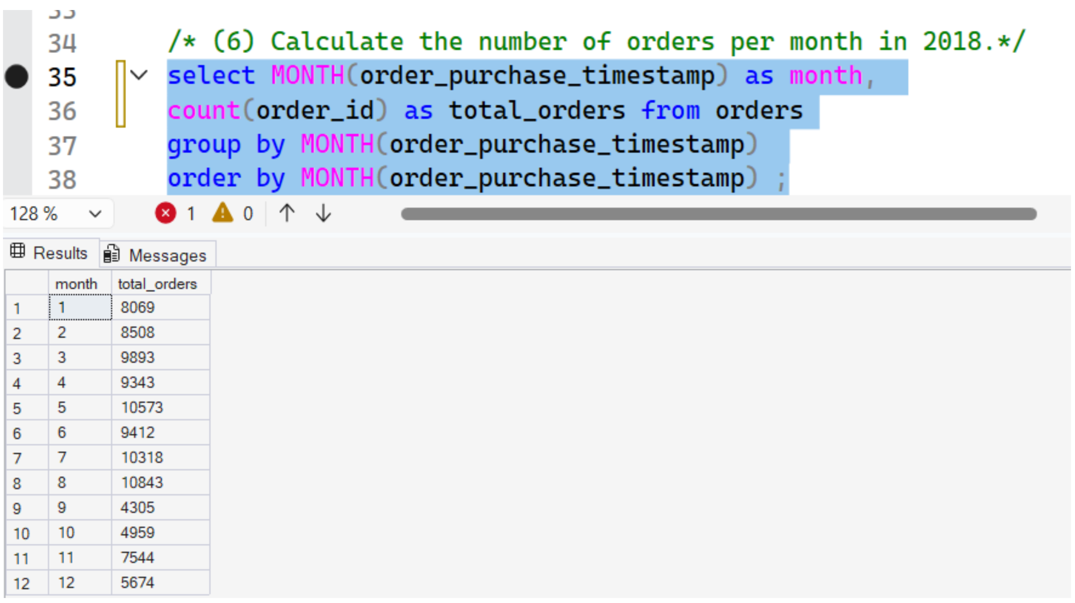
SQL Data Analysis Project

Domain: E-commerce
Prepared by: Abhishek Panwar
Role: Data Analyst (Fresher)
Tools: SQL (CTEs, Window Functions)

Q6: Monthly Order Trends (2018)

Objective: Calculate the number of orders placed per month in 2018.



```
/* (6) Calculate the number of orders per month in 2018.*/
select MONTH(order_purchase_timestamp) as month,
count(order_id) as total_orders from orders
group by MONTH(order_purchase_timestamp)
order by MONTH(order_purchase_timestamp) ;
```

