Distributed Computing

Final Assignment

Bahaar Khalilian and Hesam Mohebi

1. Problem Statement

For this project, we have chosen an E-commerce dataset. With this dataset in mind, we wish to: “Design a comprehensive data processing and analytics solution to understand customer behavior, analyze sales trends, and predict future revenue for a retail business.”

2. Dataset Description

The dataset used in this study is taken from a UK-based online retailer that specializes in selling unique all-occasion gifts. Two years ago, it transitioned from catalog-based sales to a web-based business model and markets its products through its website and Amazon.co.uk. The dataset contains all transactions from January 1, 2011, to December 31, 2011, focusing more on customers from the United Kingdom. Key details of the dataset:

- **Total Transactions:** 22,190 valid transactions.
- **Distinct Customers:** 4,381 unique postcodes.
- **Total Records:** 406,830 rows, with each row representing a specific item in a transaction.
- **Average Transactions per Customer:** Approximately 5, indicating one purchase every two months.
- **Average Items per Transaction:** 18.3, suggesting the customer base includes a significant proportion of organizational buyers.

3. Defined Tasks/Queries

Part One: Customer Insights

Query 1 identifies the top-spending customers, helping the business understand its most valuable customers. It calculates each customer's total spending by aggregating the product of UnitPrice and Quantity. The results are sorted in descending order, and the top 10 customers are displayed. Each step's execution time is recorded. Finally, the script summarizes the timing results for each configuration, comparing their performance.

Query 2 analyses customer behavior by country. First, we calculate the total spending for each transaction by multiplying the UnitPrice by Quantity. We then group the data by Country and aggregate the total expenditure and the number of orders per country. Additionally, we compute the average order size by dividing the total spending by the total number of orders for each country.

Part Two: Sales Trends and Performance

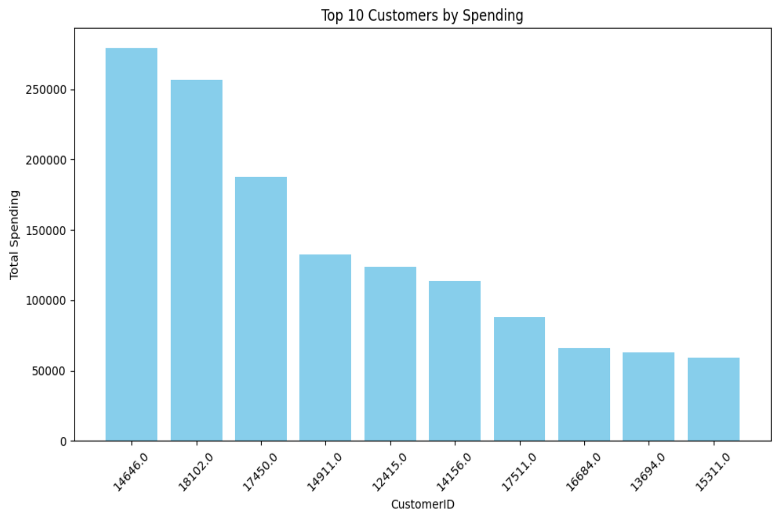
Query 3 evaluates sales performance over time by aggregating monthly sales, helping the business identify seasonal patterns and trends. We extract the year and month from the Date column using year() and month(). Then, we group the data by these columns and sum the Amount to calculate the total sales per month. And lastly, we sort the results by Total_Sales in ascending order.

Query 4 Highlights the most expensive purchases, enabling the business to focus on high-value transactions and better understand customer purchasing patterns. Here, we group the data by InvoiceNo and calculate the total sales for each invoice using the sum() function. We also retrieve the CustomerID and Country for each invoice using the first() function, as all rows for an invoice share the same values. Then, we sort the results by Total_Sales in descending order and limit the output to the top 10 invoices.

Part Three: Forecasting and Future Planning

Query 5 implements a forecasting model to predict revenue for the next 30 days, aiding in inventory planning, marketing strategies, and financial projections. Here we implement a sales forecasting pipeline using PySpark. First, we aggregate total sales by date and calculate the number of days since the earliest date in the dataset. then, we prepare the data for regression by training a linear regression model to predict total sales. Finally, we predict the sales for the next 30 days based on the model we trained, using the latest date in the dataset as a reference.

