1. provides information about product's subcategory, product's colour and total sales value; please do not use pivoting here – this will be our base query for the pivot.
2. provides information about total sales value for different colours of different product subcategories; please put colours on columns, products' subcategory names on rows, and total sales value as values; use the query from the previous point.
    a. please prepare a version of this query, where only subcategories from category *Bike* are presented.
3. (*) provides information about average sales subtotal amounts in years and months; please put months on columns, years on rows, and subtotal as values – do this without manually specifying all individual years.

TASK 1A - SOLUTIONS:

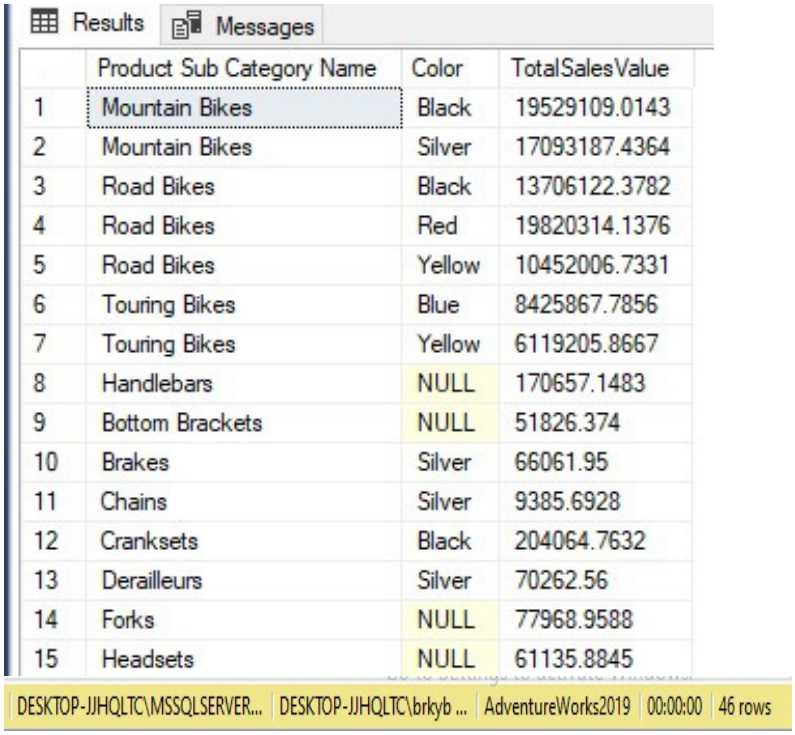
Use this section to provide your solutions, add a screenshot of several first results, and try to briefly comment your approach:

1 SQL

Below provided query is solid version without addition of Pivot table also together with to clearly see if the NULL colour values are existed. As presented there are 46 rows in the result. Because there is more than one rows for each SubCategory name in case more than one colour are existed or no color declined but the Sales value based on SubCategory name does.

```
            SELECT
    PS.Name AS [Product Sub Category Name],
    P.Color,
    SUM(SOD.UnitPrice * SOD.OrderQty) AS TotalSalesValue
FROM
    Production.Product AS P
INNER JOIN
    Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
INNER JOIN
    Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID
GROUP BY
    PS.Name,
       P.Color;
```

Moreover, Data is grouped by product subcategory name and color to enable simple analysis based on these. Below image is the representation of result in SSMS of the created query based on requirements;



Results | Messages

	Product Sub Category Name	Color	TotalSalesValue
1	Mountain Bikes	Black	19529109.0143
2	Mountain Bikes	Silver	17093187.4364
3	Road Bikes	Black	13706122.3782
4	Road Bikes	Red	19820314.1376
5	Road Bikes	Yellow	10452006.7331
6	Touring Bikes	Blue	8425867.7856
7	Touring Bikes	Yellow	6119205.8667
8	Handlebars	NULL	170657.1483
9	Bottom Brackets	NULL	51826.374
10	Brakes	Silver	66061.95
11	Chains	Silver	9385.6928
12	Cranksets	Black	204064.7632
13	Derailleurs	Silver	70262.56
14	Forks	NULL	77968.9588
15	Headsets	NULL	61135.8845

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 46 rows

For instance we are able to see first 2 rows as Mountain Bikes with 2 different color or Road bikes to rest 3 row as Black, Red, Yellow.

---

2 SQL

This query uses COALESCE instead of ISNULL to ensure consistency by returning zero instead of null when there are no sales for a particular color, improving both readability and efficiency.

Thus, in query it's not necessary in WHERE clause to check for color IS NOT NULL.

*Total Sales Value is assumed from SOD.UnitPrice * SOD.OrderQty.*

Since there is only SubCategory name, TotalSalesValue and Color are requested I joined only following tables:

1. *Production.ProductSubcategory*
2. *Production.Product*
3. *Sales.SalesOrderDetail*

