# 

1. Sales Analysis: Analyze sales data, including total sales by product, daily trends, and high-value transactions

• Aggregate sales by product.

• Use CASE to categorize sales into high, medium, and low-value bins.

• Use NTILE to divide sales into quartiles for deeper analysis.

**Aggregate Sales by Product**

*SELECT*

*ProductName,*

*SUM(SalesAmount) AS TotalSales*

*FROM*

*Sales*

*GROUP BY*

*ProductName*

*ORDER BY*

*TotalSales DESC;*

**output**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| | **ProductName** | | --- |  |  | | --- | |  |  |  |  | | --- | --- | |  |  | | | **TotalSales** | | --- | |
| |  | | --- | |  |   Laptop | |  | | --- | | 150000 | |
| Smartphone | 120000 |
| Tablet | 60000 |

**Categorize Sales into High, Medium, and Low-Value Bins Using CASE**

*SELECT*

*SaleID,*

*SalesAmount,*

*CASE*

*WHEN SalesAmount >= 10000 THEN 'High'*

*WHEN SalesAmount BETWEEN 5000 AND 9999 THEN 'Medium'*

*ELSE 'Low'*

*END AS SaleCategory*

*FROM*

*Sales;*

**Output**

|  |  |  |
| --- | --- | --- |
| **SaleID** | **SalesAmount** | **SaleCategory** |
| **1** | **15000** | **High** |
| **2** | **7000** | **Medium** |
| **3** | **3000** | **Low** |

**Divide Sales into Quartiles Using NTILE**

*SELECT*

*SaleID,*

*SalesAmount,*

*NTILE(4) OVER (ORDER BY SalesAmount DESC) AS Quartile*

*FROM*

*Sales;*

|  |  |  |
| --- | --- | --- |
| **SaleCategory** | **SalesAmount** | **Quartile** |
| **1** | **20000** | **1** |
| **2** | **15000** | **1** |
| **3** | **10000** | **2** |
| **4** | **8000** | **2** |

1. Customer Segmentation: Segment customers based on their purchasing behavior, gender, and Net Promoter Scores (NPS).

• Use CASE to standardize gender values.

• Categorize NPS survey responses into Promoters, Passives, and Detractors.

• Create flags for high-value customers.

**Standardize Gender Values Using CASE**

SELECT

CustomerID,

CASE

WHEN LOWER(Gender) IN ('male', 'm') THEN 'Male'

WHEN LOWER(Gender) IN ('female', 'f') THEN 'Female'

ELSE 'Other'

END AS

StandardizedGender

FROM

Customers;

|  |  |
| --- | --- |
| CustomerID | StandardizedGender |
| 101 | |  | | --- | |  |  |  | | --- | | Male | |
| 102 | Female |
| 103 | Other |

**Categorize NPS Survey Responses**

SELECT

CustomerID,

NPSScore,

CASE

WHEN NPSScore >= 9 THEN 'Promoter'

WHEN NPSScore BETWEEN 7 AND 8 THEN 'Passive'

ELSE 'Detractor'

END AS NPSCategory

FROM CustomerFeedback;

**into Promoters, Passives, and Detractors  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
output**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **NPSScore** | **NPSCategory** |
| **101** | **10** | **Promoter** |
| **102** | **8** | **Passive** |
| **103** | **6** | **Detractor** |

**Create Flags for High-Value Customers**

SELECT

CustomerID, TotalPurchases, NPSScore,

CASE

WHEN TotalPurchases > 10000 OR NPSScore >= 9 THEN 1

ELSE 0

END AS IsHighValueCustomer

FROM Customers;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerID** | **NPSScore** | |  | | --- | |  |  |  | | --- | | **IsHighValueCustomer** | |
| **101** | **12000** | **10** | **1** |
| **102** | **8000** | **8** | **0** |
| **103** | **5000** | **9** | **1** |

**output**

1. Fraud Detection: Detect duplicate and suspicious transactions in an orders database.

• Find duplicate orders using GROUP BY and HAVING COUNT(\*) > 1.

• Use CASE to flag transactions with unusually high amounts.

