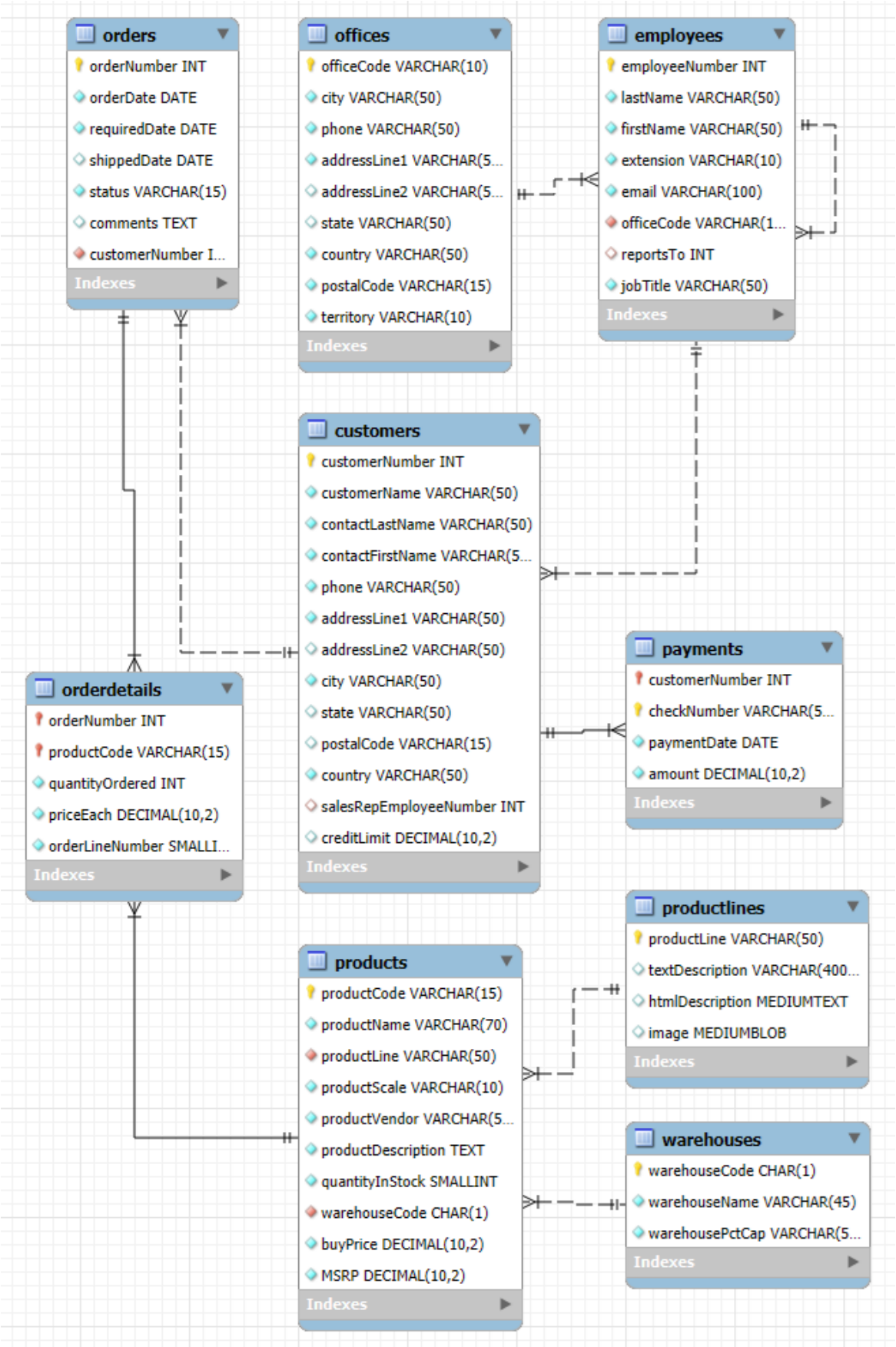
Overview

In this demo project, I will step into the shoes of an entry-level data analyst at the fictional Mint Classics Company, helping to analyze data in a relational database with the goal of supporting inventory-related business decisions that lead to the closure of a storage facility.

