TASK 6: Sales Trend Analysis Using Aggregations

Step 1 :Dataset Preparation and Import :

I organized the sales dataset into two normalized tables — `Orderstask6` and `Detailstask6` — to represent orders and transaction-level details separately. These CSV files were imported into SQLite to perform structured SQL queries. This setup allowed efficient use of JOINs, date functions, and aggregate calculations for trend analysis.
All dataset files are publicly available on my GitHub repository.
🗀 **Data path**:

📊 Step 2: SQL Query Execution and Analysis:

Using the structured Orderstask6 and Detailstask6 tables in SQLite, I executed a series of SQL queries to uncover sales patterns and trends. Key SQL operations included:

• SELECT, WHERE, ORDER BY, GROUP BY for extracting, filtering, and organizing sales data
• Aggregate functions such as SUM(), AVG(), MAX(), and COUNT() to calculate revenue, order volume, and performance metrics
• JOIN operations between Orderstask6 and Detailstask6 to combine order-level and item-level data
• Date transformation using SUBSTR() and strftime() to analyze revenue trends by month and year
• Identified top-performing months, total revenue, average order value, and unique order counts

Each query was thoroughly tested for accuracy and performance. Visuals and outputs from the queries have been documented in the sections below.

Basic Queries:

🌐 1. Joining Datasets:

```
---Join both datasets on Order ID--

SELECT *

FROM Orderstask6 o

JOIN Detailstask6 d

ON o.order_id = d.order_id;
```

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```
1 ************************************
2 ---Join both datasets on Order ID--
3 ************************************
4 SELECT *
5 FROM Orderstask6 o
6 JOIN Detailstask6 d
7 ON o.order_id = d.order_id;
8
```

Or...	Order_Date	Cust...	State	City	Order...	Amount	Profit	Quan...	Categ...	Sub-...	Paym...
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	57	7	2	Clothing	Shirt	UPI
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	94	27	2	Clothing	T-shirt	COD
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	213	4	14	Clothing	Shirt	COD
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	227	48	5	Clothing	Stole	COD
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	443	11	1	Clothing	Saree	COD
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	671	114	9	Electro...	Phones	Credit ...
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	1218	-420	8	Furniture	Bookc...	COD
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	1250	-12	2	Electro...	Printers	EMI
B-26055	10-03-2018	Hariva...	Uttar P...	Mathura	B-26055	5729	64	14	Furniture	Chairs	EMI
B-25993	03-02-2018	Madhav	Delhi	Delhi	B-25993	44	8	2	Clothing	Stole	COD
B-25993	03-02-2018	Madhav	Delhi	Delhi	B-25993	173	86	1	Electro...	Printers	Debit ...

Explanation:
This joins both tables (Orderstask6 and Detailstask6) using the order_id column. This is essential because sales data is split across two tables, and combining them gives full information for analysis

2. **Handling NULLs in Aggregations:**

--handle NULLs in aggregates

--Treat NULLs in Amount as 0

```
SELECT
  SUM(COALESCE(Amount, 0)) AS total_revenue_safe
FROM Detailstask6;
```

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```
9  ************************************************
10 --********************handle NULLs in aggregates***************
11 --********************Treat NULLs in Amount as 0***************
12 SELECT
13   SUM(COALESCE(Amount, 0)) AS total_revenue_safe
14 FROM Detailstask6;
15 ************************************************
16 --******************Aggregate functions**********************
```

total_revenue_safe
437771

🧠 **Explanation**:
Sometimes the Amount column might contain NULL. COALESCE(Amount, 0) replaces NULL with 0, ensuring SUM() doesn't skip them or give errors.

