Retail Analytics Case Study

This case study is about using data analytics to solve problems in a company.

Key Concepts:

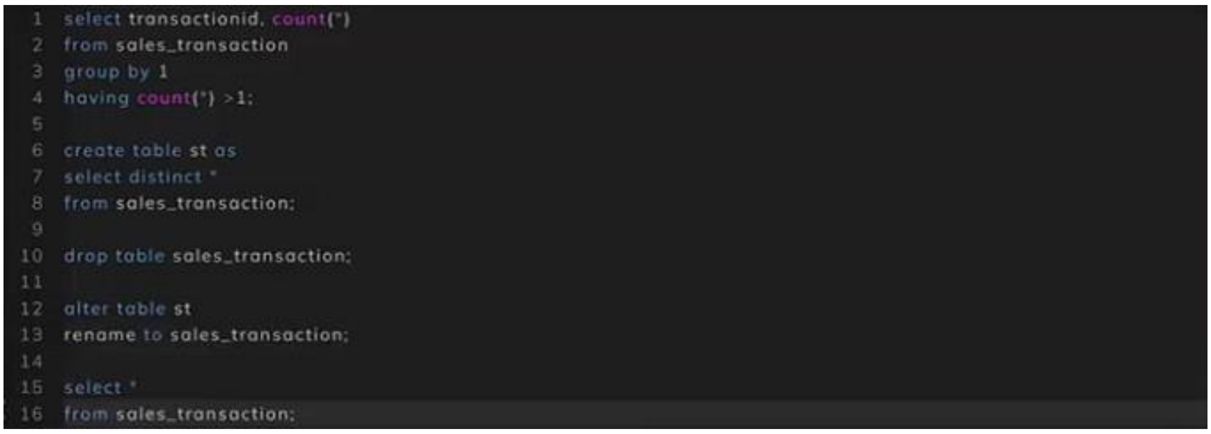
- **Product Sales Analysis:** Finding out which products sell well and which don't.
    - **Example:** If Product A sold 500 units and Product B sold only 10, focus marketing on Product A and consider removing Product B.
- **Customer Segmentation:** Group customers based on how much they buy.
    - **Example:**
    - No Orders: Customer bought 0 items.
    - Low: Bought 1-10 items.
    - Mid: Bought 10-30 items.
    - High Value: Bought more than 30 items.
      This helps target promotions (e.g., special offers for high-value customers).
- **Customer Behavior Analysis:** Study patterns like repeat purchases and loyalty.
    - **Example:** If a customer buys every month, offer them a loyalty discount to keep them coming back.
- **Data Cleaning:** Use SQL to clean data and explore trends.
    - **Example:** Remove duplicate sales records, check for missing values, and calculate total sales per product.

Datasets Used:

- **Sales Transaction (who bought what, when, and how much)**
- **Customer Profiles (customer details)**
- **Product Inventory (product details and stock)**

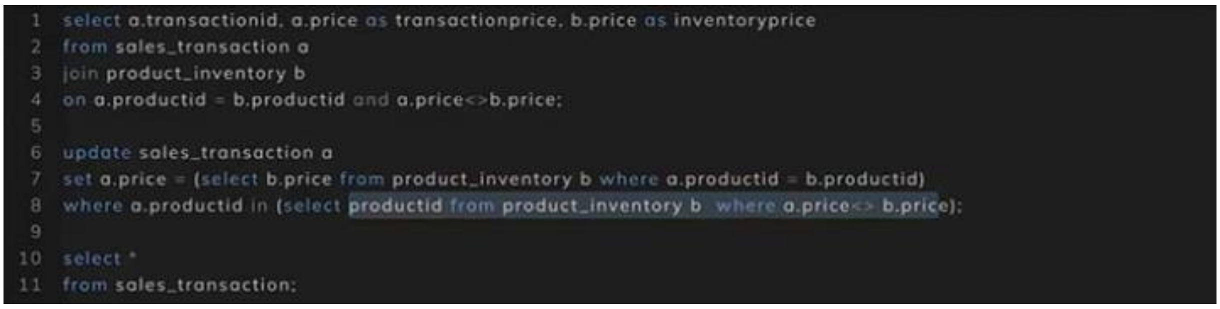
Primary Goal: Using these analyses to improve marketing, manage inventory better, and increase customer satisfaction.