Tools

- I have performed SQL queries in MySQL database.
- I stored the results and create visualizations in Microsoft Excel.
- I used generative AI (Microsoft Copilot) to explore the database, generate explanations for SQL queries (that I've created on my own) and generate explanations for the results of the queries.

EER Diagram of the database:



EER Diagram Table Descriptions

customers

- Stores information about the company's clients.
- Includes fields like customerNumber, customerName, contact details, address, and salesRepEmployeeNumber.
- Links to the employees table to identify the sales representative.
- Central for tracking customer purchases and relationships.

employees

- Contains data about company employees including employeeNumber, names, contact info, and job title.
- Includes a reportsTo field to establish a hierarchy among employees.
- Links to customers as sales reps and to offices to indicate location.
- Supports internal organization and employee management.

offices

- Holds details about the company’s physical office locations.
- Includes officeCode, address, phone, and territory.
- Connects to the employees table to show where each employee is based.
- Supports geographic organization of the business.

orders

- Tracks customer orders with fields like orderNumber, orderDate, requiredDate, shippedDate, and status.
- Includes customerNumber to link orders to customers.
- Central to order management and tracking.

orderdetails

- Junction table connecting orders and products.
- Includes orderNumber, productCode, quantityOrdered, priceEach, and orderLineNumber.
- Allows multiple products per order and tracks quantities and pricing.
- Essential for detailed order breakdowns.

payments

- Records customer payments.
- Includes customerNumber, checkNumber, paymentDate, and amount.
- Helps in financial tracking and linking payments to customers.

products

- Stores product information such as productCode, productName, productLine, productVendor, quantityInStock, buyPrice, and MSRP.
- Links to orderdetails and productlines.
- Essential for inventory management and product tracking.

productlines

- Categorizes products into lines or groups.
- Includes productLine, textDescription, htmlDescription, and image.
- Supports product catalog organization and marketing.

warehouses

- Contains data about storage locations.
- Includes warehouseCode, warehouseName, and warehousePctCap (percentage capacity).
- Links to products via warehouseCode.
- Supports inventory distribution and logistics.

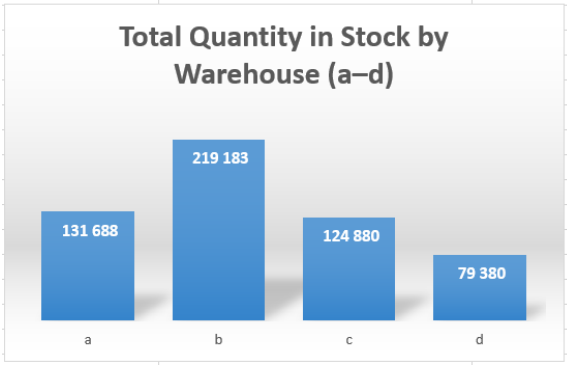
1. Warehouse Inventory and Utilization Analysis

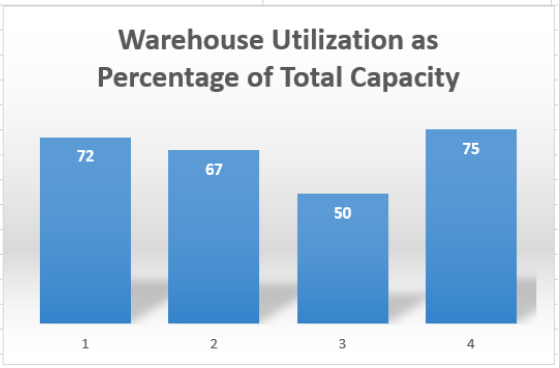
This SQL query analyzes warehouse inventory distribution and utilization. It uses a subquery to calculate the total quantity of products in stock per warehouse (totalQuantityInStockPerWarehouse) and the overall total stock across all warehouses. The outer query joins this result with the warehouses table to retrieve each warehouse's percentage capacity usage (warehousePctCap). It also computes the stock ratio of each warehouse relative to the total stock using a calculated field. The query uses aggregation (SUM), grouping (GROUP BY), subqueries, and an INNER JOIN to combine and enrich data from the products and warehouses tables.

```
SELECT  sub.warehouseCode,
        sub.totalQuantityInStockPerWarehouse,
        ware.warehousePctCap as warehouseUsagePercentOfTotalCapacity,
        sub.totalQuantityInStockPerWarehouse / sub.totalQuantityInStockOverAll * 100 AS warehouseStockRatioOverall
FROM (  SELECT  prod.warehouseCode,
        SUM(prod.quantityInStock) AS totalQuantityInStockPerWarehouse,
        (   SELECT SUM(prod.quantityInStock)
            FROM mintclassics.products AS prod  ) AS totalQuantityInStockOverAll
        FROM mintclassics.products AS prod
        GROUP BY prod.warehouseCode ) AS sub
        inner join  mintclassics.warehouses as ware on ware.warehouseCode = sub.warehouseCode;
```

