**Project: Sales Analyzer**

**Repository:** [**Datafactz Project**](https://github.com/ravish0409/datafactz-project)

**Overview**

The **Sales Data Dashboard** is an interactive application that visualizes sales data using Python's Tkinter library for the graphical user interface (GUI) and Matplotlib for generating visual charts. It allows users to explore various aspects of the sales data through charts, such as revenue by region, profit by country, sales by item type, and revenue over time. This project uses a CSV file (5000 Sales Records.csv) as the data source and processes it with Pandas.

The dashboard includes multiple types of visualizations and sorting mechanisms for easy data interpretation. Users can select different views and sort data in ascending or descending order.

**Project Dependencies**

To run this project, ensure that the following Python libraries are installed:

* tkinter: for creating the GUI
* pandas: for data manipulation and analysis
* matplotlib: for generating charts and plots
* matplotlib.backends.backend\_tkagg: for embedding Matplotlib figures in the Tkinter window

**Project Structure**

1. **Dashboard Class**: The core class responsible for the creation of the GUI, loading the data, and displaying various visualizations.

**Attributes**:

* root: Main window for the GUI.
* data: The sales data loaded from a CSV file.
* sort\_var: Controls the sorting order (ascending, descending, or none).
* active\_button: Tracks the currently active button to display the relevant chart.
* region\_var: Stores the selected region for region-based filtering.

1. **Methods**:
   * **\_\_init\_\_()**: Initializes the GUI, loads the data, and sets up the layout.
   * **load\_data()**: Reads and preprocesses the sales data from the CSV file. Handles missing or non-numeric values and converts date columns.
   * **create\_header()**: Creates the header section of the dashboard.
   * **create\_sidebar()**: Sets up the sidebar with buttons to select different visualizations and sorting options.
   * **create\_main\_content()**: Configures the main content area where charts and summary reports are displayed.
   * **update\_chart()**: Refreshes the chart and displays the updated summary based on user interaction.
   * **add\_BM()**: Utility function to format large numbers as millions (M)
   * **annotate\_bars()**: Adds labels to bars in bar charts for clarity.
   * **apply\_sorting()**: Sorts data based on user selection.
   * **sort\_with\_col()**: Sorts DataFrame columns based on the sorting option selected.
   * **highlight\_active\_button()**: Highlights the button for the currently selected chart.
   * **create\_table\_str()**: Formats data into a table-like string format for display in the summary.
   * **create\_df\_str()**: Creates a formatted string for displaying DataFrame summaries.

**Data Preprocessing**

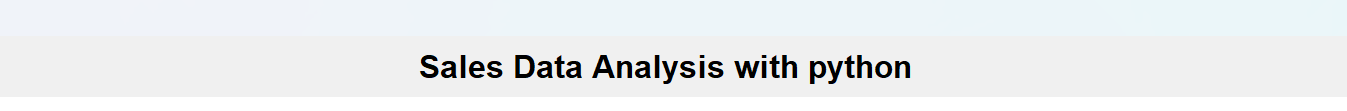
The data is loaded using the pandas library through the load\_data() method. It performs the following steps:

1. **Loading the CSV**: Attempts to load 5000 Sales Records.csv.
   * Handles errors, including FileNotFoundError and ValueError.
2. **Data Cleaning**:
   * Removes missing values and duplicate rows.
   * Converts date columns (Order Date and Ship Date) to datetime format.
   * Converts numerical columns (e.g., Units Sold, Total Revenue, etc.) to the correct data types.
   * Strips extra whitespace from string columns.

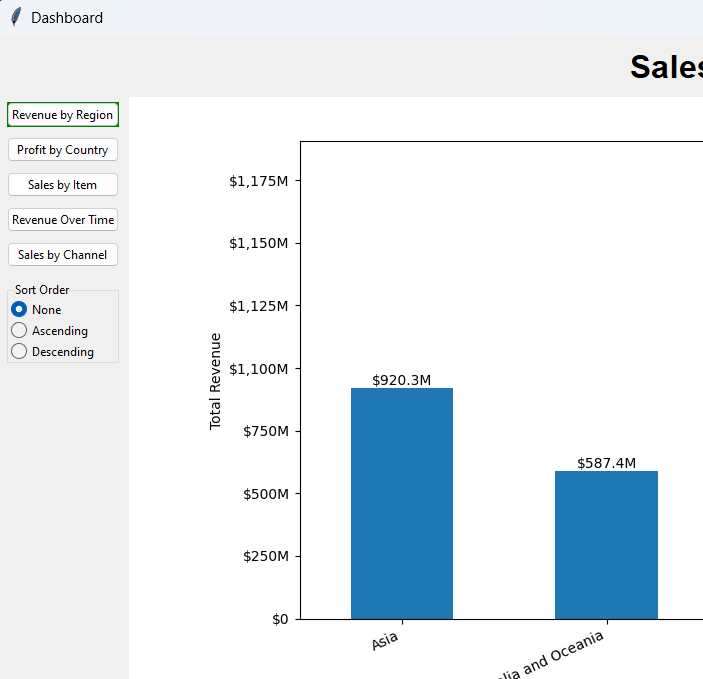
**Dashboard Layout**

The dashboard is divided into three main sections:

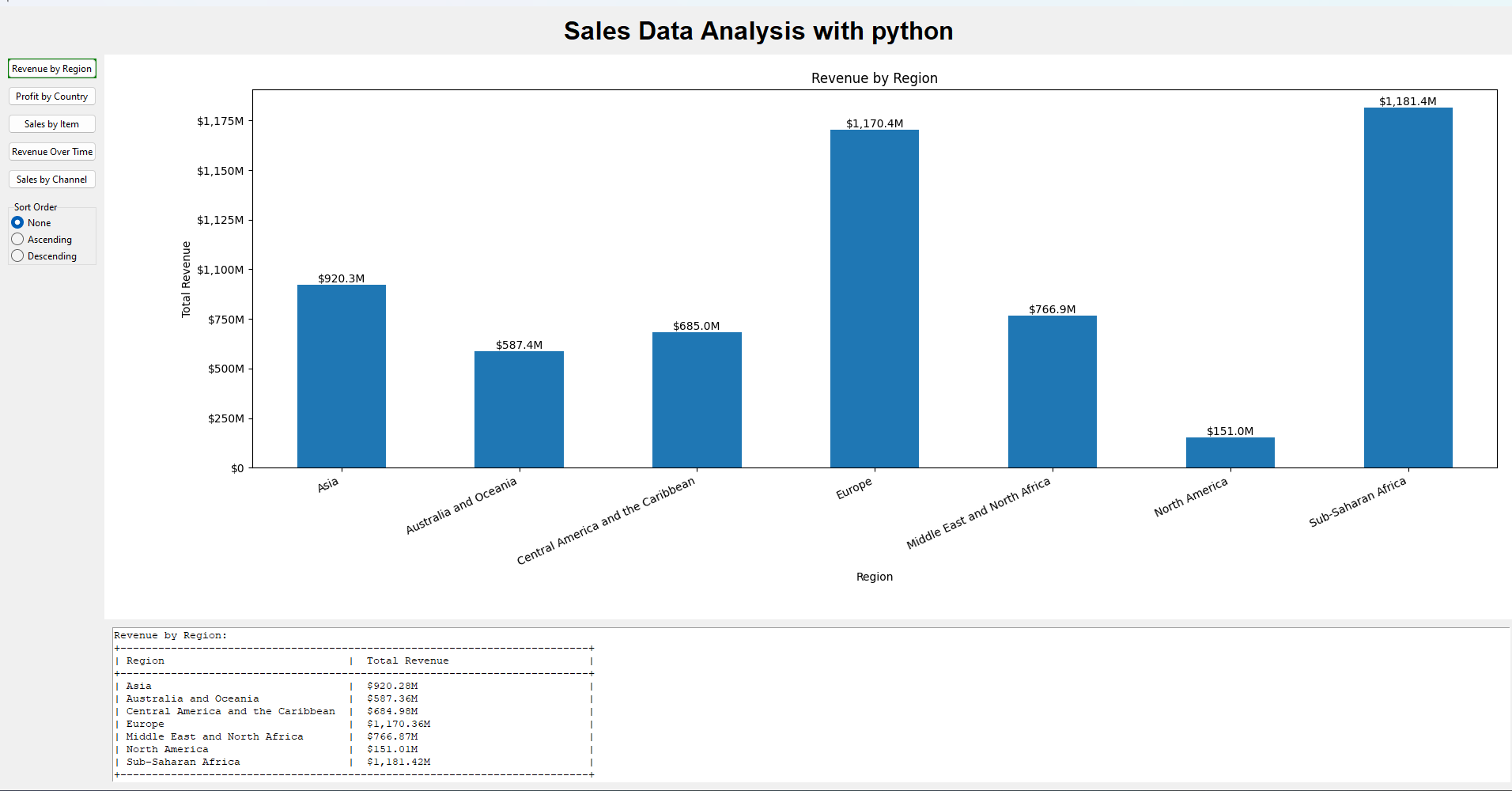
1. **Header**: Displays the title "Sales Data Analysis with Python".



1. **Sidebar**: Contains buttons to display different charts and sorting options.
   * Buttons include:
     + Revenue by Region
     + Profit by Country
     + Sales by Item
     + Revenue Over Time
     + Sales by Channel
   * Sorting Options: None, Ascending, Descending