Task 1: Identifying and Removing Duplicate Records



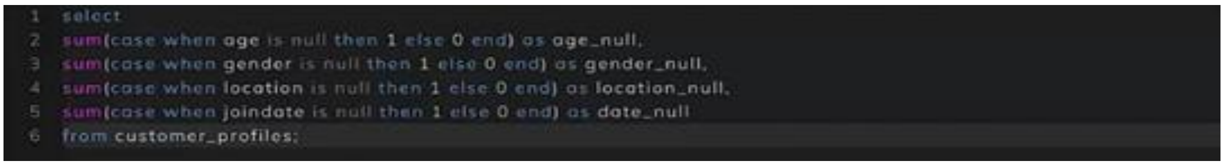
```sql
select transactionid, count(*)
from sales_transaction
group by 1
having count(*) >1;

create table st as
select distinct *
from sales_transaction;

drop table sales_transaction;

alter table st
rename to sales_transaction;

select *
from sales_transaction;
```

Task 2: Correcting Product Price Discrepancies

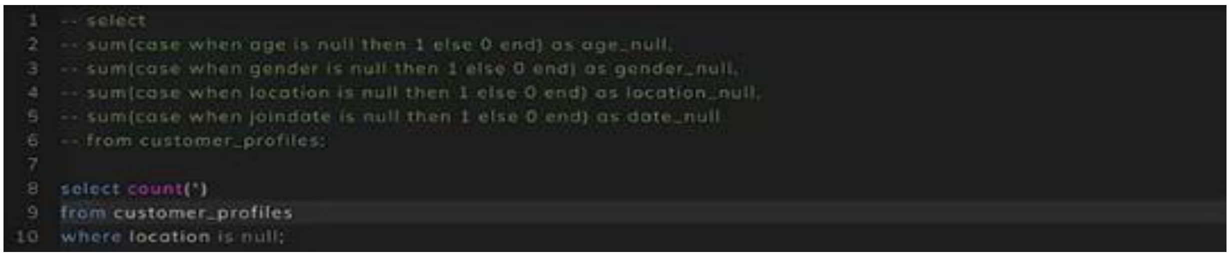


```sql
select a.transactionid, a.price as transactionprice, b.price as inventoryprice
from sales_transaction a
join product_inventory b
on a.productid = b.productid and a.price<>b.price;

update sales_transaction a
set a.price = (select b.price from product_inventory b where a.productid = b.productid)
where a.productid in (select productid from product_inventory b  where a.price<> b.price);

select *
from sales_transaction;
```

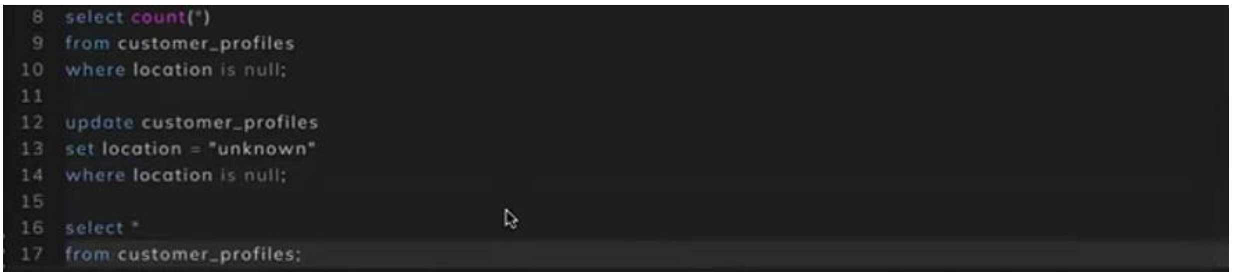
Task 3: Identifying & Handling NULL Values



```sql
select
sum(case when age is null then 1 else 0 end) as age_null,
sum(case when gender is null then 1 else 0 end) as gender_null,
sum(case when location is null then 1 else 0 end) as location_null,
sum(case when joindate is null then 1 else 0 end) as date_null
from customer_profiles;
```

