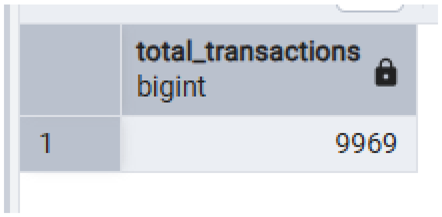
Walmart Sales Data Analysis using SQL

1. BASIC DATA UNDERSTANDING

1.1: Total number of transactions

```
SELECT COUNT(*) AS total_transactions FROM walmart;
```

Output:

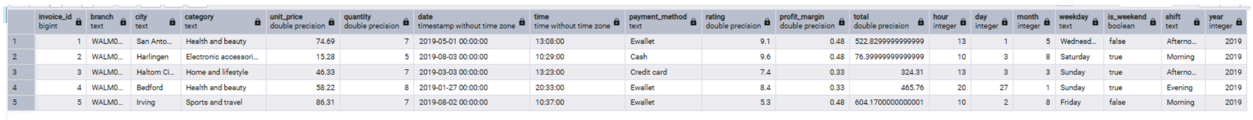


	total_transactions bigint
1	9969

1.2: View first 5 rows of data

```
SELECT * FROM walmart LIMIT 5;
```

Output:

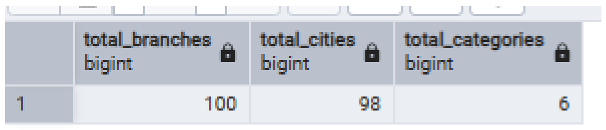


	invoice_id bigint	branch text	city text	category text	unit_price double precision	quantity double precision	date timestamp without time zone	time time without time zone	payment_method text	rating double precision	profit_margin double precision	total double precision	hour integer	day integer	month integer	weekday text	is_weekend boolean	shift text	year integer
1	1	WALM0...	San Anto...	Health and beauty	74.69	7	2019-05-01 00:00:00	13:08:00	Ewallet	9.1	0.48	522.8299999999999	13	1	5	Wednesd...	false	Afterno...	2019
2	2	WALM0...	Harlingen	Electronic accessori...	15.28	5	2019-08-03 00:00:00	10:29:00	Cash	9.6	0.48	76.39999999999999	10	3	8	Saturday	true	Morning	2019
3	3	WALM0...	Haltom Ci...	Home and lifestyle	46.33	7	2019-03-03 00:00:00	13:23:00	Credit card	7.4	0.33	324.31	13	3	3	Sunday	true	Afterno...	2019
4	4	WALM0...	Bedford	Health and beauty	58.22	8	2019-01-27 00:00:00	20:33:00	Ewallet	8.4	0.33	465.76	20	27	1	Sunday	true	Evening	2019
5	5	WALM0...	Irving	Sports and travel	86.31	7	2019-08-02 00:00:00	10:37:00	Ewallet	5.3	0.48	604.1700000000001	10	2	8	Friday	false	Morning	2019

1.3: Unique branches, cities, and categories

```
SELECT COUNT(DISTINCT branch) AS total_branches,
       COUNT(DISTINCT city) AS total_cities,
       COUNT(DISTINCT category) AS total_categories
FROM walmart;
```

Output:

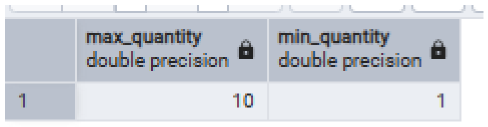


	total_branches bigint	total_cities bigint	total_categories bigint
1	100	98	6

1.4: Maximum and minimum quantity sold

```
SELECT MAX(quantity) AS max_quantity, MIN(quantity) AS min_quantity
FROM walmart;
```

Output:



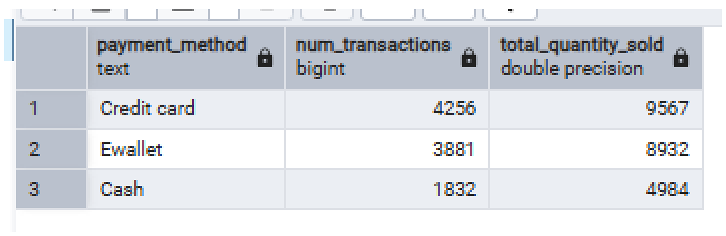
	max_quantity double precision	min_quantity double precision
1	10	1

