

SQLD / Enhance RFM analysis

SQL

-

2010~2011

가 가

가

가

E - Commerce Data , UCI Machine Learning Repository

(
features)

RFM

segmentation

)

column_name	missing_%
StockCode	0.0
Country	0.0
InvoiceNo	0.0
Quantity	0.0
InvoiceDate	0.0
<u>CustomerID</u>	<u>24.93</u>
UnitPrice	0.0
<u>Description</u>	<u>0.27</u>

행	column_name	missing_percenta...
1	StockCode	0.0
2	Country	0.0
3	InvoiceNo	0.0
4	Quantity	0.0
5	InvoiceDate	0.0
6	CustomerID	24.93
7	UnitPrice	0.0
8	Description	0.27

'85123A';
 StockCode / DISTINCT _description

작업 정보	결과	차트	JSON
행	description		
1	WHITE HANGING HEART T-LIGH...		
2	?		
3	wrongly marked carton 22804		
4	CREAM HANGING HEART T-LIG...		

--- 모든 컬럼에 대하여 COUNT[] 함수를 적용

```
-- SELECT
-- -- COUNT(*) as total_records
-- COUNT(InvoiceNo) AS count_InvoiceNo,
-- COUNT(StockCode) AS count_StockCode,
-- COUNT>Description) AS count_Description,
-- COUNT(Quantity) AS count_quantity,
-- COUNT(InvoiceDate) AS count_InvoiceDate,
-- COUNT(UnitPrice) AS count_UnitPrice,
-- COUNT(CustomerID) AS count_CustomerID,
-- COUNT(Country) AS count_Country

-- FROM `thefirst-464902.modulabs_project.data`;
-- 누락 된 값이 있는 행을 살펴본다.
-- SELECT
--     SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0
-- END) AS null_invoices,
--     SUM(CASE WHEN StockCode IS NULL THEN 1 ELSE 0
-- END) AS null_stockcodes,
--     SUM(CASE WHEN Description IS NULL THEN 1 ELSE 0
-- END) AS null_descriptions,
--     SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0
-- END) AS null_quantities,
--     SUM(CASE WHEN InvoiceDate IS NULL THEN 1 ELSE 0
-- END) AS null_invoicedates,
--     SUM(CASE WHEN UnitPrice IS NULL THEN 1 ELSE 0
-- END) AS null_unitprices,
--     SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0
-- END) AS null_customerids,
--     SUM(CASE WHEN Country IS NULL THEN 1 ELSE 0
-- END) AS null_countries
-- FROM `thefirst-464902.modulabs_project.data`;
-- 빈 문자열까지 포함한 결측치 체크
-- SELECT
--     SUM(CASE WHEN InvoiceNo IS NULL OR InvoiceNo
-- = '' THEN 1 ELSE 0 END) as missing_invoice,
--     SUM(CASE WHEN StockCode IS NULL OR StockCode
-- = '' THEN 1 ELSE 0 END) as missing_stockcode,
--     SUM(CASE WHEN Description IS NULL OR
-- Description = '' THEN 1 ELSE 0 END) as
-- missing_description,
--     -- SUM(CASE WHEN Quantity IS NULL OR Quantity
-- = '' THEN 1 ELSE 0 END) as missing_quantity,
```

```
--      -- SUM(CASE WHEN UnitPrice IS NULL OR
UnitPrice = '' THEN 1 ELSE 0 END) as
missing_unitprice,
--      -- SUM(CASE WHEN CustomerID IS NULL OR
CustomerID = '' THEN 1 ELSE 0 END) as
missing_customer,
--      SUM(CASE WHEN Country IS NULL OR Country = ''
THEN 1 ELSE 0 END) as missing_country
-- FROM `thefirst-464902.modulabs_project.data`;
--- 결측치의 비율을 확인해본다.
-- SELECT
--     'InvoiceNo' AS column_name,
--     ROUND(SUM(CASE WHEN InvoiceNo IS NULL THEN 1
ELSE 0 END) / COUNT(*) * 100, 2) AS
missing_percentage
-- FROM `thefirst-464902.modulabs_project.data`  
  

-- UNION ALL  
  

-- SELECT
--     'StockCode' AS column_name,
--     ROUND(SUM(CASE WHEN StockCode IS NULL THEN 1
ELSE 0 END) / COUNT(*) * 100, 2) AS
missing_percentage
-- FROM `thefirst-464902.modulabs_project.data`  
  

-- UNION ALL  
  

-- SELECT
--     'Description' AS column_name,
--     ROUND(SUM(CASE WHEN Description IS NULL THEN
1 ELSE 0 END) / COUNT(*) * 100, 2) AS
missing_percentage
-- FROM `thefirst-464902.modulabs_project.data`  
  

-- UNION ALL  
  

-- SELECT
--     'Quantity' AS column_name,
--     ROUND(SUM(CASE WHEN Quantity IS NULL THEN 1
ELSE 0 END) / COUNT(*) * 100, 2) AS
missing_percentage
```

```
-- FROM `thefirst-464902.modulabs_project.data`  
  
-- UNION ALL  
  
-- SELECT  
--     'InvoiceDate' AS column_name,  
--     ROUND(SUM(CASE WHEN InvoiceDate IS NULL THEN  
1 ELSE 0 END) / COUNT(*) * 100, 2) AS  
missing_percentage  
-- FROM `thefirst-464902.modulabs_project.data`  
  
-- UNION ALL  
  
-- SELECT  
--     'UnitPrice' AS column_name,  
--     ROUND(SUM(CASE WHEN UnitPrice IS NULL THEN 1  
ELSE 0 END) / COUNT(*) * 100, 2) AS  
missing_percentage  
-- FROM `thefirst-464902.modulabs_project.data`  
  
-- UNION ALL  
  
-- SELECT  
--     'CustomerID' AS column_name,  
--     ROUND(SUM(CASE WHEN CustomerID IS NULL THEN 1  
ELSE 0 END) / COUNT(*) * 100, 2) AS  
missing_percentage  
-- FROM `thefirst-464902.modulabs_project.data`  
  
-- UNION ALL  
  
-- SELECT  
--     'Country' AS column_name,  
--     ROUND(SUM(CASE WHEN Country IS NULL THEN 1  
ELSE 0 END) / COUNT(*) * 100, 2) AS  
missing_percentage  
-- FROM `thefirst-464902.modulabs_project.data`;  
  
-- SELECT  
--     StockCode, Description  
-- FROM `thefirst-464902.modulabs_project.data`  
-- WHERE StockCode = '85123A';
```

```
-- SELECT DISTINCT description
-- FROM `thefirst-464902.modulabs_project.data`
-- WHERE StockCode = '85123A';
-- DELETE FROM
`thefirst-464902.modulabs_project.data`

-- # -- 주요 필드가 NULL인 행 삭제

-- WHERE InvoiceNo IS NULL
--      OR StockCode IS NULL
--      OR UnitPrice IS NULL
--      OR Quantity IS NULL
--      OR Description IS NULL
--      OR Quantity = 0
--      OR InvoiceDate IS NULL
--      OR CustomerID IS NULL
--      OR Country IS NULL;
-- SELECT
-- COUNT(*) AS duplicated_count
-- FROM `thefirst-464902.modulabs_project.data`
-- GROUP BY InvoiceDate, StockCode, Description,
Quantity, InvoiceDate, UnitPrice, CustomerID,
Country
-- HAVING COUNT(*) >1;

-- --create replace / distinct : extract duplicated rows.
-- CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project.data` AS
-- SELECT DISTINCT *
-- FROM `thefirst-464902.modulabs_project.data`;
-- SELECT *
-- FROM `thefirst-464902.modulabs_project.data`
SELECT
COUNT(*) AS total_rows,
COUNT(DISTINCT InvoiceNo) AS unique_invoice,
COUNT(*) - COUNT(DISTINCT InvoiceNo) AS
duplicate_invoice_rows,
ROUND(COUNT(DISTINCT InvoiceNo) / COUNT(*) *100,2)
AS uniqueness_rate
FROM `thefirst-464902.modulabs_project.data`
```

```

