Case Study: Car Sales Data Analysis

Problem Statement

Given a global car-sales dataset in Excel/CSV, complete the following tasks:

1. Count the number of cars sold by **Audi** in **China** in 2025.
2. Compute the total revenue generated by **BMW** in 2025.
3. Analyze the distribution of BMW revenue across **European countries**, sorted from highest to lowest.

Functional Requirements

- Import large CSV/Excel datasets (MB/GB scale).
- Filter rows based on exact or partial/wildcard column values.
- Compute counts for specific criteria.
- Sum numeric columns grouped by a chosen column.
- Sort aggregated results by numeric values.
- Display results in a readable tabular format.
- Handle missing or invalid data gracefully.

Non-Functional Requirements

- Efficient and scalable processing for large datasets.
- Multi-threaded execution for high performance.
- Maintainable, modular, and extensible code.
- Reliable logging and error reporting.

Implementation Features

CarSalesProcessor Class

Purpose: Flexible, multi-threaded data processing with filtering, aggregation, and sorting.

1. filterRowsByColumnValues(columns, rows_to_store, threads)

- Filters dataset rows by column values.
- Supports exact matches and wildcard (*) prefix/suffix.
- Multi-threaded execution; number of threads configurable.
- Select specific columns to include in output.
- Flexible: multiple columns and multiple criteria per column.
- **Example:** Filter cars sold by "Audi" in "China" or all manufacturers starting with "Au*" or ending with “*enz”.

2. groupAndSumColumns(rows, group_by_column, columns_to_sum, threads, sorted, sort_by_col)

- Aggregates numeric columns grouped by a specified column. (eg – gives the sum for each country in the same column – grouping feature)
- Sums multiple numeric columns per group. (We can find the sum of multiple columns in 1 go)
- Multi-threaded for high performance.
- Optional sorting by any summed column.
- Robust error handling for missing columns or invalid numeric data.
- Flexible: dynamic selection of grouping and summation columns.
- **Example:** Compute total revenue per country, optionally sorted by revenue.

3. readCsv(csv_path)

- Loads CSV/Excel data efficiently.
- Reads headers and all rows for further processing.

How the Problems Were Solved

Q1 – Count Cars Sold by Audi in China in 2025

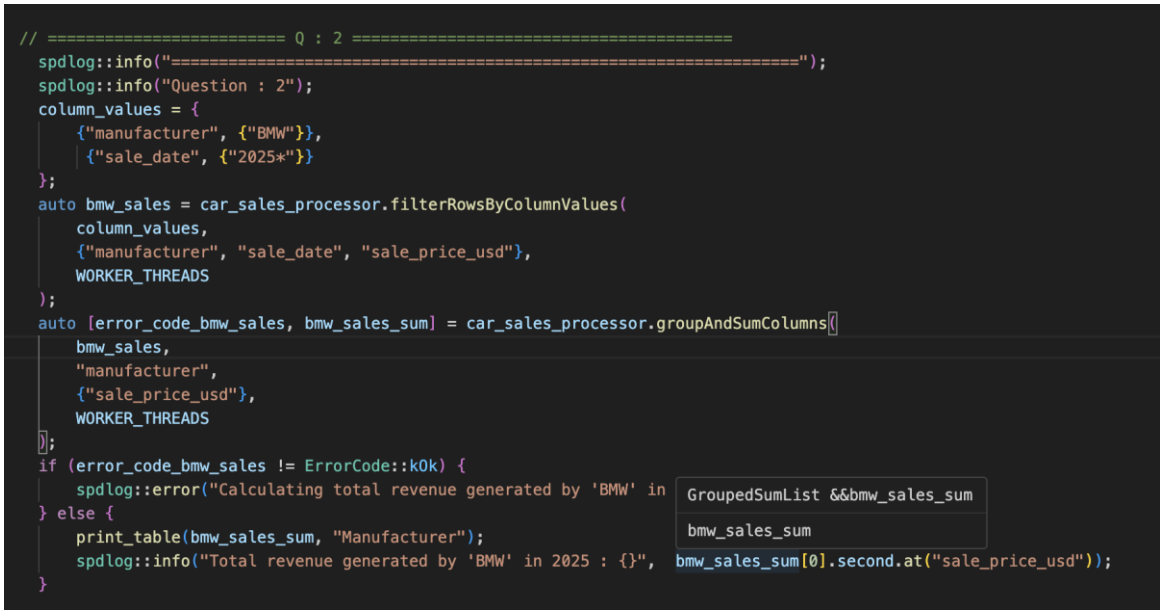
- **Criteria:**
  - Country = China
  - Manufacturer = Audi
  - Sale_date = 2025* (wildcard supports "starts with")
- Used **filterRowsByColumnValues** to retrieve matching rows. Only the selected columns {country, manufacturer, sale_date} were returned.
- **Features leveraged:**
  - Multi-threading for parallel filtering.