📊 3. Aggregate Functions Overview:

-- Aggregate functions

```
SELECT
  SUM(Amount) AS total_revenue,
  COUNT(*) AS total_rows,
  COUNT(DISTINCT order_id) AS unique_products,
  AVG(Amount) AS average_amount,
  MAX(Amount) AS max_amount,
  MIN(Amount) AS min_amount
FROM Detailstask6;
```

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```sql
--******************Aggregate functions*********************
SELECT
  SUM(Amount) AS total_revenue,
  COUNT(*) AS total_rows,
  COUNT(DISTINCT order_id) AS unique_products,
  AVG(Amount) AS average_amount,
  MAX(Amount) AS max_amount,
  MIN(Amount) AS min_amount
FROM Detailstask6;
*********************************************************
```

total_revenue	total_rows	unique_products	average_amount	max_amount	min_amount
437771	1500	500	291.8473333333…	5729	4

Explanation:
These are **standard aggregation functions**:

- SUM() → Total amount
- COUNT(*) → Number of rows
- COUNT(DISTINCT order_id) → Unique orders
- AVG() → Average
- MAX() / MIN() → Highest and lowest sale

4. Extracting Year and Month :

```
SELECT
  strftime('%Y', SUBSTR(Order_Date, 7, 4) || '-' || SUBSTR(Order_Date, 4, 2) || '-' || SUBSTR(Order_Date, 1, 2)) AS year,
  strftime('%m', SUBSTR(Order_Date, 7, 4) || '-' || SUBSTR(Order_Date, 4, 2) || '-' || SUBSTR(Order_Date, 1, 2)) AS month,
  COUNT(*) AS total_orders
FROM Orderstask6
GROUP BY year, month
ORDER BY year, month;
```

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```
25 ******************************************************
26 --******************Group data by month and year****************
27 SELECT
28   strftime('%Y', SUBSTR(Order_Date, 7, 4) || '-' || SUBSTR(Order_Date, 4, 2) || '-' || SUBSTR(Order_Date, 1, 2)) AS year,
29   strftime('%m', SUBSTR(Order_Date, 7, 4) || '-' || SUBSTR(Order_Date, 4, 2) || '-' || SUBSTR(Order_Date, 1, 2)) AS month,
30   COUNT(*) AS total_orders
31 FROM Orderstask6
32 GROUP BY year, month
33 ORDER BY year, month;
34 ************************************************************
```

year	month	total_orders
2018	01	61
2018	02	54
2018	03	58
2018	04	44
2018	05	31
2018	06	30
2018	07	31
2018	08	31
2018	09	30
2018	10	43

📅 **Explanation**:
Dates are in DD-MM-YYYY format. To convert it:

- SUBSTR() rearranges it to YYYY-MM-DD
- strftime('%Y', ...) extracts the year
- strftime('%m', ...) extracts the month

This lets you **group sales by month and year**.

Counts how many orders were made in each month, using the fixed format date for grouping.

📊 5. COUNT(*) vs COUNT(DISTINCT)

SELECT COUNT(*) AS total_rows FROM Detailstask6;

SELECT COUNT(DISTINCT Order_ID) AS unique_orders FROM Detailstask6;

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```
37 ************************************************************
38 --******************COUNT(*) vs COUNT(DISTINCT col)******************
39 -- ******************COUNT(*) example: total rows in details*****************
40 SELECT COUNT(*) AS total_rows FROM Detailstask6;
41
42 --******************COUNT(DISTINCT Order_ID): how many unique orders******************
43 SELECT COUNT(DISTINCT Order_ID) AS unique_orders FROM Detailstask6;
44 ************************************************************************
45
46
```

total_rows
1500

```
41
42 --******************COUNT(DISTINCT Order_ID): how many unique orders******************
43 SELECT COUNT(DISTINCT Order_ID) AS unique_orders FROM Detailstask6;
44 ************************************************************************
45
46
```

unique_orders
500

📌 **Explanation**:

- COUNT(*) → Counts all rows
- COUNT(DISTINCT Order_ID) → Counts only **unique orders**, avoiding duplicates
  Great to know for interview questions!

6. GROUP BY vs ORDER BY:

--ORDER BY vs GROUP BY

--GROUP BY: To get revenue by product

```
SELECT
  order_id,
  SUM(Amount) AS total_revenue
FROM Detailstask6
GROUP BY order_id;
```

-- ORDER BY: To sort by highest revenue