~~~~~
total_rows /unique_invoice/duplicate_invoice_rows/
uniqueness_rate
401604 22190 379414 5.53
~~~~~ # Unique StockCode extracted from Rows
-- 숫자가 2개 이상인 StockCode만 유지
    AND LENGTH(StockCode) -
LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) > 1;

-- 2-2. 서비스/시스템 관련 정보 제거
DELETE FROM
`thefirst-464902.modulabs_project3.data_cleaned`
WHERE Description IN (
    'NEXT DAY CARRIAGE', 'HIGH RESOLUTION IMAGE',
    'POSTAGE', 'MANUAL',
    'BANK CHARGES', 'AMAZON FEE', 'COMMISSION',
    'DAMAGES', 'DISCOUNT',
    'ADJUSTMENT', 'TEST', 'CARRIAGE', 'SHIPPING',
    'DELIVERY', 'M', 'D', 'B', 'DOT', 'C2'
)
-- OR REGEXP_CONTAINS>Description, r'^[A-Z]{1,3}$')
OR LENGTH>Description) <= 1;

-- 2-3. 중복 행 제거
CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.data_cleaned` AS
SELECT DISTINCT *
FROM
`thefirst-464902.modulabs_project3.data_cleaned`;

-- 2-4. 정상 구매 데이터만 별도 저장 (RFM 분석용)
CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.purchase_data` AS
SELECT * EXCEPT(transaction_type)
FROM
`thefirst-464902.modulabs_project3.data_cleaned`
WHERE transaction_type = 'PURCHASE';

-- 2-5. 취소/반품 데이터 별도 저장
CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.cancellation_data`

```

```

` AS
SELECT * EXCEPT(transaction_type)
FROM
`thefirst-464902.modulabs_project3.data_cleaned`
WHERE transaction_type = 'CANCELLATION';

```

쿼리 완료됨

처리 위치: US X

쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	StockCode
9	21731
10	85116
11	22805
12	22774
13	71477
14	22375
15	22773
16	22727
17	84997B

페이지당 결과 수: 50 1 – 50 (전체 3684행) |< < > >|

작업 기록 C 새로고침

행	StockCode	sell_cnt
1	85123A	2065
2	22423	1894
3	85099B	1659
4	47566	1409
5	84879	1405
6	20725	1346
7	22720	1224
8	POST	1196
9	22197	1110

- 1,0 결과를 낸 스타크코드가 어떤건지를 확인해본다.

쿼리 결과

작업 정보	결과	차트	JSON	실행 세부정보	실행
행	StockCode		number_count		
1	POST		0		
2	M		0		
3	C2		1		
4	D		0		
5	BANK CHARGES		0		
6	PADS		0		
7	DOT		0		
8	CRUK		0		

StockCode 이상치 찾아보기

```

75 --     SELECT StockCode,
76 --         LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]')
77 --     FROM `thefirst-464902.modulabs_project.data`
78 -- )
79 -- WHERE number_count >= 0 AND number_count < 2;
80 DELETE FROM `thefirst-464902.modulabs_project.data`
81 WHERE StockCode IN (
82     SELECT StockCode
83     FROM (
84         SELECT StockCode,
85             LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]')
86         FROM `thefirst-464902.modulabs_project.data`
87     )
88     WHERE number_count >= 0 AND number_count < 2
89 );
90

```

쿼리 완료됨

쿼리 위치: US X

쿼리 결과

작업 정보 **결과** 실행 세부정보 실행 그래프

i 이 문으로 data의 행 1,915개가 삭제되었습니다.