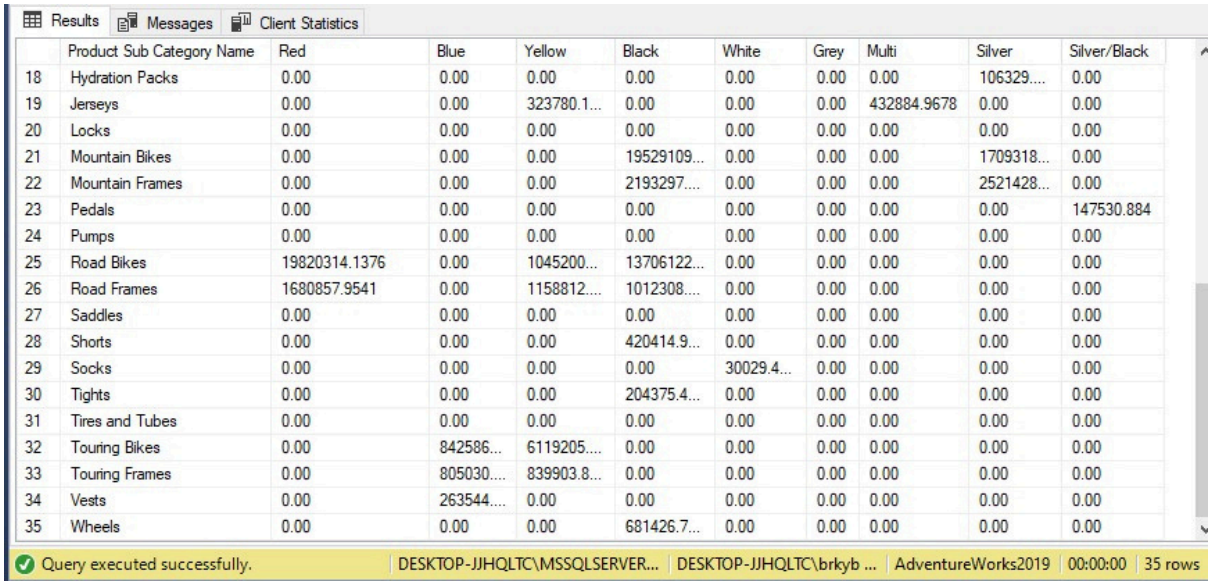
```
SELECT
    [Product Sub Category Name], COALESCE([Red],
    0) AS [Red], COALESCE([Blue], 0) AS [Blue],
    COALESCE([Yellow],     0)     AS     [Yellow],
    COALESCE([Black],      0)      AS      [Black],
    COALESCE([White],      0)      AS      [White],
    COALESCE([Grey],       0)       AS       [Grey],
    COALESCE([Multi],      0)      AS      [Multi],
    COALESCE([Silver],     0)     AS     [Silver],
    COALESCE([Silver/Black], 0) AS [Silver/Black]

FROM
    (SELECT
        PS.Name AS [Product Sub Category Name],
```

```
        P.Color,
        SUM(SOD.UnitPrice * SOD.OrderQty) AS TotalSalesValue
    FROM
        Production.Product AS P
    INNER JOIN
        Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
    INNER JOIN
        Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID
    GROUP BY
        PS.Name,
        P.Color) AS SourceTable
PIVOT
    (SUM(TotalSalesValue)
     FOR Color IN ([Red], [Blue], [Green], [Yellow], [Black], [White],
[Grey],[Multi],[Silver],[Silver/Black])) AS PivotTable;
```

when we want to see sales in different colors for a specific product category, we can used a pivot table. Because It transforms row headers (product categories) into column headers (colors), making the data more readable and informative.

Below image is the representation of result in SSMS of the created query based on requirements;



Results | Messages | Client Statistics

	Product Sub Category Name	Red	Blue	Yellow	Black	White	Grey	Multi	Silver	Silver/Black
18	Hydration Packs	0.00	0.00	0.00	0.00	0.00	0.00	0.00	106329....	0.00
19	Jerseys	0.00	0.00	323780.1...	0.00	0.00	0.00	432884.9678	0.00	0.00
20	Locks	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
21	Mountain Bikes	0.00	0.00	0.00	19529109...	0.00	0.00	0.00	1709318...	0.00
22	Mountain Frames	0.00	0.00	0.00	2193297....	0.00	0.00	0.00	2521428...	0.00
23	Pedals	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	147530.884
24	Pumps	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25	Road Bikes	19820314.1376	0.00	1045200...	13706122...	0.00	0.00	0.00	0.00	0.00
26	Road Frames	1680857.9541	0.00	1158812....	1012308....	0.00	0.00	0.00	0.00	0.00
27	Saddles	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
28	Shorts	0.00	0.00	0.00	420414.9...	0.00	0.00	0.00	0.00	0.00
29	Socks	0.00	0.00	0.00	0.00	30029.4...	0.00	0.00	0.00	0.00
30	Tights	0.00	0.00	0.00	204375.4...	0.00	0.00	0.00	0.00	0.00
31	Tires and Tubes	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
32	Touring Bikes	0.00	842586...	6119205....	0.00	0.00	0.00	0.00	0.00	0.00
33	Touring Frames	0.00	805030....	839903.8...	0.00	0.00	0.00	0.00	0.00	0.00
34	Vests	0.00	263544....	0.00	0.00	0.00	0.00	0.00	0.00	0.00
35	Wheels	0.00	0.00	0.00	681426.7...	0.00	0.00	0.00	0.00	0.00

Query executed successfully. | DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 35 rows

This result provide the rows also together with the NULL color values even if the SubCategory name doesn't include any colorful optin in the database. I provide this query due to clearly see what the database going to provide me together with the all the data.