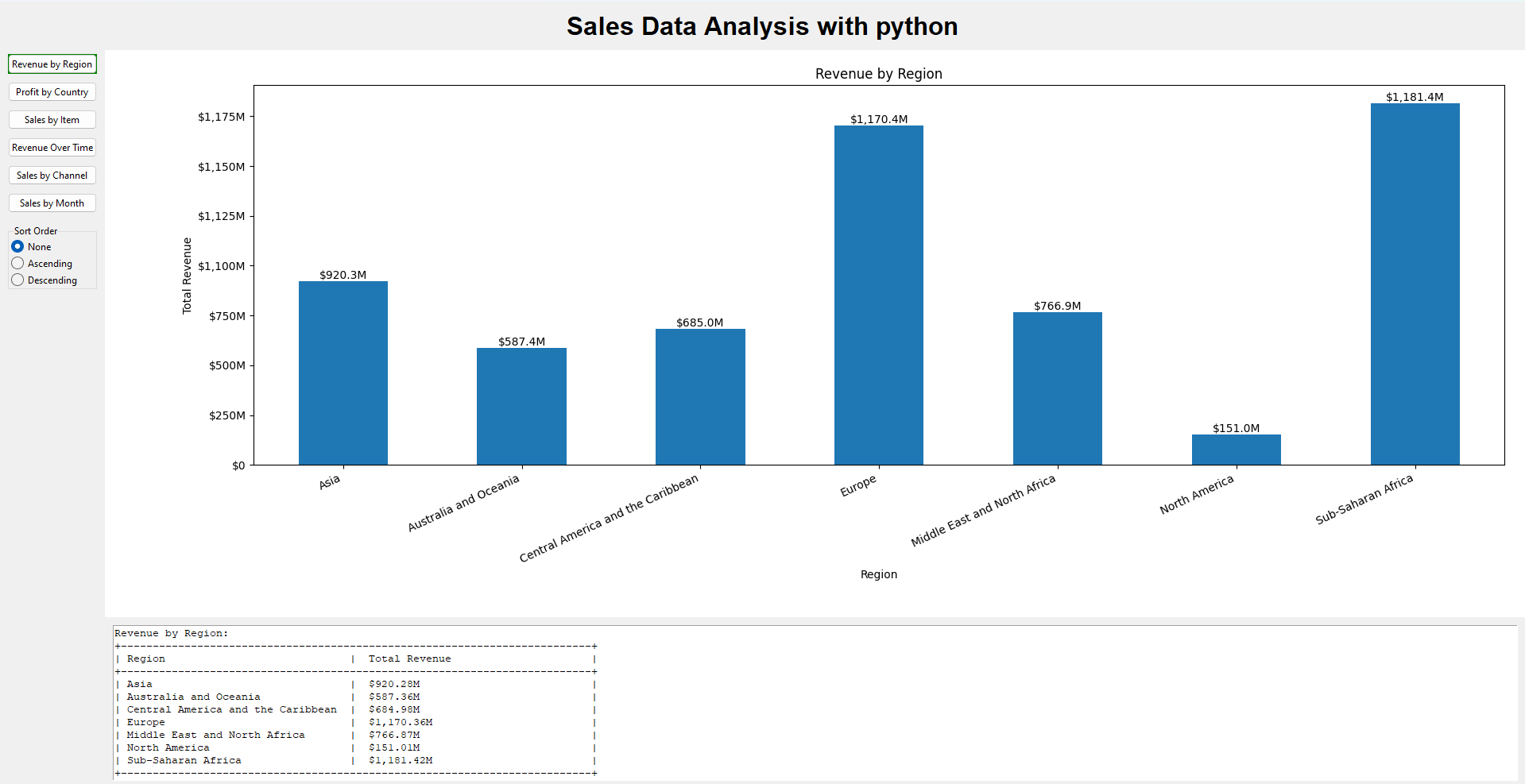


1. **Main Content Area**: Displays the chart and summary report based on the selected button.