    - Flexible to allow multiple manufacturers or countries.
- **Result:** Number of cars sold = total filtered rows.

```
// ======================== Q : 1 =====================================
spdlog::info("================================================================");
spdlog::info("Question : 1");
column_values = {
    {"country", {"China"}},
    {"manufacturer", {"Audi"}},
    {"sale_date", {"2025*"}}
};
auto china_audi_2025_sales = car_sales_processor.filterRowsByColumnValues(
    column_values,
    {"country", "manufacturer", "sale_date"},
    WORKER_THREADS
);
spdlog::info("Number of cars sold by 'Audi' in China in 2025 : {}",  china_audi_2025_sales.size());
```

Q2 – Total Revenue Generated by BMW in 2025

- **Criteria:**
    - Manufacturer = BMW
    - Sale_date = 2025* (wildcard used)
- Step 1: Use **filterRowsByColumnValues** to get filtered rows.
- Step 2: Use **groupAndSumColumns** to:
    - Group by manufacturer
    - Sum the sale_price_usd column
- **Features leveraged:**
    - Multi-threaded aggregation.
    - Dynamic grouping and summation columns.
    - Can sum multiple columns in 1 go. Not just sales, but any multiple columns can be summed together in 1 pass – at the same time grouped. Each column will be grouped.
    - Returns sums grouped by column values.
- **Result:** Total revenue = sum of sale_price_usd for BMW in 2025 (e.g., "BMW": {"sale_price_usd": 98688244}).



Q3 – Distribution of total revenue in European countries (sorted)

Criteria:

- Region = Europe

Solution Steps:

1. **Filter Rows:**
    a. Used filterRowsByColumnValues with {"region", {"Europe"}}.
    b. Selected columns returned: {country, region, manufacturer, sale_price_usd}.
    c. Multi-threaded filtering ensures fast processing for large datasets.
2. **Aggregate Revenue:**
    a. Applied groupAndSumColumns on filtered rows.
    b. Grouped by "country" and summed "sale_price_usd".
    c. Enabled sorting by "sale_price_usd" to get countries with highest revenue first.
3. **Output:**
    a. Results displayed in a tabular format using print_table.
    b. Shows total revenue per European country in descending order.

Notes:

- Multi-threaded aggregation improves performance.

- Flexible: can easily group by other columns or sum multiple numeric columns.
- Robust: handles missing or invalid data gracefully.

```cpp
// ======================== Q : 3 ======================================
spdlog::info("================================================================");
spdlog::info("Question : 3");
spdlog::info("Distribution of total revenue in European countries sorted from highest to lowest : ");
column_values = {
    {"region", {"Europe"}}
};
auto europe_sales = car_sales_processor.filterRowsByColumnValues(
    column_values,
    {"country", "region", "manufacturer", "sale_price_usd"},
    WORKER_THREADS
);
auto [error_code_europe, revenue_by_country] =
car_sales_processor.groupAndSumColumns(
    europe_sales,
    "country",
    {"sale_price_usd"},
    WORKER_THREADS,
    true,
    "sale_price_usd"
);
if (error_code_europe != ErrorCode::kOk) {
    spdlog::error("Calculating total revenue generated by 'BMW' in 2025 failed!");
} else {
    print_table(revenue_by_country, "Country");
}
```

