Elevate Labs – Day 7 Task

Technical Requirements : Get Basic Sales Summary from a Tiny SQLite Database using Python.

Tools & Libraries:

- **Python**: Core programming language used for querying and data manipulation.
- **sqlite3**: Python's built-in library to interact with SQLite databases—no extra installation required.
- **pandas**: For reading SQL outputs and handling dataframes efficiently.
- **matplotlib**: To create graphical representations like bar charts.
- **SQLite database**: File named `sales_data.db` housing your sales table.
- **Development Environment**: Jupyter Notebook or a regular `.py` script—whichever you prefer for code execution.

Step-by-Step Solution

1. Create, Populate, and Structure Your SQLite Database

Begin by setting up a SQLite database (`sales_data.db`). Inside this database, you'll need a table called `sales` with at least the following fields:

- `id`: Unique identifier (INTEGER, auto-increment)
- `product`: Name of the product (TEXT)
- `quantity`: Units sold (INTEGER)
- `price`: Sale price per unit (REAL)

Sample schema:

```
CREATE TABLE sales (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    product TEXT NOT NULL,
    quantity INTEGER NOT NULL,
    price REAL NOT NULL
);
```

Populate with dummy sales data for testing. You can use DB Browser for SQLite, SQL commands, or Python to insert records.

2. Connect Python to the SQLite Database

Establish communication with your database:

```
import sqlite3
conn = sqlite3.connect("sales_data.db")
```

This line creates a live connection object (`conn`) to your database, allowing you to execute queries and retrieve data.

3. Write and Execute SQL Queries for Sales Aggregation

Craft an SQL query to gather total quantity sold and total revenue per product:

```
query = """
SELECT
    product,
    SUM(quantity) AS total_qty,
    SUM(quantity * price) AS revenue
FROM
    sales
GROUP BY
    product
"""
```

`GROUP BY product` tells SQLite to aggregate records for each unique product, helping you view sales totals and revenue breakdowns.

4. Import Query Results into Pandas DataFrame

Load your SQL results into a pandas DataFrame for analysis and visualization:

```
import pandas as pd
df = pd.read_sql_query(query, conn)
```

This step leverages pandas' `read_sql_query()` function to directly fetch results, making subsequent data manipulations seamless.

5. Display Results and Visualize Sales Data

Print your sales summary:

```
print("Sales Summary by Product:")
print(df)
```

Generate a bar chart for revenue distribution using matplotlib:

```
import matplotlib.pyplot as plt

df.plot(kind='bar', x='product', y='revenue', legend=False)
plt.ylabel("Total Revenue")
plt.title("Product-wise Revenue Summary")
plt.tight_layout()
plt.savefig("sales_chart.png")  # Optional: save as PNG
plt.show()
```

Make sure you close your database connection at the end:

```
conn.close()
```

Interview Preparation: Core Technical Concepts

- **Connecting Python and SQLite:** Used `sqlite3.connect` for efficient database connectivity.
- **SQL Query Design:** Leveraged aggregation functions (`SUM`) and grouping (`GROUP BY`) to extract meaningful business insights—specifically sales totals and revenue per product.
- **Revenue Calculation Method:** Calculated total revenue for each product by summing `quantity * price` across all transactions.
- **Result Visualization:** Utilized `matplotlib` for a crisp, readable bar chart, complementing tabular output for stakeholder-friendly reporting.
- **Pandas Role:** Accelerates data wrangling, making the transformation from SQL outputs to analyzable formats quick and error-free.

- **Advantages of SQL in Python:** Seamlessly blends powerful querying with automated analysis and visualization, essential for modern data workflows.
- **SQL Query in DB Browser:** Absolutely—such aggregation queries run natively in DB Browser for SQLite's interactive interface.

Submission Checklist

- Organize code, database file, screenshots, and a straightforward README.md in a dedicated GitHub repository.
- When complete, submit your repo link via the provided submission portal before the daily deadline.

This approach demonstrates foundational techniques crucial for a data analyst: database operations, SQL proficiency, Python data processing, and basic visualization.