128 %   ✖ 1   ⚠ 0

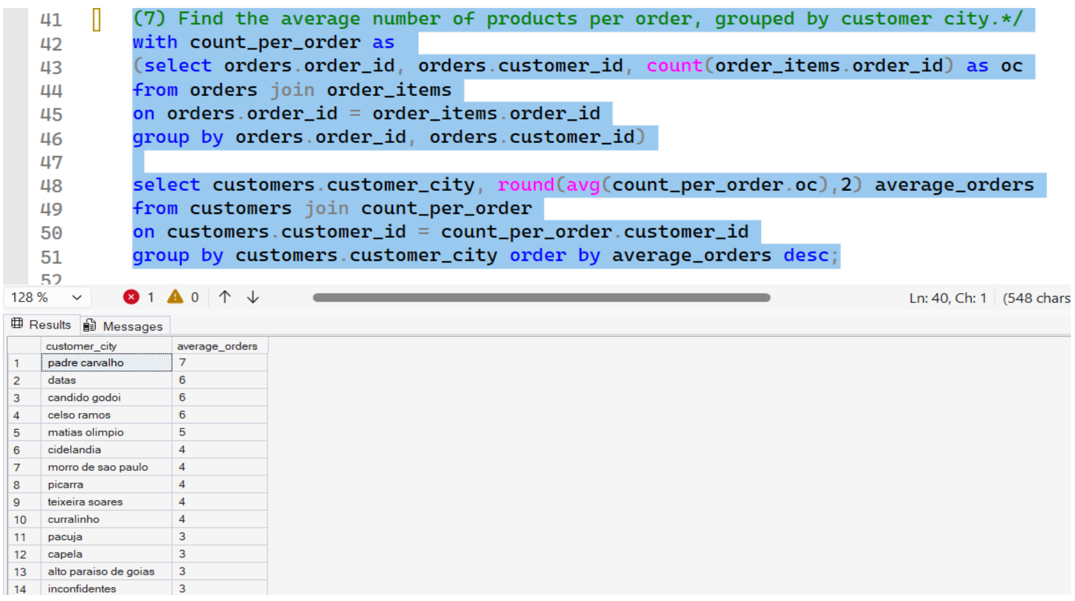
Results | Messages

	month	total_orders
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318
8	8	10843
9	9	4305
10	10	4959
11	11	7544
12	12	5674

```
SELECT MONTH(order_purchase_timestamp) AS month,
       COUNT(order_id) AS total_orders
FROM orders
GROUP BY MONTH(order_purchase_timestamp)
ORDER BY MONTH(order_purchase_timestamp);
```

Q7: Average Products per Order by City

Objective: Find average number of products per order grouped by customer city.



```
(7) Find the average number of products per order, grouped by customer city.*/
with count_per_order as
(select orders.order_id, orders.customer_id, count(order_items.order_id) as oc
from orders join order_items
on orders.order_id = order_items.order_id
group by orders.order_id, orders.customer_id)

select customers.customer_city, round(avg(count_per_order.oc),2) average_orders
from customers join count_per_order
on customers.customer_id = count_per_order.customer_id
group by customers.customer_city order by average_orders desc;
```

128 %   ✖ 1   ⚠ 0   Ln: 40, Ch: 1   (548 chars

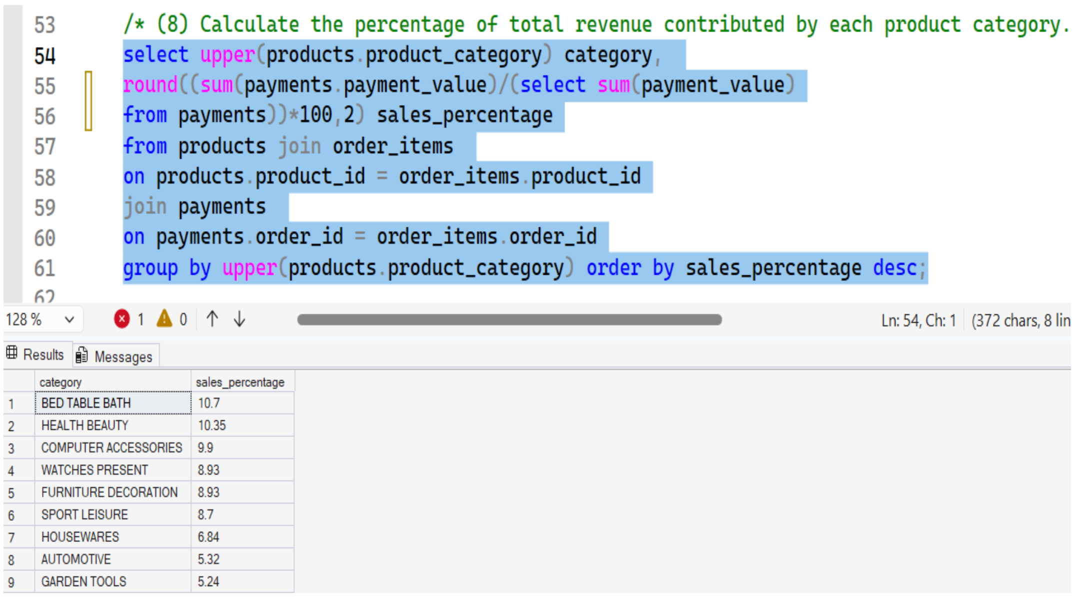
Results | Messages

	customer_city	average_orders
1	padre carvalho	7
2	datas	6
3	candido godoi	6
4	celso ramos	6
5	matias olimpio	5
6	cidelandia	4
7	morro de sao paulo	4
8	picarra	4
9	teixeira soares	4
10	curralinho	4
11	pacuja	3
12	capela	3
13	alto paraiso de goias	3
14	inconfidentes	3

```
WITH count_per_order AS (
  SELECT o.order_id, o.customer_id,
         COUNT(oi.order_id) AS oc
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY o.order_id, o.customer_id
)
SELECT c.customer_city,
       ROUND(AVG(cpo.oc), 2) AS average_orders
FROM customers c
JOIN count_per_order cpo ON c.customer_id = cpo.customer_id
GROUP BY c.customer_city
ORDER BY average_orders DESC;
```

Q8: Revenue Contribution by Category

Objective: Calculate revenue percentage contribution of each product category.