• Use PERCENT RANK to identify outliers.

**Find Duplicate Orders Using GROUP BY and HAVING COUNT(\*) > 1**

SELECT

OrderID,

CustomerID,

OrderDate,

COUNT(\*) AS DuplicateCount

FROM

Orders

GROUP BY

OrderID, CustomerID, OrderDate

HAVING

COUNT(\*) > 1;

**OUTPUT**

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID | CustomerID | OrderDate | DuplicateCount |
| 1001 | 200 | 2024-12-01 | 2 |
| 1002 | 201 | |  | | --- | | 2024-12-02 |  |  | | --- | |  | | 3 |
|  |  |  |  |

**Flag Transactions with Unusually High Amounts Using CASE**

**OUTPUT**

SELECT

TransactionID,

OrderAmount,

CASE

WHEN OrderAmount > 5000 THEN 'Suspicious'

ELSE 'Normal'

END AS TransactionFlag

FROM Transactions;

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | TransactionID |  |  | | --- | |  | | |  | | --- | | OrderAmount |  |  | | --- | |  | | | TransactionFlag | | --- |  |  | | --- | |  | |
| 3001 | 6000 | Suspicious |
| 3002 | 4500 | Normal |
| 3003 | 8000 | Suspicious |

**Identify Outliers Using PERCENT\_RANK**

SELECT

TransactionID, OrderAmount,

PERCENT\_RANK() OVER (ORDER BY OrderAmount DESC) AS Rank

FROM

Transactions

WHERE

PERCENT\_RANK() OVER (ORDER BY OrderAmount DESC) > 0.95;

**Output**

|  |  |  |
| --- | --- | --- |
| **TransactionID** | **OrderAmount** | **Rank** |
| **3003** | **8000** | **0.98** |
| **3001** | **6000** | **0.96** |

1. Employee Data Quality Check: Ensure data quality in an employee database by identifying and fixing null or inconsistent values.

• Use COALESCE and NVL to replace nulls with default values.

• Use CASE to standardize department names.

• Detect missing or duplicate records in the database.

**Replace Nulls with Default Values Using COALESCE or NVL**

SELECT

EmployeeID,

COALESCE(FirstName, 'Unknown') AS FirstName,

COALESCE(LastName, 'Unknown') AS LastName,

NVL(Salary, 0) AS Salary, COALESCE(Department, 'Not Assigned') AS Department

FROM Employees;

**Output**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EmployeeID | FirstName | LastName | Salary | Department |
| 101 | John | Dae | 50000 | HR |
| 102 | Unknown | Smith | 0 | Not Assigned |
| 103 | Jane | Unknown | 70000 | Engineering |

**Standardize Department Names Using CASE**

SELECT

EmployeeID, Department,

CASE

WHEN LOWER(Department) IN ('hr', 'human resources') THEN 'Human Resources'

WHEN LOWER(Department) IN ('eng', 'engineering') THEN 'Engineering'

WHEN LOWER(Department) IN ('sales', 'marketing') THEN 'Sales & Marketing'

ELSE 'Other'

END AS StandardizedDepartment

FROM Employees;

**Output**

|  |  |  |
| --- | --- | --- |
| EmployeeID | Department | StandardizedDepartment |
| 101 | HR | Human Resources |
| 102 | Eng | Engineering |
| 103 | marketing | Sales & Marketing |

**Detect Missing or Duplicate Records**

SELECT

EmployeeID,

FirstName,

LastName,

Department

FROM

Employees

WHERE

FirstName IS NULL

OR LastName IS NULL

OR Department IS NULL;

**Output**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | EmployeeID |  |  | | --- | |  | | |  | | --- | | FirstName |  |  | | --- | |  | | |  | | --- | | LastName |  |  | | --- | |  | | | Department | | --- |  |  | | --- | |  | |
| 102 | NULL | Smith | NULL |
|  |  |  |  |

**Detect Duplicate Records**

SELECT

EmployeeID,

COUNT(\*) AS DuplicateCount

FROM

Employees

GROUP BY

EmployeeID

HAVING

COUNT(\*) > 1;

Output

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **EmployeeID** |  |  | | --- | |  | | DuplicateCount |
| 103 | 2 |

1. 5. Retail Store Inventory: Analyze inventory data for a fruit shop, including frequency and trend analysis.