2. CUSTOMER & PAYMENT INSIGHTS

2.1: Count of transactions and quantity sold by payment method

```
SELECT
    payment_method,
    COUNT(*) AS num_transactions,
    SUM(quantity) AS total_quantity_sold
FROM walmart
GROUP BY payment_method
ORDER BY total_quantity_sold DESC;
```

Output:

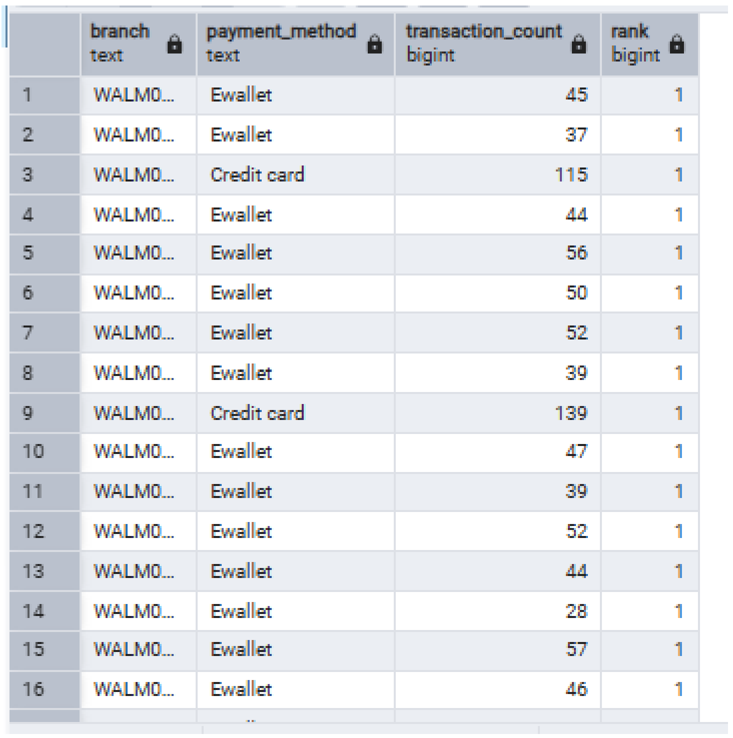


	payment_method text	num_transactions bigint	total_quantity_sold double precision
1	Credit card	4256	9567
2	Ewallet	3881	8932
3	Cash	1832	4984

2.2: Most preferred payment method per branch

```
WITH payment_ranking AS (
    SELECT
        branch,
        payment_method,
        COUNT(*) AS transaction_count,
        RANK() OVER (PARTITION BY branch ORDER BY COUNT(*) DESC) AS
rank
    FROM walmart
    GROUP BY branch, payment_method
)
SELECT * FROM payment_ranking WHERE rank = 1;
```

Output:



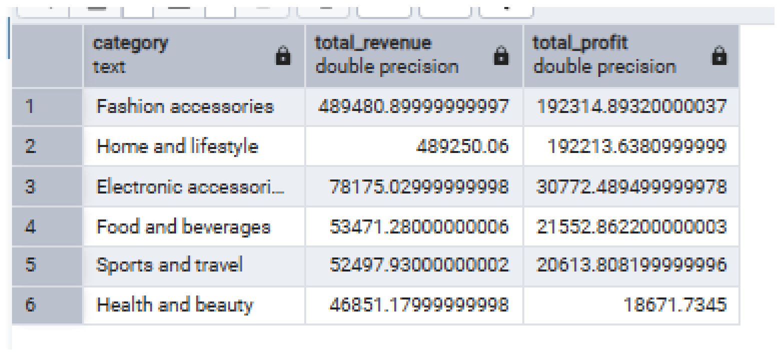
	branch text	payment_method text	transaction_count bigint	rank bigint
1	WALM0...	Ewallet	45	1
2	WALM0...	Ewallet	37	1
3	WALM0...	Credit card	115	1
4	WALM0...	Ewallet	44	1
5	WALM0...	Ewallet	56	1
6	WALM0...	Ewallet	50	1
7	WALM0...	Ewallet	52	1
8	WALM0...	Ewallet	39	1
9	WALM0...	Credit card	139	1
10	WALM0...	Ewallet	47	1
11	WALM0...	Ewallet	39	1
12	WALM0...	Ewallet	52	1
13	WALM0...	Ewallet	44	1
14	WALM0...	Ewallet	28	1
15	WALM0...	Ewallet	57	1
16	WALM0...	Ewallet	46	1