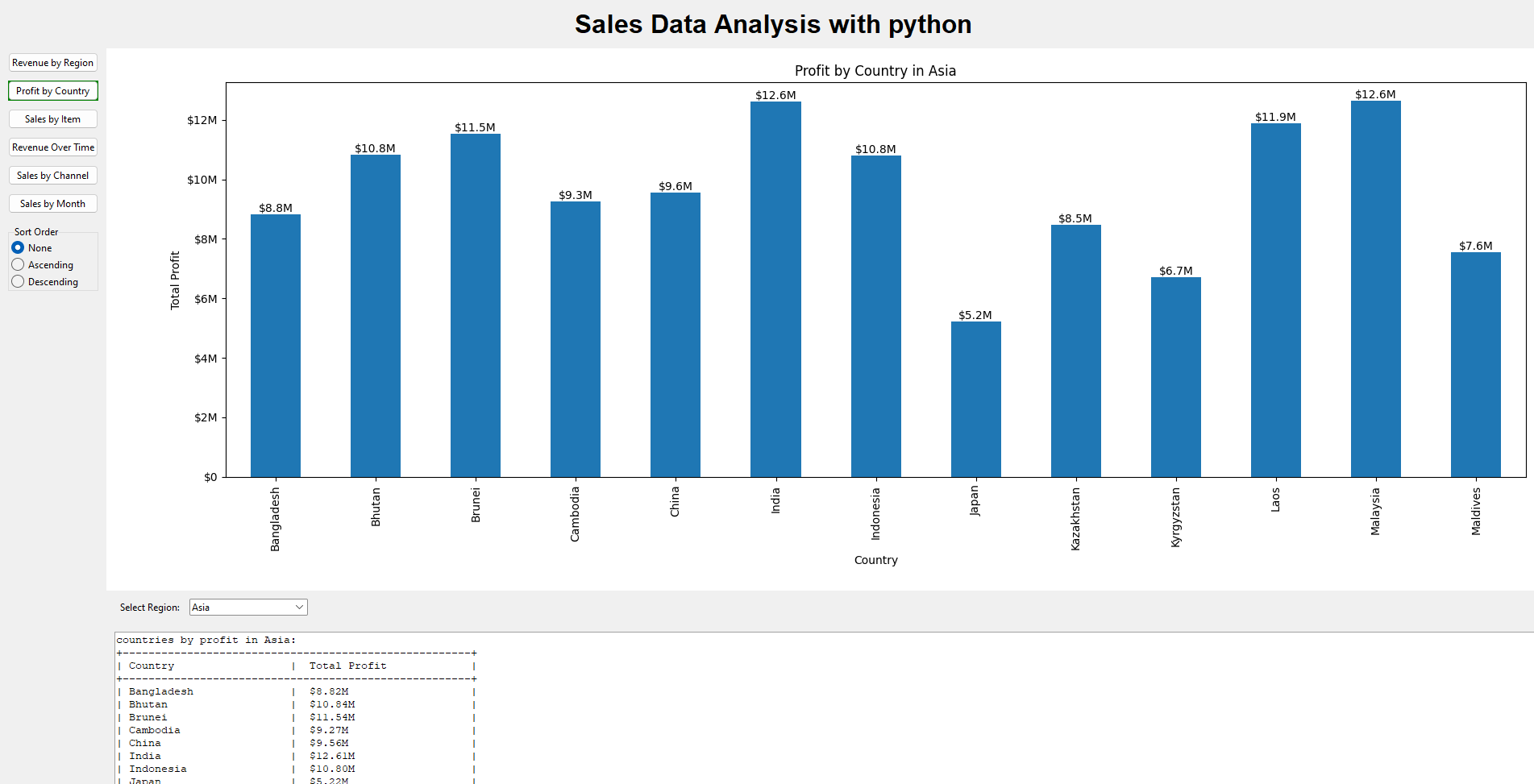


**Functionality and Visualization**

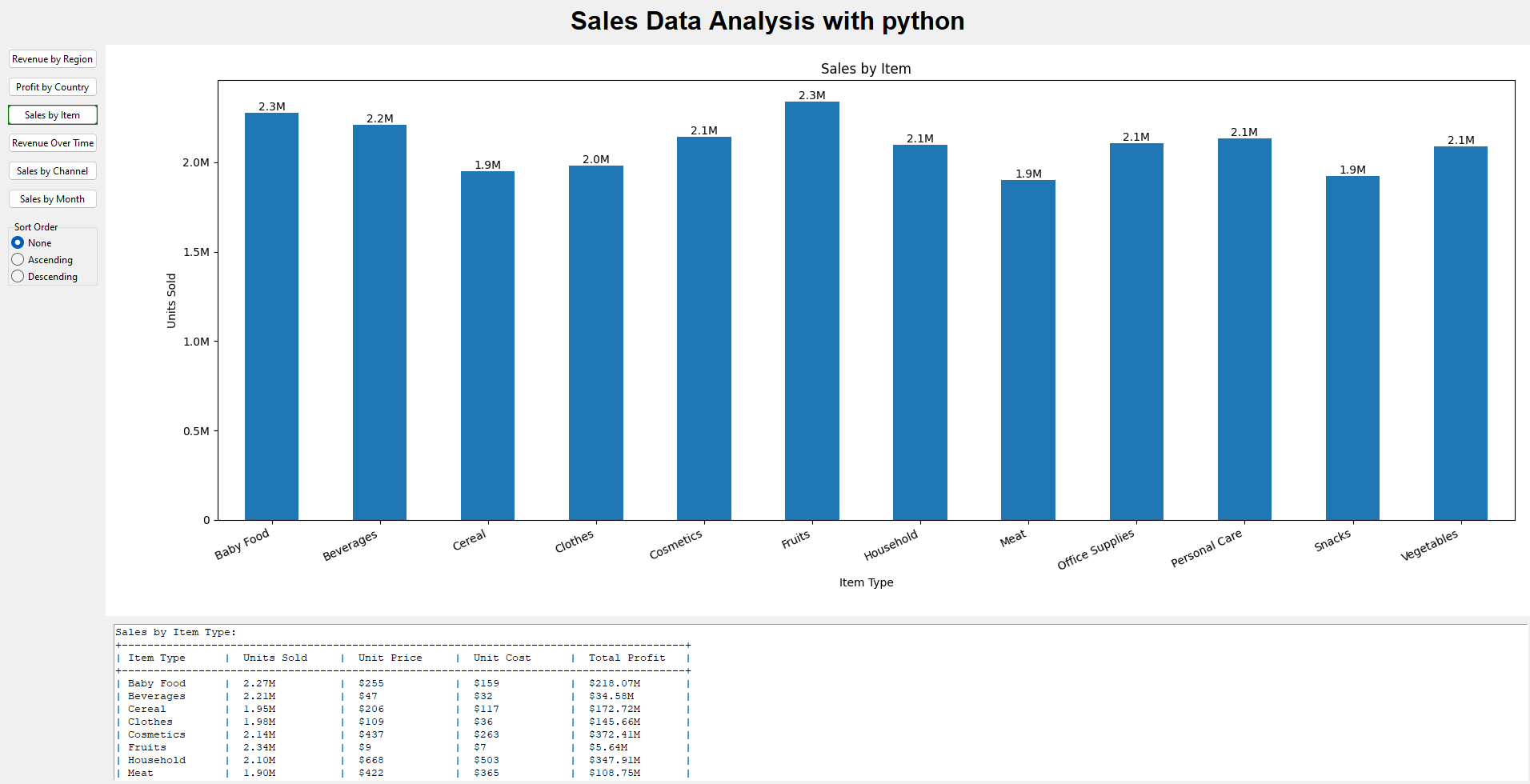
1. **Revenue by Region** (show\_revenue\_by\_region()):
   * Displays a bar chart showing total revenue for each region.
   * Includes sorting (ascending/descending) and automatic annotation for better readability.
   * The summary report is a textual representation of total revenue by region, formatted with millions (M).



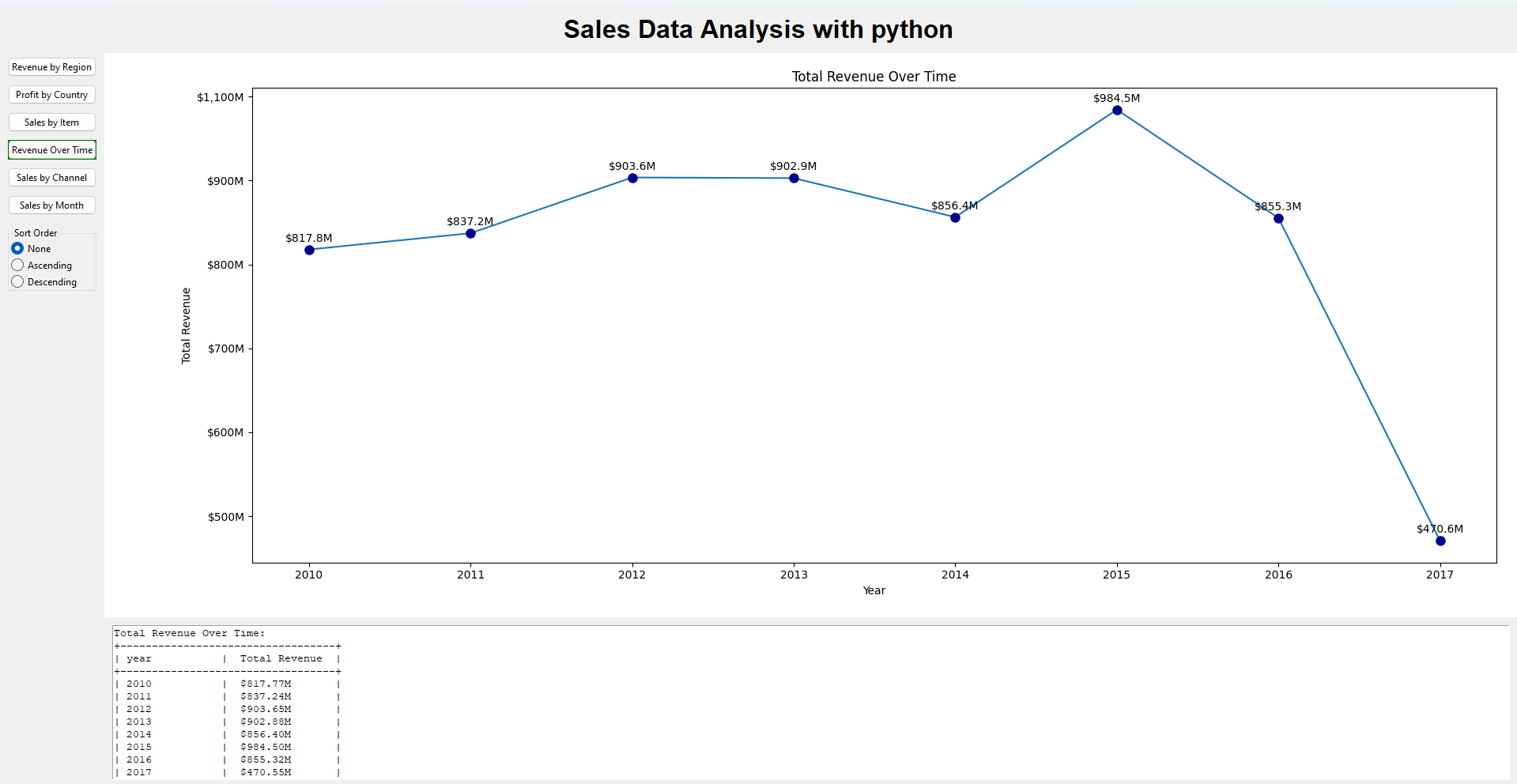
1. **Profit by Country** (show\_profit\_by\_country()):
   * Displays a bar chart of total profit by country within a selected region.
   * Users can select a region from a dropdown menu, and the chart updates accordingly.
   * The summary report shows the profit figures in the selected region.



