SUPER STORE ANALYSIS

A Business Intelligence Project powered by Excel & PostgreSQL.

This project focuses on analysing the Superstore dataset by leveraging SQL for structured queries and business insights. The journey began with comprehensive data cleaning and preparation in Excel, followed by importing the cleaned datasets into PostgreSQL for deep analytical querying.

Excel-Based Data Cleaning Highlights.

Before jumping into SQL, the raw dataset underwent careful cleaning in Excel using key data transformation functions:

- TRIM() – Removed extra spaces for clean, uniform text entries
- PROPER() – Standardized text casing (e.g., "john doe" → "John Doe")
- SUBSTITUTE() – Handled special characters and replaced problematic symbols
- COUNTBLANK() – Identified and handled missing values across columns

By using these powerful Excel formulas, the dataset was transformed into a clean, structured format, ensuring a smooth and error-free import into PostgreSQL.
From there, SQL was used to uncover patterns in sales trends, customer behaviour, discount impacts, and profitability across various product categories and regions.

Database Structure

After cleaning the Superstore dataset in Excel, the structured data was imported into **PostgreSQL** by creating three key relational tables using SQL commands. These tables are designed to store different aspects of the dataset in a normalized format:

1. **Customer Table** :-

```
Create Table CUSTOMERS(

    Customer_Id Varchar,
    Customer_Name Varchar,
    Segment Varchar);
```

Details:

This table was created using a CREATE TABLE query with three columns to store unique customer IDs, names, and the respective segment they belong to.

2. **Orders Table** :-

```
Create Table ORDERS(

    Order_ID Varchar,
    Order_Date Date,
    Ship_Date Date,
    Customer_Id Varchar,
    Region Varchar);
```

Details:

This table was created with five columns to capture order details, including order and shipping dates, customer references, and regional data..

3. **Products Table :-**

```
CREATE TABLE PRODUCTS(

  Product_ID VARCHAR,
  Category VARCHAR,
  Sub_Category VARCHAR,
  Product_Name VARCHAR);
```

Details:

The PRODUCTS table stores product-related details including their categories and names. Special attention was given to clean special characters in product names before importing.

4. **Sales_Data table :-**

```
CREATE TABLE Sales_Data (

  Order_ID VARCHAR,
  Product_ID VARCHAR,
  Sales FLOAT,
  Quantity INTEGER,
  Discount FLOAT,
  Profit FLOAT);
```

Details:

The **SALES_DATA** table captures transaction-level metrics such as sales revenue, quantity sold, discount offered, and profit earned. It links to both the ORDERS and PRODUCTS tables via foreign keys (Order_ID and Product_ID).

Data Import Process :-

Once the tables were created, the cleaned datasets were imported into each respective table using the CSV import option in PostgreSQL. The import was done through the "Import from File" feature, which allows direct uploading of .CSV files into the tables.

Each CSV file was carefully formatted to match the column structure of its respective table, ensuring that no errors occurred during the import. This method provided a quick and efficient way to load clean, structured data into the database.

Key Performance Indicators (KPIs)

1. **Total Sales :-**

```
select cast(sum(Sales) / 1000000.0 as Decimal (10,2)) as Total_Sales_Million from Sales_Data
```

	total_sales_million numeric (10,2)
1	2.30

2. **Total Profit :-**

```
select cast(sum(profit) as Decimal (10,2)) as Total_Profit from Sales_Data
```

	total_profit numeric (10,2)
1	286397.02

3. **Total Orders :-**

```
select distinct count (order_id) as Total_Orders from orders
```

	total_unique_orders bigint
1	9994

4. **Total Quantity Sold :-**

```
select count(quantity) as Total_Quntity_Sold from Sales_Data
```

	total_quntity_sold bigint
1	9994

5. **Average Sales per Order :-**

```
select cast(sum(sales)/count(distinct order_id) as decimal (10,2)) as Avg_sales_Per_Order from Sales_Data
```

	avg_sales_per_order numeric (10,2)
1	458.71

6. **Average Profit per Order :-**

```
select cast(sum(profit)/count(distinct order_id) as decimal (10,2)) as
Avg_Profit_Per_Order from Sales_Data
```