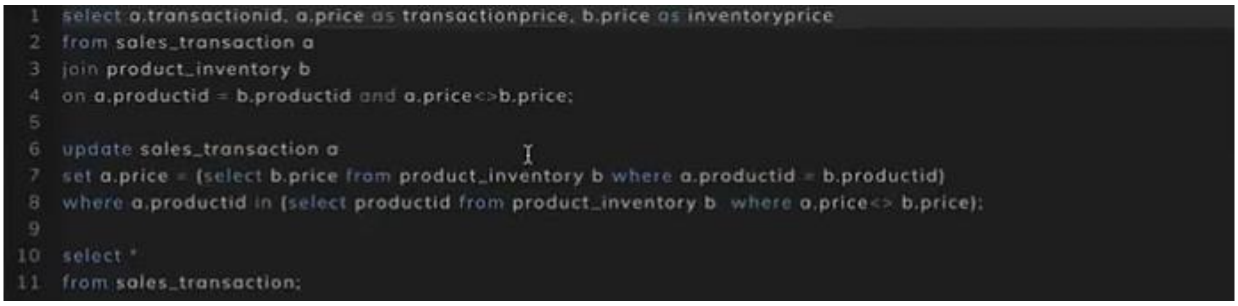


```sql
-- select
-- sum(case when age is null then 1 else 0 end) as age_null,
-- sum(case when gender is null then 1 else 0 end) as gender_null,
-- sum(case when location is null then 1 else 0 end) as location_null,
-- sum(case when joindate is null then 1 else 0 end) as date_null
-- from customer_profiles;

select count(*)
from customer_profiles
where location is null;
```



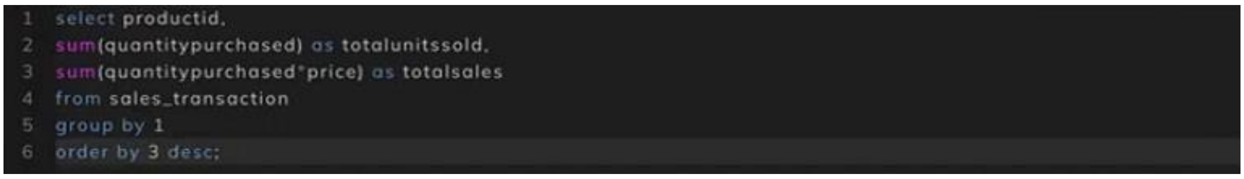
```sql
select count(*)
from customer_profiles
where location is null;

update customer_profiles
set location = "unknown"
where location is null;

select *
from customer_profiles;
```

Task 4: Cleaning and Converting Date Formats



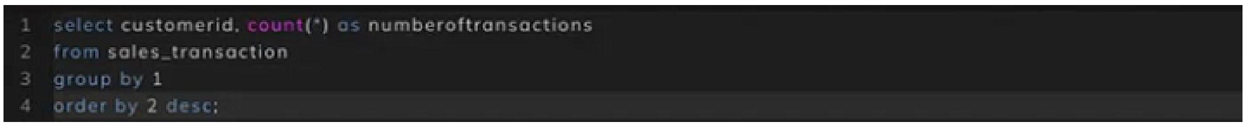
```sql
select a.transactionid, a.price as transactionprice, b.price as inventoryprice
from sales_transaction a
join product_inventory b
on a.productid = b.productid and a.price<>b.price;

update sales_transaction a
set a.price = (select b.price from product_inventory b where a.productid = b.productid)
where a.productid in (select productid from product_inventory b  where a.price<> b.price);

select *
from sales_transaction;
```

