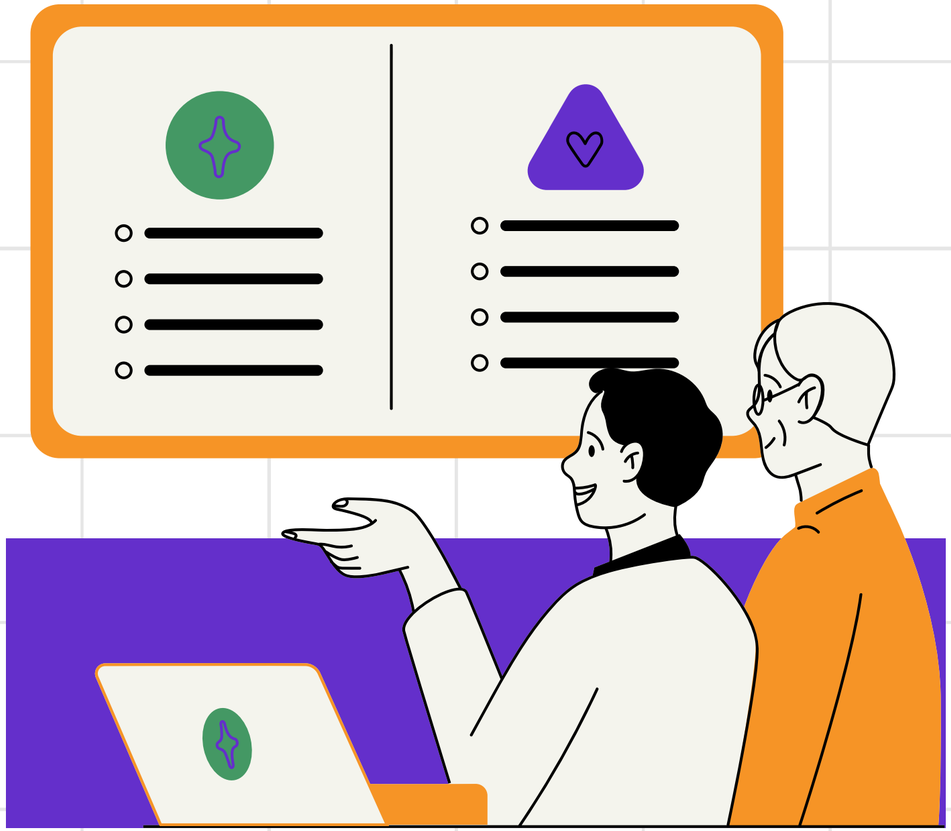
RETAIL ANALYTICS

PRESENTATION

OVERVIEW

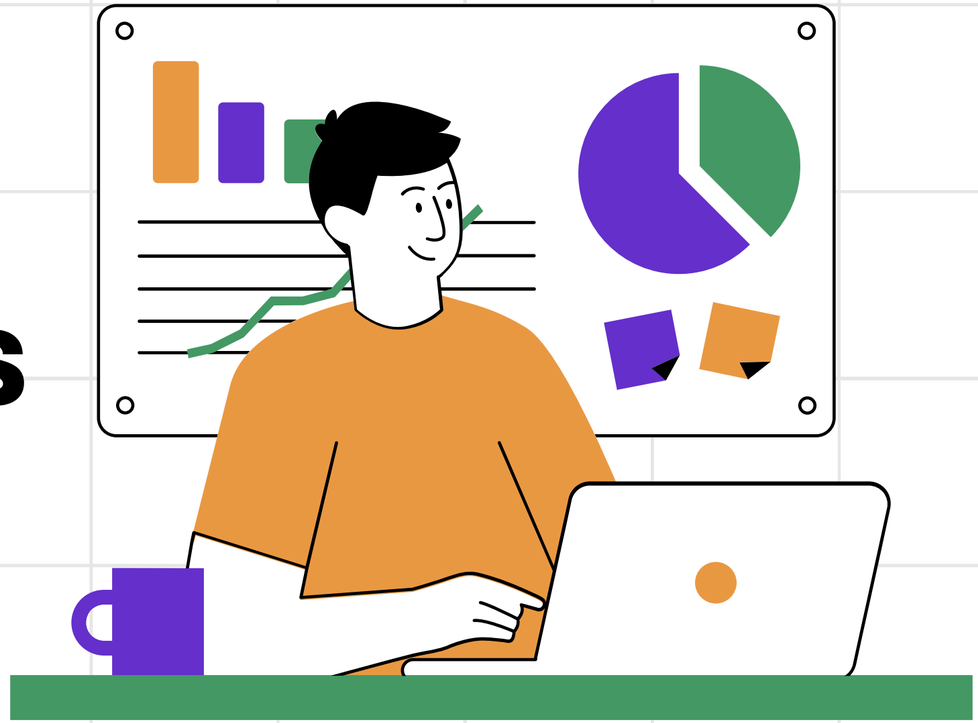
Led a comprehensive retail analytics project aimed at optimizing sales, enhancing customer segmentation, and improving inventory management. Cleaned and analyzed large datasets of sales transactions using SQL, uncovering high-performing products and identifying underperforming items to inform inventory decisions. Leveraged key retail metrics like sales trends, customer preferences, and demand forecasting to align inventory with market needs. Delivered actionable insights that streamlined operations, optimized product offerings, and maximized overall business performance.