Result:

A	B	C	D
warehouseCode	totalQuantityInStockPerWarehouse	warehouseUsagePercentOfTotalCapacity (%)	warehouseStockRatioOverall (%)
a	131 688	72	23,72
b	219 183	67	39,48
c	124 880	50	22,5
d	79 380	75	14,3





The result shows inventory and capacity usage for four warehouses (a, b, c, d):

- Warehouse b holds the most stock (219,183 units), accounting for 39.48% of total inventory, but its capacity usage is only 67%.
- Warehouse d has the least stock (79,380 units) but the highest utilization at 75%, suggesting it may be smaller or more constrained.
- Warehouse a and c have similar stock levels (~130k and ~125k), with utilization rates of 72% and 50%, respectively.

The stock ratio overall metric helps compare how much each warehouse contributes to the total inventory, while the capacity usage shows how efficiently each warehouse is being used.

Based on the data provided, the most appropriate warehouse to consider for closure is Warehouse D. Here's why:

- Lowest stock volume: Warehouse D holds only 79,380 units, the smallest share among all warehouses.
- Lowest contribution to overall stock: It accounts for just 14.3% of the total inventory, meaning its closure would have the least impact on overall stock availability.
- High utilization rate: Despite its small stock, it operates at 75% capacity, suggesting it is relatively small or inefficient in space usage.
- Efficiency trade-off: Closing a warehouse with high utilization but low stock contribution may free up operational costs without significantly disrupting inventory distribution.

This decision balances minimizing operational disruption while optimizing resource allocation.

Although this query provides an answer to the main question - which warehouse should be closed - let's continue with further analysis to uncover additional valuable insights from the data.

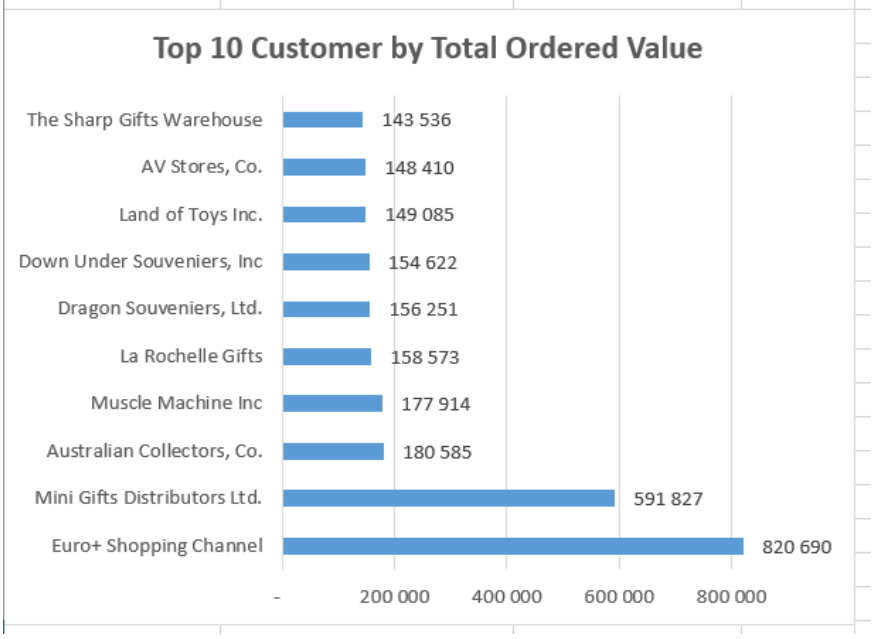
2. Top 10 Customers by Total Order Value