```
SELECT
  order_id,
  SUM(Amount) AS total_revenue
FROM Detailstask6
GROUP BY order_id
ORDER BY total_revenue DESC;
```

OUTPUT:

Screenshots of queries and their respective outputs are provided below:

```
47 --******************ORDER BY vs GROUP BY*****************
48 --******************GROUP BY: To get revenue by product*****************
49 SELECT
50   order_id,
51   SUM(Amount) AS total_revenue
52 FROM Detailstask6
53 GROUP BY order_id;
54 -- ******************ORDER BY: To sort by highest revenue*****************
55 SELECT
```

Order_ID	total_revenue
B-25601	1429
B-25602	3889
B-25603	2025
B-25604	222
B-25605	75
B-25606	87
B-25607	50
B-25608	2953
B-25609	734
B-25610	2105

```
-- ******************ORDER BY: To sort by highest revenue*****************
SELECT
  order_id,
  SUM(Amount) AS total_revenue
FROM Detailstask6
GROUP BY order_id
ORDER BY total_revenue DESC;
*****************************************************************
--******************Months by Sales*******************************
```

Order_ID	total_revenue
B-26055	9902
B-25955	6339
B-25993	6026
B-25881	5809
B-25973	5228
B-25923	4836
B-26051	4719
B-25855	4613
B-26093	4502
B-25653	4434

Explanation:

- GROUP BY is used to **aggregate data**
- ORDER BY is used to **sort data**

Example: you can group by order_id to get revenue per order, then order the result by highest to lowest revenue.

7. TOP 3 **Monthly total revenue**

SELECT

strftime('%Y', SUBSTR(o.Order_Date, 7, 4) || '-' || SUBSTR(o.Order_Date, 4, 2) || '-' || SUBSTR(o.Order_Date, 1, 2)) AS year,

strftime('%m', SUBSTR(o.Order_Date, 7, 4) || '-' || SUBSTR(o.Order_Date, 4, 2) || '-' || SUBSTR(o.Order_Date, 1, 2)) AS month,

SUM(d.Amount) AS total_revenue

FROM Orderstask6 o

JOIN Detailstask6 d ON o.Order_ID = d.Order_ID

GROUP BY year, month

ORDER BY total_revenue DESC

LIMIT 3;

OUTPUT:

Screenshots of queries and their respective outputs are provided below:

```
--*****************Months by Sales*****************************
SELECT
  strftime('%Y', SUBSTR(o.Order_Date, 7, 4) || '-' || SUBSTR(o.Order_Date, 4, 2) || '-' || SUBSTR(o.Order_Date, 1, 2)) AS year,
  strftime('%m', SUBSTR(o.Order_Date, 7, 4) || '-' || SUBSTR(o.Order_Date, 4, 2) || '-' || SUBSTR(o.Order_Date, 1, 2)) AS month,
  SUM(d.Amount) AS total_revenue
FROM Orderstask6 o
JOIN Detailstask6 d ON o.Order_ID = d.Order_ID
GROUP BY year, month
ORDER BY total_revenue DESC
LIMIT 3;
*****************************************************************
```

year	month	total_revenue
2018	01	61632
2018	03	60694
2018	11	48469

Explanation:

Counts how many orders were made in each month, using the fixed format date for grouping.

8 . Extract YEAR and MONTH from Order Date (with revenue & order volume):

SELECT

strftime('%Y', SUBSTR(o.order_date, 7, 4) || '-' || SUBSTR(o.order_date, 4, 2) || '-' || SUBSTR(o.order_date, 1, 2)) AS year,

strftime('%m', SUBSTR(o.order_date, 7, 4) || '-' || SUBSTR(o.order_date, 4, 2) || '-' || SUBSTR(o.order_date, 1, 2)) AS month,

SUM(d.amount) AS total_revenue,

COUNT(DISTINCT o.order_id) AS order_volume

FROM Orderstask6 o

JOIN Detailstask6 d ON o.order_id = d.order_id

GROUP BY year, month

ORDER BY year, month;

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```
--*****************Extract YEAR and MONTH from Order Date--*****************
SELECT
  strftime('%Y', SUBSTR(o.order_date, 7, 4) || '-' || SUBSTR(o.order_date, 4, 2) || '-' || SUBSTR(o.order_date, 1, 2)) AS year,
  strftime('%m', SUBSTR(o.order_date, 7, 4) || '-' || SUBSTR(o.order_date, 4, 2) || '-' || SUBSTR(o.order_date, 1, 2)) AS month,
  SUM(d.amount) AS total_revenue,
  COUNT(DISTINCT o.order_id) AS order_volume
FROM Orderstask6 o
JOIN Detailstask6 d ON o.order_id = d.order_id
GROUP BY year, month
ORDER BY year, month;
```