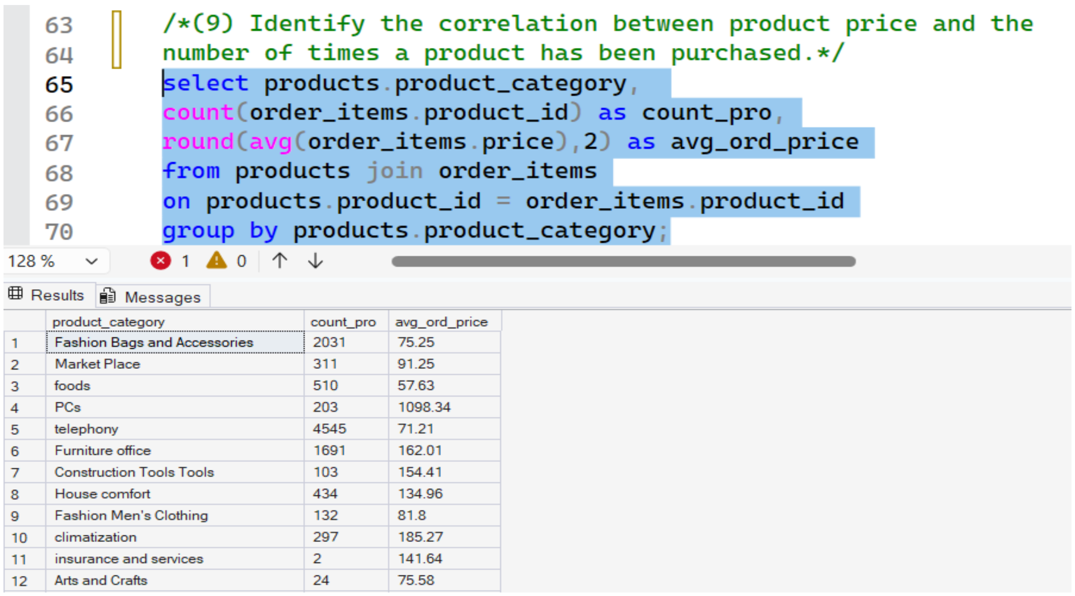
```
/* (8) Calculate the percentage of total revenue contributed by each product category.
select upper(products.product_category) category,
round((sum(payments.payment_value)/(select sum(payment_value)
from payments))*100,2) sales_percentage
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by upper(products.product_category) order by sales_percentage desc;
```

	category	sales_percentage
1	BED TABLE BATH	10.7
2	HEALTH BEAUTY	10.35
3	COMPUTER ACCESSORIES	9.9
4	WATCHES PRESENT	8.93
5	FURNITURE DECORATION	8.93
6	SPORT LEISURE	8.7
7	HOUSEWARES	6.84
8	AUTOMOTIVE	5.32
9	GARDEN TOOLS	5.24

```
SELECT UPPER(p.product_category) AS category,
ROUND((SUM(pay.payment_value) /
      (SELECT SUM(payment_value) FROM payments)) * 100, 2) AS sales_percentage
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
JOIN payments pay ON pay.order_id = oi.order_id
GROUP BY UPPER(p.product_category)
ORDER BY sales_percentage DESC;
```

Q9: Price vs Purchase Frequency

Objective: Analyze correlation between product price and purchase frequency.



```
/*(9) Identify the correlation between product price and the
number of times a product has been purchased.*/
select products.product_category,
count(order_items.product_id) as count_pro,
round(avg(order_items.price),2) as avg_ord_price
from products join order_items
on products.product_id = order_items.product_id
group by products.product_category;
```

