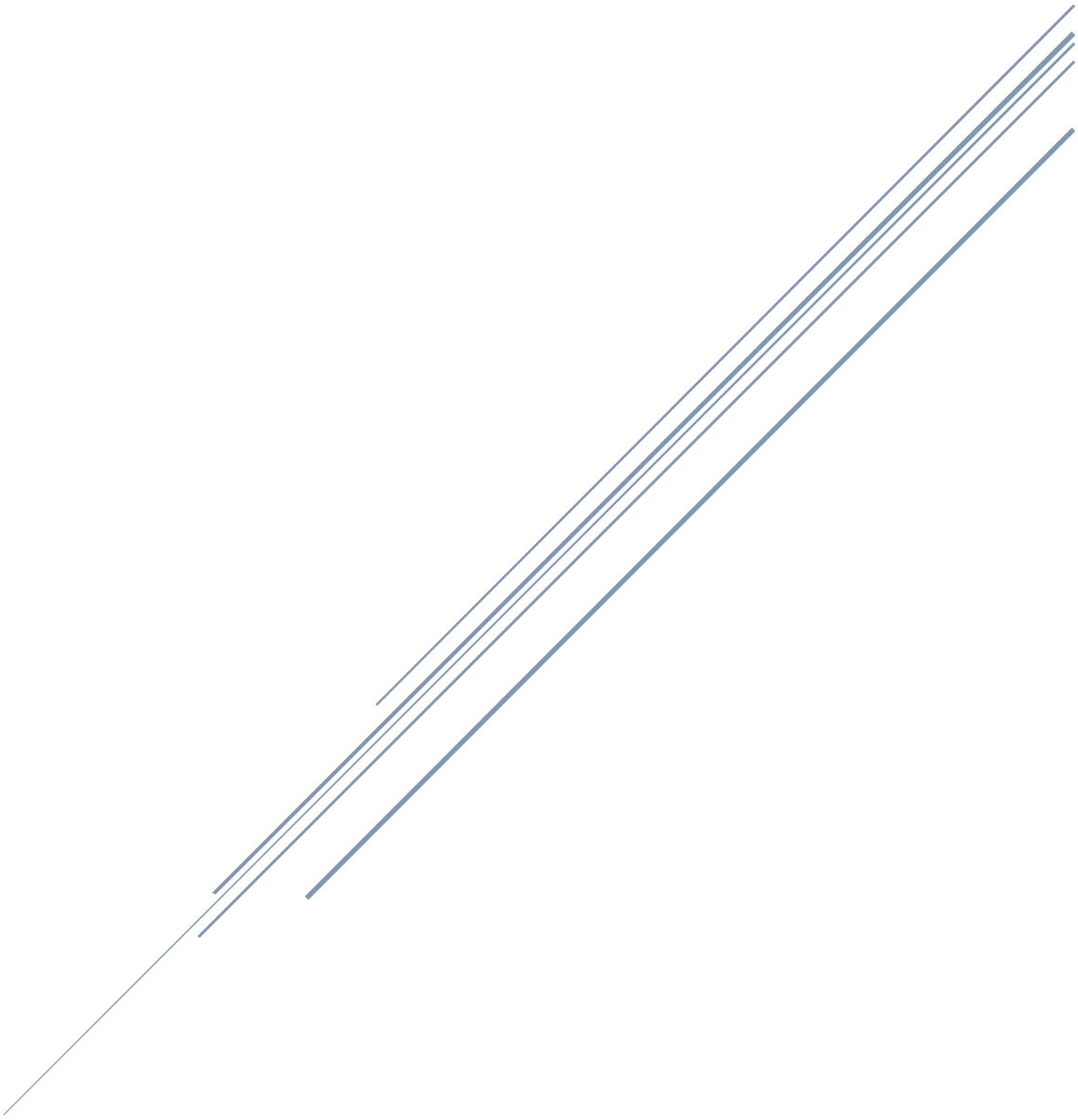
ANALYTICAL SQL PROJECT

[Analyzing Store Transactions Database]



Abdullah Attia

Table of content:

1- EDA

2- RFM

3- some insights

1- Exploratory Data Analysis

Let’s do some basic EDA!
First, Getting to know the data with some data discovery queries.

```
SELECT
    *
FROM
    TABLERETAIL
LIMIT
    5;
```

invoice character varying (50)	stockcode character varying (50)	quantity integer	invoicedate character varying (50)	price double precision	customer_id character varying (50)
537215	85124C	12	12/5/2010 15:38	2.55	12747
537215	85124B	6	12/5/2010 15:38	2.55	12747
537215	84879	16	12/5/2010 15:38	1.69	12747
537215	85062	24	12/5/2010 15:38	1.65	12747
537215	85064	6	12/5/2010 15:38	5.45	12747

So we have orders dataset of transactions, with order item as level of granularity,

```sql
SELECT
    COUNT(DISTINCT INVOICE) INVOICE_COUNT,
    COUNT(DISTINCT STOCKCODE) STOCKCODE_COUNT,
    COUNT(DISTINCT CUSTOMER_ID) CUSTOMER_COUNT,
    MIN(INVOICEDATE::DATE) AS "first order",
    MAX(INVOICEDATE::DATE) AS "last order",
    MAX(INVOICEDATE::DATE) - MIN(INVOICEDATE::DATE) "dataset time period"
FROM
    TABLERETAIL;
```

invoice_count bigint	stockcode_count bigint	customer_count bigint	first order date	last order date	dataset time period integer
717	2335	110	2010-12-01	2011-12-09	373

So in this dataset,

We have 717 orders including 2335 different products made by 110 different customers over around a year.

Now, Let’s Dive a little bit in the data to get some more insights and answer some questions!

1- what are the top selling products

```
SELECT
    STOCKCODE,
    SUM(QUANTITY) "UNITS SOLD"
FROM
    TABLERETAIL T
GROUP BY
    STOCKCODE
ORDER BY
    "UNITS SOLD" DESC
LIMIT
    5;
```

stockcode character varying (50)	UNITS SOLD bigint
84077	7824
84879	6117
22197	5918
21787	5075
21977	4691

So these are the top selling Products, we can do some further analysis, to find out why? Is it related to their prices, quality, promotions, and more.

2- Best Selling Month

a. In terms of Number of Orders

```
SELECT
    TO_CHAR(INVOICEDATE::DATE, 'Mon') "month",
    COUNT(INVOICE) "Order Count"
FROM
    TABLERETAIL T
GROUP BY
    "month"
ORDER BY
    "Order Count" DESC;
```

month text	Order Count bigint
Nov	3200
Dec	1955
Sep	1411
Oct	1003
May	983
Jun	745
Jul	743
Mar	727
Aug	595
Feb	556
Apr	479
Jan	461

b. In terms of Sales

```
SELECT
    TO_CHAR(INVOICEDATE::DATE, 'Mon') "month",
    ROUND(SUM(QUANTITY * PRICE)::NUMERIC, 2) AS SALES
FROM
    TABLERETAIL T
GROUP BY
    "month"
ORDER BY
    SALES DESC;
```

month text	sales numeric
Nov	45633.38
Aug	38374.64
Sep	27853.82
Dec	24547.09
Oct	19735.07
May	19496.18
Mar	17038.01
Jul	15664.54
Jun	13517.01
Feb	13336.84
Apr	10980.51
Jan	9541.29

As we can see November was the Best Month In terms of Both Sales And Orders Traffic, however the second and third are different in terms of number of orders, and sales meaning that the average order value for months is changing by a good amount.

3- Best Selling Day

a. In terms of Number of Orders

```
SELECT
    INVOICEDATE::DATE "date",
    COUNT(INVOICE) "Order Count"
FROM
    TABLERETAIL T
GROUP BY
    "date"
ORDER BY
    "Order Count" DESC
LIMIT
    5;
```

date date	Order Count bigint
2011-11-23	426
2011-11-17	304
2011-09-11	289
2011-11-25	268
2011-11-15	268

b. In terms of Sales

```
SELECT
    INVOICEDATE::DATE "day",
    ROUND(SUM(QUANTITY * PRICE)::NUMERIC, 2) AS SALES
FROM
    TABLERETAIL T
GROUP BY
    "day"
ORDER BY
    SALES DESC
LIMIT
    5;
```

day date	sales numeric
2011-08-04	19304.68
2011-08-11	9349.72
2011-11-09	6676.09
2011-09-22	4839.38
2011-11-17	4672.72

factors such as promotions, marketing campaigns, special events, and customer behavior can influence these differences. Analyzing both metrics separately can provide insights into different aspects of daily sales performance and help in making informed business decisions.