3-1. 고객별 RFM 지표 계산 (unique_products 포함)

```

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.user_rfm` AS
WITH purchase_summary AS (
    SELECT
        CustomerID,
        -- 가장 최근 구매일 기준 Recency
        DATE_DIFF(CURRENT_DATE(), MAX(InvoiceDate), DAY)
    AS recency,
        -- 구매 빈도 (정상 거래만)
        COUNT(DISTINCT InvoiceNo) AS purchase_cnt,
        -- 총 구매 아이템 수 (정상 거래만)
        SUM(Quantity) AS item_cnt,
        -- 총 지출액 (정상 거래만)
        ROUND(SUM(UnitPrice * Quantity), 1) AS

```

```

user_total
    FROM
`thefirst-464902.modulabs_project3.purchase_data`
    GROUP BY CustomerID
),
unique_products AS (
    SELECT
        CustomerID,
        COUNT(DISTINCT StockCode) AS unique_products
    FROM
`thefirst-464902.modulabs_project3.purchase_data`
    GROUP BY CustomerID
)
SELECT
    ps.CustomerID,
    ps.recency,
    ps.purchase_cnt,
    ps.item_cnt,
    ps.user_total,
    ROUND(ps.user_total / ps.purchase_cnt, 1) AS
user_average,
    COALESCE(up.unique_products, 0) AS unique_products
-- COALESCE 추가로 안전성 확보
FROM purchase_summary ps
LEFT JOIN unique_products up
    ON ps.CustomerID = up.CustomerID;

```

-- 4. 재방문 주기 분석

```

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.customer_revisit_
cycle` AS
WITH customer_purchase_dates AS (
    SELECT
        CustomerID,
        InvoiceDate AS purchase_date
    FROM
`thefirst-464902.modulabs_project3.purchase_data`
    GROUP BY CustomerID, InvoiceDate
    ORDER BY CustomerID, purchase_date
),

```

```

customer_purchase_intervals AS (
    SELECT
        CustomerID,
        purchase_date,
        LAG(purchase_date) OVER (PARTITION BY CustomerID
ORDER BY purchase_date) AS prev_purchase_date,
        DATE_DIFF(
            purchase_date,
            LAG(purchase_date) OVER (PARTITION BY
CustomerID ORDER BY purchase_date),
            DAY
        ) AS days_between_purchases
    FROM customer_purchase_dates
)
SELECT
    CustomerID,
    COUNT(days_between_purchases) AS total_intervals,
    ROUND(AVG(days_between_purchases), 1) AS
avg_days_between_purchases,
    MIN(days_between_purchases) AS
min_days_between_purchases,
    MAX(days_between_purchases) AS
max_days_between_purchases,
    ROUND(STDDEV(days_between_purchases), 1) AS
stddev_days_between_purchases
FROM customer_purchase_intervals
WHERE days_between_purchases IS NOT NULL
GROUP BY CustomerID
HAVING COUNT(days_between_purchases) >= 1;

```

UnitPrice / MIN, MAX, AVE

작업 정보		결과		자트		JSON		실행 세부정보		실행 그래프	
행		min_unitprice		max_unitprice		avg_unitprice					
1		0.0		649.5		2.907415195928...					

UnitPrice /

쿼리 결과

결과 저장

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프
행	// cnt_quantity ▾	// min_quantity ▾	// max_quantity ▾	// avg_quantity ▾	
1	33	1	12540	420.515151515151...	

RFM 구하기

234
✓ 쿼리 완료됨
처리 위치: US X

쿼리 결과

결과 저장

다음에서 열기

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프
행	// InvoiceDay ▾	// CustomerID ▾	// Frequency ▾	// Monetary ▾	
1	4960	12423	118	1624.110000000...	
2	4960	12433	420	13375.869999999...	
3	4960	12518	119	1840.889999999...	
4	4960	12526	68	1172.66	
5	4960	12662	222	3511.079999999...	
6	4960	12680	49	790.81	
7	4960	12713	37	794.55	
8	4960	12748	4440	29819.990000000...	
9	4960	12985	78	1215.620000000...	

The most recent invoice day

```

8 -- FROM `thefirst-464902.modulabs_project.data`
9 -- GROUP BY CustomerID
0 -- ORDER BY InvoiceDay;
1 SELECT MAX(InvoiceDate) AS most_recent_purchase
2 FROM `thefirst-464902.modulabs_project.data`;
3
4

```

쿼리 완료됨

위 위치: US X

내리 결과

결과 저장 다음에서 열기

업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

most_recent_purchase ▾

1	2011-12-09 12:50:00 UTC
---	-------------------------

Each customerID/Recent Invoiced : Recency

```

3 SELECT
4   CustomerID,
5   DATE_DIFF(CURRENT_DATE(), DATE(DATE_TRUNC(MAX(InvoiceDate), DAY), DAY)) AS InvoiceDay
6   FROM `thefirst-464902.modulabs_project.data`
7   GROUP BY CustomerID
8   ORDER BY InvoiceDay;
9
0
1
2

```

실행 시 이 쿼리가 6.1MB를 처리합니다.

위 위치: US X

내리 결과

결과 저장 다음에서 열기

업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

CustomerID ▾ InvoiceDay ▾

1	12423	4960
2	12433	4960
3	12518	4960
4	12526	4960
5	12662	4960
6	12680	4960
7	12713	4960
8	12748	4960
9	12985	4960

ORDER by DESC

```
FROM `thefirst-464902.modulabs_project.data`  
l3 SELECT  
l4     CustomerID,  
l5     DATE_DIFF(CURRENT_DATE(), DATE(DATE_TRUNC(MAX(InvoiceDate), DAY), DAY) AS  
l6 FROM `thefirst-464902.modulabs_project.data`  
l7 GROUP BY CustomerID  
l8 ORDER BY InvoiceDay DESC;  
l9  
l10  
l11  
l12
```

실행 시 이 쿼리가 6.1MB를 처리합니다.