128 % ⓧ 1 ⚠ 0

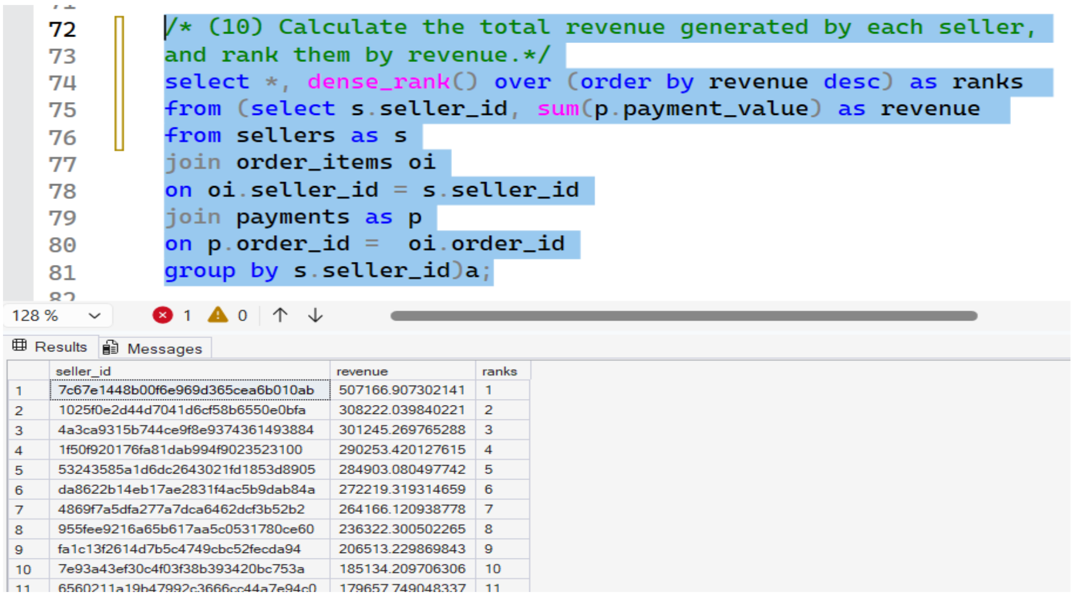
Results | Messages

	product_category	count_pro	avg_ord_price
1	Fashion Bags and Accessories	2031	75.25
2	Market Place	311	91.25
3	foods	510	57.63
4	PCs	203	1098.34
5	telephony	4545	71.21
6	Furniture office	1691	162.01
7	Construction Tools Tools	103	154.41
8	House comfort	434	134.96
9	Fashion Men's Clothing	132	81.8
10	climatization	297	185.27
11	insurance and services	2	141.64
12	Arts and Crafts	24	75.58

```
SELECT p.product_category,
       COUNT(oi.product_id) AS count_pro,
       ROUND(AVG(oi.price), 2) AS avg_ord_price
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_category;
```

Q10: Seller Revenue Ranking

Objective: Rank sellers based on total revenue generated.



```
/* (10) Calculate the total revenue generated by each seller,
and rank them by revenue.*/
select *, dense_rank() over (order by revenue desc) as ranks
from (select s.seller_id, sum(p.payment_value) as revenue
from sellers as s
join order_items oi
on oi.seller_id = s.seller_id
join payments as p
on p.order_id =  oi.order_id
group by s.seller_id)a;
```