2a)

IN order to find the Subcategories from Bike category, WHERE clause added to name column from Product.SubCategory table.
Final query is presented below;

```
    SELECT
    [Product Sub Category Name],
    COALESCE([Red], 0) AS [Red],
```

```
    COALESCE([Blue],       0)       AS       [Blue],
    COALESCE([Yellow],     0)       AS     [Yellow],
    COALESCE([Black],      0)       AS      [Black],
    COALESCE([White],      0)       AS      [White],
    COALESCE([Grey],       0)       AS       [Grey],
    COALESCE([Multi],      0)       AS      [Multi],
    COALESCE([Silver],     0)       AS     [Silver],
    COALESCE([Silver/Black], 0) AS [Silver/Black]
FROM
    (SELECT
        PS.Name AS [Product Sub Category Name],
        P.Color,
        SUM(SOD.UnitPrice * SOD.OrderQty) AS TotalSalesValue
    FROM
        Production.Product AS P
    INNER JOIN
        Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
       INNER JOIN
        Production.ProductCategory PC ON PS.ProductCategoryID = PC.ProductCategoryID
    INNER JOIN
        Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID
    WHERE
        PC.Name = 'Bikes'
    GROUP BY
        PS.Name,
        P.Color) AS SourceTable
PIVOT
    (SUM(TotalSalesValue)
     FOR Color IN ([Red], [Blue], [Green], [Yellow], [Black], [White],
[Grey],[Multi],[Silver],[Silver/Black])) AS PivotTable;
```

In order to see the Category names contains bike, I referenced SOD table through PS.ProductCategoryID. Thus 1 more INNER JOIN added to reference.

Below image is the representation of result in SSMS of the created query based on requirements;

Results | Messages

	Product Sub Category Name	Red	Blue	Yellow	Black	White	Grey	Multi	Silver	Silver/Black
1	Mountain Bikes	0.00	0.00	0.00	19529109.0143	0.00	0.00	0.00	17093187.4364	0.00
2	Road Bikes	19820314.1376	0.00	10452006.7331	13706122.3782	0.00	0.00	0.00	0.00	0.00
3	Touring Bikes	0.00	8425867.7856	6119205.8667	0.00	0.00	0.00	0.00	0.00	0.00

After sorted by the SubCategory names filtered by names contains word Bikes, above result is provided.

---

3 SQL

provides information about average sales subtotal amounts in years and months; please put months on columns, years on rows, and subtotal as values – do this without manually specifying all individual years.

```
DECLARE @Columns NVARCHAR(MAX);
DECLARE @Query NVARCHAR(MAX);
-- Dynamically generate column names for months
SET @Columns = '';
SELECT @Columns = CONCAT(@Columns,


                         QUOTENAME(Month), ',')
FROM (
    SELECT DISTINCT MONTH(OrderDate) AS Month
    FROM Sales.SalesOrderHeader
```

```
) AS Months
ORDER BY Month;

-- Remove trailing comma
SET @Columns = LEFT(@Columns, LEN(@Columns) - 1);
-- Generate list of distinct years
DECLARE @Years NVARCHAR(MAX);
SET @Years = '';
SELECT @Years = CONCAT(@Years,

                       QUOTENAME(Year), ',')
FROM (
    SELECT DISTINCT YEAR(OrderDate) AS Year
    FROM Sales.SalesOrderHeader
) AS Years
ORDER BY Year;
-- Remove trailing comma
SET @Years = LEFT(@Years, LEN(@Years) - 1);
-- Dynamic pivot query
SET @Query = '
SELECT *
FROM (

    SELECT YEAR(OrderDate) AS [Year],
           MONTH(OrderDate) AS [Month],
           SubTotal
    FROM Sales.SalesOrderHeader
) AS PivotData
PIVOT (
    AVG(SubTotal) FOR [Month] IN (' + @Columns + ')
) AS PivotTable
ORDER BY [Year];';

-- Execute dynamic query
EXEC sp_executesql @Query;
```

In this task, I was not sure about the solution. However, based on my researchents, I founda way to dynamically generating a pivot table to display average sales subtotal amounts for each month across different years.

- Pivot Query: a dynamic pivot query is constructed using the generated column names for months and years.
The PIVOT function is utilized to pivot the SubTotal values across months, with years as rows and months as columns.
-the dynamic SQL query is executed using sp_executesql to obtain the pivot table with average sales subtotal amounts for each month and year.
This approach eliminates the need to manually specify the all individual years and months.

Query result is shown in below image:

Results | Messages | Client Statistics

	Year	1	2	3	4	5	6	7	8	9	10	11	12
1	2011	NULL	NULL	NULL	NULL	11716.4166	3254.6867	8851.0822	9983.2669	3197.9225	14032.9107	3207.9992	5745.0142
2	2012	11817.343	6737.1091	9788.6455	6076.5828	10493.5249	10511.165	8877.8022	7633.8148	9812.9316	7925.5174	4889.5612	7485.1979
3	2013	5219.6811	7128.9912	7737.1178	5916.5091	7583.233	7066.8555	2813.9964	1863.5908	2530.9373	2436.8969	1574.9549	1988.0422
4	2014	2003.6515	761.8024	3008.5581	849.7276	2225.9124	52.1893	NULL	NULL	NULL	NULL	NULL	NULL

TASK 1B – CASE

Please prepare for each subtask – query, result (or at least its excerpt). Please use the data structures created the previous Lab Assignments. ***Use Case expression*** *– prepare SQL queries which as a result:*

1. provides different product's price categories:
   a. ListPrice < 20.00 – Inexpensive
   b. 20.00 < ListPrice < 75.00 – Regular
   c. 75 < ListPrice < 750.00 – High
   d. 750.00 < ListPrice – Expensive
2. provides information about total volume of product for different price categories and different product categories – use price categories in columns, product categories in rows, and total volume as values (0 - never sold a product from a given category); please use CASE to put years on columns

TASK 1B - SOLUTIONS:

Use this section to provide your solutions, add a screenshot of several first results, and try to briefly comment your approach:

1 SQL

I assumed by created data structure in previous assignment is that view statement created to analyse in Excel.