	avg_profit_per_order numeric (10,2)
1	57.19

7. **Highest Selling Product :-**

```
SELECT
   a.product_id,
   a.product_name,
   CAST(SUM(b.sales) AS DECIMAL(10,2)) AS
Highest_Selling_Product
FROM sales_data AS b
INNER JOIN products AS a ON a.product_id =
b.product_id
GROUP BY a.product_id, a.product_name
ORDER BY Highest_Selling_Product DESC
LIMIT 1;
```

	product_id character varying	product_name character varying	highest_selling_product numeric (10,2)
1	TEC-CO-10004722	Canon imageCLASS 2200 Advanced Copier	307999.12

8. **Most Profitable Product :-**

```
SELECT
   a.product_id,
   a.product_name,
   CAST(SUM(b.profit) AS DECIMAL(10,2)) AS Highest_Profitable_Product
FROM sales_data AS b
INNER JOIN products AS a ON a.product_id = b.product_id
GROUP BY a.product_id, a.product_name
ORDER BY Highest_Profitable_Product DESC
LIMIT 1;
```

	product_id character varying	product_name character varying	highest_profitable_product numeric (10,2)
1	TEC-CO-10004722	Canon imageCLASS 2200 Advanced Copier	125999.64

9. Sales by Region :-

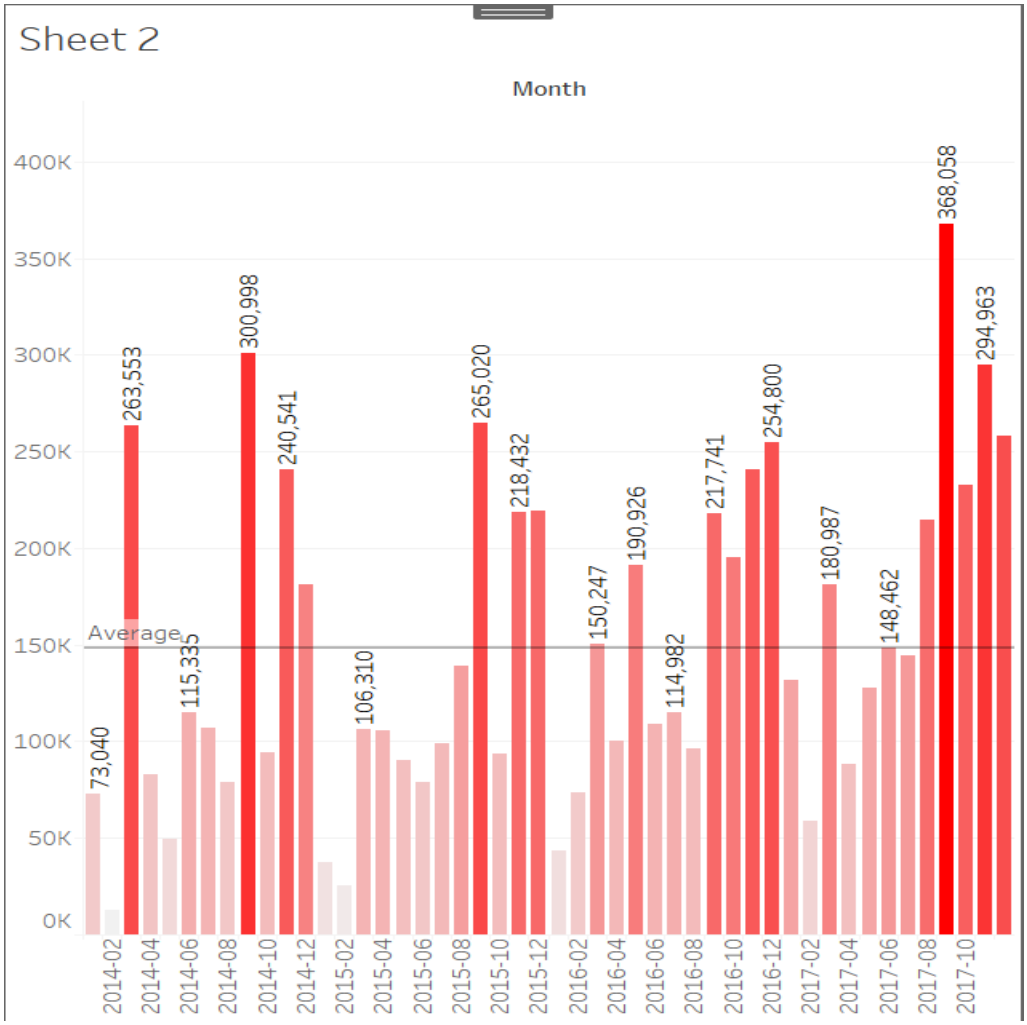
```
Select
a.region,
cast(sum(b.sales) / 1000000.02 as decimal (10,2)) as
Region_Wise_Sales_Million
from sales_data as b join orders as a on a.order_id = b.order_id
group by a.region
order by Region_Wise_Sales_Million desc
```