리 위치: US X

쿼리 결과

💾 결과 저장

업 정보				결과	차트	JSON	실행 세부정보	실행 그래프
		CustomerID	InvoiceDay					
1		12791	5333					
2		13065	5333					
3		13747	5333					
4		14142	5333					
5		14237	5333					
6		14729	5333					
7		15165	5333					
8		15350	5333					
9		16274	5333					

#가장 최근 일자(most_recent_date)와 유저별 마지막 구매일(InvoiceDay)
간의 차이를 계산

```

3 WITH MostRecent AS (
1   |   SELECT MAX(DATE(InvoiceDate)) AS most_recent_date
2   |   FROM `thefirst-464902.modulabs_project.data`
3 )
4 SELECT
5   |   CustomerID,
6   |   DATE_DIFF((SELECT most_recent_date FROM MostRecent), MAX(DATE(InvoiceDate)), DAY) AS recency
7 FROM `thefirst-464902.modulabs_project.data`
8 GROUP BY CustomerID
9 ORDER BY recency DESC;
3

```

실행 시 이 쿼리가 6.1MB를 처리합니다.

| 위치: US X

쿼리 결과

결과 저장 ▾ 다음에서 열기 ▾

업 정보 **결과** 차트 JSON 실행 세부정보 실행 그래프

	CustomerID	recency	
1	12791	373	
2	13065	373	
3	13747	373	
4	14142	373	
5	14237	373	
6	14729	373	
7	15165	373	
8	15350	373	
9	16274	373	

```

9 -- ORDER BY recency DESC;
0 CREATE OR REPLACE TABLE `thefirst-464902.modulabs_project.user_r` AS
1 WITH MostRecent AS (
2   |   SELECT MAX(DATE(InvoiceDate)) AS most_recent_date
3   |   FROM `thefirst-464902.modulabs_project.data`
4 )
5 SELECT
6   |   CustomerID,
7   |   DATE_DIFF((SELECT most_recent_date FROM MostRecent), MAX(DATE(InvoiceDate)), DAY) AS recency
8 FROM `thefirst-464902.modulabs_project.data`
9 GROUP BY CustomerID;
a

```

쿼리 완료됨

| 위치: US X

쿼리 결과

결과 저장 ▾ 다음에서 열기 ▾

업 정보 **결과** 실행 세부정보 실행 그래프

▶ 이 문으로 이름이 user_r인 새 테이블이 생성되었습니다.

테이블로 이동

```

268 -- FROM `thefirst-464902.modulabs_project.data`
269 -- GROUP BY CustomerID;
270 SELECT
271   CustomerID,
272   COUNT(DISTINCT InvoiceNo) AS purchase_cnt
273 FROM `thefirst-464902.modulabs_project.data`
274 GROUP BY CustomerID;
275
276

```

✓ 실행 시 이 쿼리가 6.11MB를 처리합니다.

처리 위치: US X

쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	CustomerID	purchase_cnt
1	12346	2
2	12347	7
3	12348	4
4	12349	1
5	12350	1
6	12352	8
7	12353	1
8	12354	1
9	12355	1

```

274 -- GROUP BY CustomerID;
275 SELECT
276   CustomerID,
277   SUM(Quantity) AS item_cnt -- Total quantity of items purchased by each customer
278 FROM `thefirst-464902.modulabs_project.data`
279 GROUP BY CustomerID
280 ORDER BY item_cnt DESC;
281
282

```

✓ 쿼리 완료됨

처리 위치: US X

쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	CustomerID	item_cnt
1	14646	196556
2	12415	76946
3	14911	76823
4	17450	69021
5	18102	64124
6	17511	63014
7	13694	61904
8	14298	58021
9	14156	56896

- 5. 취소 분석 (수정된 버전)

CREATE OR REPLACE TABLE

`thefirst-464902.modulabs_project3.customer_cancel_analysis` AS

```

WITH customer_transaction_summary AS (
    SELECT
        CustomerID,
        -- 전체 거래 건수 (구매 + 취소)
        COUNT(*) AS total_transactions,
        -- 취소 건수
        SUM(CASE WHEN transaction_type = 'CANCELLATION'
THEN 1 ELSE 0 END) AS cancel_frequency,
        -- 정상 구매 건수
        SUM(CASE WHEN transaction_type = 'PURCHASE' THEN
1 ELSE 0 END) AS normal_transactions,
        -- 취소된 총 수량
        SUM(CASE WHEN transaction_type = 'CANCELLATION'
THEN Quantity ELSE 0 END) AS
total_cancelled_quantity,
        -- 취소된 총 금액
        ROUND(SUM(CASE WHEN transaction_type =
'CANCELLATION' THEN UnitPrice * Quantity ELSE 0
END), 2) AS total_cancelled_amount
    FROM
`thefirst-464902.modulabs_project3.data_cleaned`
    GROUP BY CustomerID
)
SELECT
    CustomerID,
    total_transactions,
    cancel_frequency,
    normal_transactions,
    total_cancelled_quantity,
    total_cancelled_amount,
    -- 취소 비율 계산
    ROUND(
        CASE
            WHEN total_transactions > 0 THEN
cancel_frequency / total_transactions * 100
            ELSE 0
        END,
        2
    ) AS cancel_rate,
    -- 취소 성향 분류
    CASE
        WHEN cancel_frequency = 0 THEN '취소 없음'

```

```

        WHEN cancel_frequency <= 2 AND
cancel_frequency / total_transactions <= 0.1 THEN
'저빈도 취소'
        WHEN cancel_frequency <= 5 AND
cancel_frequency / total_transactions <= 0.2 THEN
'중빈도 취소'
        ELSE '고빈도 취소'
    END AS cancel_behavior_type
FROM customer_transaction_summary;