128 % ⓧ 1 ⚠ 0

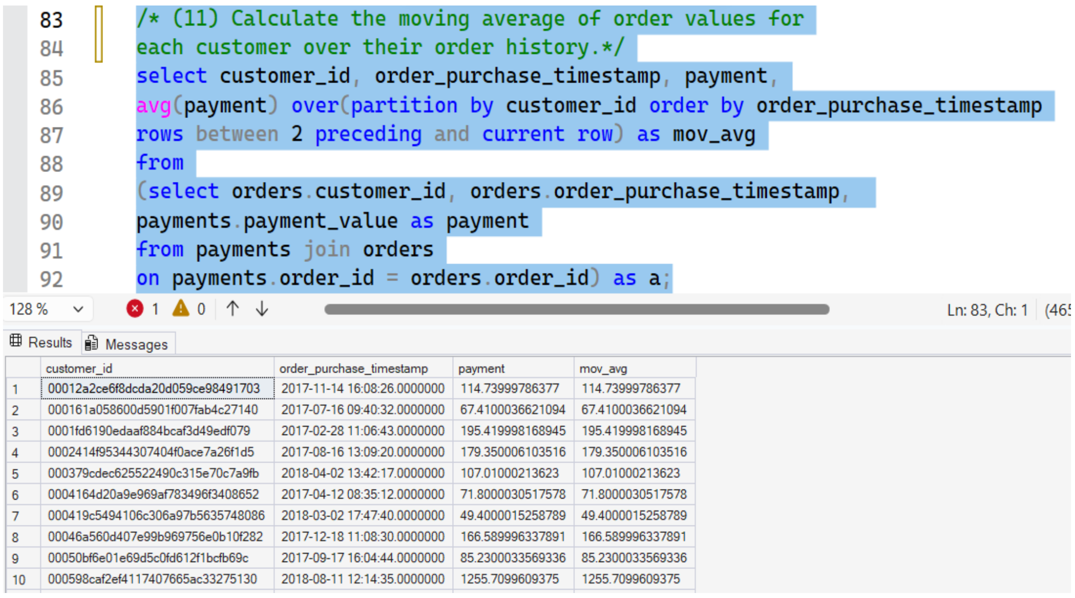
Results | Messages

	seller_id	revenue	ranks
1	7c67e1448b00f6e969d365cea6b010ab	507166.907302141	1
2	1025f0e2d44d7041d6cf58b6550e0bfa	308222.039840221	2
3	4a3ca9315b744ce9f8e9374361493884	301245.269765288	3
4	1f50f920176fa81dab994f9023523100	290253.420127615	4
5	53243585a1d6dc2643021fd1853d8905	284903.080497742	5
6	da8622b14eb17ae2831f4ac5b9dab84a	272219.319314659	6
7	4869f7a5dfa277a7dca6462dcf3b52b2	264166.120938778	7
8	955fee9216a65b617aa5c0531780ce60	236322.300502265	8
9	fa1c13f2614d7b5c4749cbc52fecda94	206513.229869843	9
10	7e93a43ef30c4f03f38b393420bc753a	185134.209706306	10
11	6560211a19b47992c3666cc44a7e94c0	179657.749048337	11

```
SELECT *, DENSE_RANK() OVER (ORDER BY revenue DESC) AS ranks
FROM (
  SELECT s.seller_id, SUM(p.payment_value) AS revenue
  FROM sellers s
  JOIN order_items oi ON s.seller_id = oi.seller_id
  JOIN payments p ON p.order_id = oi.order_id
  GROUP BY s.seller_id
) a;
```

Q11: Moving Average of Order Values

Objective: Calculate moving average of customer order values.



```
/* (11) Calculate the moving average of order values for
each customer over their order history.*/
select customer_id, order_purchase_timestamp, payment,
avg(payment) over(partition by customer_id order by order_purchase_timestamp
rows between 2 preceding and current row) as mov_avg
from
(select orders.customer_id, orders.order_purchase_timestamp,
payments.payment_value as payment
from payments join orders
on payments.order_id = orders.order_id) as a;
```