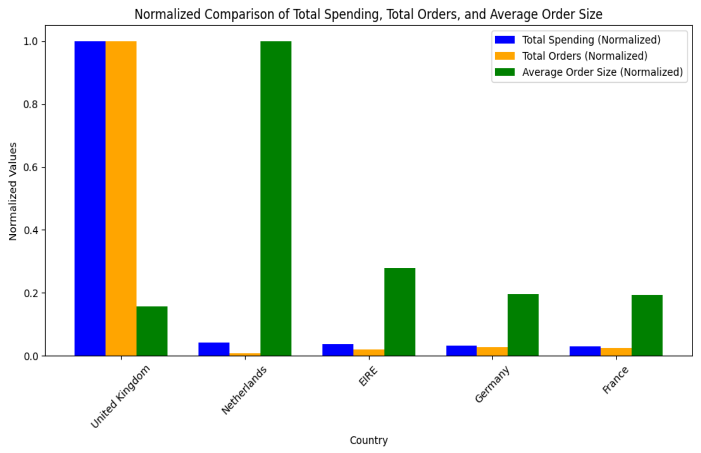
4. Results



Query 1 indicates that the highest spender is **CustomerID 14646**, with total spending of 279,489.02 followed by **CustomerID 18102** with 256,438.49. The plot shows that a small subset of customers contributes disproportionately to total revenue, highlighting a typical **Pareto distribution** (80/20 rule) in customer spending behavior.

Configuration	Data Loading(s)	Data Preprocessing	Data Processing	Result Display	Total Time (s)
Local [2]	2.48	0.01	0.04	2.03	4.56
Local [4]	5.2	0.01	0.05	2.02	7.28
Local [*]	2.55	0.01	0.04	1.86	4.46
Cluster	12.44	0.02	0.13	9.26	21.85

The timetable indicates that increasing the number of threads (**local [4]**) does not guarantee faster execution, especially if the task size is relatively small or if there's additional overhead in thread management. **Local [*]** shows promising results utilizes all available cores and provides the fastest execution. **Cluster** configuration indicates worst performance in every stage of processing which is something expected due to the overhead associated with managing a distributed system and also the fact the local device performs better are smaller dataset.



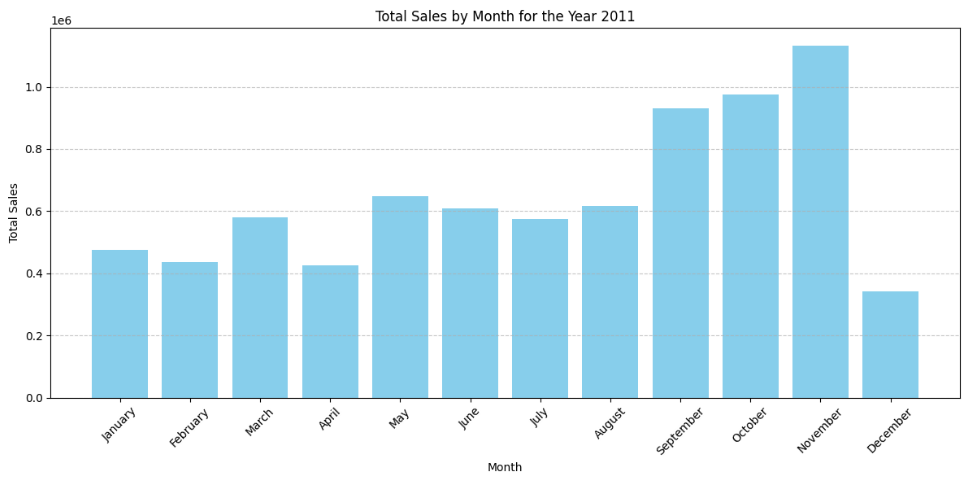
Query2 states that **UK** dominates with the highest total spending and largest number of orders. **Netherland** has the highest average order size, indicates fewer yet more expensive transactions. Counties like **EEIRE, Germany,** and **France** have a balanced mixed of spending and order count.

