis labels.
* **Purpose:** This bar plot shows how sales are distributed across different regions, highlighting the regions with the highest and lowest sales.

**🎯 Profit Distribution (Histogram)**

python

CopyEdit

plt.figure(figsize=(8,5))

sns.histplot(df["Profit"], bins=30, kde=True, color="purple")

plt.title("Profit Distribution")

plt.xlabel("Profit")

plt.ylabel("Frequency")

plt.show()

* **Explanation:**
  + sns.histplot(df["Profit"], bins=30, kde=True, color="purple") creates a histogram of the Profit column with 30 bins. The kde=True argument adds a kernel density estimate (KDE) curve, providing a smooth estimate of the distribution.
  + plt.title(), plt.xlabel(), and plt.ylabel() add title and axis labels.
* **Purpose:** This plot shows the distribution of profit across all sales transactions. The histogram displays how frequently different profit values appear, and the KDE curve helps visualize the overall distribution pattern.

**🎯 Sales vs Profit Scatter Plot**

python

CopyEdit

plt.figure(figsize=(8,5))

sns.scatterplot(x=df["Sales"], y=df["Profit"], alpha=0.5)

plt.title("Sales vs Profit")

plt.xlabel("Sales")

plt.ylabel("Profit")

plt.show()

* **Explanation:**
  + sns.scatterplot(x=df["Sales"], y=df["Profit"], alpha=0.5) creates a scatter plot where each point represents a sale, with the Sales value on the x-axis and the Profit value on the y-axis. The alpha=0.5 argument makes the points semi-transparent, helping to visualize overlapping points.
  + plt.title(), plt.xlabel(), and plt.ylabel() add title and axis labels.
* **Purpose:** This scatter plot helps visualize the relationship between Sales and Profit, allowing us to see whether higher sales correlate with higher profits or if there are anomalies in the data.

**Summary of All Graphs:**

1. **Sales Trend Over Time:** A line plot showing total sales over time, which helps identify trends, spikes, or drops in sales.
2. **Sales Trend Over Time (Monthly Aggregation):** A line plot showing total sales aggregated by month, offering insights into monthly patterns.
3. **Sales by Region:** A bar chart comparing total sales by region, highlighting regional performance differences.
4. **Profit Distribution:** A histogram and KDE plot that shows the distribution of profits across all transactions, helping identify profit patterns.
5. **Sales vs Profit Scatter Plot:** A scatter plot that visualizes the relationship between sales and profit, helping detect any correlations or outliers.

4o mini