DATA ANALYSIS

USING SQL

INTRODUCTION OF PROJECT

The objective is to draw actionable insights from raw transactional data — including trends in product performance, customer behavior, sales growth, and country-level metrics.
Through powerful SQL queries (joins, aggregations, window functions, CTEs, and views), we uncover meaningful patterns to support strategic decisions in product planning, marketing, and customer segmentation.

DATASET STRUCTURE

The dataset used in this project follows a star schema model and contains three primary tables — one fact table and two dimension tables — as described below:

- gold.fact_sales - Fact Table
- gold.dim_customers - Dimension Table
- gold.dim_products - Dimension Table

gold.fact_sales

Column Name	Data Type	Description
order_number	INTEGER	Unique identifier for each order
order_date	DATE	Date when the order was placed
sales_amount	DECIMAL	Total value of the sale
quantity	INTEGER	Number of products in the order
product_key	INTEGER (FK)	Links to the product ordered
customer_key	INTEGER (FK)	Links to the customer who placed the order

gold.dim_customers

Column Name	Data Type	Description
customer_key	INT	Unique customer identifier
customer_name	TEXT	Full name of the customer
gender	TEXT	Gender of the customer
birthdate	DATE	Customer's date of birth
country	TEXT	Country where the customer is located

gold.dim_products

Column Name	Data Type	Description
product_key	INT	Unique product identifier
product_name	TEXT	Name of the product
category	TEXT	Main product category
subcategory	TEXT	Product subcategory
cost	FLOAT	Base cost or procurement price of product

RETRIEVE A LIST OF ALL TABLES IN THE DATABASE

```
SELECT
    TABLE_CATALOG,
    TABLE_SCHEMA,
    TABLE_NAME,
    TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES;
```

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1	master	dbo	spt_fallback_db	BASE TABLE
2	master	dbo	spt_fallback_dev	BASE TABLE
3	master	dbo	spt_fallback_usg	BASE TABLE
4	master	dbo	spt_values	VIEW
5	master	dbo	spt_monitor	BASE TABLE
6	master	dbo	MSreplication_options	BASE TABLE

FIND TOTAL CUSTOMERS BY COUNTRIES

```
SELECT
    country,
    COUNT(customer_key) AS total_customers
FROM gold.dim_customers
GROUP BY country
ORDER BY total_customers DESC;
```

	country	total_customers
1	United States	7482
2	Australia	3591
3	United Kingdom	1913
4	France	1810
5	Germany	1780
6	Canada	1571
7	n/a	337

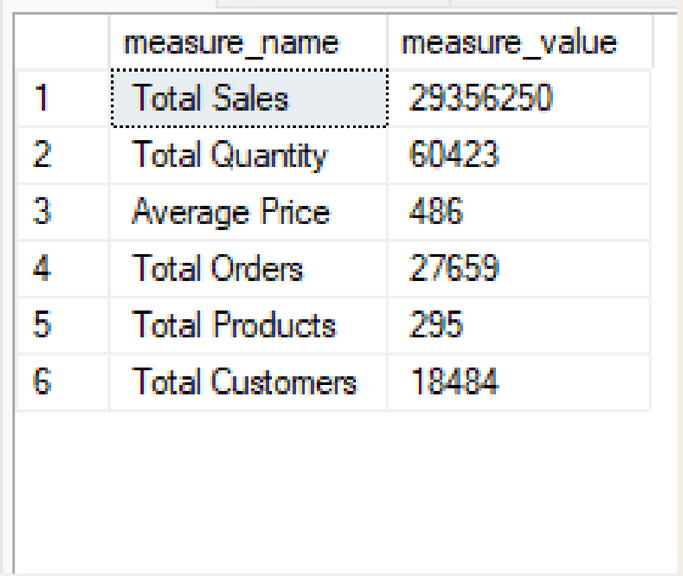
FIND THE YOUNGEST AND OLDEST CUSTOMER BASED ON BIRTHDATE

```
SELECT
    MIN(birthdate) AS oldest_birthdate,
            DATEDIFF(YEAR,  MIN(birthdate),  GETDATE())  AS
oldest_age,
    MAX(birthdate) AS youngest_birthdate,
            DATEDIFF(YEAR,  MAX(birthdate),  GETDATE())  AS
youngest_age
FROM gold.dim_customers;
```

	oldest_birthdate	oldest_age	youngest_birthdate	youngest_age
1	1916-02-10	109	1986-06-25	39

GENERATE A REPORT THAT SHOWS ALL KEY METRICS OF THE BUSINESS

```
SELECT 'Total Sales' AS measure_name, SUM(sales_amount) AS measure_value FROM gold.fact_sales
UNION ALL
SELECT 'Total Quantity', SUM(quantity) FROM gold.fact_sales
UNION ALL
SELECT 'Average Price', AVG(price) FROM gold.fact_sales
UNION ALL
SELECT 'Total Orders', COUNT(DISTINCT order_number) FROM gold.fact_sales
UNION ALL
SELECT 'Total Products', COUNT(DISTINCT product_name) FROM gold.dim_products
UNION ALL
SELECT 'Total Customers', COUNT(customer_key) FROM gold.dim_customers;
```



	measure_name	measure_value
1	Total Sales	29356250
2	Total Quantity	60423
3	Average Price	486
4	Total Orders	27659
5	Total Products	295
6	Total Customers	18484