• Count occurrences of each fruit in the inventory.

• Create a histogram of sales using GROUP BY.

• Use NTILE to group inventory levels into deciles.

**Count occurrences of each fruit in the inventory**

Output

SELECT

FruitName,

COUNT(\*) AS FruitCount

FROM

FruitInventory

GROUP BY

FruitName

ORDER BY

FruitCount DESC;

|  |  |  |  |
| --- | --- | --- | --- |
| **FruitName** | |  | | --- | |  |  |  | | --- | | **FruitCount** | |
| Apple | 50 |
| Banana | 30 |
| Orange | 20 |

Create a Histogram of Sales Using GROUP BY

SELECT

SalesRange,

COUNT(\*) AS Frequency

FROM (

SELECT

CASE

WHEN SalesQuantity BETWEEN 1 AND 10 THEN '1-10'

WHEN SalesQuantity BETWEEN 11 AND 20 THEN '11-20'

WHEN SalesQuantity BETWEEN 21 AND 30 THEN '21-30'

ELSE '31+'

END AS SalesRange

FROM

Sales

) AS SalesHistogram

GROUP BY SalesRange ORDER BY SalesRange;

Output

|  |  |
| --- | --- |
| SalesRange | Frequency |
| 1-10 | 15 |
| 11-20 | 10 |
| 21-30 | 5 |
| 31+ | 2 |

Group Inventory Levels into Deciles Using NTILE

SELECT

FruitName,

InventoryLevel,

NTILE(10) OVER (ORDER BY InventoryLevel DESC) AS Decile FROM

Inventory;

Output

|  |  |  |
| --- | --- | --- |
| FruitName | InventoryLevel | Decile |
| Apple | 100 | 1 |
| Banana | 80 | 2 |
| Orange | 60 | 3 |

1. Transaction Binning and Quartile Analysis: Categorize transactions into bins and analyze quartiles for financial transactions.

• Use NTILE to divide transactions into quartiles.

• Use CASE to categorize transactions as small, medium, or large.

• Find lower and upper bounds for each bin.

Divide Transactions into Quartiles Using NTILE

SELECT

TransactionID,

TransactionAmount,

NTILE(4) OVER (ORDER BY TransactionAmount ASC) AS Quartile

FROM

Transactions;

Output

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TransactionID** | |  | | --- | |  |  |  | | --- | | **TransactionAmount** | | **Quartile** |
| 101 | 100 | 1 |
| 102 | 300 | 1 |
| 103 | 500 | 2 |
| 104 | 700 | 3 |
| 105 | 1000 | 4 |

Categorize Transactions as Small, Medium, or Large Using CASE

SELECT

TransactionID,

TransactionAmount,

CASE

WHEN TransactionAmount <= 500 THEN 'Small'

WHEN TransactionAmount BETWEEN 501 AND 1000 THEN 'Medium'

ELSE 'Large'

END AS TransactionCategory

FROM

Transactions;

Output

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TransactionID** | |  | | --- | |  |  |  | | --- | | **TransactionAmount** | | **TransactionCategory** |
| 101 | 100 | Small |
| 102 | 300 | Small |
| 103 | 600 | Medium |
| 104 | 900 | Medium |
| 105 | 1500 | Large |

Find Lower and Upper Bounds for Each Bin

WITH BinnedTransactions AS (

SELECT

NTILE(4) OVER (ORDER BY TransactionAmount ASC) AS Quartile, TransactionAmount

FROM Transactions )

Output

SELECT

Quartile,

MIN(TransactionAmount) AS LowerBound,

MAX(TransactionAmount) AS UpperBound

FROM

BinnedTransactions

GROUP BY

Quartile

ORDER BY

Quartile;

|  |  |  |
| --- | --- | --- |
| Quartile | LowerBound | UpperBound |
| 1 | 100 | 300 |
| 2 | 301 | 500 |
| 3 | 501 | 700 |
| 4 | 701 | 1500 |

1. Historical Transaction Trends: Analyze historical sales trends using pivoting and unpivoting techniques.

• Use PIVOT to summarize sales trends by date and product type.