Below query is edited version based on requirements.

```
SELECT
    P.ProductID,
    P.Name AS ProductName,
    PC.Name AS [Product Category Name],
    PS.Name AS [Product SubCategory Name],
    P.Color,
    P.Weight,
    SOH.Status,
    SOH.SalesOrderID AS [SalesOrderID],
    YEAR(SOH.OrderDate) AS [Year],
    MONTH(SOH.OrderDate) AS [Month],
    DAY(SOH.OrderDate) AS "Day",
    SUM(SOD.OrderQty) AS [Volume],
    SUM(SOD.LineTotal) AS [Sales Amount],
    COUNT(DISTINCT SOD.SalesOrderID) AS [Number of Orders],

       P.ListPrice,
    CASE
    WHEN AVG(P.ListPrice) < 20.00 THEN 'Inexpensive'
    WHEN AVG(P.ListPrice) >= 20.00 AND AVG(P.ListPrice) < 75.00 THEN 'Regular'
    WHEN AVG(P.ListPrice) >= 75.00 AND AVG(P.ListPrice) < 750.00 THEN 'High'
    ELSE 'Expensive'
    END AS PriceCategory
FROM
    Production.Product AS P
INNER JOIN
    Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
INNER JOIN
    Production.ProductCategory AS PC ON PS.ProductCategoryID = PC.ProductCategoryID
INNER JOIN
    Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID
INNER JOIN
    Sales.SalesOrderHeader AS SOH ON SOH.SalesOrderID = SOD.SalesOrderID
GROUP BY
```

```
P.ProductID,    P.Name,    PC.Name,    PS.Name,
P.Color,  P.Weight,  P.ListPrice,  --  Include
ListPrice     column     here     SOH.Status,
SOH.SalesOrderID,          YEAR(SOH.OrderDate),
MONTH(SOH.OrderDate),
DAY(SOH.OrderDate);
```

At the above edited version of previous assignment query I added Case statement and Production.ListPrice column to present it in the result.

Below CASE statement was added…

```
CASE
    WHEN AVG(P.ListPrice) < 20.00 THEN 'Inexpensive'
    WHEN AVG(P.ListPrice) >= 20.00 AND AVG(P.ListPrice) < 75.00 THEN 'Regular'
    WHEN AVG(P.ListPrice) >= 75.00 AND AVG(P.ListPrice) < 750.00 THEN 'High'
    ELSE 'Expensive'
    END AS PriceCategory
```

This case statement takes the average list price of products and assigns them to different price categories based on previously defined price ranges. If the average list price doesn't fall into any of the specified ranges than it is categorizing as 'Expensive'. if > 750.00.

Query result is shown in below image:

Results | Messages | Client Statistics

	ProductID	ProductName	Product Category Name	Product SubCategory Name	Color	Weight	Status	SalesOrderID	Year	Month	Day	Volume	Sales Amount	Number of Orders	ListPrice	PriceCategory
1	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43677	2011	5	31	1	20.186500	1	34.99	Regular
2	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43680	2011	5	31	4	80.746000	1	34.99	Regular
3	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43683	2011	5	31	3	60.559500	1	34.99	Regular
4	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43693	2011	5	31	2	40.373000	1	34.99	Regular
5	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43849	2011	7	1	1	20.186500	1	34.99	Regular
6	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43872	2011	7	1	4	80.746000	1	34.99	Regular
7	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43875	2011	7	1	2	40.373000	1	34.99	Regular
8	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43901	2011	7	1	2	40.373000	1	34.99	Regular
9	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	43907	2011	7	1	2	40.373000	1	34.99	Regular
10	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	44086	2011	8	1	5	100.932500	1	34.99	Regular
11	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	44109	2011	8	1	4	80.746000	1	34.99	Regular
12	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	44115	2011	8	1	3	60.559500	1	34.99	Regular
13	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	44128	2011	8	1	1	20.186500	1	34.99	Regular
14	707	Sport-100 Helmet, Red	Accessories	Helmets	Red	NULL	5	44131	2011	8	1	2	40.373000	1	34.99	Regular

Query executed successfully. DESKTOP-JJHQLTC\MSSQLSERVER... DESKTOP-JJHQLTC\

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:02 | 121,317 rows

The requirements can bee seen as last 2 column in the result image above.

List price and then Price Category based on the price amount is presented weather as inexpensive, Regular, High, Expensive

2 SQL

Final query is presented below;

```
SELECT
    [Product Category Name],
    [Year],
    COALESCE([Inexpensive], 0) AS [Inexpensive],
    COALESCE([Regular], 0) AS [Regular],
    COALESCE([High], 0) AS [High],
    COALESCE([Expensive], 0) AS [Expensive]
FROM (
    SELECT
        PC.Name AS [Product Category Name],
        YEAR(SOH.OrderDate) AS [Year],
        CASE
        WHEN P.ListPrice < 20.00 THEN 'Inexpensive'
        WHEN P.ListPrice >= 20.00 AND P.ListPrice < 75.00 THEN 'Regular'
        WHEN P.ListPrice >= 75.00 AND P.ListPrice < 750.00 THEN 'High'
        ELSE 'Expensive'
        END AS [Price Category],
        SOD.OrderQty AS [Total Volume]
    FROM
        Production.Product AS P
        INNER JOIN Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID
        INNER JOIN Sales.SalesOrderHeader AS SOH ON SOD.SalesOrderID =
SOH.SalesOrderID
        INNER JOIN Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
        INNER JOIN Production.ProductCategory AS PC ON PS.ProductCategoryID =
PC.ProductCategoryID
) AS SourceData
PIVOT (
    SUM([Total Volume])
    FOR [Price Category] IN ([Inexpensive], [Regular], [High], [Expensive])
) AS PivotTable
ORDER BY
    [Product Category Name],
    [Year];
```

the result set is formatted to display the product category name, year, and the total sales volume for each price category (Inexpensive, Regular, High, and Expensive) within each product category. Any missing sales volumes for a particular price category in a product category are filled with 0 using the COALESCE function. The result set is ordered by product category name and year.