4- Traffic Over the Hour Of Day

a. Number of Orders

```
SELECT
    TO_CHAR(INVOICEDATE::TIMESTAMP, 'HH24') AS "Hour",
    COUNT(INVOICE) AS "count of orders"
FROM
    TABLERETAIL
GROUP BY
    TO_CHAR(INVOICEDATE::TIMESTAMP, 'HH24')
ORDER BY
    "Hour";
```

Hour text	count of orders bigint
07	1
08	80
09	550
10	944
11	1713
12	2439
13	2173
14	1710
15	1827
16	698
17	372
18	120
19	199
20	32

b. Sales

```
SELECT
    TO_CHAR(INVOICEDATE::TIMESTAMP, 'HH24') AS "Hour",
    ROUND(SUM(QUANTITY * PRICE)::NUMERIC, 2) AS SALES
FROM
    TABLERETAIL
GROUP BY
    TO_CHAR(INVOICEDATE::TIMESTAMP, 'HH24')
ORDER BY
    "Hour"
```

Hour text	sales numeric
07	535.50
08	2065.29
09	15988.75
10	28699.67
11	21820.18
12	39473.63
13	44414.87
14	24235.95
15	35772.63
16	11249.71
17	8260.93
18	21755.01
19	1362.57
20	83.69

The sales amount is highest during the late morning and early afternoon hours, specifically between 10 and 14. This suggests that these hours are the busiest in terms of sales activity

However, after this time sales start to decrease so we might want to figure out why or add some promotions and discounts to try to push the sales further.

5- Top 5 Customers

a. In terms of Number of Orders

```
SELECT
    CUSTOMER_ID,
    COUNT(*) "count of orders"
FROM
    TABLERETAIL T
GROUP BY
    CUSTOMER_ID
ORDER BY
    "count of orders" DESC
LIMIT
    5;
```

customer_id character varying (50)	count of orders bigint
12748	4596
12921	720
12867	538
12841	420
12856	314

b. In terms of Sales

```
SELECT
    CUSTOMER_ID,
    ROUND(SUM(quantity*price)::numeric, 2) sales
FROM
    TABLERETAIL T
GROUP BY
    CUSTOMER_ID
ORDER BY
    sales DESC
LIMIT
    5;
```

customer_id character varying (50)	sales numeric
12931	42055.96
12748	33719.73
12901	17654.54
12921	16587.09
12939	11581.80

We can spot the top customers through some customer Segmentation or even RFM analysis to identify their behavior and give them some sort of discounts to motivate their behavior more and see their percentage from the total income, etc. Will see more on this later.

2- RFM

And now we will segment all the customers in the database based on their behavior, mainly focusing on:

- Recency:
  When was the most recent order?
- Frequency:
  How often does the customer buy from the Business?
- Monetary:
  How much has the customer paid to the Business in total?

Normally for each Value we have a Score, then we combine these into some groups that we add the customers to.

However, in this Analysis, we will combine the Frequency and Monetary scores as they sort of decide the value of the customer (the customer importance)

And the recency as it is what decides how recent the customer and its status (recent, lost, etc.)

So, Based on these 2 values for each customer.

We will decide the customer segment.

Group name	Recency score	AVG(Frequency & Monetary) score
Champions	5	5
	5	4
	4	5
Potential Loyalists	5	2
	4	2
	3	3
	4	3
Loyal Customers	5	3
	4	4
	3	5
	3	4
Recent Customers	5	1
Promising	4	1
	3	1
Customers Needing Attention	3	2
	2	3
	2	2
At Risk	2	5
	2	4
	1	3
Cant Lose Them	1	5
	1	4
Hibernating	1	2
Lost	1	1

Now the Code:

```
WITH
    RFM_STATS AS (
        SELECT
            CUSTOMER_ID AS CUSTOMER,
            MAX(INVOICEDATE::DATE) AS "latest_order",
            CURRENT_DATE - MAX(INVOICEDATE::DATE) AS RECENCY,
            COUNT(DISTINCT INVOICE) AS FREQUENCY,
            ROUND(SUM(PRICE * QUANTITY)::NUMERIC, 2) AS MONETARY
        FROM
            TABLERETAIL T
        GROUP BY
            CUSTOMER
    ),
```

customer character varying (50)	latest_order date	recency integer	frequency bigint	monetary numeric
12747	2011-12-07	4472	11	4196.01
12748	2011-12-09	4470	210	33719.73
12749	2011-12-06	4473	5	4090.88

