```sql
/* CAPSTONE PROJECT_1 */

/* Project Title: Adventure Works*/


/* Create a View to combine sales data of 2015, 2016 and 2017  */

USE AdventureWorks;

CREATE VIEW Combined_Sales_Data AS

SELECT * FROM AdventureWorks_Sales_2015
UNION ALL
SELECT * FROM AdventureWorks_Sales_2016
UNION ALL
SELECT * FROM AdventureWorks_Sales_2017



/* Find the to return quantity and amount of each model */

SELECT
    p.ModelName,
    SUM(r.ReturnQuantity) AS TotalReturnQuantity,
    SUM(r.ReturnQuantity * (p.ProductPrice - p.ProductCost)) AS TotalReturnAmount
FROM
    [AdventureWorks].[dbo].[AdventureWorks_Returns] r
JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Products] p ON r.ProductKey =
      p.ProductKey
GROUP BY
    p.ModelName;



/* Find the least selling product category of 2016 */

SELECT
    pc.CategoryName,
    SUM(s.OrderQuantity) AS TotalQuantity
FROM
    [AdventureWorks].[dbo].[AdventureWorks_Sales_2016] s
INNER JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Products] p ON s.ProductKey =
      p.ProductKey
INNER JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Product_Subcategories] ps ON
      p.ProductSubcategoryKey = ps.ProductSubcategoryKey
INNER JOIN
```

```
    [AdventureWorks].[dbo].[AdventureWorks_Product_Categories] pc ON
      ps.ProductCategoryKey = pc.ProductCategoryKey
GROUP BY
    pc.CategoryName
ORDER BY
    TotalQuantity;



 /*Create a view to identify top selling products based onorder quantity */

CREATE VIEW TopSellingProducts AS
WITH RankedProducts AS (
    SELECT
        s.ProductKey,
        p.ProductName,
        s.OrderQuantity,
        RANK() OVER (ORDER BY s.OrderQuantity DESC) AS Ranking
    FROM
        [AdventureWorks].[dbo].[AdventureWorks_Sales_2017] s
    INNER JOIN
        [AdventureWorks].[dbo].[AdventureWorks_Products] p ON s.ProductKey =
          p.ProductKey
)
SELECT
    ProductKey,
    ProductName,
    OrderQuantity,
    Ranking
FROM
    RankedProducts
WHERE
    Ranking = 1; -- Change this number to get top N selling products



/* Show the name of the month and their respective average sales and return
  quantity */

SELECT
    FORMAT(s.[OrderDate], 'MMMM') AS MonthName,
    AVG(s.[OrderQuantity]) AS AverageSales,
    AVG(r.[ReturnQuantity]) AS AverageReturnQuantity
FROM
    [AdventureWorks].[dbo].[AdventureWorks_Sales_2017] s
LEFT JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Returns] r ON s.[OrderDate] = r.
      [ReturnDate]
```

```
GROUP BY
    FORMAT(s.[OrderDate], 'MMMM')
ORDER BY
    FORMAT(s.[OrderDate], 'MM');



/* Show the Total Order quantity of each product where order hasbeen placed from
  United States or Canada and Order the resultsby Total sales of the product*/

SELECT
    p.[ProductName],
    SUM(s.[OrderQuantity]) AS TotalSales
FROM
    [AdventureWorks].[dbo].[AdventureWorks_Sales_2017] s
INNER JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Products] p ON s.[ProductKey] = p.
      [ProductKey]
INNER JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Customers] c ON s.[CustomerKey] = c.
      [CustomerKey]
INNER JOIN
    [AdventureWorks].[dbo].[AdventureWorks_Territories] t ON s.[TerritoryKey] = t.
      [SalesTerritoryKey]
WHERE
    t.[Country] IN ('United States', 'Canada')
GROUP BY
    p.[ProductName]
ORDER BY
    TotalSales DESC;


/* Rank the model's name by their total profitability and partition bytheir colour
  and order by their total order quantity  */


WITH ProfitabilityCTE AS (
    SELECT
        p.[ModelName],
        p.[ProductColor],
        SUM(s.[OrderQuantity] * (p.[ProductPrice] - p.[ProductCost])) AS
          TotalProfit,
        SUM(s.[OrderQuantity]) AS TotalOrderQuantity
    FROM
        [AdventureWorks].[dbo].[AdventureWorks_Sales_2017] s
    INNER JOIN
        [AdventureWorks].[dbo].[AdventureWorks_Products] p ON s.[ProductKey] = p.