year	month	total_revenue	order_volume
2018	01	61632	61
2018	02	38962	54
2018	03	60694	58
2018	04	34330	44
2018	05	29093	31
2018	06	23658	30
2018	07	12966	31
2018	08	31492	31
2018	09	27283	30
2018	10	31613	43

Explanation:

▶ Extracts year & month from order_date, then calculates:

- Total revenue
- Unique orders per month

☑ 9. Aggregate SUM(amount) for Total Revenue:

SELECT

substr(o.order_date, 7, 4) AS year,

substr(o.order_date, 4, 2) AS month,

SUM(d.amount) AS total_revenue

FROM

Orderstask6 o

JOIN

Detailstask6 d ON o.order_id = d.order_id

GROUP BY

year, month

ORDER BY

year, month;

OUTPUT:
Screenshots of queries and their respective outputs are provided below:

```
92 ---*****************Aggregate SUM(amount) for total revenue*****************
93 SELECT
94   substr(o.order_date, 7, 4) AS year,
95   substr(o.order_date, 4, 2) AS month,
96   SUM(d.amount) AS total_revenue
97 FROM
98   Orderstask6 o
99 JOIN
00   Detailstask6 d ON o.order_id = d.order_id
01 GROUP BY
02   year, month
03 ORDER BY
04   year, month;
05 ********************************************************************************
```

year	month	total_revenue
2018	01	61632
2018	02	38962
2018	03	60694
2018	04	34330
2018	05	29093
2018	06	23658
2018	07	12966
2018	08	31492

Explanation:

- **Joins** both tables on order_id
- **Extracts year & month** using SUBSTR
- **Calculates total revenue** per month
- **Counts unique orders** per month as **order volume**