This presentation is allows for easy comparison of product sales volumes across different price categories and product categories, facilitating analysis.

Query result is shown in below image:

Results | Messages | Client Statistics

	Product Category Name	Year	Inexpensive	Regular	High	Expensive
1	Accessories	2011	0	1032	0	0
2	Accessories	2012	793	4957	0	0
3	Accessories	2013	13906	15974	2273	0
4	Accessories	2014	11126	10729	1142	0
5	Bikes	2011	0	0	0	7963
6	Bikes	2012	0	0	0	28494
7	Bikes	2013	0	0	9111	28637
8	Bikes	2014	0	0	5039	11024
9	Clothing	2011	1219	1027	0	0
10	Clothing	2012	2571	14476	2181	0
11	Clothing	2013	6492	29752	936	0
12	Clothing	2014	3246	11762	8	0
13	Components	2011	0	0	1113	534
14	Components	2012	0	1506	11240	2361
15	Components	2013	0	5460	15200	4047
16	Components	2014	0	1965	4301	1317

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 16 rows

1. provides total sales amount for different product categories along with a total value of sales for all categories.
2. provides total sales amount for each product category and color of products (include also products without specified color), for each color, total for each category and total sales amount:
    a. Please use grouping function to identify the total sales amount from the sales amount for products without specified color:
        i. https://docs.microsoft.com/en-us/sql/t-sql/functions/grouping-transact-sql
3. provides total sales amount for different products (use product's name) in different product categories and subcategories – please provide sales summaries for each category and subcategory and a total value.

1 SQL

Final query is presented below;

```
SELECT
    COALESCE(PC.Name, 'Products with a null category') AS [Product Category],
    SUM(SOD.UnitPrice * SOD.OrderQty) AS [Total Sales Amount]
FROM
    Sales.SalesOrderDetail AS SOD
INNER JOIN
    Production.Product AS P ON SOD.ProductID = P.ProductID
```

```
INNER JOIN
    Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
INNER JOIN
    Production.ProductCategory AS PC ON PS.ProductCategoryID = PC.ProductCategoryID
GROUP BY
    ROLLUP (PC.Name);
```

Results | Messages | Client Statistics

	Product Category	Total Sales Amount
1	Accessories	1278760.9125
2	Bikes	95145813.3519
3	Clothing	2141507.0245
4	Components	11807808.0245
5	Products with a null category	110373889.3134

- The query retrieves data from the SalesOrderDetail, Product, ProductSubcategory, and ProductCategory tables.

- It joins these tables to get relevant information about sales orders, products, and their categories.

- The data is then grouped by the product category (PC.Name). The ROLLUP function is used to include an additional row that provides the total sales amount for all categories.

- By `COALESCE(PC.Name, 'Products with a null category') AS [Product Category],` function is used to replace any NULL values in the product category with the string 'Products with a null category', ensuring that all categories are accounted for in the results.

---

2 SQL

...

Final query is presented below;

```
SELECT

    COALESCE(PC.Name, 'TotalSalesAmount of AllCategories') AS [Product Category],
    COALESCE(P.Color, 'No Color Specified') AS [Product Color],
    SUM(SOD.UnitPrice * SOD.OrderQty) AS [Total Sales Amount]
FROM
    Production.Product AS P
INNER JOIN
    Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
PS.ProductSubcategoryID
INNER JOIN
    Production.ProductCategory AS PC ON PS.ProductCategoryID = PC.ProductCategoryID
LEFT JOIN
      Sales.SalesOrderDetail SOD ON P.ProductID = sod.ProductID
GROUP BY
    PC.Name, P.Color WITH ROLLUP;
```

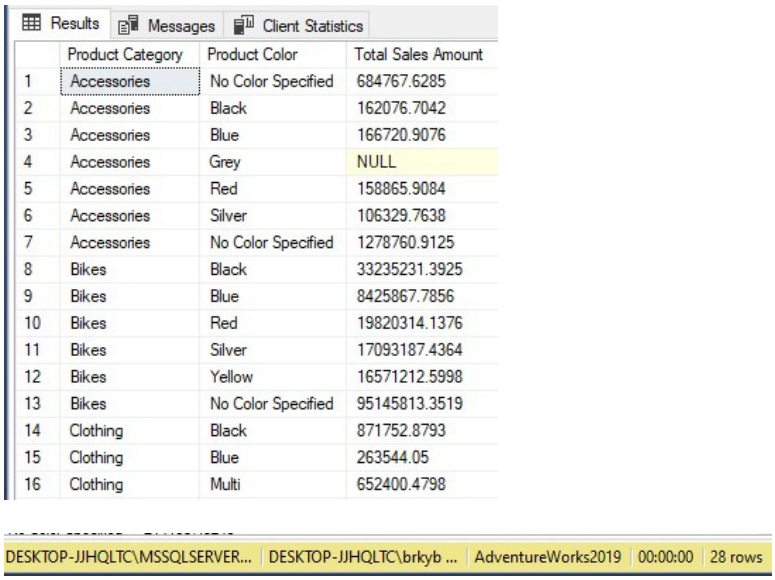
- The query joins these tables to get relevant information about products, their categories, and sales details. -> Product, ProductSubcategory, ProductCategory, and SalesOrderDetail

- The data is grouped by product category (PC.Name) and product color (P.Color), using the GROUP BY clause.