• Use UNION ALL to unpivot data for vertical analysis

• Combine historical and current data for trend comparisons.

Use PIVOT to Summarize Sales Trends by Date and Product Type

SELECT

SaleDate, [Electronics] AS ElectronicsSales, [Clothing] AS ClothingSales, [Groceries] AS GroceriesSales

FROM

(SELECT SaleDate, ProductType, SalesAmount FROM Sales) AS SourceTable

PIVOT

(SUM(SalesAmount) FOR ProductType IN ([Electronics], [Clothing], [Groceries])) AS PivotTable ORDER BY

SaleDate;

Output

|  |  |  |  |
| --- | --- | --- | --- |
| SaleDate | ElectronicsSales | ClothingSales | GroceriesSales |
| 2024-12-01 | 5000 | 3000 | 2000 |
| 2024-12-02 | 4000 | 2500 | 1500 |
| 2024-12-03 | 6000 | 3500 | 3000 |

Use UNION ALL to Unpivot Data for Vertical Analysis

SELECT

SaleDate,

'Electronics' AS ProductType,

ElectronicsSales AS SalesAmount

FROM

(SELECT

SaleDate,

[Electronics] AS ElectronicsSales,

[Clothing] AS ClothingSales,

[Groceries] AS GroceriesSales

FROM

(SELECT

SaleDate,

ProductType,

SalesAmount

FROM

Sales) AS SourceTable

PIVOT

(SUM(SalesAmount) FOR ProductType IN ([Electronics], [Clothing], [Groceries])) AS PivotTable

) AS PivotedData

UNION ALL

SELECT

SaleDate, 'Clothing', ClothingSales

FROM PivotedData

UNION ALL

SELECT

SaleDate, 'Groceries', GroceriesSales

FROM PivotedData

ORDER BY

SaleDate, ProductType;

Output

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SaleDate** | |  | | --- | |  |  |  | | --- | | **ProductType** | | **SalesAmount** |
| 2024-12-01 | |  | | --- | |  |  |  | | --- | | Electronics | | 5000 |
| 2024-12-01 | Clothing | 3000 |
| 2024-12-01 | Groceries | 2000 |
| 2024-12-02 | Electronics | 4000 |
| 2024-12-02 | |  | | --- | |  |  |  | | --- | | Clothing | | 2500 |

Combine Historical and Current Data for Trend Comparisons

SELECT

SaleDate,

ProductType,

SUM(SalesAmount) AS TotalSales,

CASE

WHEN SaleDate < '2024-12-01' THEN 'Historical'

ELSE 'Current'

END AS TimePeriod

FROM

(SELECT SaleDate, ProductType, SalesAmount FROM HistoricalSales

UNION ALL SELECT SaleDate, ProductType, SalesAmount FROM CurrentSales) AS

Output

CombinedData

GROUP BY

SaleDate, ProductType,

CASE

WHEN SaleDate < '2024-12-01' THEN 'Historical'

ELSE 'Current'

END

ORDER BY

SaleDate, ProductType;

|  |  |  |  |
| --- | --- | --- | --- |
| SaleDate | ProductType | TotalSales | TimePeriod |
| 2024-11-30 | Electronics | 4000 | Historical |
| 2024-12-01 | Electronics | 5000 | Current |
| 2024-12-01 | Clothing | 3000 | Current |

1. Missing Data Imputation: Handle missing data in a database, filling in gaps for null or incomplete entries.

• Identify null values using IS NULL.

• Use COALESCE or NVL to replace missing values with defaults.

• Update missing salaries or attributes with calculated estimates.