This SQL query identifies the top 10 customers based on the total value of their orders. It joins three tables: orderDetails (which contains product quantities and prices), orders (which links orders to customers), and customers (which provides customer details). The query calculates the total ordered value per customer by multiplying quantityOrdered by priceEach and summing the result. It uses GROUP BY to aggregate data per customer, ORDER BY to sort them by total value in descending order, and LIMIT to return only the top 10.

```
select  cust.customerNumber,
        cust.customerName,
        sum((ordd.quantityOrdered * ordd.priceEach)) as totalOrderedValue
from    mintclassics.orderDetails ordd
        inner join mintclassics.orders as ords on ordd.orderNumber = ords.orderNumber
        inner join mintclassics.customers AS cust on cust.customerNumber = ords.customerNumber
group by    cust.customerNumber,
            cust.customerName
order by    sum((ordd.quantityOrdered * ordd.priceEach)) desc
limit       10;
```

Result:

A	B	C	D
customerNumbe	customerName	totalOrderedValue	
141	Euro+ Shopping Channel	820 690	
124	Mini Gifts Distributors Ltd.	591 827	
114	Australian Collectors, Co.	180 585	
151	Muscle Machine Inc	177 914	
119	La Rochelle Gifts	158 573	
148	Dragon Souveniers, Ltd.	156 251	
323	Down Under Souveniers, Inc	154 622	
131	Land of Toys Inc.	149 085	
187	AV Stores, Co.	148 410	
450	The Sharp Gifts Warehouse	143 536	



The result highlights the top 10 customers by total purchase value:

- Euro+ Shopping Channel is the most valuable customer, with a total ordered value of $820,690.
- Mini Gifts Distributors Ltd. follows with $591,827, showing a significant gap from the top.
- The remaining customers have total values ranging from $180,585 down to $143,536.
- The bar chart visually emphasizes the dominance of the top two customers, suggesting they are key accounts.
- This insight can guide customer segmentation, priority support, or targeted marketing strategies.

3. Monthly Sales Trends for Key Customers

This SQL query analyzes monthly sales performance for five specific customers by calculating their total ordered value per month. It joins the orderDetails, orders, and customers tables to access product pricing, order dates, and customer identities. The query uses CASE WHEN statements inside SUM() functions to isolate and aggregate order values for each customer. It groups results by month (yearMonth) using DATE_FORMAT, and sorts them chronologically. This technique is useful for time series analysis and customer-specific trend tracking.

```
select  DATE_FORMAT(ords.orderDate, '%Y-%m') as yearMonth,
        sum( case when cust.customerNumber = 141 then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueEuroPlus,
        sum( case when cust.customerNumber = 124 then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueMinGifts,
        sum( case when cust.customerNumber = 114 then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueAustralianCollectors,
        sum( case when cust.customerNumber = 151 then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueMuscleMacine,
        sum( case when cust.customerNumber = 119 then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueLaRochelle
from    mintclassics.orderDetails ordd
        inner join mintclassics.orders as ords on ordd.orderNumber = ords.orderNumber
        inner join mintclassics.customers AS cust on cust.customerNumber = ords.customerNumber
group by    DATE_FORMAT(ords.orderDate, '%Y-%m')
order by    DATE_FORMAT(ords.orderDate, '%Y-%m');
```

Result:

A	B	C	D	E	F
yearMonth	totalOrderedValue EuroPlus	totalOrderedValue MinGifts	totalOrderedValue AustralianCollectors	totalOrderedValue MuscleMacine	totalOrderedValue LaRochelle
2003.01.01	40 206	-	-	-	-
2003.02.01	-	-	-	-	-
2003.03.01	-	11 044	-	-	-
2003.04.01	-	-	45 864	-	-
2003.05.01	-	-	7 565	-	-
2003.06.01	36 251	-	-	58 841	-
2003.07.01	-	55 602	-	-	-
2003.08.01	-	56 053	-	-	-
2003.09.01	44 940	-	-	-	-
2003.10.01	4 600	-	-	-	-
2003.11.01	10 722	45 084	-	-	-
2003.12.01	53 122	-	-	58 794	-
2004.01.01	59 831	-	-	-	-
2004.02.01	-	-	44 895	-	-
2004.03.01	-	43 369	-	-	-
2004.04.01	26 156	-	-	-	-
2004.05.01	35 421	-	-	-	-
2004.06.01	47 065	-	-	-	-
2004.07.01	-	37 431	-	20 314	47 924
2004.08.01	20 010	47 980	-	-	-
2004.09.01	-	-	-	-	-
2004.10.01	36 140	55 640	-	-	19 502
2004.11.01	-	6 466	82 261	-	-
2004.12.01	116 208	40 676	-	39 965	-
2005.01.01	-	49 012	-	-	-
2005.02.01	120 167	52 232	-	-	49 524
2005.03.01	65 071	83 598	-	-	-
2005.04.01	-	-	-	-	-
2005.05.01	104 781	7 639	-	-	41 623
Control sum	**820 690**	**591 827**	**180 585**	**177 914**	**158 573**

The result presents a month-by-month breakdown of total ordered values for five key customers from January 2003 to May 2005. At the bottom, a control sum row shows the cumulative total for each customer:

- EuroPlus Shopping Channel leads with $820,690, indicating it's the most valuable customer over time.
- Mini Gifts Distributors Ltd. follows with $591,827, also showing consistent monthly activity.
- The other three customers—Australian Collectors, Muscle Machine, and La Rochelle Gifts—have lower totals, ranging from $158,573 to $180,585.

This result helps identify sales trends, seasonal patterns, and customer value over time, which are crucial for forecasting and strategic planning.

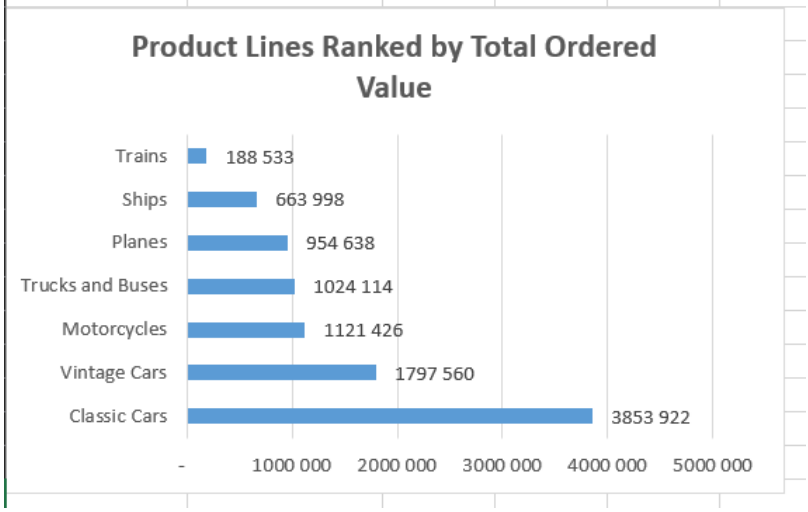
4. Total Sales by Product Line

This SQL query calculates the total ordered value for each product line by joining the orderDetails and products tables. It multiplies quantityOrdered by priceEach to compute the value of each order line, then aggregates these values using SUM() grouped by productLine. The results are sorted in descending order to highlight the most profitable product lines. This approach uses aggregation, joins, and ordering to perform a category-level sales analysis.

```sql
select  prod.productLine,
        sum((ordd.quantityOrdered * ordd.priceEach)) as totalOrderedValue
from    mintclassics.orderDetails ordd
        inner join mintclassics.products AS prod on prod.productCode = ordd.productCode
group by    prod.productLine
order by    sum((ordd.quantityOrdered * ordd.priceEach)) desc;
```

Result:

productLine	totalOrderedVa
Classic Cars	3 853 922
Vintage Cars	1 797 560
Motorcycles	1 121 426
Trucks and Buses	1 024 114
Planes	954 638
Ships	663 998
Trains	188 533



The result shows that Classic Cars are the top-performing product line with a total ordered value of $3,853,922, significantly ahead of the next category, Vintage Cars at $1,797,560. Other notable categories include Motorcycles, Trucks and Buses, and Planes, each contributing over $950,000. Ships and Trains have the lowest totals, with Trains at just $188,533. The bar chart visually reinforces this ranking, making it clear which product lines drive the most revenue. This insight is valuable for inventory prioritization, marketing focus, and strategic planning.