	region character varying	region_wise_sales_million numeric (10,2)
1	East	2.19
2	West	2.15
3	Central	1.49
4	South	1.28

10. Monthly Sales Trend:-

```
SELECT
  TO_CHAR(A.order_date, 'YYYY-MM') AS
Month,
  ROUND(SUM(B.sales)::numeric, 2) AS
Total_Sales
FROM sales_data AS B
JOIN orders AS A ON A.order_id = B.order_id
GROUP BY TO_CHAR(A.order_date, 'YYYY-MM')
ORDER BY Month;
```

	month text	total_sales numeric
1	2014-01	73040.00
2	2014-02	12548.97
3	2014-03	263552.52
4	2014-04	82745.70
5	2014-05	49727.38
6	2014-06	115335.00
7	2014-07	106880.44
8	2014-08	78774.67
9	2014-09	300997.57
10	2014-10	94373.56
11	2014-11	240540.86
12	2014-12	181123.71
13	2015-01	37719.78
14	2015-02	25533.63
15	2015-03	106309.88
16	2015-04	105900.62
17	2015-05	90569.04
18	2015-06	78672.60
19	2015-07	98629.17
20	2015-08	139371.54
21	2015-09	265020.19
22	2015-10	93383.68
23	2015-11	218431.60
24	2015-12	219279.66
25	2016-01	43192.63
26	2016-02	73663.01
27	2016-03	150247.28
28	2016-04	100356.46
29	2016-05	190926.20
30	2016-06	108804.76
31	2016-07	114982.38
32	2016-08	96178.17
33	2016-09	217741.23
34	2016-10	195082.92
35	2016-11	240767.58
36	2016-12	254799.88
37	2017-01	131782.32
38	2017-02	58598.83
39	2017-03	180987.32
40	2017-04	88238.68
41	2017-05	127919.58
42	2017-06	148462.44
43	2017-07	144396.89
44	2017-08	214403.35
45	2017-09	368057.74
46	2017-10	233012.12
47	2017-11	294962.55
48	2017-12	258344.81



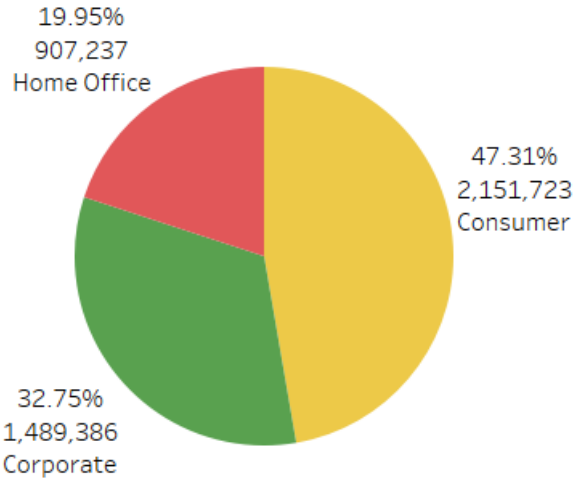
Description :-

This query shows the **monthly sales trend** by summing up total sales for each month using SUM(). The TO_CHAR function converts the order date into 'YYYY-MM' format to group data by month. ROUND(SUM(B.sales)::numeric, 2) ensures sales are shown with two decimal places. The result helps visualize how sales performed month-over-month