128 % 1 0 Ln: 83, Ch: 1

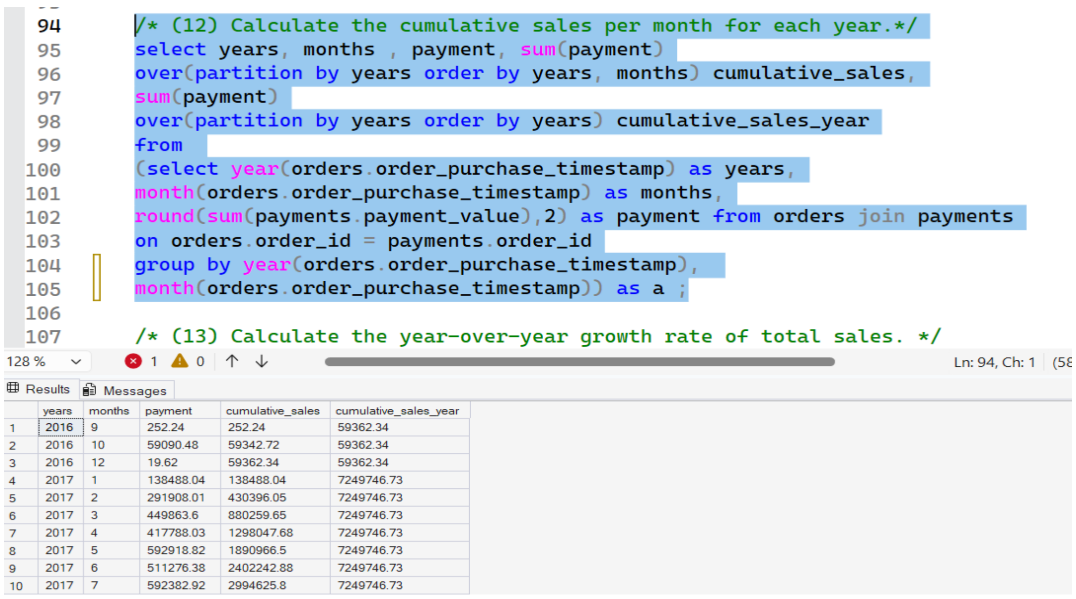
Results | Messages

	customer_id	order_purchase_timestamp	payment	mov_avg
1	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26.0000000	114.73999786377	114.73999786377
2	000161a058600d5901f007fab4c27140	2017-07-16 09:40:32.0000000	67.4100036621094	67.4100036621094
3	0001fd6190edaaf884bcaf3d49edf079	2017-02-28 11:06:43.0000000	195.419998168945	195.419998168945
4	0002414f95344307404f0ace7a26f1d5	2017-08-16 13:09:20.0000000	179.350006103516	179.350006103516
5	000379cdec625522490c315e70c7a9fb	2018-04-02 13:42:17.0000000	107.01000213623	107.01000213623
6	0004164d20a9e969af783496f3408652	2017-04-12 08:35:12.0000000	71.8000030517578	71.8000030517578
7	000419c5494106c306a97b5635748086	2018-03-02 17:47:40.0000000	49.4000015258789	49.4000015258789
8	00046a560d407e99b969756e0b10f282	2017-12-18 11:08:30.0000000	166.589996337891	166.589996337891
9	00050bf6e01e69d5c0fd612f1bcfb69c	2017-09-17 16:04:44.0000000	85.2300033569336	85.2300033569336
10	000598caf2ef4117407665ac33275130	2018-08-11 12:14:35.0000000	1255.7099609375	1255.7099609375

```
SELECT customer_id, order_purchase_timestamp, payment,
AVG(payment) OVER (
  PARTITION BY customer_id
  ORDER BY order_purchase_timestamp
  ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
) AS mov_avg
FROM (
  SELECT o.customer_id, o.order_purchase_timestamp,
         p.payment_value AS payment
  FROM payments p
  JOIN orders o ON p.order_id = o.order_id
) a;
```

Q12: Cumulative Sales per Month

Objective: Calculate cumulative sales per month for each year.



```
/* (12) Calculate the cumulative sales per month for each year.*/
select years, months , payment, sum(payment)
over(partition by years order by years, months) cumulative_sales,
sum(payment)
over(partition by years order by years) cumulative_sales_year
from
(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by year(orders.order_purchase_timestamp),
month(orders.order_purchase_timestamp)) as a ;

/* (13) Calculate the year-over-year growth rate of total sales. */
```

128 % ✕ 1 ⚠ 0 Ln: 94, Ch: 1

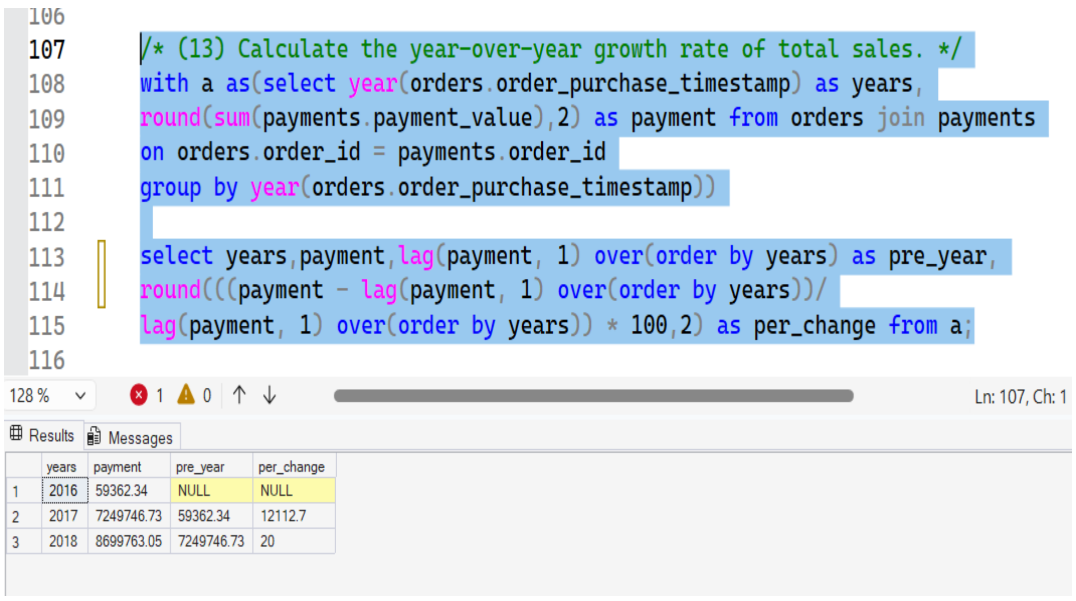
Results Messages

	years	months	payment	cumulative_sales	cumulative_sales_year
1	2016	9	252.24	252.24	59362.34
2	2016	10	59090.48	59342.72	59362.34
3	2016	12	19.62	59362.34	59362.34
4	2017	1	138488.04	138488.04	7249746.73
5	2017	2	291908.01	430396.05	7249746.73
6	2017	3	449863.6	880259.65	7249746.73
7	2017	4	417788.03	1298047.68	7249746.73
8	2017	5	592918.82	1890966.5	7249746.73
9	2017	6	511276.38	2402242.88	7249746.73
10	2017	7	592382.92	2994625.8	7249746.73

```
SELECT years, months, payment,
SUM(payment) OVER(PARTITION BY years ORDER BY years, months) AS cumulative_sales
FROM (
  SELECT YEAR(o.order_purchase_timestamp) AS years,
         MONTH(o.order_purchase_timestamp) AS months,
         ROUND(SUM(p.payment_value), 2) AS payment
  FROM orders o
  JOIN payments p ON o.order_id = p.order_id
  GROUP BY YEAR(o.order_purchase_timestamp),
           MONTH(o.order_purchase_timestamp)
) a;
```