Configuration	Data Loading(s)	Data Preprocessing	Data Processing	Result Display	Total Time (s)
Local [2]	2.48	0.01	0.04	2.03	4.56
Local [4]	5.2	0.01	0.05	2.02	7.28
Local [*]	2.55	0.01	0.04	1.86	4.46
Cluster	11.99	0.03	0.16	9.50	21.68

The **Cluster** configuration exhibits a significantly longer total runtime of 21.68 seconds, primarily due to increased durations in data loading (11.99 seconds) and result display (9.50 seconds). This outcome aligns with expectations, as running small datasets in cluster mode often incurs additional overhead from managing distributed resources, leading to slower performance compared to local modes.

In contrast, the local configurations, especially **Local [2]** and **Local [*]**, demonstrate more efficient processing times for this dataset size. The reduced overhead in local modes allows for quicker data loading and result display, making them more suitable for handling smaller datasets.

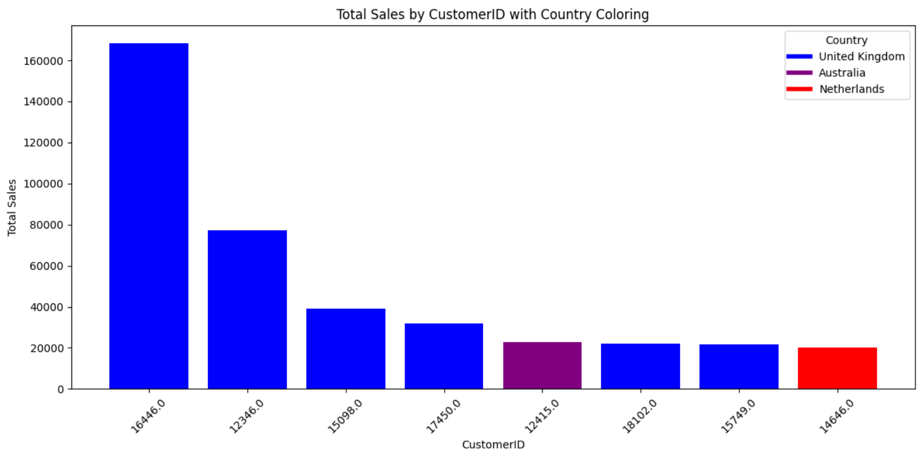
For **Query3**, November 2011 saw the highest sales (1,132,407.74), likely caused by holiday shopping. Also, Oct and Sep had strong performance, indicating sustained sales before year-end.



Configuration	Data Loading(s)	Data Preprocessing	Year and month extraction	Calculate total sales	Show total sale	Sort sales	Show sorted sales	Total Time (s)
Local [2]	3.06	0.01	0.02	0.02	1.74	0.01	1.97	6.63
Local [4]	2.94	0.01	0.02	0.02	3.25	0.02	3.36	9.62
Local [*]	3.27	0.01	0.02	0.02	1.94	0.01	1.9	7.17
Cluster	12.50	0.03	0.09	0.06	9.19	0.04	7.74	30.65

Again in **Cluster** configuration we can clearly observe the significant draw back in every stage of processing data compared the local configuration. The time gap is specifically visible in **Data Loading** and showing the plots.

on the local device, **Local [4]** had the longest runtime, due to slower display operations. **local[2]** and **local[*]** had similar runtimes, with **local[2]** being slightly faster in displaying results.



For **Query4**, the most expensive purchase belongs to CustomerID 16446 in the UK. Majority of the high-value invoices are from the UK, with two exceptions (Australia and Netherlands).

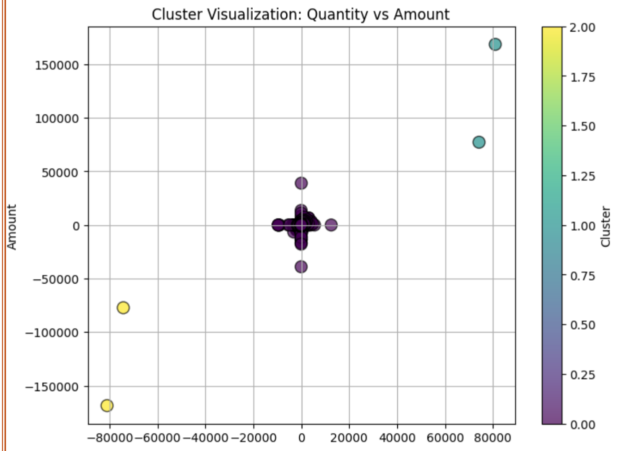
Configuration	Data Loading(s)	Data Preprocessing	Calculate Total Sales per Invoice	Sort and Filter Top 10	Show Results (s)	Total Time (s)
Local [2]	4.42	0.01	0.03	0.01	2.59	7.06
Local [4]	2.98	0.01	0.03	0.01	2.76	5.79
Local [*]	4.35	0.02	0.05	0.02	2.80	7.24
Cluster	14.43	0.02	0.10	0.03	16.94	31.52