11. Profit By Segments :-

```
select
A.segment,
cast(sum(B.profit) AS decimal (10,2)) AS
Segment_Wise_Profit
from sales_data as B join customers as A on A.customer_id =
B.customer_id
Group by A.segment
order by Segment_Wise_Profit desc
```

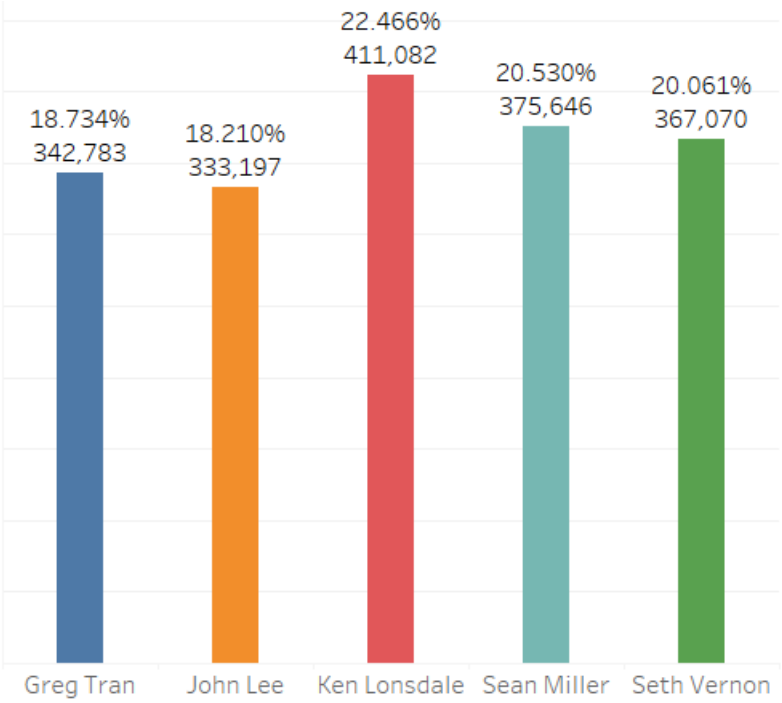
	segment character varying	segment_wise_profit numeric (10,2)
1	Consumer	2151723.42
2	Corporate	1489385.66
3	Home Office	907236.86



12. Top 5 Customers by Sales

```
select
A.customer_id,
A.customer_name,
cast(sum(B.sales) as decimal (10,2)) as
Top_5_customer_By_Sales
from sales_data as B join customers as A on A.customer_id =
B.customer_id
Group by A.customer_id, A.customer_name
order by Top_5_customer_By_Sales desc
limit 5
```

	customer_id character varying	customer_name character varying	top_5_customer_by_sales numeric (10,2)
1	KL-16645	Ken Lonsdale	411081.64
2	SM-20320	Sean Miller	375645.75
3	SV-20365	Seth Vernon	367070.40
4	GT-14710	Greg Tran	342783.48
5	JL-15835	John Lee	333197.38



13. Profit Margin :-

```
Select
cast((sum(profit) / sum(sales) * 100)
as decimal (10,2))Profit_Margin_Percentage from
sales_data
```

	profit_margin_percentage numeric (10,2)
1	12.47

Conclusion :-

This project offered a comprehensive analysis of the Superstore dataset using **SQL, Excel and Tableau**. By calculating key performance indicators (**KPIs**) such as **Total Sales, Profit Margin, Average Sales per Order, and identifying the Highest Selling Products and Top Customers**, we uncovered important business insights. SQL queries with **JOINs, GROUP BY, ORDER BY,** and functions like **ROUND(), TO_CHAR(), and CAST()** were used effectively to clean, aggregate, and analyze the data. The resulting visualizations provided a clear picture of trends across segments, categories, and time periods, helping to identify areas of strength and opportunities for improvement. This kind of data-driven decision-making is essential for business growth and strategic planning.