Task 5: Summarising Total Sales and Quantity Per Product



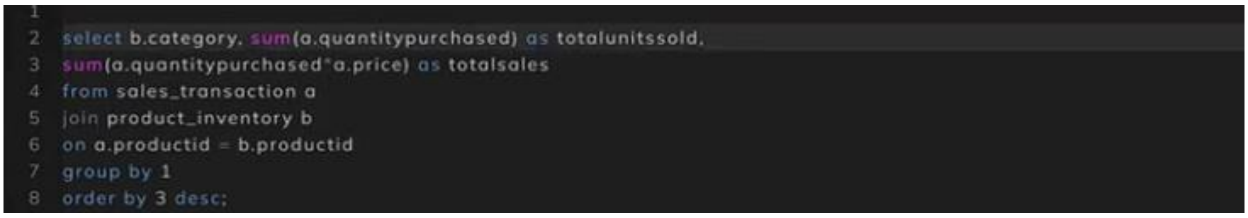
```sql
select productid,
sum(quantitypurchased) as totalunitssold,
sum(quantitypurchased*price) as totalsales
from sales_transaction
group by 1
order by 3 desc;
```

Taks 6: Analyzing Customer Purchase Frequency by Counting Transactions



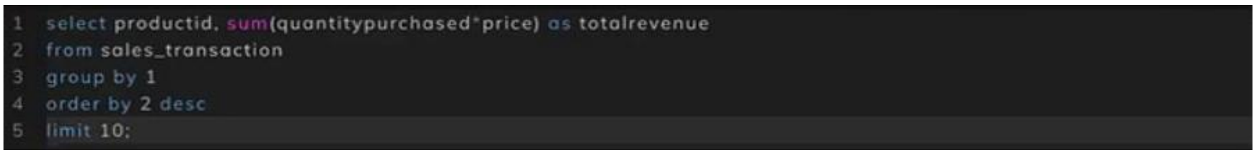
```sql
select customerid, count(*) as numberoftransactions
from sales_transaction
group by 1
order by 2 desc;
```

Task 7: Evaluating Product Category Performance using JOINS



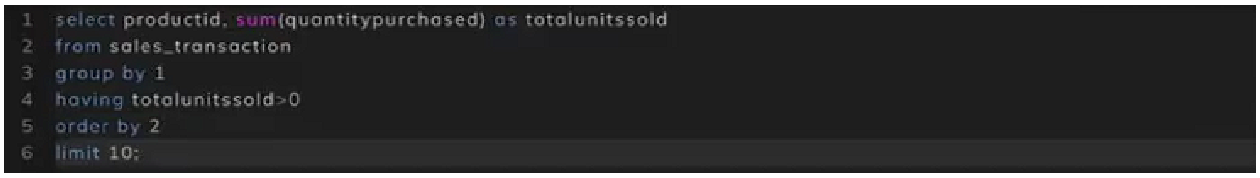
```sql

select b.category, sum(a.quantitypurchased) as totalunitssold,
sum(a.quantitypurchased*a.price) as totalsales
from sales_transaction a
join product_inventory b
on a.productid = b.productid
group by 1
order by 3 desc;
```

Task 8: Identifying Top 10 Products by Revenue



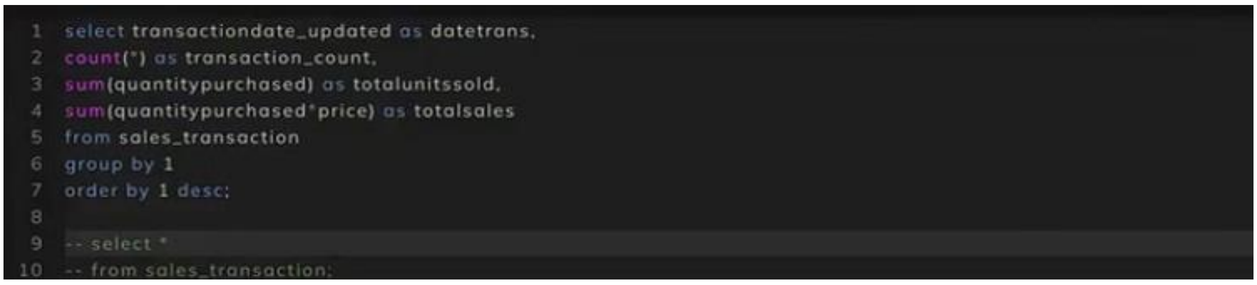
```sql
select productid, sum(quantitypurchased*price) as totalrevenue
from sales_transaction
group by 1
order by 2 desc
limit 10;
```