          [ProductKey]
    GROUP BY
```

```
        p.[ModelName], p.[ProductColor]
)

SELECT
    [ModelName],
    [ProductColor],
    TotalProfit,
    TotalOrderQuantity,
    RANK() OVER(PARTITION BY [ProductColor] ORDER BY TotalProfit DESC) AS
      ProfitabilityRank
FROM
    ProfitabilityCTE
ORDER BY
    TotalOrderQuantity DESC;


/* Using window functions, Find the Region wise Average profitpartition by Product
  */


SELECT
    Region,
    ProductKey,
    AVG(Profit) OVER (PARTITION BY Region, ProductKey) AS AverageProfit
FROM
    (
        SELECT
            t.Region,
            s.ProductKey,
            (s.OrderQuantity * (p.ProductPrice - p.ProductCost)) AS Profit
        FROM
            [AdventureWorks].[dbo].[AdventureWorks_Sales_2017] s
        INNER JOIN
            [AdventureWorks].[dbo].[AdventureWorks_Products] p ON s.ProductKey =
              p.ProductKey
        INNER JOIN
            [AdventureWorks].[dbo].[AdventureWorks_Territories] t ON s.TerritoryKey
               = t.SalesTerritoryKey
    ) AS Profitability
ORDER BY
    Region, ProductKey;


/* Category and Round of the profit by two decimal places. */


SELECT
    Region,
    CategoryName,
```

```
    ROUND(AverageProfit, 2) AS RoundedAverageProfit
FROM
    (
        SELECT
            t.Region,
            pc.CategoryName,
            (s.OrderQuantity * (p.ProductPrice - p.ProductCost)) AS Profit,
            AVG(s.OrderQuantity * (p.ProductPrice - p.ProductCost)) OVER (PARTITION
               BY t.Region, pc.CategoryName) AS AverageProfit
        FROM
            [AdventureWorks].[dbo].[AdventureWorks_Sales_2017] s
        INNER JOIN
            [AdventureWorks].[dbo].[AdventureWorks_Products] p ON s.ProductKey =
              p.ProductKey
        INNER JOIN
            [AdventureWorks].[dbo].[AdventureWorks_Product_Subcategories] ps ON
              p.ProductSubcategoryKey = ps.ProductSubcategoryKey
        INNER JOIN
            [AdventureWorks].[dbo].[AdventureWorks_Product_Categories] pc ON
              ps.ProductCategoryKey = pc.ProductCategoryKey
        INNER JOIN
            [AdventureWorks].[dbo].[AdventureWorks_Territories] t ON s.TerritoryKey
               = t.SalesTerritoryKey
    ) AS Profitability
ORDER BY
    Region, CategoryName;


/* Create a Procedure to update the salary of the customer withcustomer id and roll
   back if the new salary is less than theexisting salary */

CREATE PROCEDURE UpdateCustomerSalary
    @CustomerID INT,
    @NewSalary DECIMAL(18, 2)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ExistingSalary DECIMAL(18, 2);

    -- Get the existing salary for the given customer ID
    SELECT @ExistingSalary = Salary
    FROM AdventureWorks.dbo.Customers
    WHERE CustomerID = @CustomerID;

    -- Check if the new salary is less than the existing salary
    IF @NewSalary < @ExistingSalary
    BEGIN
        RAISERROR('New salary cannot be less than existing salary. Rolling back
```

```
            transaction.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END

    -- Update the salary
    UPDATE AdventureWorks.dbo.Customers
    SET Salary = @NewSalary
    WHERE CustomerID = @CustomerID;

    SELECT 'Salary updated successfully.' AS Result;
END;
GO


/* Create a Triggers to log the details of the customer into a newtable called
  customer_logs when any existing customer isdeleted from Customers table */

CREATE TRIGGER LogDeletedCustomer
ON AdventureWorks_Customers
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO customer_logs (CustomerID, FirstName, LastName, EmailAddress,
      DeletedDate)
    SELECT CustomerKey, FirstName, LastName, EmailAddress, GETDATE()
    FROM deleted;
END;
GO



  /* CAPSTONE PROJECT_1 THE END THANKYOU VERTOCITY */
```