**Identify Null Values Using IS NULL**

SELECT

EmployeeID, FirstName, LastName, Salary, Department

FROM Employees

WHERE

FirstName IS NULL

OR LastName IS NULL

OR Salary IS NULL

OR Department IS NULL;

**Output**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EmployeeID | FirstName | LastName | Salary | Department |
| 101 | NULL | Smith | 50000 | HR |
| 102 | jane | NULL | NULL | Engineering |

**Replace Missing Values Using COALESCE or NVL**

SELECT

EmployeeID,

COALESCE(FirstName, 'Unknown') AS FirstName,

COALESCE(LastName, 'Unknown') AS LastName,

COALESCE(Salary, (SELECT AVG(Salary) FROM Employees WHERE Salary IS NOT NULL)) AS Salary,

COALESCE(Department, 'Not Assigned') AS Department

FROM Employees;

**Output**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | EmployeeID |  |  | | --- | |  | | FirstName | LastName | Salary | Department |
| 101 | Unknown | Smith | 50000 | HR |
| 102 | Jane | Unknown | 55000 | Engineering |

**Update Missing Salaries or Attributes with Calculated Estimates**

UPDATE

Employees

SET

Salary = (SELECT AVG(Salary) FROM Employees WHERE Salary IS NOT NULL)

WHERE

Salary IS NULL;

UPDATE

Employees

SET

FirstName = 'Unknown'

WHERE

FirstName IS NULL;

UPDATE

Employees

SET

LastName = 'Unknown'

WHERE

LastName IS NULL;

UPDATE

Employees

SET

Department = 'Not Assigned'

WHERE

Department IS NULL;

1. Sales Trend Analysis: Analyze and visualize trends in total sales over time for different businesses.

• Plot total sales trends by month and year.

• Highlight seasonal patterns or long-term growth.

• Trendline charts, seasonal sales predictions.

**Plot Total Sales Trends by Month and Year**

SELECT

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(SalesAmount) AS TotalSales

FROM

Sales

GROUP BY

YEAR(SaleDate), MONTH(SaleDate)

ORDER BY

SaleYear, SaleMonth;

**Output**

|  |  |  |
| --- | --- | --- |
| SaleYear | SaleMonth | TotaSale |
| 2023 | 1 | 5000 |
| 2023 | 2 | 6200 |
| 2023 | 3 | 7000 |
| 2024 | 1 | 5500 |

**Highlight Seasonal Patterns or Long-Term Growth**

SELECT

MONTH(SaleDate) AS SaleMonth,

AVG(SalesAmount) AS AvgMonthlySales

FROM

Sales

GROUP BY

MONTH(SaleDate)

ORDER BY

SaleMonth;

**Output**

|  |  |
| --- | --- |
| SaleMonth | AvgMonthSales |
| 1 | 5500 |
| 2 | 6200 |
| 3 | 7100 |

**Add a Trendline and Seasonal Predictions**

[Grab your reader’s attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.holtwinters import ExponentialSmoothing

data={'YearMonth': ['2023-01', '2023-02', '2023-03', '2024-01'], 'TotalSales': [5000, 6200, 7000, 5500]}

df = pd.DataFrame(data)

df['YearMonth'] = pd.to\_datetime(df['YearMonth'])

model = ExponentialSmoothing(df['TotalSales'], seasonal='add', seasonal\_periods=3).fit() df['Forecast'] = model.fittedvalues

plt.figure(figsize=(10, 6))

plt.plot(df['YearMonth'], df['TotalSales'], label='Actual Sales')

plt.plot(df['YearMonth'], df['Forecast'], label='Trendline (Forecast)', linestyle='--')

plt.title('Sales Trend Analysis')

plt.xlabel('Time')

plt.ylabel('Sales Amount')

plt.legend()

plt.show()

INSERT INTO PredictedSales (SaleDate, PredictedSalesAmount)

VALUES ('2024-02-01', 5800),

('2024-03-01', 6300),

('2024-04-01', 7000);

1. Gender-Based Clothing Sales Analysis: Compare sales between men’s and women’s clothing stores.

• Calculate yearly and cumulative sales differences.

• Identify which category contributes more to revenue growth.

• Line charts showing cumulative and relative sales differences.