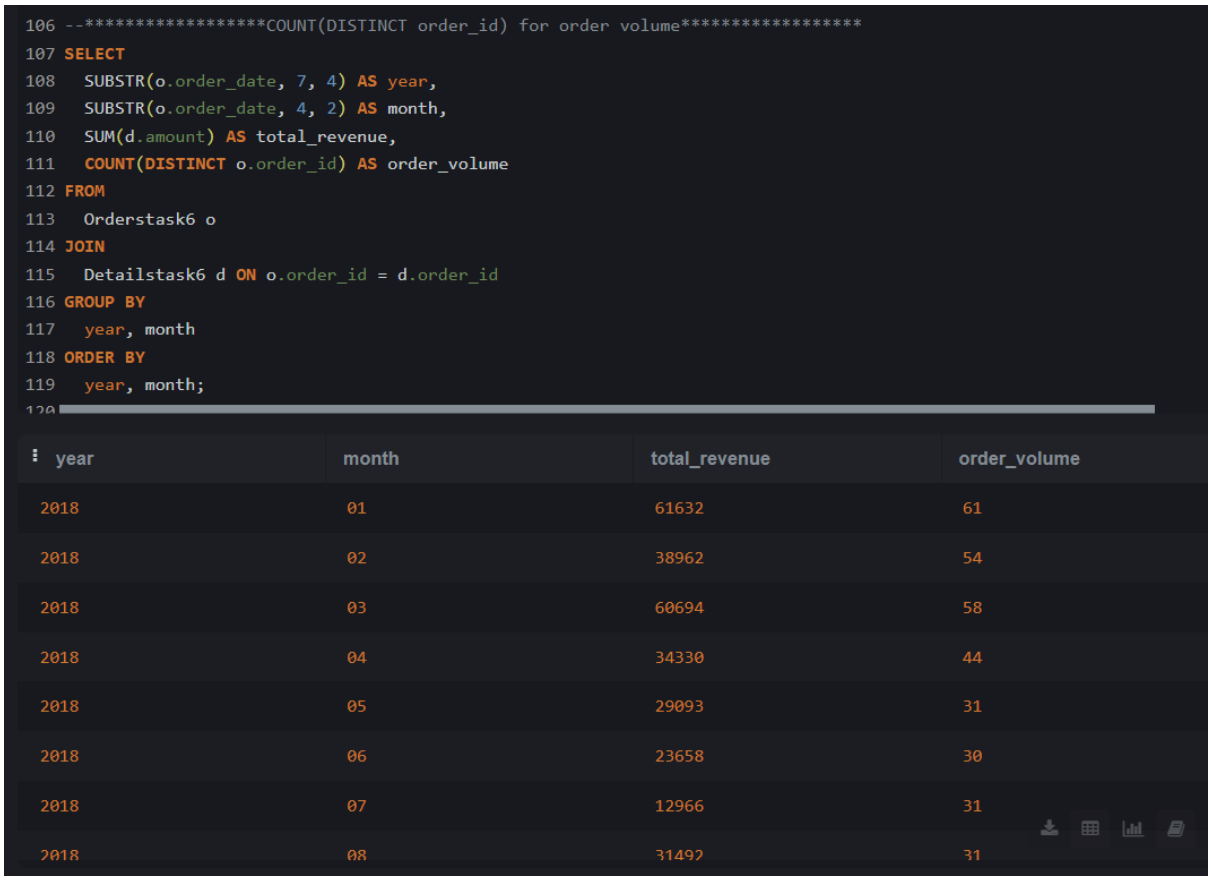
10 . COUNT(DISTINCT order_id) for order volume :

SELECT

```
  SUBSTR(o.order_date, 7, 4) AS year,
  SUBSTR(o.order_date, 4, 2) AS month,
  SUM(d.amount) AS total_revenue,
  COUNT(DISTINCT o.order_id) AS order_volume
FROM
  Orderstask6 o
JOIN
  Detailstask6 d ON o.order_id = d.order_id
GROUP BY
  year, month
ORDER BY
  year, month;
```

OUTPUT:

Screenshots of queries and their respective outputs are provided below:



```
106 --*****************COUNT(DISTINCT order_id) for order volume*****************
107 SELECT
108   SUBSTR(o.order_date, 7, 4) AS year,
109   SUBSTR(o.order_date, 4, 2) AS month,
110   SUM(d.amount) AS total_revenue,
111   COUNT(DISTINCT o.order_id) AS order_volume
112 FROM
113   Orderstask6 o
114 JOIN
115   Detailstask6 d ON o.order_id = d.order_id
116 GROUP BY
117   year, month
118 ORDER BY
119   year, month;
```

year	month	total_revenue	order_volume
2018	01	61632	61
2018	02	38962	54
2018	03	60694	58
2018	04	34330	44
2018	05	29093	31
2018	06	23658	30
2018	07	12966	31
2018	08	31492	31

Explanation:

- Converts order_date to **year** and **month**

- Sums up the **amount** for monthly **revenue**
- Counts **distinct orders** per month as **order volume**
- Shows **monthly performance over time**

☑ Conclusion :

Through structured SQL analysis on the `Orderstask6` and `Detailstask6` tables, I was able to uncover key sales trends, such as total revenue, peak sales months, and order volumes. By leveraging `JOINs`, aggregate functions, and date-based grouping, I gained meaningful insights into business performance over time. This project reinforced my SQL skills in real-world data analysis scenarios and demonstrated how structured querying can drive actionable business decisions.