Console output

```
nothing added to commit but untracked files present (use "git add" to track)
diljithkd@Diljiths-MacBook-Pro EigenRiskCarSales % mkdir build
diljithkd@Diljiths-MacBook-Pro EigenRiskCarSales % cd build
diljithkd@Diljiths-MacBook-Pro build % cmake ..
-- The CXX compiler identification is AppleClang 16.0.0.16000026
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done (1.2s)
-- Generating done (0.0s)
-- Build files have been written to: /Users/diljithkd/Desktop/Diljith_KD_EigenRisk_CarSales/EigenRiskCarSales/build
diljithkd@Diljiths-MacBook-Pro build % make -j8
[ 66%] Building CXX object CMakeFiles/car_sales_app.dir/src/main.cpp.o
[ 66%] Building CXX object CMakeFiles/car_sales_app.dir/src/car_sales_processor.cpp.o
[100%] Linking CXX executable car_sales_app
[100%] Built target car_sales_app
diljithkd@Diljiths-MacBook-Pro build % ./car_sales_app
[2025-11-22 23:26:54.115] [info] Staring Car Sales Processor
[2025-11-22 23:26:54.116] [info] Reading CSV File : ../data/world_car_sales_1m.csv
[2025-11-22 23:26:55.384] [info] CSV has 1000001 rows
[2025-11-22 23:26:55.384] [info] CSV has 43 columns
[2025-11-22 23:26:55.384] [info] CSV File Read Complete : ../data/world_car_sales_1m.csv
[2025-11-22 23:26:55.384] [info] ================================================================
[2025-11-22 23:26:55.384] [info] Question : 1
[2025-11-22 23:27:00.162] [info] Number of cars sold by 'Audi' in China in 2025 : 373
[2025-11-22 23:27:00.162] [info] ================================================================
[2025-11-22 23:27:00.162] [info] Question : 2
[2025-11-22 23:27:05.345] [info] +-----------------------------------------+
[2025-11-22 23:27:05.345] [info] |    Manufacturer    |   sale_price_usd   |
[2025-11-22 23:27:05.345] [info] +-----------------------------------------+
[2025-11-22 23:27:05.345] [info] |        BMW         |      98688244      |
[2025-11-22 23:27:05.345] [info] +-----------------------------------------+
[2025-11-22 23:27:05.345] [info] Total revenue generated by 'BMW' in 2025 : 98688244
[2025-11-22 23:27:05.345] [info] ================================================================
[2025-11-22 23:27:05.345] [info] Question : 3
[2025-11-22 23:27:05.345] [info] Distribution of total revenue in European countries sorted from highest to lowest :
[2025-11-22 23:27:10.594] [info] +-----------------------------------------+
[2025-11-22 23:27:10.594] [info] |      Country       |   sale_price_usd   |
[2025-11-22 23:27:10.594] [info] +-----------------------------------------+
[2025-11-22 23:27:10.594] [info] |       Italy        |     620416419      |
[2025-11-22 23:27:10.594] [info] |      Germany       |     618549868      |
[2025-11-22 23:27:10.594] [info] |       Sweden       |     615825350      |
[2025-11-22 23:27:10.594] [info] |    Netherlands     |     614994351      |
[2025-11-22 23:27:10.594] [info] |       France       |     614030545      |
[2025-11-22 23:27:10.594] [info] |   United Kingdom   |     611249629      |
[2025-11-22 23:27:10.594] [info] |       Spain        |     611169909      |
[2025-11-22 23:27:10.594] [info] +-----------------------------------------+
diljithkd@Diljiths-MacBook-Pro build %
```

Multithreading Analysis

Table 1: filterRowsByColumnValues – Execution Time vs Threads

Question / Task	No of Threads	Execution Time (ms)	Observation
Q1 – Cars sold by Audi in China 2025	1	30129	Baseline (single-thread)
	4	7503	~4x faster with 4 threads
	8	5671	~5x faster with 8 threads

Q2 – Filter rows for BMW 2025 revenue	1	29007	Baseline
	4	7402	~4x faster
	8	5724	~5x faster
Q3 – European revenue distribution	1	27957	Baseline
	4	5724	~5x faster
	8	5509	~5x faster; marginal gain over 4 threads

Table 2: groupAndSumColumns – Execution Time vs Threads

Question / Task	No of Threads	Execution Time (ms)	Observation
Q2 – Total revenue for BMW 2025	1	9	Baseline
	4	2	~4x faster
	8	2	No significant gain beyond 4 threads
Q3 – European revenue distribution	1	395	Baseline
	4	109	~3.5x faster
	8	79	~5x faster; best performance