```

customer_rfm_enhanced 테이블 생성

```

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.customer_rfm_enha
nced` AS
SELECT
    rfm.CustomerID,
    rfm.recency,
    rfm.purchase_cnt,
    rfm.item_cnt,
    rfm.user_total,
    rfm.user_average,
    rfm.unique_products, -- 이 컬럼이 포함되어야 함
    -- 재방문 주기 정보
    COALESCE(rc.avg_days_between_purchases, NULL) AS
avg_revisit_days,
    COALESCE(rc.total_intervals, 0) AS
purchase_intervals_count,
    COALESCE(rc.min_days_between_purchases, NULL) AS
min_revisit_days,
    COALESCE(rc.max_days_between_purchases, NULL) AS
max_revisit_days,
    COALESCE(rc.stddev_days_between_purchases, NULL)
AS stddev_revisit_days,
    -- 취소 정보
    COALESCE(ca.cancel_frequency, 0) AS
cancel_frequency,
    COALESCE(ca.cancel_rate, 0.00) AS cancel_rate,
    COALESCE(ca.total_cancelled_quantity, 0) AS
total_cancelled_quantity,
    COALESCE(ca.total_cancelled_amount, 0.00) AS
total_cancelled_amount,
    COALESCE(ca.cancel_behavior_type, '취소 없음') AS

```

```

cancel_behavior_type,
-- 재방문 패턴 분류
CASE
    WHEN rc.avg_days_between_purchases IS NULL THEN
'일회성 고객'
    WHEN rc.avg_days_between_purchases <= 7 THEN '주
간 고객'
    WHEN rc.avg_days_between_purchases <= 30 THEN
'월간 고객'
    WHEN rc.avg_days_between_purchases <= 90 THEN
'계절 고객'
    ELSE '비정기 고객'
END AS revisit_pattern
FROM `thefirst-464902.modulabs_project3.user_rfm` rfm
LEFT JOIN
`thefirst-464902.modulabs_project3.customer_revisit_
cycle` rc
    ON rfm.CustomerID = rc.CustomerID
LEFT JOIN
`thefirst-464902.modulabs_project3.customer_cancel_a
nalysis` ca
    ON rfm.CustomerID = ca.CustomerID;

```

User_rf 총 합한 테이블로 생성

```

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.user_data` AS
WITH customer_all_metrics AS (
    SELECT
        rfm_enhanced.CustomerID,
        rfm_enhanced.purchase_cnt AS frequency,
        rfm_enhanced.item_cnt,
        rfm_enhanced.recency,
        rfm_enhanced.user_total AS monetary,
        rfm_enhanced.user_average,
        rfm_enhanced.unique_products,
        -- 재방문 주기 정보
        rfm_enhanced.avg_revisit_days,
        rfm_enhanced.purchase_intervals_count,
        rfm_enhanced.min_revisit_days,
        rfm_enhanced.max_revisit_days,

```

```

rfm_enhanced.stddev_revisit_days,
rfm_enhanced.revisit_pattern,
-- 취소 정보
rfm_enhanced.cancel_frequency,
rfm_enhanced.cancel_rate,
rfm_enhanced.total_cancelled_quantity,
rfm_enhanced.total_cancelled_amount,
rfm_enhanced.cancel_behavior_type,
-- RFM 점수 계산
CASE
    WHEN rfm_enhanced.recency <= 30 THEN 5
    WHEN rfm_enhanced.recency <= 60 THEN 4
    WHEN rfm_enhanced.recency <= 120 THEN 3
    WHEN rfm_enhanced.recency <= 240 THEN 2
    ELSE 1
END AS R_score,
CASE
    WHEN rfm_enhanced.purchase_cnt >= 20 THEN 5
    WHEN rfm_enhanced.purchase_cnt >= 10 THEN 4
    WHEN rfm_enhanced.purchase_cnt >= 5 THEN 3
    WHEN rfm_enhanced.purchase_cnt >= 2 THEN 2
    ELSE 1
END AS F_score,
CASE
    WHEN rfm_enhanced.user_total >= 2000 THEN 5
    WHEN rfm_enhanced.user_total >= 1000 THEN 4
    WHEN rfm_enhanced.user_total >= 500 THEN 3
    WHEN rfm_enhanced.user_total >= 200 THEN 2
    ELSE 1
END AS M_score
FROM
`thefirst-464902.modulabs_project3.customer_rfm_enha
nced` rfm_enhanced
)
SELECT
*, 
CONCAT(R_score, F_score, M_score) AS RFM_code,
-- 고객 세그먼트 분류
CASE
    WHEN R_score >= 4 AND F_score >= 4 AND M_score
>= 4 THEN 'VIP 고객'
    WHEN R_score >= 4 AND F_score >= 3 THEN '활성 고

```

```

    고객'
        WHEN R_score <= 2 AND F_score >= 4 THEN '휴면 우수
고객'
        WHEN R_score <= 2 AND F_score <= 2 AND M_score
>= 4 THEN '고가치 휴면 고객'
        WHEN R_score >= 4 AND F_score <= 2 THEN '신규 고
객'
        WHEN R_score >= 3 AND F_score >= 2 AND M_score
>= 2 THEN '잠재 고객'
        ELSE '일반 고객'
    END AS customer_segment,
    -- 상품 다양성 지수
    ROUND(unique_products / frequency, 2) AS
product_diversity_index,
    -- 평균 상품별 구매량
    ROUND(item_cnt / unique_products, 1) AS
avg_quantity_per_product,
    -- 다음 구매 예상일 계산
CASE
    WHEN avg_revisit_days IS NOT NULL THEN
        DATE_ADD(CURRENT_DATE(), INTERVAL
CAST(avg_revisit_days - recency AS INT64) DAY)
    ELSE NULL
END AS predicted_next_purchase_date,
    -- 재방문 예측 상태
CASE
    WHEN avg_revisit_days IS NULL THEN '예측 불가'
    WHEN recency > avg_revisit_days * 1.5 THEN '재방
문 지연'
    WHEN recency > avg_revisit_days THEN '재방문 예상'
    ELSE '정상 주기'
END AS revisit_status,
    -- 고객 리스크 레벨
CASE
    WHEN cancel_rate >= 30 THEN '고위험 고객'
    WHEN cancel_rate >= 15 THEN '중위험 고객'
    WHEN cancel_rate >= 5 THEN '저위험 고객'
    ELSE '안전 고객'
END AS risk_level,
    -- 고객 가치 등급
CASE
    WHEN monetary >= 2000 AND frequency >= 10 AND

```

```

unique_products >= 20 THEN '최고가치 고객'
    WHEN monetary >= 1000 AND frequency >= 5 AND
unique_products >= 10 THEN '고가치 고객'
    WHEN monetary >= 500 OR frequency >= 3 OR
unique_products >= 5 THEN '중가치 고객'
    ELSE '저가치 고객'
END AS customer_value_tier
FROM customer_all_metrics;

```

Screenshot of a data visualization tool showing a table named 'user_rf'. The table has columns: 행 (Row), CustomerID, purchase_cnt, item_cnt, recency. The '미리보기' (Preview) tab is selected.