OBJECTIVE

- Optimize sales performance by identifying high- and low-performing products through data analysis.
- Enhance customer segmentation using insights derived from sales transaction data.
- Improve inventory management by aligning stock levels with demand trends and market needs.
- Provide actionable insights to streamline operations and boost overall business efficiency.

HERE ARE SOME KEY SQL QUERIES AND ANALYTICAL TASKS TYPICALLY INVOLVED IN A RETAIL ANALYTICS PROJECT:



WRITE A QUERY TO IDENTIFY THE NUMBER OF DUPLICATES IN "SALES_TRANSACTION" TABLE. ALSO, CREATE A SEPARATE TABLE CONTAINING THE UNIQUE VALUES AND REMOVE THE THE ORIGINAL TABLE FROM THE DATABASES AND REPLACE THE NAME OF THE NEW TABLE WITH THE ORIGINAL NAME.

```sql
SELECT
    transactionID, COUNT(*)
FROM
    sales_transaction
GROUP BY transactionID
HAVING COUNT(*) > 1;

CREATE TABLE new_tbl AS SELECT DISTINCT * FROM
    sales_transaction;

drop table sales_transaction;

alter table new_tbl
rename to sales_transaction;

SELECT
    *
FROM
    sales_transaction;
```

WRITE A QUERY TO IDENTIFY THE DISCREPANCIES IN THE PRICE OF THE SAME PRODUCT IN "SALES_TRANSACTION" AND "PRODUCT_INVENTORY" TABLES. ALSO, UPDATE THOSE DISCREPANCIES TO MATCH THE PRICE IN BOTH THE TABLES.

```sql
SELECT
    TransactionID,
    trans.Price AS TransactionPrice,
    invent.Price AS InventoryPrice
FROM
    Sales_transaction AS trans
        JOIN
    product_inventory AS invent ON trans.ProductID = invent.ProductID
WHERE
    trans.Price != invent.Price;

UPDATE Sales_transaction AS trans
        JOIN
    product_inventory AS invent ON trans.ProductID = invent.ProductID
SET
    trans.Price = invent.Price
WHERE
    trans.Price != invent.Price;

SELECT
    *
FROM
    Sales_transaction;
```

WRITE A SQL QUERY TO IDENTIFY THE NULL VALUES IN THE DATASET AND REPLACE THOSE BY “UNKNOWN”.

Don't Limit

```sql
SELECT
    *
FROM
    customer_profiles
WHERE
    CustomerID IS NULL OR Age IS NULL
        OR Gender IS NULL
        OR Location IS NULL
        OR JoinDate IS NULL;

SELECT
    COUNT(*)
FROM
    customer_profiles
WHERE
    Location IS NULL;

SELECT
    CustomerID,
    Age,
    Gender,
    IFNULL(Location, 'unknown') AS Location,
    JoinDate
FROM
    customer_profiles;
```