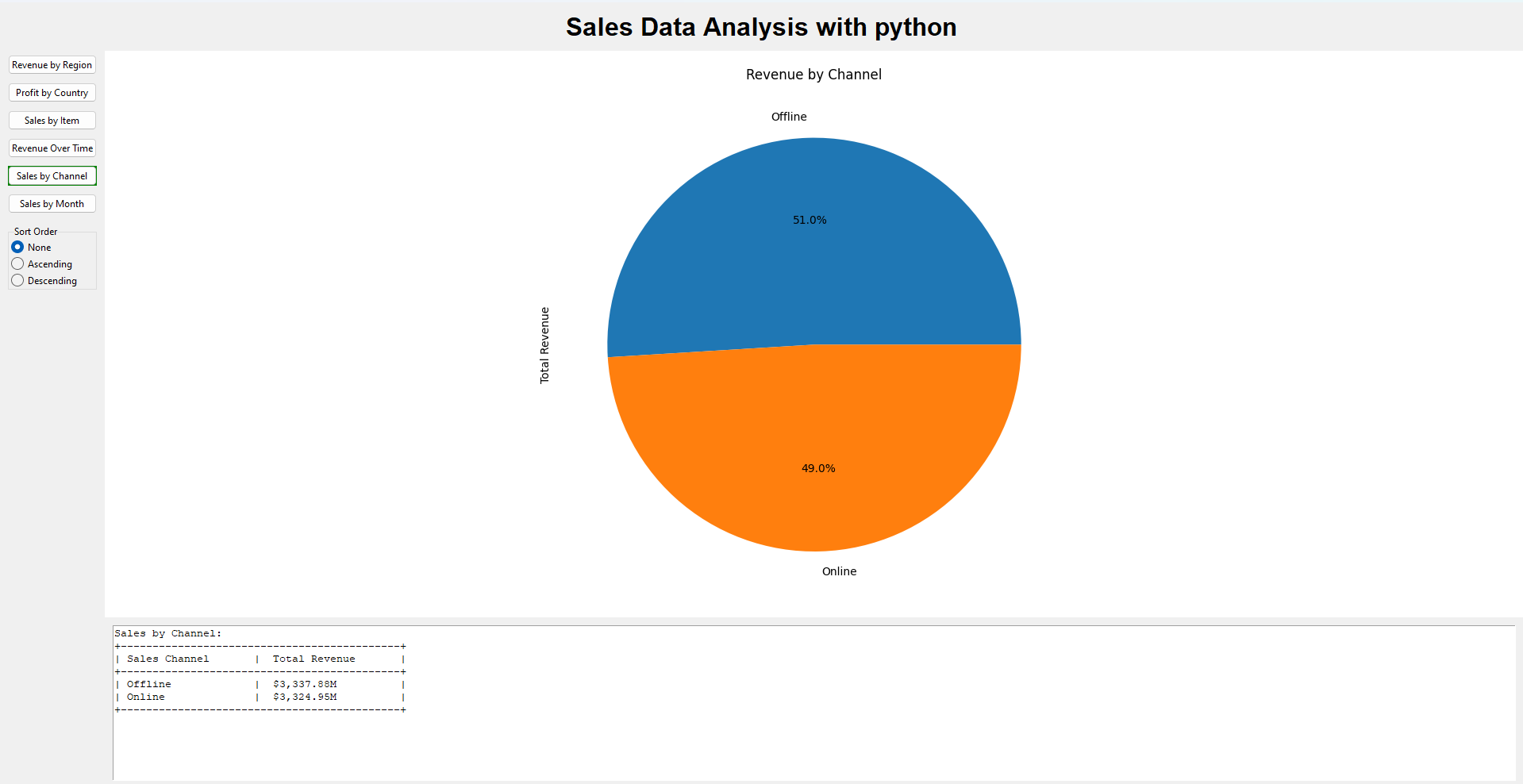
1. **Sales by Item** (show\_sales\_by\_item()):
   * Displays a bar chart of units sold by item type.
   * Includes additional metrics in the summary: average unit price, average unit cost, and total profit for each item type.