행	CustomerID	purchase_cnt	item_cnt	recency
1	12713	1	505	0
2	15520	1	314	1
3	13298	1	96	1
4	13436	1	76	1
5	14569	1	79	1
6	14204	1	72	2
7	15471	1	256	2
8	15195	1	1404	2
9	12478	1	233	3
10	12442	1	181	3
11	12650	1	250	3
12	16569	1	93	3
13	16528	1	171	3
14	15318	1	642	3
15	14578	1	240	3
16	15992	1	17	3
17	17914	1	457	3
18	16597	1	184	4
19	13790	1	748	4
20	18015	1	157	4
21	14219	1	78	4
22	15097	1	170	4
23	12367	1	172	4

Monetary 고객이 지불한 총 금액

```
312 SELECT
313   Custo -- 리소스를 나란히 업니다. (cmd + click) 3 total_expenditure -- Total expenditure rounded to 1 decimal
314   ROUND(
315     FROM `thefirst-464902.modulabs_project.data`
316   GROUP BY CustomerID;
```

▶ 실행 시 이 쿼리가 9.15MB를 처리합니다.

처리 위치: US X

쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

	CustomerID	total_expenditure	
1	12346	0.0	
2	12347	4310.0	
3	12348	1437.2	
4	12349	1457.5	
5	12350	294.4	
6	12352	1265.4	
7	12353	89.0	
8	12354	1079.4	
9	12355	459.4	

```
|6 GROUP BY CustomerID  
|7 ORDER BY total_expenditure;  
|8  
|9  
|10  
|11  
|12  
|13
```

쿼리 완료됨

리 위치: US

쿼리 결과

업 정보 결과 차트 JSON 실행 세부정보 실행 그래

	CustomerID	total_expenditure
1	14213	-1192.2
2	15823	-811.9
3	16742	-464.9
4	16252	-295.1
5	12666	-227.4
6	17307	-152.6
7	17548	-141.5
8	15728	-134.8
9	13958	-102.5

RFM 통합 테이블 출력하기

user_rfm

쿼리 다음에서 열기 ▾ 공유 복사 스냅샷 삭제

스키마 세부정보 미리보기 테이블 탐색기 미리보기 통계 계보 데이터 프로필

필터 속성 이름 또는 값 입력

필드 이름	유형	모드	키	대조	기본값	정책 태그
CustomerID	INTEGER	NULLABLE	-	-	-	-
purchase_cnt	INTEGER	NULLABLE	-	-	-	-
item_cnt	INTEGER	NULLABLE	-	-	-	-
recency	INTEGER	NULLABLE	-	-	-	-
user_total	FLOAT	NULLABLE	-	-	-	-
user_average	FLOAT	NULLABLE	-	-	-	-

스키마 수정 행 액세스 정책 보기

위와 같은 과정을 거쳐서 생성된 **최종 user_rfm 테이블을 출력해주겠습니다.**

행	CustomerID	purchase_cnt	item_cnt	recency	
1	12713	1	505	0	
2	15520	1	314	1	
3	13298	1	96	1	
4	13436	1	76	1	
5	14569	1	79	1	
6	14204	1	72	2	
7	15471	1	256	2	
8	15195	1	1404	2	
9	12478	1	233	3	
10	12442	1	181	3	
11	12650	1	250	3	
12	16569	1	93	3	
13	16528	1	171	3	
14	15318	1	642	3	
15	14578	1	240	3	
16	15992	1	17	3	
17	17914	1	457	3	
18	16597	1	184	4	
19	13790	1	748	4	
20	18015	1	157	4	
21	14219	1	78	4	
22	15097	1	170	4	
23	12367	1	172	4	
24	17383	1	148	4	

페이지당 결과 수: 50 ▼ 1 – 50 (전체 4362행)

고유한 유저의 수는 몇 명인가요? 4362
pattern of purchase by customerID

```

338
339 -- 고객 세그먼트 / 구매 패턴 설정 1) 고객 별로 구매한 상품들의 고유한 수를 계산
340 SELECT
341   · CustomerID,
342   · COUNT(DISTINCT StockCode) AS unique_products_purchased
343 FROM `thefirst-464902.modulabs_project.data`
344 GROUP BY CustomerID
345 ORDER BY unique_products_purchased DESC;
346
347

```

쿼리 완료됨

처리 위치: US X

쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	CustomerID	unique_products...
1	14911	1791
2	12748	1767
3	17841	1330
4	14096	1118
5	14298	884
6	14606	829
7	14769	717
8	14156	714
9	14646	699

쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	15657	1	24	22	30.0	30.0	1
2	16454	1	2	64	5.9	5.9	1
3	14351	1	12	164	51.0	51.0	1
4	13703	1	10	318	99.5	99.5	1
5	18068	1	6	289	101.7	101.7	1
6	13366	1	144	50	56.2	56.2	1
7	13747	1	8	373	79.6	79.6	1
8	18141	1	-12	360	-35.4	-35.4	1
9	14576	1	12	372	35.4	35.4	1

CustomerID purchase_cnt item_cnt recency user_total
user_average unique_products

15657	1	24	22	30.0	30.0	1
16454	1	2	64	5.9	5.9	1
14351	1	12	164	51.0	51.0	1
13703	1	10	318	99.5	99.5	1
18068	1	6	289	101.7	101.7	1
13366	1	144	50	56.2	56.2	1
13747	1	8	373	79.6	79.6	1
18141	1	-12	360	-35.4	-35.4	1
14576	1	12	372	35.4	35.4	1
16765	1	4	294	34.0	34.0	1

[[오류 해결 과정]]

```
*****문제 발생 : 중간 ["""DELETE FROM
`thefirst-464902.modulabs_project.data`(
    -- WHERE StockCode IN (
        -- SELECT StockCode
        -- FROM (
            -- SELECT StockCode,
            -- LENGTH(StockCode) -
LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS
number_count
        -- FROM
`thefirst-464902.modulabs_project.data`(
            -- )
            -- WHERE number_count >= 0 AND
number_count < 2
        -- );"""]
```

: 문제 이해를 잘못해서, 0과 1의 결과를[숫자를 제외한 문자형개수]제외를 하고 남은 데이터를 저장해야하는데 모두 삭제함.