WRITE A SQL QUERY TO SUMMARIZE THE TOTAL SALES AND QUANTITIES SOLD PER PRODUCT BY THE COMPANY.

```sql
SELECT
    ProductID,
    SUM(quantitypurchased) AS TotalUnitsSold,
    SUM(price * quantitypurchased) AS TotalSales
FROM
    Sales_transaction
GROUP BY ProductID
ORDER BY TotalSales DESC;
```

Result Grid | Filter Rows:

ProductID	TotalUnitsSold	TotalSales
51	55	512160
17	100	9450
87	92	7817.2399999
179	86	7388.2599999
96	72	7132.3200000
54	86	7052.8600000

WRITE A SQL QUERY TO COUNT THE NUMBER OF TRANSACTIONS PER CUSTOMER TO UNDERSTAND PURCHASE FREQUENCY.

```sql
SELECT
    customerID, COUNT(TransactionID) AS NumberOfTransactions
FROM
    sales_transaction
GROUP BY customerID
ORDER BY NumberOfTransactions DESC;
```

Result Grid | Filter Rows:

customerID	NumberOfTransactions
664	14
958	12
99	12
113	12
929	12
936	12

WRITE A SQL QUERY TO EVALUATE THE PERFORMANCE OF THE PRODUCT CATEGORIES BASED ON THE TOTAL SALES WHICH HELP US UNDERSTAND THE PRODUCT CATEGORIES WHICH NEEDS TO BE PROMOTED IN THE MARKETING CAMPAIGNS.

```sql
SELECT
    category,
    SUM(quantitypurchased) AS TotalUnitsSold,
    SUM(sales_transaction.price * quantitypurchased) AS TotalSales
FROM
    sales_transaction
        JOIN
    product_inventory ON sales_transaction.productID = product_inventory.productID
GROUP BY category
ORDER BY TotalSales DESC;
```

Result Grid | Filter Rows: | Expo

category	TotalUnitsSold	TotalSales
Clothing	2810	669912.610000001
Home & Kitchen	3477	217755.94000000026
Electronics	3037	177548.48000000007
Beauty & Health	3005	143932.26999999947

WRITE A SQL QUERY TO FIND THE TOP 10 PRODUCTS WITH THE HIGHEST TOTAL SALES REVENUE FROM THE SALES TRANSACTIONS. THIS WILL HELP THE COMPANY TO IDENTIFY THE HIGH SALES PRODUCTS WHICH NEEDS TO BE FOCUSED TO INCREASE THE REVENUE OF THE COMPANY.

```sql
SELECT
    ProductID, SUM(price * quantitypurchased) AS TotalRevenue
FROM
    Sales_transaction
GROUP BY ProductID
ORDER BY TotalRevenue DESC
LIMIT 10;
```

Result Grid | Filter Rows:

ProductID	TotalRevenue
51	512160
17	9450
87	7817.239999999998
179	7388.259999999998
96	7132.3200000000015
54	7052.8600000000015
187	6915.880000000003
156	6827.840000000002
57	6622.199999999999

WRITE A SQL QUERY TO FIND THE TEN PRODUCTS WITH THE LEAST AMOUNT OF UNITS SOLD FROM THE SALES TRANSACTIONS, PROVIDED THAT AT LEAST ONE UNIT WAS SOLD FOR THOSE PRODUCTS.

```sql
SELECT
    productID, SUM(quantitypurchased) AS TotalUnitsSold
FROM
    Sales_transaction
GROUP BY productID
HAVING COUNT(quantitypurchased) >= 1
ORDER BY TotalUnitsSold ASC
LIMIT 10;
```

Result Grid | Filter Rows:

productID	TotalUnitsSold
142	27
33	31
174	33
60	35
41	35
91	35
198	36
159	37

WRITE A SQL QUERY TO IDENTIFY THE SALES TREND TO UNDERSTAND THE REVENUE PATTERN OF THE COMPANY.

```sql
SELECT
    TransactionDate_updated AS DATETRANS,
    COUNT(TransactionID) AS Transaction_count,
    SUM(QuantityPurchased) AS TotalUnitsSold,
    SUM(price * QuantityPurchased) AS TotalSales
FROM
    Sales_transaction
GROUP BY DATETRANS
ORDER BY DATETRANS DESC;
```