The **cluster** mode took the longest (31.52s), mainly due to slow data loading (14.43s) and result display (16.94s). As seen before, the overhead of distributed processing isn't worth it for a small dataset. **Local [4]** was the fastest (5.79s), while **Local [2]** and **Local [*]** had similar times (~7s). For this dataset, local execution is the better choice.

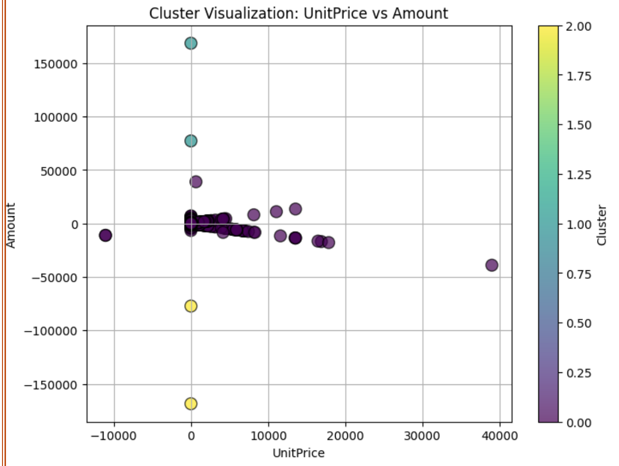
Using MLlib, we implemented a Regression model that predicts sales the next 20 days. with values ranging from **37,858.36** to **38,968.65**, the predicted values indicate consistent growth of prices over the next 20 days.

Configuration	Data Loading(s)	Data Preprocessing	Aggregate Sales (s)	Calculate Days Since Start	Assemble Features	Train Model	Predict Sales	Display Results	Result Display
Local [2]	2.57	0.01	0.01	2.93	0.04	6.90	0.10	0.04	13.88
Local [4]	2.44	0.01	0.01	3.00	0.05	7.52	0.11	0.05	14.89
Local [*]	3.44	0.01	0.03	5.85	0.06	5.19	0.09	0.04	15.84
Cluster	14.59	0.11	0.07	24.42	0.60	29.1	0.17	0.97	70.03

The **Cluster** mode took significantly longer, mainly due to slow data loading (14.59s), since start calculation (24.42s), and model training (29.1s). **Local [2]** was the fastest (13.88s), with **Local [4]** (14.89s) and **Local [*]** (15.84s) slightly slower due to overhead in training and feature processing. Once again, local execution is more efficient for this dataset.



In first plot**, Cluster purple** shows the majority of data points are near the origin, indicating group of transactions with relatively small quantities, which corresponds to regular low-value purchases. **Cluster Yellow**, represents unusual transactions with very large negative values in Quantity/Amount. **Cluster Cyan**, shows high value transactions with large quantities and amount, pointing to bulk purchases.



In the second plot, **Cluster purple** again shows common inexpensive transactions. The density and concentration of these data points show that the majority of purchases belong to low-value items. **Cluster Yellow** with negative values indicates customer returning products. **Cluster Cyan** like before shows the high-value item purchased.

Configuration	Data Loading(s)	Data Preprocessing	Feature Assembling	Model Training	Clustering	Show Results (s)	Total Time (s)
Local [2]	2.48	0.02	0.06	12.9	0.07	3.27	18.80
Local [4]	3.67	0.03	0.10	13.50	0.10	4.04	21.44
Local [*]	2.49	0.04	0.06	13.72	0.11	2.64	19.06
Cluster	15.37	0.20	0.58	30.88	0.08	13.07	60.18

The **cluster** mode was the slowest at (60.18s), mainly due to long data loading (15.37s), model training (30.88s), and result display (13.07s). **Local [2]** was the fastest (18.80s), while **Local [*]** (19.06s) and **Local [4]** (21.44s) were slightly slower. Model training dominated execution time (~60-70%), making local execution the better choice for this dataset.