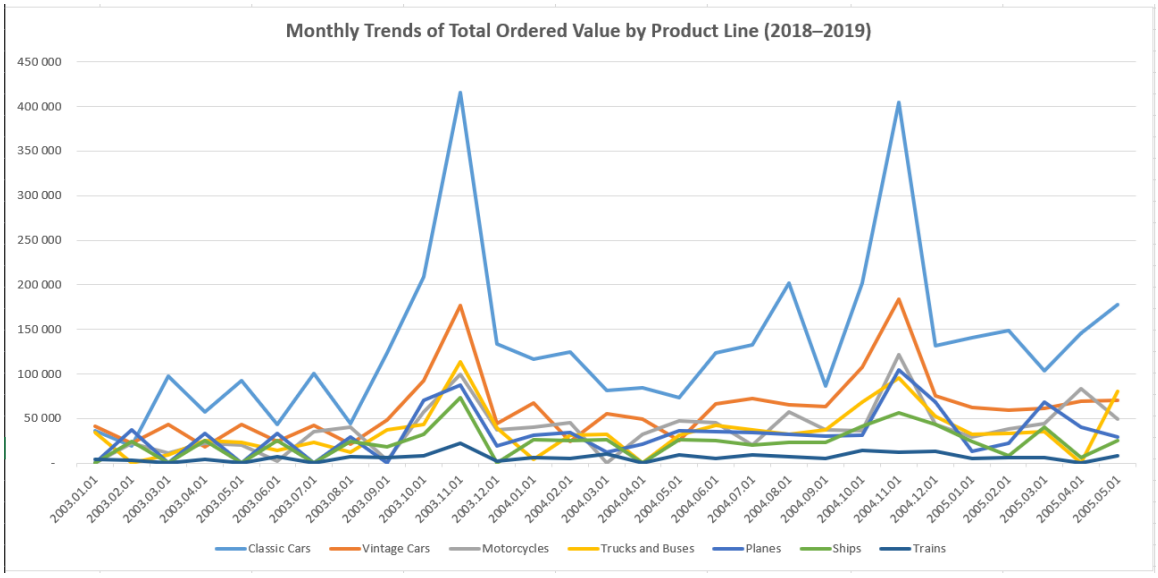
5. Monthly Sales Trends by Product Line

This SQL query analyzes monthly sales trends across different product lines. It joins the orderDetails, products, and orders tables to access order dates, product categories, and pricing. Using CASE WHEN inside SUM() functions, it calculates the total ordered value for each product line per month. The query groups results by month (yearMonth) using DATE_FORMAT, and sorts them chronologically. It also includes a grand total column to show the overall monthly sales across all product lines.

```sql
select  DATE_FORMAT(ords.orderDate, '%Y-%m') as yearMonth,
        sum( case when prod.productLine = "Classic Cars" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueClassicCars,
        sum( case when prod.productLine = "Vintage Cars" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueVintageCars,
        sum( case when prod.productLine = "Motorcycles" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueMotorcycles,
        sum( case when prod.productLine = "Trucks and Buses" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueTrucksandBuses,
        sum( case when prod.productLine = "Planes" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValuePlanes,
        sum( case when prod.productLine = "Ships" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueShips,
        sum( case when prod.productLine = "Trains" then (ordd.quantityOrdered * ordd.priceEach) else 0 end) as totalOrderedValueTrains,
        sum(ordd.quantityOrdered * ordd.priceEach) as totalOrderedGrandTotal
from    mintclassics.orderDetails ordd
        inner join mintclassics.products AS prod on prod.productCode = ordd.productCode
        inner join mintclassics.orders as ords on ordd.orderNumber = ords.orderNumber
group by    DATE_FORMAT(ords.orderDate, '%Y-%m')
order by    DATE_FORMAT(ords.orderDate, '%Y-%m');
```

Result:

A	B	C	D	E	F	G	H
yearMonth	totalOrderedValue ClassicCars	totalOrderedValue VintageCars	totalOrderedValue Motorcycles	totalOrderedValue TrucksandBuses	totalOrderedValue Planes	totalOrderedValue Ships	totalOrderedValue Trains
2003.01.01	36 552	40 951	-	34 713	-	-	4 477
2003.02.01	18 941	22 253	22 293	-	37 136	24 447	3 334
2003.03.01	97 088	43 779	10 770	8 881	-	-	-
2003.04.01	57 545	18 135	21 815	24 942	33 693	25 207	4 511
2003.05.01	92 906	43 611	19 815	23 103	-	-	-
2003.06.01	43 840	24 518	1 861	14 759	32 895	24 964	7 633
2003.07.01	101 011	42 271	35 108	23 549	-	-	-
2003.08.01	44 312	21 049	40 042	12 708	28 847	24 273	7 027
2003.09.01	123 645	48 573	2 834	37 585	-	18 123	5 937
2003.10.01	209 334	92 620	57 450	43 527	70 847	32 021	8 539
2003.11.01	415 953	177 114	99 080	113 664	87 193	73 147	21 875
2003.12.01	133 704	44 290	37 842	39 225	19 174	-	2 489
2004.01.01	116 763	67 164	39 987	4 616	31 159	26 310	6 387
2004.02.01	124 584	24 433	45 694	31 134	34 000	24 894	4 763
2004.03.01	81 639	55 500	-	32 193	12 114	26 367	9 879
2004.04.01	84 319	49 260	32 229	-	21 768	-	-
2004.05.01	73 228	24 835	47 873	31 729	35 898	25 877	8 886
2004.06.01	123 358	66 279	45 309	41 967	35 684	25 410	5 364
2004.07.01	133 007	72 418	19 847	36 967	34 079	20 260	8 985
2004.08.01	201 860	65 019	57 601	32 147	32 083	23 485	7 132
2004.09.01	86 729	62 984	37 008	37 720	30 634	23 114	5 611
2004.10.01	202 056	107 121	36 694	68 620	31 081	40 881	13 781
2004.11.01	404 161	183 657	121 934	95 685	104 816	56 891	12 148
2004.12.01	131 433	75 882	43 069	52 612	68 654	43 838	13 350
2005.01.01	140 731	62 516	29 552	32 571	12 849	24 257	5 261
2005.02.01	148 696	59 305	38 813	33 126	22 537	8 230	6 485
2005.03.01	103 494	61 220	44 019	35 636	68 317	40 537	6 490
2005.04.01	145 358	69 840	83 717	-	39 871	6 034	-
2005.05.01	177 675	70 965	49 171	80 734	29 308	25 432	8 190



The result shows how each product line performed month by month from January 2003 to May 2005. The line graph visualizes these trends, revealing that:

- Classic Cars consistently dominate in total ordered value, showing strong and stable demand.
- Vintage Cars, Motorcycles, and Trucks and Buses follow with moderate but steady performance.
- Planes, Ships, and Trains contribute less and show more fluctuation over time.
- The graph helps identify seasonal patterns, growth trends, and potential dips in product line performance.

This analysis is ideal for sales forecasting, inventory planning, and strategic product focus.

6. Customer Diversity in Product Line Orders

This SQL query creates a view named countOfOrderedProductLinesByCustomers that tracks how many distinct product lines each customer has ordered from. It joins the customers, orders, orderDetails, and products tables to trace each customer's purchases down to the product line level. The query uses COUNT(DISTINCT ...) to ensure only unique product lines are counted per customer. The results are grouped by customer and sorted in descending order of product line diversity, helping identify the most broadly engaged customers.

```
create view mintclassics.countOfOrderedProductLinesByCustomers as
select  cust.customerNumber,
        cust.customerName,
        count(distinct prod.productLine) as countOfOrderedProductLines
from    mintclassics.customers as cust
        inner join mintclassics.orders as ords on ords.customerNumber = cust.customerNumber
        inner join mintclassics.orderDetails as ordd on ords.orderNumber = ordd.orderNumber
        inner join mintclassics.products as prod on prod.productCode = ordd.productCode
group by    cust.customerNumber,
            cust.customerName
order by    count(distinct prod.productLine) desc;

select  *
from    mintclassics.countOfOrderedProductLinesByCustomers;
```