3. PRODUCT CATEGORY & PROFIT ANALYSIS

3.1: Revenue and profit by category

```
SELECT
    category,
    SUM(total) AS total_revenue,
    SUM(total * profit_margin) AS total_profit
FROM walmart
GROUP BY category
ORDER BY total_profit DESC;
```

Output:

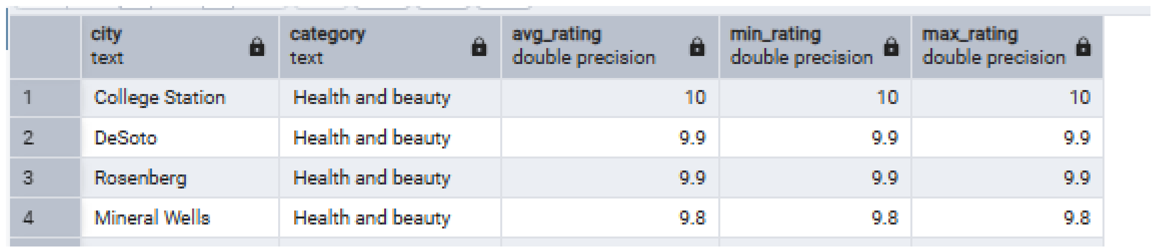


	category text	total_revenue double precision	total_profit double precision
1	Fashion accessories	489480.89999999997	192314.89320000037
2	Home and lifestyle	489250.06	192213.6380999999
3	Electronic accessori...	78175.02999999998	30772.489499999978
4	Food and beverages	53471.28000000006	21552.862200000003
5	Sports and travel	52497.93000000002	20613.808199999996
6	Health and beauty	46851.17999999998	18671.7345

3.2: Average, min, max rating per city-category pair

```
SELECT
    city,
    category,
    AVG(rating) AS avg_rating,
    MIN(rating) AS min_rating,
    MAX(rating) AS max_rating
FROM walmart
GROUP BY city, category
ORDER BY avg_rating DESC;
```

Output:



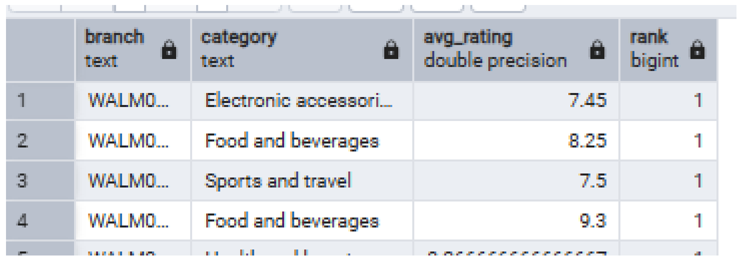
	city text	category text	avg_rating double precision	min_rating double precision	max_rating double precision
1	College Station	Health and beauty	10	10	10
2	DeSoto	Health and beauty	9.9	9.9	9.9
3	Rosenberg	Health and beauty	9.9	9.9	9.9
4	Mineral Wells	Health and beauty	9.8	9.8	9.8

3.3: Highest-rated product category per branch

```
WITH category_rating AS (
    SELECT
```

```
        branch,
        category,
        AVG(rating) AS avg_rating,
        RANK() OVER (PARTITION BY branch ORDER BY AVG(rating) DESC)
AS rank
    FROM walmart
    GROUP BY branch, category
)
SELECT * FROM category_rating WHERE rank = 1;
```

Output:

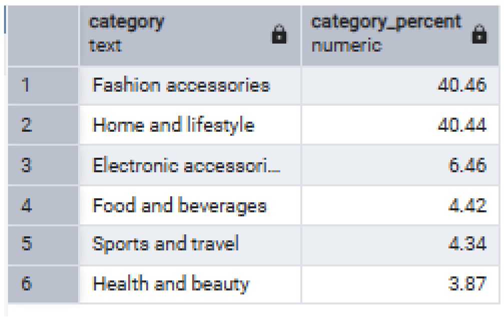


	branch text	category text	avg_rating double precision	rank bigint
1	WALM0...	Electronic accessori...	7.45	1
2	WALM0...	Food and beverages	8.25	1
3	WALM0...	Sports and travel	7.5	1
4	WALM0...	Food and beverages	9.3	1

3.4: Category share in total revenue

```
SELECT
    category,
    ROUND((SUM(total) * 100.0 / (SELECT SUM(total) FROM
walmart))::numeric, 2) AS category_percent
FROM walmart
GROUP BY category
ORDER BY category_percent DESC;
```

Output:



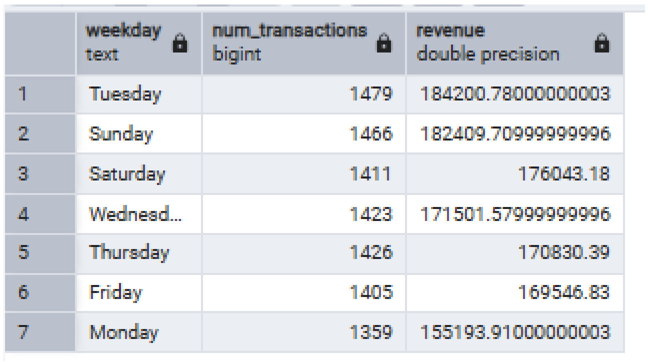
	category text	category_percent numeric
1	Fashion accessories	40.46
2	Home and lifestyle	40.44
3	Electronic accessori...	6.46
4	Food and beverages	4.42
5	Sports and travel	4.34
6	Health and beauty	3.87

4. TIME-BASED SALES TRENDS

4.1 Sales by day of the week

```
SELECT
    weekday,
    COUNT(*) AS num_transactions,
    SUM(total) AS revenue
FROM walmart
GROUP BY weekday
ORDER BY revenue DESC;
```

Output:

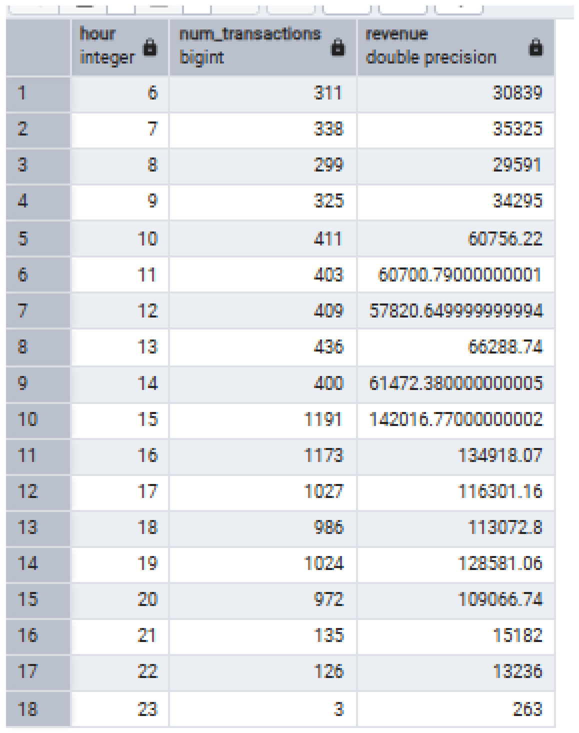


	weekday text	num_transactions bigint	revenue double precision
1	Tuesday	1479	184200.78000000003
2	Sunday	1466	182409.70999999996
3	Saturday	1411	176043.18
4	Wednesd...	1423	171501.57999999996
5	Thursday	1426	170830.39
6	Friday	1405	169546.83
7	Monday	1359	155193.91000000003

4.2: Sales by hour

```
SELECT
    hour,
    COUNT(*) AS num_transactions,
    SUM(total) AS revenue
FROM walmart
GROUP BY hour
ORDER BY hour;
```

Output:



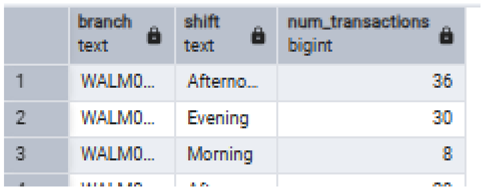
	hour integer	num_transactions bigint	revenue double precision
1	6	311	30839
2	7	338	35325
3	8	299	29591
4	9	325	34295
5	10	411	60756.22
6	11	403	60700.79000000001
7	12	409	57820.649999999994
8	13	436	66288.74
9	14	400	61472.380000000005
10	15	1191	142016.77000000002
11	16	1173	134918.07
12	17	1027	116301.16
13	18	986	113072.8
14	19	1024	128581.06
15	20	972	109066.74
16	21	135	15182
17	22	126	13236
18	23	3	263

4.3: Sales by shift (Morning, Afternoon, Evening)

```
SELECT
    branch,
    shift,
    COUNT(*) AS num_transactions
FROM walmart
```

```
GROUP BY branch, shift
ORDER BY branch, num_transactions DESC;
```

Output:

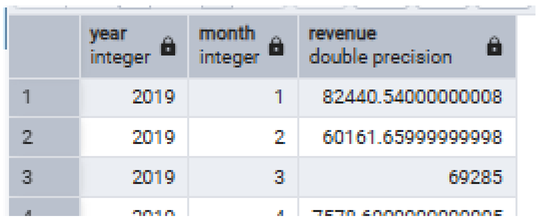


	branch text	shift text	num_transactions bigint
1	WALM0...	Afterno...	36
2	WALM0...	Evening	30
3	WALM0...	Morning	8

4.4 Month-over-month revenue trend

```
SELECT
    year,
    month,
    SUM(total) AS revenue
FROM walmart
GROUP BY year, month
ORDER BY year, month;
```

Output:



	year integer	month integer	revenue double precision
1	2019	1	82440.54000000008
2	2019	2	60161.65999999998
3	2019	3	69285

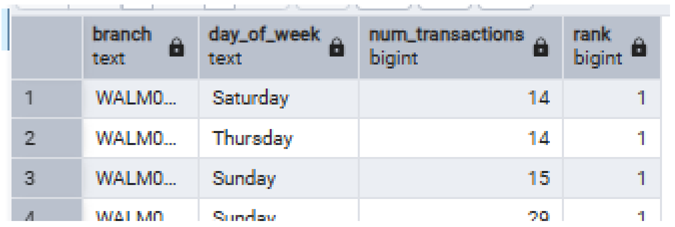
5. ADVANCED KPIs & STRATEGIC INSIGHTS

5.1: Busiest day (most transactions) for each branch

```
WITH daily_rank AS (
    SELECT
```

```
        branch,
        weekday AS day_of_week,
        COUNT(*) AS num_transactions,
        RANK() OVER (PARTITION BY branch ORDER BY COUNT(*) DESC) AS
rank
    FROM walmart
    GROUP BY branch, weekday
)
SELECT * FROM daily_rank WHERE rank = 1;
```

Output:

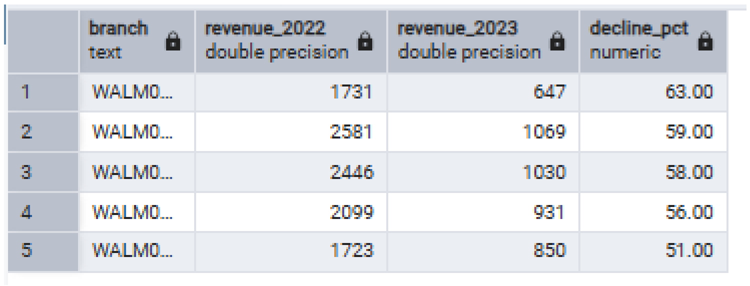


	branch text	day_of_week text	num_transactions bigint	rank bigint
1	WALM0...	Saturday	14	1
2	WALM0...	Thursday	14	1
3	WALM0...	Sunday	15	1
4	WALM0	Sunday	29	1

