→ This is my second project for my SQL + Bi Portfolio.

# Project - II :)

## Revenue Segmentation Engine

Goal:)

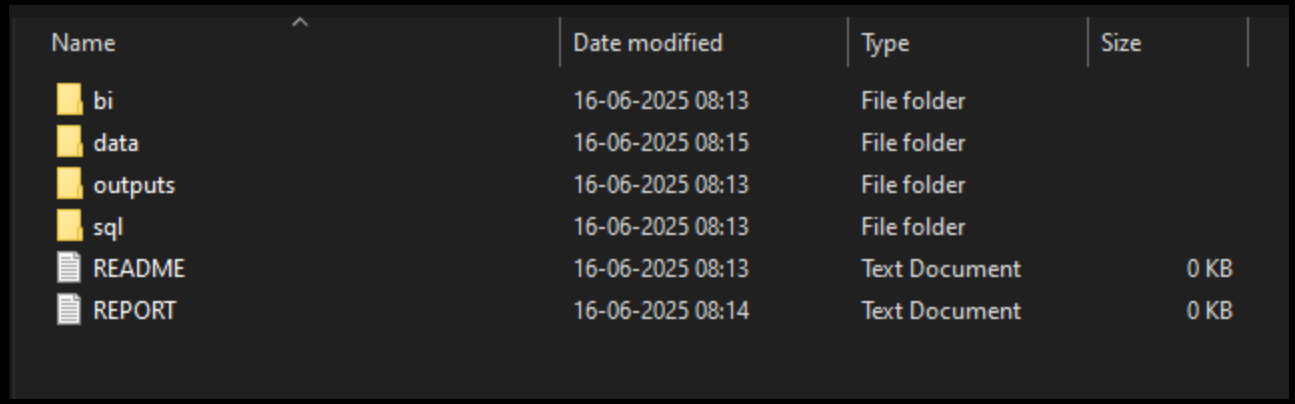
Analyze customer spend behavior, create tiers, find repeat buyers, and build a modular segmentation engine.

Key deliverable:

- first-orders : Identify customer signup date.
- revenue-summary : Aggregate spend, frequency, recency.
- tier-assignment : Assign revenue tier (e.g. Booze / Silver / Gold)
- repeat-buyers : Classify first-time vs repeat customers.
- churn-labels : Optional flags for inactive high-spenders.

# Concepts used (SQL)

- Case statements → for tiers assignment
- Percentile_cont() → Percentile-based tiers
- RANK()/NTILE() → Tiers distribution
- JOINs, GroupBY → Aggregation
- CTEs → Modular layers for reusability



| Name | Date modified | Type | Size |
|---|---|---|---|
| bi | 16-06-2025 08:13 | File folder | |
| data | 16-06-2025 08:15 | File folder | |
| outputs | 16-06-2025 08:13 | File folder | |
| sql | 16-06-2025 08:13 | File folder | |
| README | 16-06-2025 08:13 | Text Document | 0 KB |
| REPORT | 16-06-2025 08:14 | Text Document | 0 KB |

This is the folder structure I'm using.
This way is very efficient in overall upload to Git to user understanding.
(Industry level segregation)

# Power BI :-

- Revenue distribution by customer
- Active vs inactive customer segments
- Spend over time by tier
- Repeat vs first-time buyer heatmaps.

So, I've uploaded my csv. file, already converted "InvoiceDate" to timestamp from python.

```sql
-- Table: public.ecommerce_data_cleaned

-- DROP TABLE IF EXISTS public.ecommerce_data_cleaned;

CREATE TABLE IF NOT EXISTS public.ecommerce_data_cleaned
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.ecommerce_data_cleaned
    OWNER to postgres;
```

```sql
-- Table: public.ecommerce_data_cleaned

-- DROP TABLE IF EXISTS public.ecommerce_data_cleaned;

CREATE TABLE IF NOT EXISTS public.ecommerce_data_cleaned
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.ecommerce_data_cleaned
    OWNER to postgres;
```

Business Problem:

"We want to segment customers based on:

i) total spending

ii) purchase frequency

iii) recency

iv) tiers assignment like

- champion
- Loyal
- Potential
- Need Attention
- Require activation

v) Identify repeat buyers

```sql
-- Table: public.ecommerce_data_cleaned

-- DROP TABLE IF EXISTS public.ecommerce_data_cleaned;

CREATE TABLE IF NOT EXISTS public.ecommerce_data_cleaned
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.ecommerce_data_cleaned
    OWNER to postgres;
```

lets calculate these parameters first:

- total_orders
- total_quantity
- total_spend
- first_order_date
- last_order_date

→ these parameters are gonna aggregates customer behavior and gonna give us nice base for our future scenarios.