Data Warehouses - Report 3

- The COALESCE function is used to replace any NULL values in the product category and color columns with appropriate placeholder values ('TotalSalesAmount of AllCategories' for product category and 'No Color Specified' for product color).

- The WITH ROLLUP modifier in the GROUP BY clause is used to include additional rows that provide subtotal and total values for each category and color, as well as an overall total sales amount.
Query result is shown in below image:



Results | Messages | Client Statistics

	Product Category	Product Color	Total Sales Amount
1	Accessories	No Color Specified	684767.6285
2	Accessories	Black	162076.7042
3	Accessories	Blue	166720.9076
4	Accessories	Grey	NULL
5	Accessories	Red	158865.9084
6	Accessories	Silver	106329.7638
7	Accessories	No Color Specified	1278760.9125
8	Bikes	Black	33235231.3925
9	Bikes	Blue	8425867.7856
10	Bikes	Red	19820314.1376
11	Bikes	Silver	17093187.4364
12	Bikes	Yellow	16571212.5998
13	Bikes	No Color Specified	95145813.3519
14	Clothing	Black	871752.8793
15	Clothing	Blue	263544.05
16	Clothing	Multi	652400.4798

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 28 rows

2A)

Final query is presented below;

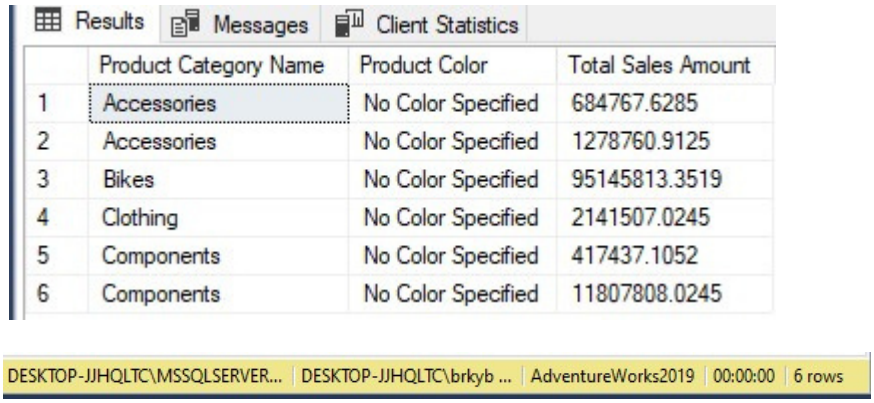
```
SELECT
    pc.Name AS [Product Category Name],
    COALESCE(p.Color, 'No Color Specified') AS [Product Color],
    SUM(sod.UnitPrice * sod.OrderQty) AS [Total Sales Amount]
FROM
    Production.Product AS p
    INNER JOIN Production.ProductSubcategory AS ps ON p.ProductSubcategoryID =
ps.ProductSubcategoryID
    INNER JOIN Production.ProductCategory AS pc ON ps.ProductCategoryID =
pc.ProductCategoryID
    LEFT JOIN Sales.SalesOrderDetail AS sod ON p.ProductID = sod.ProductID
GROUP BY
    GROUPING SETS (
        (PC.Name, P.Color),
        (PC.Name)
    )
HAVING
    P.Color IS NULL;
```

I used LEFT JOIN to make sure that every product from the Production.Product table appears in the results, whether or not there's a match in the Sales.SalesOrderDetail table. This means that products without any sales history will still show up.

Also, by using a LEFT JOIN, we're able to handle cases where the product color might be missing (NULL). We then use the COALESCE function to replace these missing values with a clear label like 'No Color Specified'. So the LEFT JOIN ensures that all products are included in our analysis, allowing us to thoroughly examine product categories and colors, even if some products haven't been sold before.

the HAVING clause filters the results to include only rows where the product color is NULL, meaning products without specified colors.

Query result is shown in below image:



Results | Messages | Client Statistics

	Product Category Name	Product Color	Total Sales Amount
1	Accessories	No Color Specified	684767.6285
2	Accessories	No Color Specified	1278760.9125
3	Bikes	No Color Specified	95145813.3519
4	Clothing	No Color Specified	2141507.0245
5	Components	No Color Specified	417437.1052
6	Components	No Color Specified	11807808.0245

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 6 rows

---

3 SQL

Final query is presented below;

```
SELECT
    (CASE
                WHEN PS.Name IS NULL THEN PC.Name
                ELSE PS.Name
            END) AS [Product Subcategory Name],
    PC.Name AS [Product Category Name],
    P.Name AS [Product Name],
    SUM(SOD.UnitPrice * SOD.OrderQty) AS [Total Sales Amount]
FROM
    Production.Product AS p
LEFT JOIN
    Production.ProductSubcategory AS ps ON p.ProductSubcategoryID =
ps.ProductSubcategoryID
LEFT JOIN
    Production.ProductCategory AS pc ON ps.ProductCategoryID = pc.ProductCategoryID
INNER JOIN
    Sales.SalesOrderDetail AS sod ON p.ProductID = sod.ProductID
GROUP BY
    GROUPING SETS (
        (PC.Name, P.Name, PS.Name),
        (PC.Name, P.Name),
              (PC.Name)

    );
```

- This query provides total sales amounts for different products, utilizing the product names, across various product categories and subcategories. It enables the aggregation of sales summaries for each category and subcategory, along with a total value, by utilizing GROUP BY and GROUPING SETS. This mechanism allows for grouping products by categories and subcategories and summing up the sales amounts, thereby obtaining sales summaries and total values for each category and subcategory.