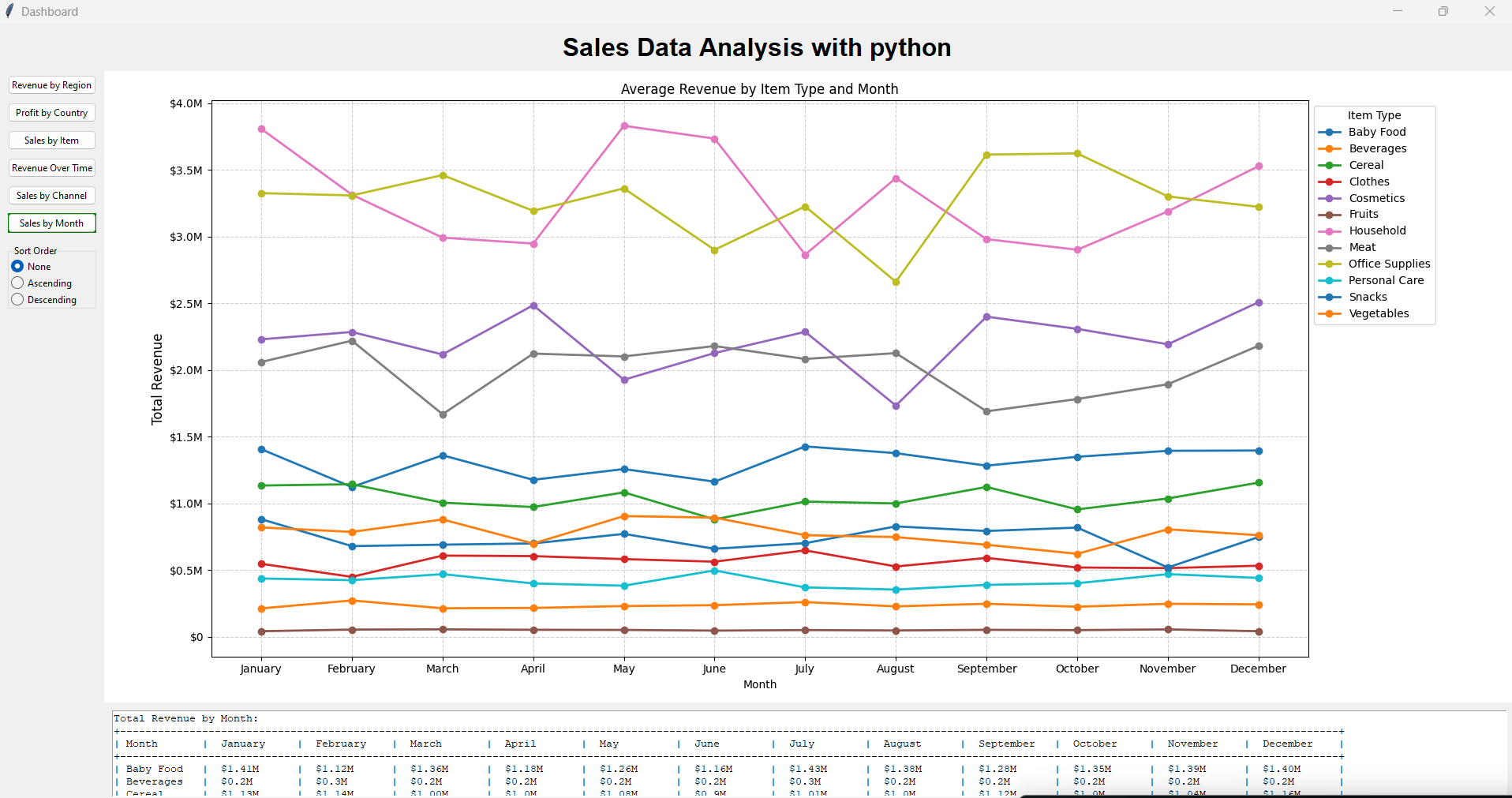
1. **Revenue Over Time** (show\_sales\_over\_time()):
   * Displays a line chart of total revenue over time, aggregated by year.
   * The chart includes annotations for each year’s revenue.
   * The summary report shows revenue by year, with sorting options applied if selected.



1. **Sales by Channel** (show\_sales\_by\_channel()):
   * Displays a pie chart showing the percentage of total revenue from different sales channels (e.g., Online and Offline).
   * The summary report presents revenue figures for each channel