# Now we gonna create an very professional, real world applicable RFM model.

R F M → Monetary [How much total has been spent]

↓ ↓

Recency Frequency [How often they purchased]

↓

[How recently they've ordered something]

To create this we have 3 layers :-

- Metric layer : r_fm_score
- Scoring layer : r_fm_scored
- Segmentation : CASE + r_fm_total.

Output

```sql
-- Table: public.ecommerce_data_cleaned

-- DROP TABLE IF EXISTS public.ecommerce_data_cleaned;

CREATE TABLE IF NOT EXISTS public.ecommerce_data_cleaned
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.ecommerce_data_cleaned
    OWNER to postgres;
```

Let's focus on first layer :- rfm_base.

```sql
with rfm_base as (
select "CustomerID",
Now() - Max("InvoiceDate") as recency,
count(Distinct("InvoiceNo")) as frequency,
sum("Quantity" * "UnitPrize") as monetary
from public.ecommerce_data_cleaned
where "CustomerID" is Not Null
group by "CustomerID"
)
```

In this Metric layer we basically calculating all the necessary fields need for our rfm_model (Recency, frequency, monetary).

```sql
-- Table: public.ecommerce_data_cleaned

-- DROP TABLE IF EXISTS public.ecommerce_data_cleaned;

CREATE TABLE IF NOT EXISTS public.ecommerce_data_cleaned
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.ecommerce_data_cleaned
    OWNER to postgres;
```

& coding layer.

```
with rfm_scores as (
    select (*),
    ntile(5) over(order by recency Desc)
        as r_score
    ntile(5)over(order by frequency)
        as f_score
    ntile(5) over (order by monetary)
        as m_score
    from rfm_base
)
```

* ntile(5) will make partition of 5-20% segmentation like top 20% till bottom 20%.

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

Now we gonna construct our final layer : Segmentation layer.

```sql
select (*),
r_score + f_score + m_score as rfm_total,
case
    when r_score >= 4 and f_score >= 4
    and m_score >= 4 then 'champions'
    when r_score >= 3 and f_score >= 3,
    and m_score >= 3 then 'Loyal'
    when r_score >= 3 and m_score >= 3
    then 'Potential'
    when f_score >= 3 and m_score >= 3
    then 'Need attention'
```

Here we are using Case function to categorizing all the segments on the basis of filter metric.

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

Else 'Require Attention'
End as segment
from rfm_scores
);

Till here we made industry level, production ready, rfm_model now to access the full power or potential of rfm model we can also pair it with repeat_buyer_flag logic to mark customers who repeat in short time like within month.

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- Servers (2)
  - PostgreSQL 17
  - ecommerce_project
    - Databases (2)
      - Revenue Segmentation Project
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions
            - Materialized Views
            - Operators
            - Procedures
            - Sequences
            - Tables (1)
              - ecommerce_data_cleaned
                - Columns
                - Constraints
                - Indexes
                - RLS Policies
                - Rules
                - Triggers
            - Trigger Functions
            - Types
            - Views
        - Subscriptions
      - postgres
        - Casts
        - Catalogs

Dashboard | Properties | SQL | Statistics | Dependencies | Dependents | Processes | repeat_customer.sql* | postgres/postgres...

Revenue Segmentation Project/postgres@ecommerce_proj...

No limit

Query | Query History

Scratch Pad

```sql
WITH rfm_base AS (
  SELECT
    "CustomerID",
    NOW() - MAX("InvoiceDate") AS recency,
    COUNT(DISTINCT "InvoiceNo") AS frequency,
    SUM("Quantity" * "UnitPrice") AS monetary
  FROM public.ecommerce_data_cleaned
  GROUP BY "CustomerID"
),
rfm_scores AS (
  SELECT *,
    NTILE(5) OVER (ORDER BY recency DESC) AS r_score,
    NTILE(5) OVER (ORDER BY frequency) AS f_score,
    NTILE(5) OVER (ORDER BY monetary) AS m_score
  FROM rfm_base
),
rfm_segments AS (
  SELECT *,
    r_score + f_score + m_score AS rfm_total,
    CASE
      WHEN r_score >= 4 AND f_score >= 4 AND m_score >= 4 THEN 'Champion'
      WHEN r_score >= 3 AND f_score >= 3 AND m_score >= 3 THEN 'Loyal'
      WHEN r_score >= 3 AND m_score >= 3 THEN 'Potential'
      WHEN f_score >= 3 AND m_score >= 3 THEN 'Need Attention'
      ELSE 'Require Activation'
    END AS segment
  FROM rfm_scores
),
```