WRITE A SQL QUERY TO UNDERSTAND THE MONTH ON MONTH GROWTH RATE OF SALES OF THE COMPANY WHICH WILL HELP UNDERSTAND THE GROWTH TREND OF THE COMPANY.

```sql
with cte as
(
select month(transactionDate_updated) as month,
sum(price*quantitypurchased) as total_sales,
lag(sum(price*quantitypurchased)) over (order by month(transactionDate_updated)) as previous_month_sale
from Sales_transaction
group by month
)
select month,total_sales,previous_month_sales,
((total_sales-previous_month_sales)/previous_month_sales*100) as mom_growth_percentage
from cte
order by month;
```

WRITE A SQL QUERY THAT DESCRIBES THE NUMBER OF TRANSACTION ALONG WITH THE TOTAL AMOUNT SPENT BY EACH CUSTOMER WHICH ARE ON THE HIGHER SIDE AND WILL HELP US UNDERSTAND THE CUSTOMERS WHO ARE THE HIGH FREQUENCY PURCHASE CUSTOMERS IN THE COMPANY.

```sql
SELECT
    CustomerID,
    COUNT(transactionID) AS NumberOfTransactions,
    SUM(price * QuantityPurchased) AS TotalSpent
FROM
    Sales_transaction
GROUP BY CustomerID
HAVING NumberOfTransactions > 10
    AND TotalSpent > 1000
ORDER BY TotalSpent DESC;
```

WRITE A SQL QUERY THAT DESCRIBES THE NUMBER OF TRANSACTION ALONG WITH THE TOTAL AMOUNT SPENT BY EACH CUSTOMER, WHICH WILL HELP US UNDERSTAND THE CUSTOMERS WHO ARE OCCASIONAL CUSTOMERS OR HAVE LOW PURCHASE FREQUENCY IN THE COMPANY.

```sql
SELECT
    customerID,
    COUNT(transactionID) AS NumberOfTransactions,
    SUM(price * quantitypurchased) AS TotalSpent
FROM
    sales_transaction
GROUP BY customerID
HAVING NumberOfTransactions <= 2
ORDER BY NumberOfTransactions , TotalSpent DESC;
```

WRITE A SQL QUERY THAT DESCRIBES THE TOTAL NUMBER OF PURCHASES MADE BY EACH CUSTOMER AGAINST EACH PRODUCTID TO UNDERSTAND THE REPEAT CUSTOMERS IN THE COMPANY.

```sql
SELECT
    customerID,
    productID,
    COUNT(quantitypurchased) AS TimesPurchased
FROM
    sales_transaction
GROUP BY customerID , productID
HAVING TimesPurchased > 1
ORDER BY TimesPurchased DESC;
```

WRITE A SQL QUERY THAT DESCRIBES THE DURATION BETWEEN THE FIRST AND THE LAST PURCHASE OF THE CUSTOMER IN THAT PARTICULAR COMPANY TO UNDERSTAND THE LOYALTY OF THE CUSTOMER.

```sql
with cte as
(
select customerID,
min(transactiondate_updated) as FirstPurchase,
max(transactiondate_updated) as LastPurchase
from Sales_transaction
group by customerID

)
select customerID,FirstPurchase,LastPurchase,
timestampdiff(day,FirstPurchase,LastPurchase) as DaysBetweenPurchases
from cte
group by customerID
having DaysBetweenPurchases>0
order by DaysBetweenPurchases desc;
```

WRITE AN SQL QUERY THAT SEGMENTS CUSTOMERS BASED ON THE TOTAL QUANTITY OF PRODUCTS THEY HAVE PURCHASED. ALSO, COUNT THE NUMBER OF CUSTOMERS IN EACH SEGMENT WHICH WILL HELP US TARGET A PARTICULAR SEGMENT FOR MARKETING.

```sql
with cte as
(
    select sales_transaction.customerID,sum(quantitypurchased) as qnt
     from customer_profiles join
     Sales_transaction
     on customer_profiles.customerID=Sales_transaction.customerID
     group by sales_transaction.customerID
)
select case when qnt
     between 1 and 9 then 'Low'
     when  qnt between 10 and 30 then 'Med' else 'High' end as CustomerSegment
     ,count(*) from cte
     group by CustomerSegment;
```

THANK YOU!