5.2: Five branches with revenue decrease from 2022 to 2023

```
WITH rev_2022 AS (
    SELECT branch, SUM(total) AS revenue_2022
    FROM walmart
    WHERE year = 2022
    GROUP BY branch
),
rev_2023 AS (
    SELECT branch, SUM(total) AS revenue_2023
    FROM walmart
    WHERE year = 2023
    GROUP BY branch
)
SELECT
    rev_2022.branch,
    rev_2022.revenue_2022,
    rev_2023.revenue_2023,
    ROUND(
        ((rev_2022.revenue_2022 - rev_2023.revenue_2023) /
rev_2022.revenue_2022)::numeric,
        2
    ) * 100 AS decline_pct
FROM rev_2022
JOIN rev_2023 ON rev_2022.branch = rev_2023.branch
WHERE rev_2023.revenue_2023 < rev_2022.revenue_2022
ORDER BY decline_pct DESC
LIMIT 5;
```

Output:

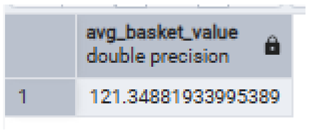


	branch text	revenue_2022 double precision	revenue_2023 double precision	decline_pct numeric
1	WALM0...	1731	647	63.00
2	WALM0...	2581	1069	59.00
3	WALM0...	2446	1030	58.00
4	WALM0...	2099	931	56.00
5	WALM0...	1723	850	51.00

5.3: Average basket value

```
SELECT
    AVG(total) AS avg_basket_value
FROM walmart;
```

Output:

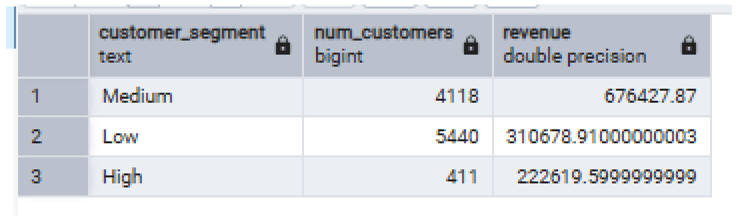


	avg_basket_value double precision
1	121.34881933995389

5.4: Segmenting customers by purchase value

```
SELECT
    CASE
        WHEN total < 100 THEN 'Low'
        WHEN total BETWEEN 100 AND 300 THEN 'Medium'
        ELSE 'High'
    END AS customer_segment,
    COUNT(*) AS num_customers,
    SUM(total) AS revenue
FROM walmart
GROUP BY customer_segment
ORDER BY revenue DESC;
```

Output:

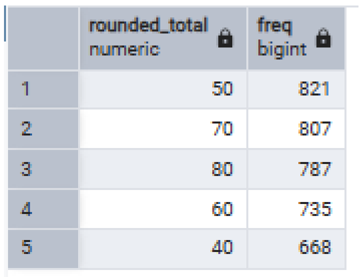


	customer_segment text	num_customers bigint	revenue double precision
1	Medium	4118	676427.87
2	Low	5440	310678.91000000003
3	High	411	222619.5999999999

5.5: Most common invoice value range (rounded)

```
SELECT
    ROUND(total::numeric, -1) AS rounded_total,
    COUNT(*) AS freq
FROM walmart
GROUP BY rounded_total
ORDER BY freq DESC
LIMIT 5;
```

Output:

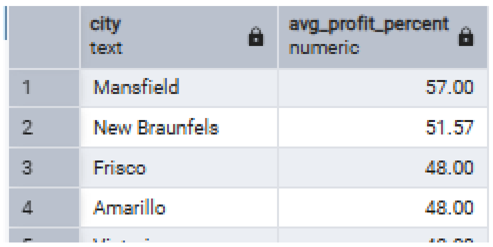


	rounded_total numeric	freq bigint
1	50	821
2	70	807
3	80	787
4	60	735
5	40	668

5.6: City-wise average profit margin

```
SELECT
    city,
    ROUND((AVG(profit_margin) * 100)::numeric, 2) AS
avg_profit_percent
FROM walmart
GROUP BY city
ORDER BY avg_profit_percent DESC;
```

Output:



	city text	avg_profit_percent numeric
1	Mansfield	57.00
2	New Braunfels	51.57
3	Frisco	48.00
4	Amarillo	48.00