Total rows: 4373 | Query complete 00:00:05.109 | CRLF | Ln 70, Col 1

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

Now lets construct Repeat_buyer flag logic.

So our approach should be like this,

1.) firstly we should numerize the invoice Date in natural order by using Row_Number() window function, It'll just assign no. to the orders date from starting like

1 - 11/06/2010
2 - 19/07/2010

} - something like this.

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

→ Lets construct our logic into code.

```sql
Purchase as (
 select "CustomerID",
"InvoiceDate",
Row_Number() over (
 Partition by "CustomerID"
 order by "InvoiceDate")
as order_rank
from public.revenue_segmentation_
                project
),
```

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

-> Now, after purchase_date CTE we also have filter it further for only first_two_orders.

```sql
first_two_orders as (
  select (*) from purchase_dates
  where order_rank <= 2
)
```

-I think this is very straight forward we just want first & second order info of users.

```
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

⇒ Now we have to convert these rows first_order_date & second_order_date in to columns and for that we'll make pivoted_orders CTE and by using Max

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

```sql
pivoted_orders AS (
  SELECT "CustomerID",
  MAX(CASE WHEN order_rank = 1
    THEN "InvoiceDate" END) AS
    first_order_date,
  MAX(CASE WHEN order_rank = 2
    THEN "InvoiceDate" END) AS
    second_order_date
  FROM first_two_orders
  GROUP BY "CustomerID"
),
```

-> MAX(...)

- We use this because if case when returns Null for rows that don't match, so max() picks the non-null value for each CustomerID.

- without max(), we can't aggregate across rows in a group by.

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

-> I think this is straightforward.

- Now the important part of actually finding the repeat customers within 30 days interval.

```
repeat_flag as (
  select *,
    case
      when second_order_date
  is not Null and second_order_date
  - first_order_date <= Interval
      '30 days'
    then 1
    else 0
  end as is_repeat_buyer
from pivoted_orders)
```

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

Now we gonna do left join on rfm_segments with repeat_flag to get repeat customers along with their segment on CustomerID by both.

```sql
select s.*,
    r.is_repeat_buyer
from rfm_segments s
left join repeat_flag r on s."CustomerID"
    = r."CustomerID";
```

-> this the way we can get all rfm_segments fields

- This is final rfm_model code

```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

```sql
WITH rfm_base AS (
  SELECT
    "CustomerID",
    NOW() - MAX("InvoiceDate") AS recency,
    COUNT(DISTINCT "InvoiceNo") AS frequency,
    SUM("Quantity" * "UnitPrice") AS monetary
  FROM public.ecommerce_data_cleaned
  GROUP BY "CustomerID"
),
rfm_scores AS (
  SELECT *,
    NTILE(5) OVER (ORDER BY recency DESC) AS r_score,
    NTILE(5) OVER (ORDER BY frequency) AS f_score,
    NTILE(5) OVER (ORDER BY monetary) AS m_score
  FROM rfm_base
),
rfm_segments AS (
  SELECT *,
    r_score + f_score + m_score AS rfm_total,
    CASE
      WHEN r_score >= 4 AND f_score >= 4 AND m_score >= 4 THEN 'Champion'
      WHEN r_score >= 3 AND f_score >= 3 AND m_score >= 3 THEN 'Loyal'
      WHEN r_score >= 3 AND m_score >= 3 THEN 'Potential'
      WHEN f_score >= 3 AND m_score >= 3 THEN 'Need Attention'
      ELSE 'Require Activation'
    END AS segment
  FROM rfm_scores
),
```