[수정]

```
SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode,
r'[0-9]', '')) AS number_count
FROM `thefirst-464902.modulabs_project.data`'
WHERE LENGTH(StockCode) -
    LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) >1;
```

[데이터 일부분만 교체를 함에 있어서 추후 잘못 정제된 데이터로 인한 손실을 막을 수 있다.]

CREATE OR REPLACE TABLE

```
`thefirst-464902.modulabs_project.data` AS
SELECT *
FROM `thefirst-464902.modulabs_project.data`'
WHERE LENGTH(StockCode) -
    LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) > 1; --
Exclude rows with 0 or 1 digits
Description 살펴보기
```

[이번엔 Description 컬럼을 더 깊이 있게 살펴보겠습니다.]

먼저 데이터셋에서 고유한 Description 별 출현 빈도를 계산하고 상위 30개

```
SELECT Description, COUNT(*) AS description_cnt
FROM `thefirst-464902.modulabs_project.data`'
```

```
GROUP BY Description  
ORDER BY description_cnt DESC;
```

대소문자가 혼합된 Description이 있는지 확인

```
SELECT DISTINCT Description  
FROM project_name.modulabs_project.data  
WHERE REGEXP_CONTAINS>Description, r'[a-z]');
```

'3 TRADITIONAL BISCUIT CUTTERS SET'는 모두 대문자로 작성된 거 아냐?"라고 생각할 수 있지만 TRADITIONA'l'에서 'l'은 대문자 I(아이)가 아니라 소문자 l(엘)입니다.

실제 제품 정보와 관련이 없어 보입니다. 아마도 다른 유형의 정보나 서비스 세부사항을 나타내는 것일 수도 있습니다. 이를 처리하기 위한 몇 가지 전략들을 생각해볼 수 있습니다.

1. '**Next Day Carriage**'와 '**High Resolution Image**'와 같은 서비스 관련 정보를 포함하는 행들을 제거합니다.
2. 대소문자를 혼합해서 사용하는 경우, 대문자로 표준화하여 데이터셋 전체에서 일관성을 유지할 수 있습니다. 이는 대소문자에 의한 중복 항목의 가능성을 줄이는 데에도 도움이 될 것입니다.

이러한 전략을 선택함으로써, 데이터셋의 품질을 향상시킬 수 있고 프로젝트의 분석 단계에 더 적합한 데이터셋을 완성할 수 있습니다.

[서비스 관련 정보를 포함하는 행들을 제거]

```
DELETE  
FROM project_name.modulabs_project.data  
WHERE Description IN('Next DayCarriage', 'High Resolution Image');
```

[대소문자를 혼합하고 있는 데이터를 대문자로 표준화]

```
CREATE OR REPLACE TABLE `project_name.modulabs_project.data`  
AS  
SELECT  
    * EXCEPT (Description),  
    UPPER>Description AS Description -- Convert Description to  
    uppercase  
FROM `project_name.modulabs_project.data`;
```

```
-- 서비스/시스템 관련 정보 삭제
DELETE FROM `thefirst-464902.modulabs_project3.data`
WHERE Description IN (
    'NEXT DAY CARRIAGE', 'HIGH RESOLUTION IMAGE', 'POSTAGE',
    'BANK CHARGES', 'AMAZON FEE', 'COMMISSION', 'DAMAGES', 'DISCOUNT',
    'ADJUSTMENT', 'TEST', 'CARRIAGE', 'SHIPPING', 'DELIVERY', 'M', 'D', 'B', 'DOT', 'C2'
)
OR Description LIKE '%CARRIAGE%'
OR Description LIKE '%POSTAGE%'
OR Description LIKE '%SHIPPING%'
OR Description LIKE '%DELIVERY%'
OR Description LIKE '%FEE%'
OR Description LIKE '%CHARGES%'
OR Description LIKE '%COMMISSION%'
OR LENGTH>Description) <= 1;

-- Description 대문자 변환
```

[UnitPrice 살펴보기]

UnitPrice에서 이상치를 찾아봅시다. 최솟값, 최댓값, 평균 데이터를 확인해 봄으로써, 단위 가격의 요약 통계량

UnitPrice의 **최솟값, 최댓값, 평균**

SELECT

```
MIN(UnitPrice) AS min_unitprice,
MAX(UnitPrice) AS max_unitprice,
AVG(UnitPrice) AS avg_unitprice
FROM `project_name.modulabs_project.data`;
```

```
-- 2-6. UnitPrice 정리
-- UnitPrice 분포 확인

SELECT
    MIN(UnitPrice) AS min_unitprice,
    MAX(UnitPrice) AS max_unitprice,
    AVG(UnitPrice) AS avg_unitprice
FROM `thefirst-464902.modulabs_project3.data`;

-- UnitPrice가 0인 데이터 확인

SELECT
    COUNT(Quantity) AS cnt_quantity,
    MIN(Quantity) AS min_quantity,
    MAX(Quantity) AS max_quantity,
    AVG(Quantity) AS avg_quantity
```

완료됨

| 결과

정보	결과	차트	JSON	실행 세부정보	실행 그래프
	// min_unitprice // max_unitprice // avg_unitprice //			0.0 649.5 2.905632575814...	