**Calculate Yearly Sales Totals**

SELECT

YEAR(SaleDate) AS SaleYear,

SUM(CASE WHEN ProductCategory = 'Men' THEN SalesAmount ELSE 0 END) AS MenSales,

SUM(CASE WHEN ProductCategory = 'Women' THEN SalesAmount ELSE 0 END) AS WomenSales,

SUM(CASE WHEN ProductCategory = 'Men' THEN SalesAmount ELSE 0 END) –

SUM(CASE WHEN ProductCategory = 'Women' THEN SalesAmount ELSE 0 END) AS YearlyDifference

FROM Sales

WHERE ProductCategory IN ('Men', 'Women')

GROUP BY YEAR(SaleDate)

ORDER BY SaleYear;

**output**

|  |  |  |  |
| --- | --- | --- | --- |
| **SaleYear** | **MenSales** | **WomenSales** | **YearlyDiffrence** |
| 2022 | 50000 | 40000 | 10000 |
| 2023 | 60000 | 70000 | -10000 |

**Calculate Cumulative Sales**

**output**

WITH YearlySales AS (

SELECT

YEAR(SaleDate) AS SaleYear,

SUM(CASE WHEN ProductCategory = 'Men' THEN SalesAmount ELSE 0 END) AS MenSales,

SUM(CASE WHEN ProductCategory = 'Women' THEN SalesAmount ELSE 0 END) AS WomenSales

FROM Sales

WHERE ProductCategory IN ('Men', 'Women')

GROUP BY YEAR(SaleDate) )

SELECT

SaleYear,

SUM(MenSales) OVER (ORDER BY SaleYear) AS CumulativeMenSales,

SUM(WomenSales) OVER (ORDER BY SaleYear) AS CumulativeWomenSales,

SUM(MenSales - WomenSales) OVER (ORDER BY SaleYear) AS CumulativeDifference

FROM YearlySales;

|  |  |  |  |
| --- | --- | --- | --- |
| SaleYear | CumulativeMenSales | CumulativeWomenSales | CumulativeDiffrence |
| 2022 | 50000 | 40000 | 10000 |
| 2023 | 110000 | 110000 | 0 |

**Revenue Contribution by Category**

SELECT

ProductCategory,

SUM(SalesAmount) AS TotalSales,

(SUM(SalesAmount) \* 100.0 /

(SELECT SUM(SalesAmount) FROM Sales WHERE ProductCategory IN ('Men', 'Women'))) AS ContributionPercentage

FROM Sales

WHERE ProductCategory IN ('Men', 'Women')

GROUP BY ProductCategory

ORDER BY TotalSales DESC;

|  |  |  |
| --- | --- | --- |
| ProductCatagory | TotalSales | ContributionPercentage |
| Women | 150000 | 57.69 |
| Men | 110000 | 42.31 |

**Relative Sales Differences**

SELECT

YEAR(SaleDate) AS SaleYear,

SUM(CASE WHEN ProductCategory = 'Men' THEN SalesAmount ELSE 0 END) AS MenSales,

SUM(CASE WHEN ProductCategory = 'Women' THEN SalesAmount ELSE 0 END) AS WomenSales,

(SUM(CASE WHEN ProductCategory = 'Men' THEN SalesAmount ELSE 0 END) –

SUM(CASE WHEN ProductCategory = 'Women' THEN SalesAmount ELSE 0 END)) AS RelativeDifference

FROM sales

WHERE ProductCategory IN ('Men', 'Women')

GROUP BY YEAR(SaleDate)

ORDER BY SaleYear;

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| SaleYear | MenSale | WomenSales | ReletiveDiffrence |
| 2022 | 50000 | 40000 | 10000 |
| 2023 | 60000 | 70000 | -10000 |

11 .Rolling Window Sales Analysis: Calculate moving averages for sales to smooth out short-term fluctuations.

• Compute rolling averages over 3, 6, and 12 months.

• Visualize moving averages for smoother trendlines.

• Smoothed trendlines showing rolling averages.