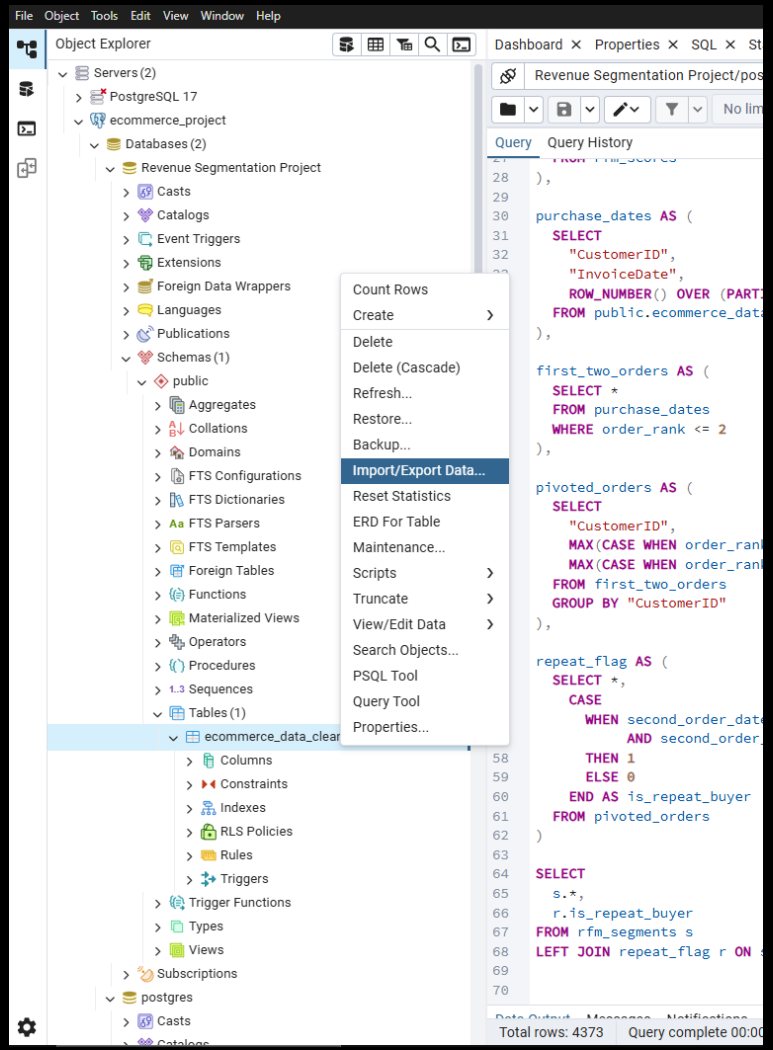
```sql
CREATE TABLE IF NOT EXISTS public.revenue_segmentation_project
(
    "InvoiceNo" text COLLATE pg_catalog."default",
    "StockCode" text COLLATE pg_catalog."default",
    "Description" text COLLATE pg_catalog."default",
    "Quantity" integer,
    "InvoiceDate" timestamp without time zone,
    "UnitPrice" numeric,
    "CustomerID" text COLLATE pg_catalog."default",
    "Country" text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.revenue_segmentation_project
    OWNER to postgres;
```

this is final repeat users code :

```sql
purchase_dates AS (
  SELECT
    "CustomerID",
    "InvoiceDate",
    ROW_NUMBER() OVER (PARTITION BY "CustomerID" ORDER BY "InvoiceDate") AS order_rank
  FROM public.ecommerce_data_cleaned
),

first_two_orders AS (
  SELECT *
  FROM purchase_dates
  WHERE order_rank <= 2
),

pivoted_orders AS (
  SELECT
    "CustomerID",
    MAX(CASE WHEN order_rank = 1 THEN "InvoiceDate" END) AS first_order_date,
    MAX(CASE WHEN order_rank = 2 THEN "InvoiceDate" END) AS second_order_date
  FROM first_two_orders
  GROUP BY "CustomerID"
),

repeat_flag AS (
  SELECT *,
    CASE
      WHEN second_order_date IS NOT NULL
           AND second_order_date - first_order_date <= INTERVAL '30 days'
      THEN 1
      ELSE 0
    END AS is_repeat_buyer
  FROM pivoted_orders
)

SELECT
  s.*,
  r.is_repeat_buyer
FROM rfm_segments s
LEFT JOIN repeat_flag r ON s."CustomerID" = r."CustomerID";
```

So here is our end of SQL part of the project.

→ Now export Data by right clicking on table name




→ then this will pop up and here you can choose the format either text or csv then export to your desired location