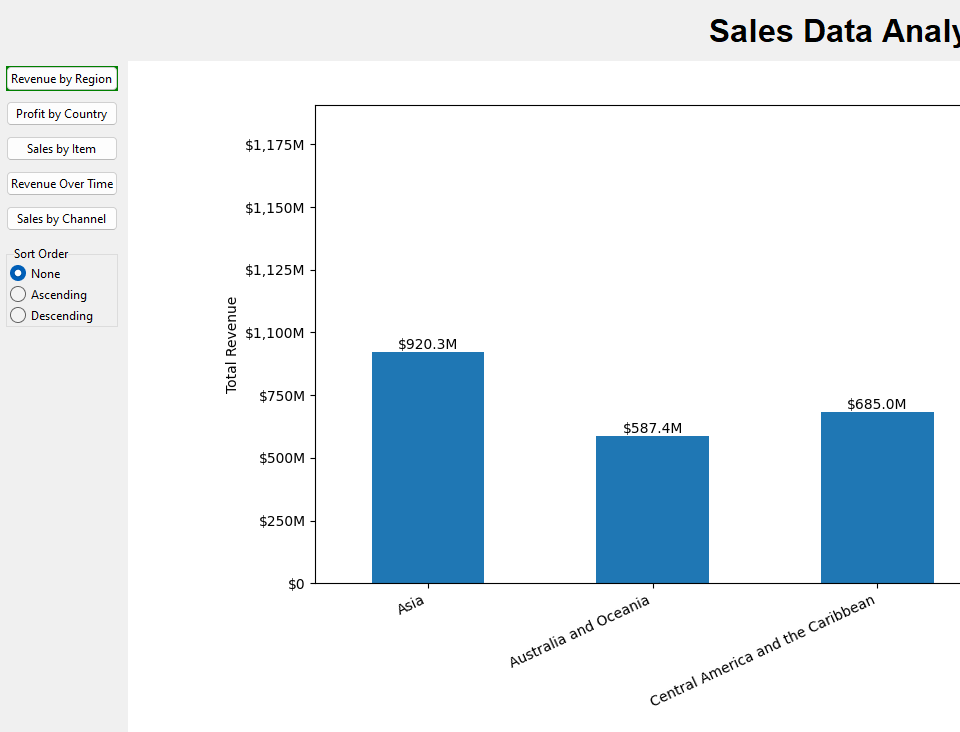
1. **Sales by month** (show\_sales\_by\_month()):
   * This will display the line chart, showing each item type’s sales vary from month to month.
   * allowing users to observe **seasonal trends** in sales for different item types.
   * A summary table is generated, showing the average revenue values for each item type across the months. The values are formatted for easy reading.



**Sorting Mechanism**

The sorting feature allows users to view data in ascending, descending, or no particular order. This functionality is controlled by the apply\_sorting() and sort\_with\_col() methods. Depending on the sorting selection, the data is reordered before visualization and shown in the chart accordingly.

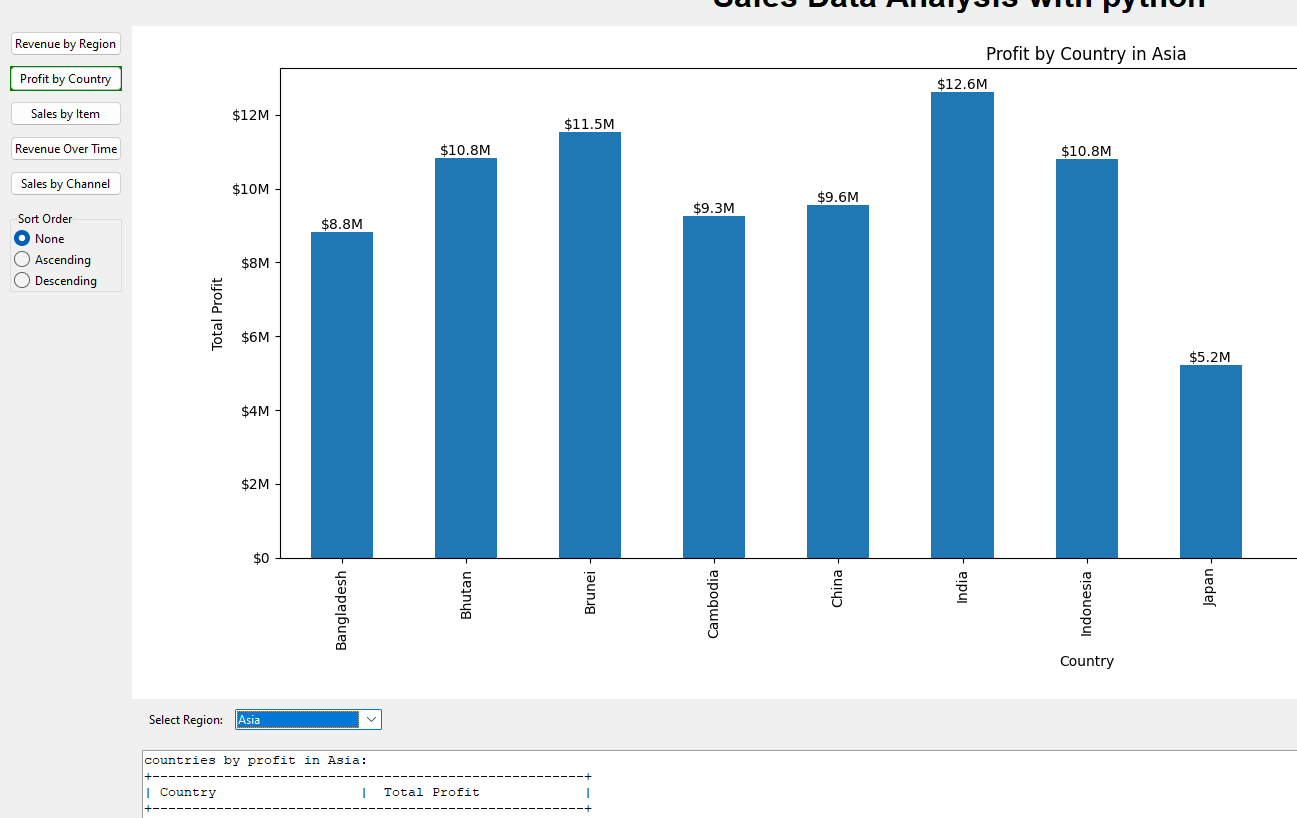
* **None**: No sorting applied (default order).
* **Ascending**: Data is sorted in increasing order.
* **Descending**: Data is sorted in decreasing order.



**Chart Customization**

Each chart is dynamically generated using Matplotlib, and the following features are included for better visualization:

* **Bar Chart Annotation**: Bars are annotated with values (e.g., millions) for quick reference.
* **Y-Axis Formatting**: Y-axis values are formatted in an easily readable format, such as millions (M). using mtick.FuncFormatter().
* **Interactivity**: Dropdowns (e.g., region selection) dynamically update the charts based on the user's selection.



