1. Find the Top N Records (e.g., Top 5 Customers by Sales)

```
sql
Copy code
SELECT customer_id, SUM(sales) AS total_sales
FROM sales_data
GROUP BY customer_id
ORDER BY total_sales DESC
LIMIT 5;
```

---

2. Identify Trends Over Time (e.g., Monthly Sales Trends)

```
sql
Copy code
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month, SUM(sales) AS total_sales
FROM sales_data
GROUP BY DATE_FORMAT(order_date, '%Y-%m')
ORDER BY month;
```

---

3. Find Frequent Items or Categories (e.g., Most Sold Products)

```
sql
Copy code
SELECT product_id, COUNT(*) AS product_count
FROM sales_data
GROUP BY product_id
ORDER BY product_count DESC
LIMIT 10;
```

---

4. Perform Market Basket Analysis (e.g., Association Rules)

Identify products frequently bought together:

```
sql
Copy code
SELECT a.product_id AS product_a, b.product_id AS product_b, COUNT(*) AS
pair_count
FROM transactions a
JOIN transactions b ON a.transaction_id = b.transaction_id AND a.product_id <
b.product_id
GROUP BY a.product_id, b.product_id
ORDER BY pair_count DESC
LIMIT 10;
```

---

5. Cluster Customers by Spending (e.g., Customer Segmentation)

Cluster customers based on their total spending:

```
sql
Copy code
SELECT customer_id,
       CASE
           WHEN SUM(sales) < 100 THEN 'Low Spenders'
           WHEN SUM(sales) BETWEEN 100 AND 500 THEN 'Medium Spenders'
           ELSE 'High Spenders'
       END AS spending_category
FROM sales_data
GROUP BY customer_id;
```

---

6. Identify Outliers (e.g., Detect Abnormal Transactions)

Find transactions significantly higher than the average:

```
sql
Copy code
SELECT *
FROM transactions
WHERE amount > (SELECT AVG(amount) + 3 * STDDEV(amount) FROM transactions);
```

---

7. Cohort Analysis (e.g., Retention Rates by Signup Month)

```
sql
Copy code
SELECT DATE_FORMAT(signup_date, '%Y-%m') AS cohort,
DATE_FORMAT(activity_date, '%Y-%m') AS activity_month, COUNT(DISTINCT
user_id) AS active_users
FROM user_activity
GROUP BY cohort, activity_month
ORDER BY cohort, activity_month;
```

---

8. Predictive Analysis (e.g., Trend Forecasting)

Generate a linear trend:

```
sql
Copy code
SELECT month, total_sales,
       ROW_NUMBER() OVER (ORDER BY month) AS time_index,
       AVG(total_sales) OVER (ORDER BY month ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW) AS moving_average
FROM (
    SELECT DATE_FORMAT(order_date, '%Y-%m') AS month, SUM(sales) AS
total_sales
    FROM sales_data
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
) AS monthly_data;
```

---

9. Find Correlations Between Variables

Example: Analyze the relationship between price and quantity sold.

```
sql
Copy code
SELECT price, quantity_sold,
       (AVG(price * quantity_sold) - AVG(price) * AVG(quantity_sold)) /
       (STDDEV(price) * STDDEV(quantity_sold)) AS correlation_coefficient
FROM sales_data;
```

---

10. Customer Lifetime Value (CLV)

Calculate the total revenue generated by each customer:

```
sql
Copy code
SELECT customer_id, SUM(sales) AS lifetime_value
FROM sales_data
GROUP BY customer_id
ORDER BY lifetime_value DESC;
```

---

11. Find Churned Customers

Identify customers who haven't purchased in the last 6 months:

```
sql
Copy code
SELECT customer_id
FROM sales_data
GROUP BY customer_id
HAVING MAX(order_date) < DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

---

12. Calculate Customer Retention Rate

```
sql
Copy code
WITH first_purchase AS (
    SELECT customer_id, MIN(order_date) AS first_order
    FROM sales_data
    GROUP BY customer_id
)
SELECT DATE_FORMAT(first_order, '%Y-%m') AS signup_month,
       COUNT(DISTINCT CASE WHEN order_date >= first_order THEN customer_id
END) / COUNT(DISTINCT customer_id) AS retention_rate
FROM sales_data
JOIN first_purchase ON sales_data.customer_id = first_purchase.customer_id
GROUP BY signup_month;
```

---

13. Find Most Active Users

```
sql
Copy code
SELECT user_id, COUNT(*) AS activity_count
FROM user_logs
GROUP BY user_id
ORDER BY activity_count DESC
LIMIT 10;
```

---

14. Analyze Product Performance by Region

```
sql
Copy code
SELECT region, product_id, SUM(sales) AS total_sales
FROM sales_data
GROUP BY region, product_id
ORDER BY region, total_sales DESC;
```

---

15. Sentiment Analysis (Basic Example)

Identify positive/negative keywords in text:

```
sql
Copy code
SELECT review_id,
       CASE
           WHEN review_text LIKE '%good%' OR review_text LIKE '%excellent%'
THEN 'Positive'
           WHEN review_text LIKE '%bad%' OR review_text LIKE '%poor%' THEN
'Negative'
           ELSE 'Neutral'
       END AS sentiment
FROM product_reviews;
```

---

16. Analyze Seasonal Patterns

```
sql
Copy code
SELECT MONTH(order_date) AS month, SUM(sales) AS total_sales
FROM sales_data
GROUP BY MONTH(order_date)
ORDER BY month;
```

---

17. Calculate Revenue Growth

```
sql
Copy code
SELECT current_month.month,
       (current_month.revenue - previous_month.revenue) /
previous_month.revenue AS growth_rate
FROM (
    SELECT DATE_FORMAT(order_date, '%Y-%m') AS month, SUM(sales) AS revenue
    FROM sales_data
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
) AS current_month
LEFT JOIN (
    SELECT DATE_FORMAT(order_date, '%Y-%m') AS month, SUM(sales) AS revenue
    FROM sales_data
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
) AS previous_month
ON current_month.month = DATE_FORMAT(DATE_SUB(previous_month.month, INTERVAL
1 MONTH), '%Y-%m');
```

---

18. Find Gaps in Data (e.g., Missing Dates)

```
sql
Copy code
WITH date_series AS (
    SELECT MIN(order_date) AS start_date, MAX(order_date) AS end_date FROM
sales_data
    UNION ALL
    SELECT DATE_ADD(start_date, INTERVAL 1 DAY) AS start_date, end_date
    FROM date_series WHERE start_date < end_date
)
SELECT start_date
FROM date_series
LEFT JOIN sales_data ON date_series.start_date = sales_data.order_date
WHERE sales_data.order_date IS NULL;
```

---

19. Calculate Average Order Value (AOV)

```
sql
Copy code
SELECT AVG(order_value) AS avg_order_value
FROM (
    SELECT order_id, SUM(sales) AS order_value
    FROM sales_data
    GROUP BY order_id
) AS order_totals;
```

---

20. Compare Year-over-Year Performance

```
sql
Copy code
SELECT YEAR(order_date) AS year, MONTH(order_date) AS month, SUM(sales) AS
total_sales
```

```
FROM sales_data
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY year, month;
```

These SQL queries provide a strong foundation for data mining tasks, from basic aggregations to complex analyses like trends, segmentation, and predictive modeling.