Result:

customerNumber	customerName	countOfOrderedProductLines
131	Land of Toys Inc.	7
141	Euro+ Shopping Channel	7
398	Tokyo Collectables, Ltd	7
353	Reims Collectables	7
319	Mini Classics	6
320	Mini Creations Ltd.	6
189	Clover Collections, Co.	6
119	La Rochelle Gifts	6
496	Kelly's Gift Shop	6
124	Mini Gifts Distributors Ltd.	6
282	Souveniers And Things Co.	6
448	Scandinavian Gift Ideas	6
333	Australian Gift Network, Co	6
148	Dragon Souveniers, Ltd.	6
386	L'ordine Souveniers	6
157	Diecast Classics Inc.	6
146	Saveley & Henriot, Co.	5
250	Lyon Souveniers	5

The result of the view shows a list of customers and the number of different product lines they have ordered from:

- Customers like Land of Toys Inc., Euro+ Shopping Channel, Tokyo Collectables, Ltd, and Reims Collectables have ordered from 7 distinct product lines, the highest in the dataset.
- Several others, including Mini Classics, Mini Creations Ltd., and La Rochelle Gifts, have ordered from 6 product lines, indicating a wide interest in the product catalog.
- This metric is useful for identifying diverse buyers who may be more loyal or open to cross-selling opportunities.

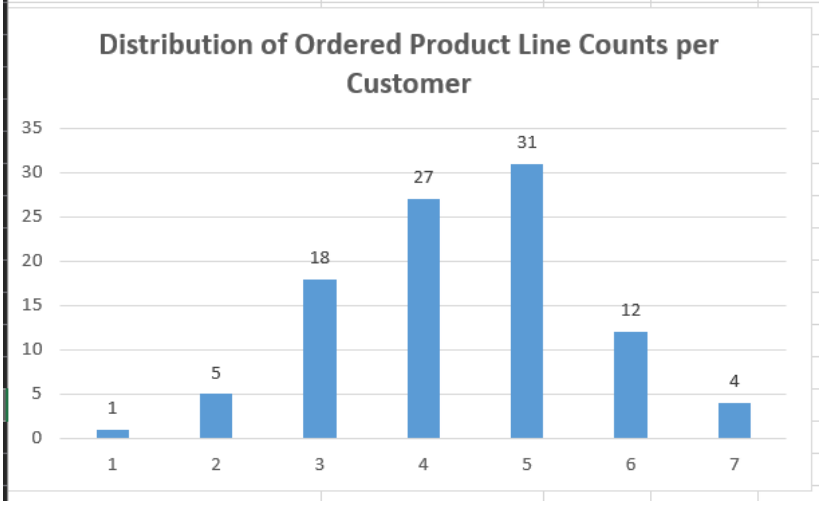
7. Distribution of Customer Product Line Diversity

This SQL query analyzes the distribution of customer diversity in terms of product lines ordered. It queries the previously created view countOfOrderedProductLinesByCustomers, which contains the number of distinct product lines each customer has ordered. The query groups customers by this count and uses COUNT(*) to determine how many customers fall into each group. The result is sorted by the number of product lines, making it easy to visualize the spread of customer ordering behavior.

```
select  countOfOrderedProductLines,
        count(*) as countOfDataInBin
from    mintclassics.countOfOrderedProductLinesByCustomers
group by countOfOrderedProductLines
order by countOfOrderedProductLines;
```

Result:

countOfOrderedProductLin	countOfDataInBin
1	1
2	5
3	18
4	27
5	31
6	12
7	4



The result shows how many customers have ordered from a given number of distinct product lines:

- The most common group is customers who ordered from 5 product lines (31 customers).
- This is followed by those who ordered from 4 lines (27 customers) and 3 lines (18 customers).
- Only 4 customers ordered from all 7 product lines, indicating high engagement.
- On the other end, 1 customer ordered from just 1 product line, showing minimal diversity.

The accompanying bar chart clearly illustrates this distribution, highlighting that most customers tend to order from 3 to 5 product lines, which can inform segmentation strategies or cross-selling opportunities.