Task 9: Filtering Aggregates: Understanding WHERE vs. HAVING



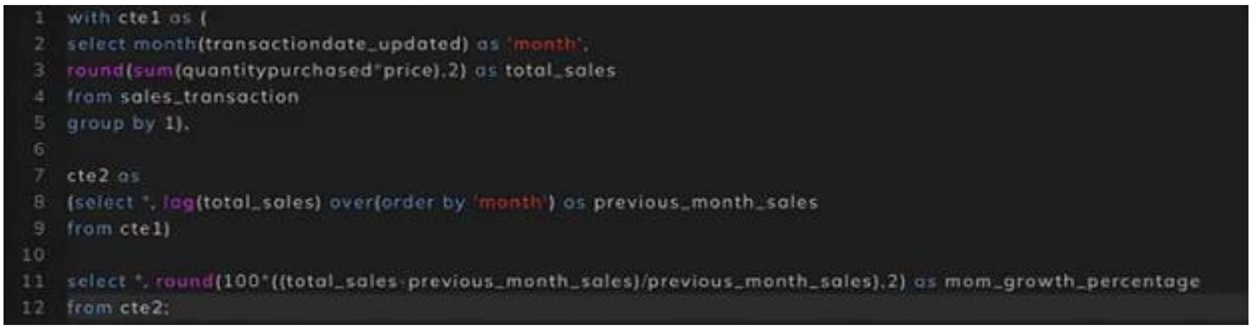
```sql
select productid, sum(quantitypurchased) as totalunitssold
from sales_transaction
group by 1
having totalunitssold>0
order by 2
limit 10;
```

Taks 10: Calculating Daily Sales Metrics



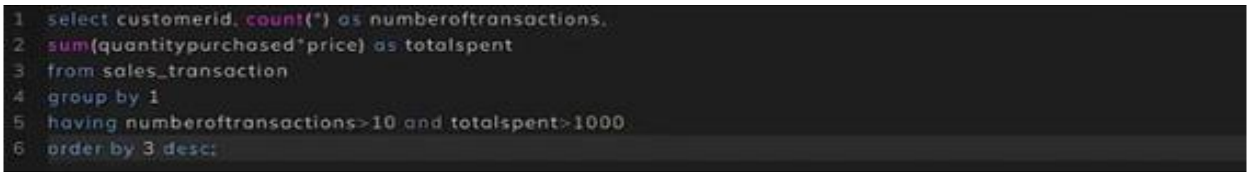
```sql
select transactiondate_updated as datetrans,
count(*) as transaction_count,
sum(quantitypurchased) as totalunitssold,
sum(quantitypurchased*price) as totalsales
from sales_transaction
group by 1
order by 1 desc;

-- select *
-- from sales_transaction;
```

Task 11: Calculating Month-on-Month Sales Growth with Window Functions



```sql
with cte1 as (
select month(transactiondate_updated) as 'month',
round(sum(quantitypurchased*price),2) as total_sales
from sales_transaction
group by 1),

cte2 as
(select *, lag(total_sales) over(order by 'month') as previous_month_sales
from cte1)

select *, round(100*((total_sales-previous_month_sales)/previous_month_sales),2) as mom_growth_percentage
from cte2;
```

Task 12: Identifying High-Value Customers with Multiple Conditions



```sql
select customerid, count(*) as numberoftransactions,
sum(quantitypurchased*price) as totalspent
from sales_transaction
group by 1
having numberoftransactions>10 and totalspent>1000
order by 3 desc;
```