**Compute Rolling Averages Over 3, 6, and 12 Months**

SELECT

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(SalesAmount) AS MonthlySales,

AVG(SUM(SalesAmount)) OVER (ORDER BY YEAR(SaleDate), MONTH(SaleDate) ROWS 2 PRECEDING) AS Rolling3MonthAvg,

AVG(SUM(SalesAmount)) OVER (ORDER BY YEAR(SaleDate), MONTH(SaleDate) ROWS 5 PRECEDING) AS Rolling6MonthAvg,

AVG(SUM(SalesAmount)) OVER (ORDER BY YEAR(SaleDate), MONTH(SaleDate) ROWS 11 PRECEDING) AS Rolling12MonthAvg

FROM Sales

GROUP BY YEAR(SaleDate), MONTH(SaleDate)

ORDER BY SaleYear, SaleMonth;

**Output**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SaleYear | SaleMonth | MonthSales | Rolling3MonthAvg | Rolling6MonthAvg | Rolling12MonthAvg |
| 2023 | 1 | 10000 | 10000.0 | 10000.0 | 10000.0 |
| 2023 | 2 | 12000 | 11000.0 | 11000.0 | 11000.0 |
| 2023 | 3 | 15000 | 12333.33 | 12333.33 | 12333.33 |
| 2023 | 4 | 18000 | 15000.0 | 13750.0 | 13750.0 |
| 2023 | 5 | 20000 | 17666.67 | 15833.33 | 15833.3 |

12.Year-to-Date (YTD) Sales Performance: Track cumulative sales progress throughout the year.

• Calculate YTD sales for each month.

• Compare YTD sales across different years.

• Cumulative sales plots.

**Calculate YTD Sales for Each Month**

SELECT

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(SalesAmount) AS MonthlySales,

SUM(SUM(SalesAmount)) OVER (PARTITION BY YEAR(SaleDate) ORDER BY MONTH(SaleDate)) AS YTDSales

FROM Sales

GROUP BY YEAR(SaleDate), MONTH(SaleDate)

ORDER BY SaleYear, SaleMonth;

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| SaleYear | SaleMonth | MonthSales | YTDSales |
| 2023 | 1 | 15000 | 15000 |
| 2023 | 2 | 12000 | 27000 |
| 2023 | 3 | 180001 | 45000 |

**Compare YTD Sales Across Different Years**

SELECT

YEAR(SaleDate) AS SaleYear,

SUM(SalesAmount) AS TotalYTD

FROM Sales

WHERE MONTH(SaleDate) <= 12

GROUP BY YEAR(SaleDate)

ORDER BY SaleYear;

**Output**

|  |  |
| --- | --- |
| **SaleYear** | **TotalYTD** |
| **2002** | **150000** |
| **2023** | **175000** |
| **2024** | **120000** |

**Cumulative Sales Plot (Visualization Suggestion)**

SELECT

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(SalesAmount) AS CumulativeSales

FROM

Sales

GROUP BY

YEAR(SaleDate), MONTH(SaleDate)

ORDER BY

SaleYear, SaleMonth;

Output

|  |  |  |
| --- | --- | --- |
| SaleYear | SaleMonth | CumulativeSales |
| 2023 | 1 | 15000 |
| 2023 | 2 | 27000 |
| 2023 | 3 | 45000 |

13. Customer Retention Analysis: Analyze purchase retention over 6 months, 1 year, and 2 years.

• Identify how many customers return for repeat purchases.

• what time frames they are most likely to return in.

**Identify How Many Customers Return for Repeat Purchases**

SELECT

CustomerID,

COUNT(DISTINCT OrderID) AS RepeatPurchases

FROM Orders

GROUP BY CustomerID HAVING

COUNT(DISTINCT OrderID) > 1;

Output

|  |  |
| --- | --- |
| CustomerID | RepeatPurchases |
| 101 | 2 |
| 102 | 3 |
| 103 | 2 |

**Identify Time Frames for Repeat Purchases (6 Months)**

SELECT

CustomerID,

MIN(OrderDate) AS FirstPurchaseDate,

MAX(OrderDate) AS LastPurchaseDate,

DATEDIFF(MAX(OrderDate), MIN(OrderDate)) AS DaysBetweenPurchases

FROM

Orders

GROUP BY

CustomerID

HAVING

DATEDIFF(MAX(OrderDate), MIN(OrderDate)) <= 180;