This CTE just gets the customer and its status that will be scored on later.

```
RFM_SCORES AS (
    SELECT
        CUSTOMER,
        "latest_order",
        RECENCY,
        NTILE(5) OVER (
            ORDER BY
                RECENCY DESC
        ) AS R_SCORE,
        FREQUENCY,
        NTILE(5) OVER (
            ORDER BY
                FREQUENCY
        ) AS F_SCORE,
        MONETARY,
        ROUND(
            PERCENT_RANK() OVER (
                ORDER BY
                    MONETARY
            )::NUMERIC,
            3
        ) AS M_SCORE
    FROM
        RFM_STATS
),
```

customer character varying (50)	latest_order date	recency integer	r_score integer	frequency bigint	f_score integer	monetary numeric	m_score numeric
12855	2010-12-02	4842	1	1	1	38.10	0.000
12967	2010-12-16	4828	1	2	2	1660.90	0.706
12829	2011-01-07	4806	1	2	3	293.00	0.156

Setting the Scores based on the customer stats.

```
R_FM_SCORES AS (
    SELECT
        CUSTOMER,
        RECENCY,
        FREQUENCY,
        MONETARY,
        M_SCORE,
        R_SCORE,
        ROUND((SUM(M_SCORE + F_SCORE) / 2)::NUMERIC) AS FM_SCORE
    FROM
        RFM_SCORES
    GROUP BY
        CUSTOMER,
        RECENCY,
        FREQUENCY,
        MONETARY,
        M_SCORE,
        R_SCORE
)
```

customer character varying (50)	recency integer	frequency bigint	monetary numeric	m_score numeric	r_score integer	fm_score numeric
12747	4472	11	4196.01	0.917	5	3
12748	4470	210	33719.73	0.991	5	3
12749	4473	5	4090.88	0.899	5	2

Combining the FREQUENCY and MONETARY Values

```sql
SELECT
    CUSTOMER,
    RECENCY,
    FREQUENCY,
    MONETARY,
    R_SCORE,
    FM_SCORE,
    CASE
        WHEN R_SCORE = 5
        AND FM_SCORE IN (5, 4) THEN 'champions'
        WHEN R_SCORE = 4
        AND FM_SCORE = 5 THEN 'champions'
        WHEN R_SCORE = 5
        AND FM_SCORE = 2 THEN 'potential loyalist'
        WHEN R_SCORE = 4
        AND FM_SCORE IN (2, 3) THEN 'potential loyalist'
        WHEN R_SCORE = 3
        AND FM_SCORE = 3 THEN 'potential loyalist'
        WHEN R_SCORE = 5
        AND FM_SCORE = 3 THEN 'loyal customers'
        WHEN R_SCORE = 4
        AND FM_SCORE = 4 THEN 'loyal customers'
        WHEN R_SCORE = 3
        AND FM_SCORE IN (5, 4) THEN 'loyal customers'
        WHEN R_SCORE = 5
        AND FM_SCORE = 1 THEN 'recent customer'
        WHEN R_SCORE = 4
        AND FM_SCORE = 1 THEN 'promising'
        WHEN R_SCORE = 3
        AND FM_SCORE = 1 THEN 'promising'
        WHEN R_SCORE = 2
        AND FM_SCORE IN (3, 2) THEN 'needs attention'
        WHEN R_SCORE = 3
        AND FM_SCORE = 2 THEN 'needs attention'
        WHEN R_SCORE = 2
        AND FM_SCORE IN (5, 4) THEN 'At Risk'
        WHEN R_SCORE = 1
        AND FM_SCORE = 3 THEN 'At Risk'
        WHEN R_SCORE = 1
        AND FM_SCORE IN (5, 4) THEN 'cant lose them'
        WHEN R_SCORE = 1
        AND FM_SCORE = 2 THEN 'Hibernating'
        WHEN R_SCORE = 2
        AND FM_SCORE = 1 THEN 'Hibernating'
        WHEN R_SCORE = 1
        AND FM_SCORE = 1 THEN 'lost'
    END AS CUSTOMER_SEGMENTATION
FROM
    R_FM_SCORES
ORDER BY
    CUSTOMER_SEGMENTATION;
```

customer character varying (50)	recency integer	frequency bigint	monetary numeric	r_score integer	fm_score numeric	customer_segmentation text
12830	4508	6	6814.64	3	2	needs attention
12822	4541	2	948.88	3	2	needs attention
12856	4478	6	2179.93	5	2	potential loyalist

And now the final Segmentation.