Q13: Year-over-Year Growth

Objective: Calculate YoY sales growth.



```
/* (13) Calculate the year-over-year growth rate of total sales. */
with a as(select year(orders.order_purchase_timestamp) as years,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by year(orders.order_purchase_timestamp))

select years,payment,lag(payment, 1) over(order by years) as pre_year,
round(((payment - lag(payment, 1) over(order by years))/
lag(payment, 1) over(order by years)) * 100,2) as per_change from a;
```

128 % ✕ 1 ⚠ 0 Ln: 107, Ch: 1

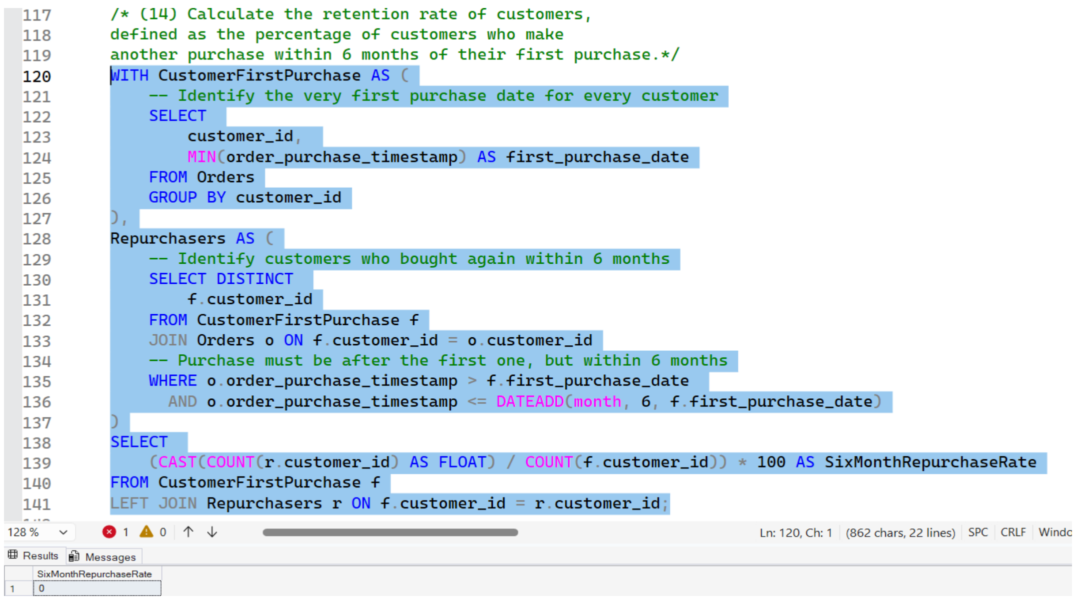
Results Messages

	years	payment	pre_year	per_change
1	2016	59362.34	NULL	NULL
2	2017	7249746.73	59362.34	12112.7
3	2018	8699763.05	7249746.73	20

```
WITH a AS (
SELECT YEAR(o.order_purchase_timestamp) AS years,
ROUND(SUM(p.payment_value), 2) AS payment
FROM orders o
JOIN payments p ON o.order_id = p.order_id
GROUP BY YEAR(o.order_purchase_timestamp)
)
SELECT years, payment,
LAG(payment) OVER(ORDER BY years) AS pre_year,
ROUND(((payment - LAG(payment) OVER(ORDER BY years)) /
LAG(payment) OVER(ORDER BY years)) * 100, 2) AS per_change
FROM a;
```

Q14: Customer Retention Rate (6 Months)

Objective: Calculate customer retention within 6 months of first purchase.