Output

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | FirstPurchaseDate | LastPurchaseDate | DaysBetweenPurchases |
| 101 | 2023-01-01 | |  | | --- | |  |  |  | | --- | | 2023-06-01 | | 150 |
| 102 | 2023-03-01 | 2023-06-01 | 90 |

**Identify Time Frames for Repeat Purchases (1 Year)**

SELECT

CustomerID,

MIN(OrderDate) AS FirstPurchaseDate,

MAX(OrderDate) AS LastPurchaseDate,

DATEDIFF(MAX(OrderDate), MIN(OrderDate)) AS DaysBetweenPurchases FROM Orders GROUP BY CustomerID HAVING

DATEDIFF(MAX(OrderDate), MIN(OrderDate)) <= 365;

**Output**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerID** | |  | | --- | |  |  |  | | --- | | **FirstPurchaseDate** | | **LastPurchaseDate** | **DaysBetweenPurchases** |
| **101** | **2023-01-01** | **2023-12-01** | **335** |
| **103** | **2022-02-01** | |  | | --- | |  |  |  | | --- | | **2023-01-01** | | **365** |

**Identify Time Frames for Repeat Purchases (2 Years)**

SELECT

CustomerID,

MIN(OrderDate) AS FirstPurchaseDate,

MAX(OrderDate) AS LastPurchaseDate,

DATEDIFF(MAX(OrderDate), MIN(OrderDate)) AS DaysBetweenPurchases

FROM

Orders

GROUP BY

CustomerID

HAVING

DATEDIFF(MAX(OrderDate), MIN(OrderDate)) <= 730;

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **FirstPurchaseDate** | **LastPurchaseDate** | **DaysBetweenPurchases** |
| **102** | |  | | --- | |  |  |  | | --- | | **2022-03-01** | | **2024-03-01** | **730** |
| **104** | |  | | --- | |  |  |  | | --- | | **2021-05-01** | | **2023-06-01** | **730** |

**14.** **Employee Tenure and Retention Analysis: Measure employee retention by cohort year of hiring. • Analyze how long employees stay with the organization • Identify trends in employee turnover**

**Analyze How Long Employees Stay with the Organization**

SELECT

EmployeeID,

YEAR(HireDate) AS CohortYear,

DATEDIFF(YEAR, HireDate, COALESCE(EndDate, GETDATE())) AS TenureYears

FROM

Employees;

**Output**

|  |  |  |
| --- | --- | --- |
| **EmployeeID** | **CohortYear** | **TenureYears** |
| **101** | **2015** | **9** |
| **102** | **2017** | **7** |
| **103** | **2020** | **4** |

**Employee Retention by Hiring Cohort**

SELECT

YEAR(HireDate) AS CohortYear,

COUNT(\*) AS TotalHired,

COUNT(CASE WHEN EndDate IS NULL THEN 1 END) AS StillEmployed,

COUNT(CASE WHEN EndDate IS NOT NULL THEN 1 END) AS LeftOrganization,

ROUND(100.0 \* COUNT(CASE WHEN EndDate IS NULL THEN 1 END) / COUNT(\*), 2) AS RetentionRate

FROM

Employees

GROUP BY

YEAR(HireDate)

ORDER BY

CohortYear;

**Output**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CohortYear** | **TotalHired** | **StillEmployed** | **LeftOrganization** | **RetentionRate** |
| **2015** | **20** | **15** | **5** | **75.00%** |
| **2017** | **30** | **20** | **10** | **66.67%** |
| **2020** | **25** | **22** | **3** | **88.00%** |

**Identify Trends in Employee Turnover**

SELECT

YEAR(EndDate) AS TurnoverYear,

COUNT(EmployeeID) AS EmployeesLeft

FROM

Employees

WHERE

EndDate IS NOT NULL

GROUP BY

YEAR(EndDate)

ORDER BY

TurnoverYear;

**Output**

|  |  |
| --- | --- |
| **TurnoverYear** | **EmployeesLeft** |
| **2022** | **10** |
| **2023** | **15** |
| **2024** | **12** |