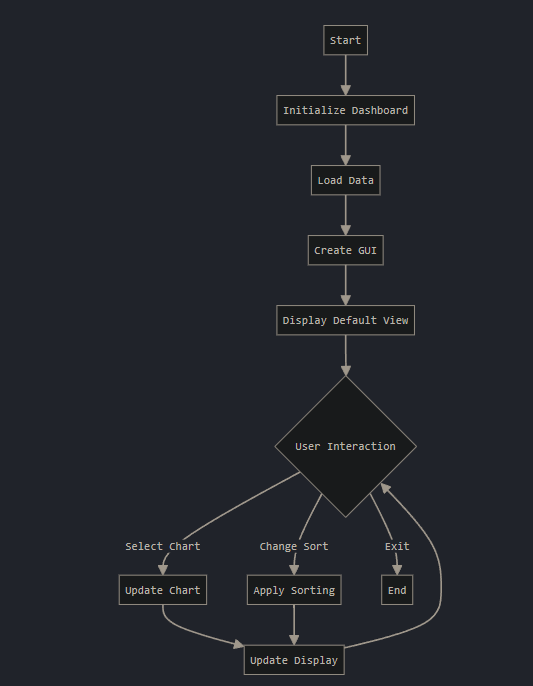
**Bar Chart Annotation**

**Y-Axis Formatting**

**Error Handling**

The application includes basic error handling in the load\_data() method. If the CSV file is not found or contains invalid data, appropriate error messages are displayed. For example:

* **FileNotFoundError**: If the file 5000 Sales Records.csv is not found, an error message is printed.
* **ValueError**: If there are any issues with data types (e.g., non-numeric values in numeric columns), they are handled gracefully.

****

**Execution**

**1. Clone the Repository:**

First, clone the repository from GitHub to your local machine:

Copy and paste this in cmd:

|  |  |
| --- | --- |
| $  $ | git clone https://github.com/ravish0409/datafactz-project.git  cd datafactz-project |

**2. Set Up a Virtual Environment (Optional but Recommended):**

Setting up a virtual environment helps manage dependencies and avoid conflicts with other projects.

|  |  |
| --- | --- |
| $  $ | python -m venv venv  venv\Scripts\activate |

**3. Install Dependencies:**

Once the environment is active, install the required dependencies listed in the requirements.txt file:

|  |  |
| --- | --- |
| $ | pip install -r requirements.txt |

**4. Run the Project:**

After installing the dependencies, you can launch the dashboard by running the sales\_analyze.py script:

|  |  |
| --- | --- |
| $ | python sales\_analyze.py |

**Testing**

Testing ensures that everything works as expected.

Manual Testing:

* UI Testing: Check that all parts of the interface work properly.

Steps:

* + Run the sales\_analyze.py script.
  + Make sure the window opens correctly.
  + Test loading data and using the sorting feature.
  + Check that graphs update correctly when data is changed.
  + Ensure the app doesn't crash during use.
* Data Verification: Compare the loaded data with the source to make sure it is accurate.

**. Future Enhancements**

* Additional Filtering: Add more ways to filter the data, like by product type or date.
* Exporting Reports: Allow users to save the graphs and insights as PDF or Excel files.
* Improving the performance for larger datasets by implementing lazy loading or data chunking

**Project Timeline**

This project was developed over several days, with key milestones documented in the commit history on GitHub. Below is a summary of the significant updates:

* **August 26, 2024**: Initial dashboard setup.
* **August 27, 2024:** created functions and filtering methods for visualization
* **August 28, 2024**: Added summary report and clarified graph visuals.
* **August 29, 2024**: Finalize the project name and integrated data cleaning scripts.
* **August 30, 2024**: Added radio buttons for improved user interaction.
* **August 31, 2024**: Structured the summary report and added a selector field.
* **September 2, 2024**: Updated README and project requirements for documentation.

**Challenges Faced**

1. **Data Cleaning**:
   * **Inconsistent Data**: Handling missing values and duplicates in the sales data was a challenge. The load\_data() function had to ensure that only clean and valid data was used for analysis.
   * **Converting Data Types**: Certain columns, such as dates and numeric values, required careful conversion to ensure proper formatting and usability for calculations and plotting.
2. **Handling Large Numbers**:
   * Formatting large financial values (millions or billions) on the charts and summary tables in a clear, concise way required the creation of custom utility functions (add\_BM()) to display numbers in a readable format (e.g., 1M for million, 1B for billion).
3. **UI Layout and Chart Display**:
   * **Chart Overlap**: Ensuring that the charts fit properly within the Tkinter window without overlap or being cut off required the use of proper layout management (pack() and grid() methods) and adjusting chart sizes dynamically.
   * **Embedding Matplotlib**: Integrating Matplotlib charts into the Tkinter application using FigureCanvasTkAgg posed some challenges related to the smooth updating and rendering of charts within the GUI.
4. **Sorting and Interactivity**:
   * Ensuring that sorting options (ascending/descending) were correctly applied to different visualizations required careful management of the sort\_var and dynamically applying sorting logic without breaking the visualization flow.
   * **Updating Views**: Handling user interaction smoothly, such as selecting different charts or regions, required the development of robust methods like update\_current\_view() and update\_chart() to refresh the display without crashing or lagging.
5. **Month-Based Data Grouping**:
   * **Correct Month Order**: When grouping the data by month, ensuring that months appear in chronological order (January to December) rather than alphabetically was a challenge. The solution involved using pd.Categorical to maintain the correct order of months.
   * **Complexity of Multi-level Grouping**: Grouping the data by both Item Type and Month for the **Sales by Month** feature required careful use of pandas' groupby() and unstack() methods to ensure the data was structured correctly for plotting.

**Conclusion**

* The dashboard provides an easy way to explore sales data through **interactive charts**.
* Key metrics like **revenue, profit**, and **sales trends** are visualized clearly.
* Sorting features help users to analyze the data in different orders.
* The tool is **user-friendly** and allows for quick **data analysis** without complex commands.