Query result is shown in below image:

Results | Messages | Client Statistics

	Product Subcategory Name	Product Category Name	Product Name	Total Sales Amount
1	Bike Stands	Accessories	All-Purpose Bike Stand	39591.00
2	Accessories	Accessories	All-Purpose Bike Stand	39591.00
3	Cleaners	Accessories	Bike Wash - Dissolver	18518.571
4	Accessories	Accessories	Bike Wash - Dissolver	18518.571
5	Locks	Accessories	Cable Lock	16264.00
6	Accessories	Accessories	Cable Lock	16264.00
7	Fenders	Accessories	Fender Set - Mountain	46619.58
8	Accessories	Accessories	Fender Set - Mountain	46619.58
9	Bike Racks	Accessories	Hitch Rack - 4-Bike	239437.20
10	Accessories	Accessories	Hitch Rack - 4-Bike	239437.20
11	Tires and Tubes	Accessories	HL Mountain Tire	48860.00

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 536 rows

TASK 2

TASK 2 - **DETAILS**: *Prepare a report (using proper SQL queries) to assess the yearly performance of individual sales representatives*
working in AdventureWorks company. The key metrices that we are focused on are the total sales and number of orders made by employees. The report should contain data as shown in Table 1.

Table 1. Result structure for task 1

Sales Person	Employee ID	Year	Sub Total	Number of orders
...	...	...	...	...

1. Prepare the report without using windowed functions (OVER clause)
2. Prepare the report using windowed functions (OVER clause)
3. Prepare the report by using CTE, where first you aggregate the sales data to establish yearly performance metrices, and only then attaching the details of the sales person

TASK 2 - SOLUTIONS:

Use this section to provide your solutions, add a screenshot of several first results, and try to briefly comment your approach:

1 SQL WITHOUT USING OVER

Final query is presented below;

```
SELECT

    CONCAT(p.FirstName, ' ', p.LastName) AS [Sales Person Name],
    soh.SalesPersonID AS [Employee ID],
    YEAR(soh.OrderDate) AS [Year],
    SUM(sod.UnitPrice * sod.OrderQty) AS [Sub Total],
    COUNT(DISTINCT soh.SalesOrderID) AS [Number of Orders]
```

```
FROM
    Sales.SalesOrderHeader soh
    INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
    INNER JOIN Person.Person p ON soh.SalesPersonID = p.BusinessEntityID
GROUP BY
    CONCAT(p.FirstName, ' ', p.LastName),
    soh.SalesPersonID,
    YEAR(soh.OrderDate)
ORDER BY
    [Employee ID],
    [Year];
```

- This SQL query is designed to generate a report on the yearly performance of individual sales without using windowed functions like the OVER clause for aggregation.

- GROUP BY clause is used to group the data by the salesperson's name, employee ID, and year. This allows for the aggregation of sales data to be performed separately for each salesperson and year combination.

- ORDER BY clause is applied to the result set to organize the data by employee ID and year. Used to ensure that report is in logical sequence and make it easier to analyse.

Query result is shown in below image:

Results | Messages | Client Statistics

	Sales Person Name	Employee ID	Year	Sub Total	Number of Orders
1	Stephen Jiang	274	2011	28926.2465	4
2	Stephen Jiang	274	2012	470380.7027	22
3	Stephen Jiang	274	2013	433119.2727	14
4	Stephen Jiang	274	2014	179335.7826	8
5	Michael Blythe	275	2011	875831.735	65
6	Michael Blythe	275	2012	3392172.0579	148
7	Michael Blythe	275	2013	3995891.5411	175
8	Michael Blythe	275	2014	1060241.8254	62
9	Linda Mitchell	276	2011	1149715.3253	46
10	Linda Mitchell	276	2012	3868319.3774	151
11	Linda Mitchell	276	2013	4145256.5031	162
12	Linda Mitchell	276	2014	1281334.5927	59

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 58 rows

---

2 SQL WITH USING OVER

Final query is presented below;

```
SELECT
    [Sales Person Name],
    [Employee ID],
    [Year],
    SUM([Sub Total]) OVER (PARTITION BY [Employee ID], [Year]) AS [Sub Total],
    COUNT([Sub Total]) OVER (PARTITION BY [Employee ID], [Year]) AS [Number of Orders]
FROM (
```

```
    SELECT
        CONCAT(p.FirstName, ' ', p.LastName) AS [Sales Person Name],
        soh.SalesPersonID AS [Employee ID],
        YEAR(soh.OrderDate) AS [Year],
        SUM(sod.UnitPrice * sod.OrderQty) AS [Sub Total]
    FROM
        Sales.SalesOrderHeader soh
        INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
        INNER JOIN Person.Person p ON soh.SalesPersonID = p.BusinessEntityID
    GROUP BY
        CONCAT(p.FirstName, ' ', p.LastName),
        soh.SalesPersonID,
        YEAR(soh.OrderDate)
) AS SalesData
ORDER BY
    [Employee ID],
    [Year];
```

- Above query employs window functios with the OVER clause to achieve the same result. It calculates the total sales and number of orders for each salesperson in each year by partitioning the data based on the employee ID and year.

- SUM(), COUNT() functions are applied over the partitions, allowing for the aggregation ofsub-total sales and counting of orders without the need for explicit grouping.

- Using window functions simplifies us to syntax and structure of the query by eliminating the GROUP BY clause and subquer.