3- Answering Business Questions

- First Question)

  What is the Maximum Number of Consecutive Days a customer made Purchases?

```sql
WITH
    T1 AS (
        SELECT
            CUSTOMER,
            AMOUNT,
            ORDER_DATE,
            EXTRACT(
                DAY
                FROM
                    (
                        ORDER_DATE - (
                            ROW_NUMBER() OVER (
                                PARTITION BY
                                    CUSTOMER
                                ORDER BY
                                    ORDER_DATE
                            ) * INTERVAL '1 day'
                        )
                    )
            ) AS DIFFERENCE
        FROM
            TRANSACTIONS
    ),
    MAX_CON_DAYS AS (
        SELECT
            CUSTOMER,
            ORDER_DATE,
            COUNT(DIFFERENCE) OVER (
                PARTITION BY
                    DIFFERENCE,
                    CUSTOMER
            ) CONSECUTIVE_DAYS
        FROM
            T1
        WHERE
            AMOUNT > 0
    )
SELECT
    CUSTOMER,
    MAX(CONSECUTIVE_DAYS) MAX_CON_DAYS
FROM
    MAX_CON_DAYS
GROUP BY
    CUSTOMER
ORDER BY
    MAX_CON_DAYS DESC;
```

customer integer	max_con_days bigint
105358175	61
170731910	61
13503299	60

The Algorithm Behind:

First, we are getting a base to the days that are calculating on HOW?
Let's say we're having a customer who has the following Pattern.

1/1
2/1
3/1
5/1
6/1

We can add a row number first

1/1 - 1
2/1 - 2
3/1 - 3
5/1 - 4
6/1 - 5

Then subtract the date from the row number as difference:

1/1 - 1, 0
2/1 - 2, 0
3/1 - 3, 0
5/1 - 4, 1
6/1 - 5, 1

And this way we are grouping the consecutive days, and customer for using partition by and counting each group.

Then Removing the people who didn't purchase.

- Second Question)

On average, how many transactions does it take a customer to reach a spent threshold of 250 L.E?

```sql
WITH
    RUNNING_TOTAL AS (
        SELECT
            CUSTOMER,
            SUM(AMOUNT) OVER (
                PARTITION BY
                    CUSTOMER
                ORDER BY
                    ORDER_DATE
            ) RUNNING_TOTAL_SPENT,
            ORDER_DATE
        FROM
            TRANSACTIONS T
    ),
    TARGET_CUSTOMERS AS ( --customers who spent more than 250
        SELECT DISTINCT
            CUSTOMER
        FROM
            RUNNING_TOTAL
        WHERE
            RUNNING_TOTAL_SPENT > 250
    ),
    DAYS_BEFORE_250 AS (
        SELECT
            CUSTOMER,
            RUNNING_TOTAL_SPENT,
            ROW_NUMBER() OVER (
                PARTITION BY
                    CUSTOMER
                ORDER BY
                    ORDER_DATE
            ) AS ORDER_COUNT_BEFORE_250
        FROM
            RUNNING_TOTAL
        WHERE
            RUNNING_TOTAL_SPENT < 250
            AND CUSTOMER IN (
                SELECT
                    CUSTOMER
                FROM
                    TARGET_CUSTOMERS
            )
    ),
    AVG_DAYS_BEFORE_250 AS (
        SELECT
            CUSTOMER,
            MAX(ORDER_COUNT_BEFORE_250) AS NUMBER_OF_ORDERS_BEFORE_250
        FROM
            DAYS_BEFORE_250
        GROUP BY
            CUSTOMER
    )
SELECT
    ROUND(AVG(NUMBER_OF_ORDERS_BEFORE_250)::NUMERIC)
FROM
    AVG_DAYS_BEFORE_250
```

The Algorithm Behind:

We first calculate the running total per transaction for each customer. Then, we isolate the running total of transactions before reaching the 250 threshold per customer. For customers who surpass this threshold, we assign row numbers to transactions and identify the maximum number within each group. Finally, we compute the average of these maximum numbers across all customers, providing insights into transaction behavior around the 250 thresholds.