WHAT IS THE TOTAL REVENUE GENERATED BY EACH CUSTOMER?

```
SELECT
    c.customer_key,
    c.first_name,
    c.last_name,
    SUM(f.sales_amount) AS total_revenue
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
    ON c.customer_key = f.customer_key
GROUP BY
    c.customer_key,
    c.first_name,
    c.last_name
ORDER BY total_revenue DESC;
```

	customer_key	first_name	last_name	total_revenue
1	1133	Kaitlyn	Henderson	13294
2	1302	Nichole	Nara	13294
3	1309	Margaret	He	13268
4	1132	Randall	Dominguez	13265
5	1301	Adriana	Gonzalez	13242
6	1322	Rosa	Hu	13215
7	1125	Brandi	Gill	13195
8	1308	Brad	She	13172
9	1297	Francisco	Sara	13164
10	434	Maurice	Shan	12914
11	440	Janet	Munoz	12488
12	242	Lisa	Cai	11468
13	418	Lacey	Zheng	11248
14	421	Jordan	Turner	11200
15	243	Larry	Munoz	11067
16	1656	Larry	Vazquez	10899
17	2264	Kate	Anand	10871
18	1324	Lawrence	Alonso	10836
19	1334	Terrance	Rodriguez	10829
20	1651	Aaron	Wright	10813

CALCULATE THE TOTAL SALES PER MONTH AND THE RUNNING TOTAL OF SALES OVER TIME

```
SELECT
 order_date,
 total_sales,
  SUM(total_sales) OVER (ORDER BY order_date) AS running_total_sales,
  AVG(avg_price) OVER (ORDER BY order_date) AS moving_average_price
FROM
(
    SELECT
        DATETRUNC(year, order_date) AS order_date,
        SUM(sales_amount) AS total_sales,
        AVG(price) AS avg_price
    FROM gold.fact_sales
    WHERE order_date IS NOT NULL
    GROUP BY DATETRUNC(year, order_date)
) t
```

	order_date	total_sales	running_total_sales	moving_average_price
1	2010-01-01	43419	43419	3101
2	2011-01-01	7075088	7118507	3146
3	2012-01-01	5842231	12960738	2670
4	2013-01-01	16344878	29305616	2080
5	2014-01-01	45642	29351258	1668

ANALYSE SALES PERFORMANCE OVER TIME
QUICK DATE FUNCTIONS

```
SELECT
    YEAR(order_date) AS order_year,
    MONTH(order_date) AS order_month,
    SUM(sales_amount) AS total_sales,
    COUNT(DISTINCT customer_key) AS total_customers,
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY YEAR(order_date), MONTH(order_date);
```

	order_year	order_month	total_sales	total_customers	total_quantity
1	2010	12	43419	14	14
2	2011	1	469795	144	144
3	2011	2	466307	144	144
4	2011	3	485165	150	150
5	2011	4	502042	157	157
6	2011	5	561647	174	174
7	2011	6	737793	230	230
8	2011	7	596710	188	188
9	2011	8	614516	193	193
10	2011	9	603047	185	185
11	2011	10	708164	221	221
12	2011	11	660507	208	208
13	2011	12	669395	222	222
14	2012	1	495363	252	252
15	2012	2	506992	260	260
16	2012	3	373478	212	212
17	2012	4	400324	219	219
18	2012	5	358866	207	207
19	2012	6	555142	318	318
20	2012	7	444533	246	246
21	2012	8	523887	294	294
22	2012	9	486149	269	269

COMPLEX BUT FLEXIBLY RANKING USING WINDOW FUNCTIONS

```
SELECT *
FROM (
    SELECT
        p.product_name,
        SUM(f.sales_amount) AS total_revenue,
        RANK() OVER (ORDER BY SUM(f.sales_amount) DESC)
AS rank_products
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON p.product_key = f.product_key
    GROUP BY p.product_name
) AS ranked_products
WHERE rank_products <= 5;
```

	product_name	total_revenue	rank_products
1	Mountain-200 Black- 46	1373454	1
2	Mountain-200 Black- 42	1363128	2
3	Mountain-200 Silver- 38	1339394	3
4	Mountain-200 Silver- 46	1301029	4
5	Mountain-200 Black- 38	1294854	5

CONCLUSION & LEARNINGS

This project showcased how structured SQL queries can transform raw retail sales data into meaningful business intelligence.
Using a star schema data model and advanced SQL techniques, we extracted insights related to customer behavior, product performance, sales trends, and segment-based strategies.

📌 What I Learned:

- Writing optimized SQL queries across fact and dimension tables
- Using window functions and CTEs to rank, segment, and analyze data
- Creating reusable views for dashboarding and reporting
- Deriving actionable business insights from structured data

🚀 Real-World Impact:

These techniques are directly applicable in business analyst, BI developer, and data engineering roles — where SQL remains the backbone of decision support systems.

🔹 Analysed by Harsh Gandhi | 🔸 SQL • Data Analytics
🔗 https://github.com/harshgandhi7/Data-Analysis-SQL.git

THANK YOU.