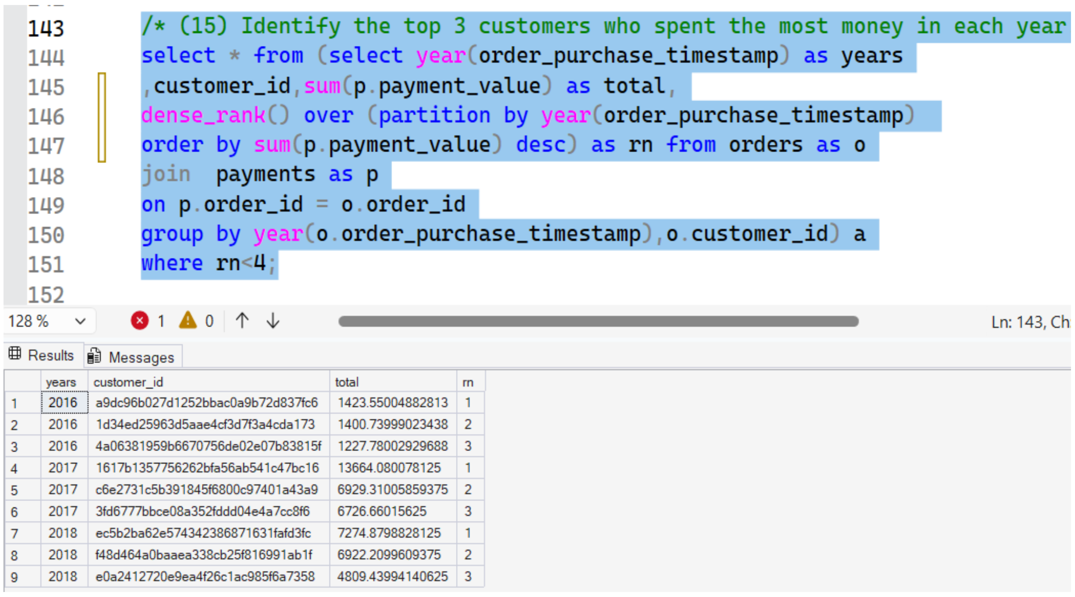


```
/* (14) Calculate the retention rate of customers,
defined as the percentage of customers who make
another purchase within 6 months of their first purchase.*/
WITH CustomerFirstPurchase AS (
    -- Identify the very first purchase date for every customer
    SELECT
        customer_id,
        MIN(order_purchase_timestamp) AS first_purchase_date
    FROM Orders
    GROUP BY customer_id
),
Repurchasers AS (
    -- Identify customers who bought again within 6 months
    SELECT DISTINCT
        f.customer_id
    FROM CustomerFirstPurchase f
    JOIN Orders o ON f.customer_id = o.customer_id
    -- Purchase must be after the first one, but within 6 months
    WHERE o.order_purchase_timestamp > f.first_purchase_date
      AND o.order_purchase_timestamp <= DATEADD(month, 6, f.first_purchase_date)
)
SELECT
    (CAST(COUNT(r.customer_id) AS FLOAT) / COUNT(f.customer_id)) * 100 AS SixMonthRepurchaseRate
FROM CustomerFirstPurchase f
LEFT JOIN Repurchasers r ON f.customer_id = r.customer_id;
```

```
128 %    1  0        Ln: 120, Ch: 1  (862 chars, 22 lines)  SPC  CRLF  Windc
Results  Messages
   SixMonthRepurchaseRate
1  0
```

Q15: Top 3 Customers by Year

Objective: Identify top 3 customers by yearly spending.



```
/* (15) Identify the top 3 customers who spent the most money in each year
select * from (select year(order_purchase_timestamp) as years
,customer_id,sum(p.payment_value) as total,
dense_rank() over (partition by year(order_purchase_timestamp)
order by sum(p.payment_value) desc) as rn from orders as o
join  payments as p
on p.order_id = o.order_id
group by year(o.order_purchase_timestamp),o.customer_id) a
where rn<4;
```

128 % 1 0 Ln: 143, Ch

Results Messages

	years	customer_id	total	rn
1	2016	a9dc96b027d1252bbac0a9b72d837fc6	1423.55004882813	1
2	2016	1d34ed25963d5aae4cf3d7f3a4cda173	1400.73999023438	2
3	2016	4a06381959b6670756de02e07b83815f	1227.78002929688	3
4	2017	1617b1357756262bfa56ab541c47bc16	13664.080078125	1
5	2017	c6e2731c5b391845f6800c97401a43a9	6929.31005859375	2
6	2017	3fd6777bbce08a352fddd04e4a7cc8f6	6726.66015625	3
7	2018	ec5b2ba62e574342386871631fafd3fc	7274.8798828125	1
8	2018	f48d464a0baaea338cb25f816991ab1f	6922.2099609375	2
9	2018	e0a2412720e9ea4f26c1ac985f6a7358	4809.43994140625	3

Conclusion

This project demonstrates advanced SQL skills including window functions, CTEs, ranking, and business-driven analysis suitable for Data Analyst roles.