Query result is shown in below image:

	Sales Person Name	Employee ID	Year	Total Sales	Number of Orders
1	Stephen Jiang	274	2011	28926.2465	1
2	Stephen Jiang	274	2012	470380.7027	1
3	Stephen Jiang	274	2013	433119.2727	1
4	Stephen Jiang	274	2014	179335.7826	1
5	Michael Blythe	275	2011	875831.735	1
6	Michael Blythe	275	2012	3392172.0579	1
7	Michael Blythe	275	2013	3995891.5411	1
8	Michael Blythe	275	2014	1060241.8254	1
9	Linda Mitchell	276	2011	1149715.3253	1
10	Linda Mitchell	276	2012	3868319.3774	1
11	Linda Mitchell	276	2013	4145256.5031	1
12	Linda Mitchell	276	2014	1281334.5927	1

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 58 rows

3 SQL WITH USING CTE

Final query is presented below;

```
WITH SalesSummary AS (
```

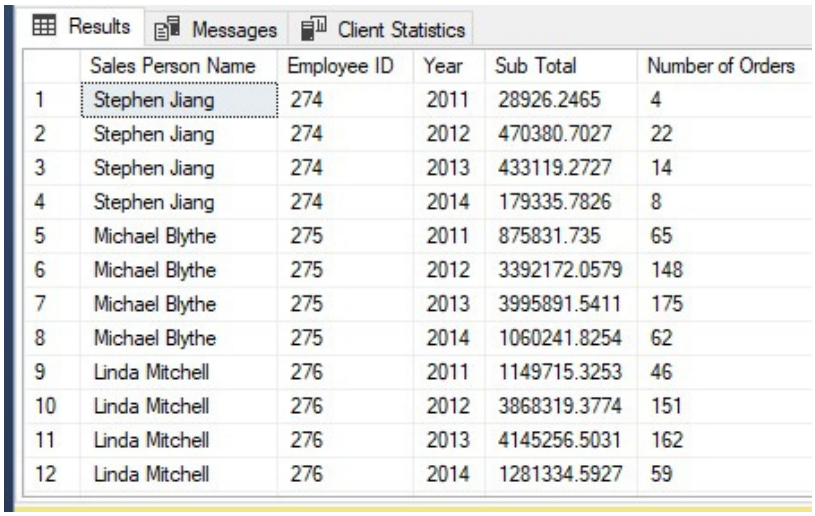
```
    SELECT
        soh.SalesPersonID AS [Employee ID],
        YEAR(soh.OrderDate) AS [Year],
        SUM(sod.UnitPrice * sod.OrderQty) AS [Sub Total],
        COUNT(DISTINCT soh.SalesOrderID) AS [Number of Orders]
    FROM
        Sales.SalesOrderHeader soh
        INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
    GROUP BY
        soh.SalesPersonID,
        YEAR(soh.OrderDate)
),
SalesData AS (
SELECT
        CONCAT(p.FirstName, ' ', p.LastName) AS [Sales Person Name],
        ss.[Employee ID],
        ss.[Year],
        ss.[Sub Total],
        ss.[Number of Orders]
    FROM
        SalesSummary ss
        INNER JOIN Person.Person p ON ss.[Employee ID] = p.BusinessEntityID
)
SELECT
    [Sales Person Name],
    [Employee ID],
    [Year],
    [Sub Total],
    [Number of Orders]
FROM
    SalesData
ORDER BY
    [Employee ID],
    [Year];
```

SalesSummary CTE: This CTE aggregates the sales data to establish yearly performance metrics. It selects the SalesPersonID, Year, Sub Total (total sales), and Number of Orders (count of distinct sales orders) from the SalesOrderHeader and SalesOrderDetail tables.

- SalesData CTE; This CTE attaches the details of the salesperson by joining the aggregated data from the SalesSummary CTE with the Person.Person table. It retrieves the Sales Person Name by concatenating the first name and last name from the Person.Person table using the BusinessEntityID (Employee ID) obtained from the

- MainQuery: selects columns from the SalesData CTE, including Sales Person Name, Employee ID, Year, Sub Total, and Number of Orders. It orders the result by Employee ID and Year.

Query result is shown in below image:



Results | Messages | Client Statistics

	Sales Person Name	Employee ID	Year	Sub Total	Number of Orders
1	Stephen Jiang	274	2011	28926.2465	4
2	Stephen Jiang	274	2012	470380.7027	22
3	Stephen Jiang	274	2013	433119.2727	14
4	Stephen Jiang	274	2014	179335.7826	8
5	Michael Blythe	275	2011	875831.735	65
6	Michael Blythe	275	2012	3392172.0579	148
7	Michael Blythe	275	2013	3995891.5411	175
8	Michael Blythe	275	2014	1060241.8254	62
9	Linda Mitchell	276	2011	1149715.3253	46
10	Linda Mitchell	276	2012	3868319.3774	151
11	Linda Mitchell	276	2013	4145256.5031	162
12	Linda Mitchell	276	2014	1281334.5927	59

DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 58 rows

Above result provides the same result as the previous ones but utilizes CTEs to organize and simplify the process of aggregating sales data and attaching salesperson details. This approach enhances readability and maintainability.

GENERAL CONCLUSIONS:

1 overall*, these tasks have provided practical experience in manipulating and summarizing data using SQL, which is essential for data analysis and reporting tasks. The challenges that I have faced through those tasks was understanding the syntax and logic behind pivot operations, writing complex 'CASE' expressions, and structuring queries to produce the desired output format. However, the tasks were great opportunity to leading to greater proficiency in SQL data manipulation.*

I used three different SQL approaches to assess yearly sales performance at database. We employed simple aggregation, window functions, and Common Table Expressions(CTE) to create reports. Challenges included managing to write the syntaxes and learning the benefit of these methods. CTE's helps simplify the data organization and window functions that are makes the calculations easier. This exercise improved my understanding of SQL business analysis.