MIN 0원인 데이터가 존재한다는

[**MIN 0원인 거래의 개수, 구매 수량(Quantity)의 최솟값, 최댓값, 평균**]

SELECT

```
COUNT(Quantity) AS cnt_quantity,
MIN(Quantity) AS min_quantity,
MAX(Quantity) AS max_quantity,
AVG(Quantity) AS avg_quantity
FROM `project_name.modulabs_project.data` 
WHERE UnitPrice = 0;
```

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프
행	// cnt_quantity // min_quantity // max_quantity // avg_quantity //			1 33 1 12540 420.5151515151...	

UnitPrice가 0인 행의 수는 33개로 비교적 적음을 알 수 있습니다. 구매 수량(Quantity)은 최소 1개부터 최대 12,540개에 이르기까지 굉장히 큰 편차를 가집니다.

데이터의 수가 적은 걸 보니 무료 제품이라기보다 데이터 오류일 가능성이 더 높을 것 같습니다. 그래서 이 데이터(UnitPrice = 0)를 제거하고 일관된 데이터셋을 유지하도록 하겠습니다.

```

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project.data` AS
SELECT *
FROM `thefirst-464902.modulabs_project.data`
WHERE UnitPrice != 0;

```

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프	
행	invoice_prefix	record_count	negative_quantity...	avg_quantity	min_quantity	max_quantity
1	5	388326	0	13.15	1	80995
2	C	8382	8382	-31.85	-80995	-1

[-- 0제거 이후에 최소값 순위10개보기.

```

SELECT DISTINCT UnitPrice
FROM `thefirst-464902.modulabs_project3.data`
WHERE UnitPrice > 0
ORDER BY UnitPrice ASC
LIMIT 10;
]

```

작업 정보	결과	차트	JSON
행	UnitPrice		
1	0.03		
2	0.04		
3	0.06		
4	0.07		
5	0.08		
6	0.09		
7	0.1		
8	0.11		
9	0.12		
10	0.14		

1-7. RFM 스코어

Recency

'마지막 구매일로부터 현재까지 경과한 일수' 낮은 값일수록 고객이 최근에 구매를 했음을 의미하며, 제품이나 서비스에 더 관심을 보인다고 예측할 수 있습니다.

InvoiceDate 컬럼을 연월일 자료형으로 변경

행	InvoiceDate	InvoiceDate_conv...	original_type	converted_type
1	2010-12-01	2010-12-01	DATE	DATE
2	2010-12-01	2010-12-01	DATE	DATE
3	2010-12-03	2010-12-03	DATE	DATE
4	2010-12-05	2010-12-05	DATE	DATE
5	2010-12-06	2010-12-06	DATE	DATE

필터 속성 이름 또는 값 입력							
<input type="checkbox"/> 필드 이름	유형	모드	키	대조	기본값	정책 태그	설명
<input type="checkbox"/> CustomerID	INTEGER	NULABLE	-	-	-	-	-
<input type="checkbox"/> recency	INTEGER	NULABLE	-	-	-	-	-

SELECT

```
-- Calculate Recency (days since the last purchase)
DATE_DIFF(CURRENT_DATE(),
DATE(DATE_TRUNC(MAX(InvoiceDate), DAY)), DAY) AS InvoiceDay,
CustomerID,
COUNT(InvoiceNo) AS Frequency,
SUM(UnitPrice * Quantity) AS Monetary
FROM `thefirst-464902.modulabs_project.data`
GROUP BY CustomerID
ORDER BY InvoiceDay;
```

가장 최근 구매일자 확인

작업 정보	결과	차트	JSON	실행 세부정보	실행
행	most_recent_purc...				
1	2011-12-09				

최종 구매일로부터 꽤 오랜 시간이 지난 데이터라는 특성이 있습니다. 그래서 이번 프로젝트에서는 모든 고객들을 통틀어 가장 최근 구매 일자를 기준으로 **Recency**를 구하려고 합니다.

SELECT

```

MAX(InvoiceDate) AS most_recent_date, -- Most recent purchase
date for each customer
DATE_DIFF(CURRENT_DATE(),
DATE(DATE_TRUNC(MAX(InvoiceDate), DAY)), DAY) AS InvoiceDay,
-- Recency: days since most recent purchase
CustomerID, -- Assuming CustomerID is the identifier for each
customer
COUNT(InvoiceNo) AS Frequency, -- Frequency: number of
purchases
SUM(UnitPrice * Quantity) AS Monetary -- Monetary: total spend
FROM ``
GROUP BY CustomerID
ORDER BY InvoiceDay;
```

```

258 -- 3-4. CURRENT_DATE() 기준 Recency 계산 (문서 요구사항)
259 -- 최종 구매일로부터 꽤 오랜 시간이 지난 데이터이므로 현재 날짜 기준으로 계산
260 SELECT
261   MAX(InvoiceDate) AS most_recent_date,
262   DATE_DIFF(CURRENT_DATE(), MAX(InvoiceDate), DAY) AS InvoiceDay,
263   CustomerID,
264   COUNT(InvoiceNo) AS Frequency,
265   SUM(UnitPrice * Quantity) AS Monetary
266 FROM `thefirst-464902.modulabs_project3.data`
267 GROUP BY CustomerID
268 ORDER BY InvoiceDay;

```

▶ 쿼리 완료됨

▶ 쿼리 결과

결과

	작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프
1		most_recent_date	InvoiceDay	CustomerID	Frequency	Monetary
1	2011-12-09		4961	13069	466	3694.419999999...
2	2011-12-09		4961	12748	4401	29470.68000000...
3	2011-12-09		4961	14446	275	1003.00000000...
4	2011-12-09		4961	12518	115	1779.689999999...
5	2011-12-09		4961	15804	273	3848.549999999...
6	2011-12-09		4961	12433	415	12953.46999999...
7	2011-12-09		4961	17364	409	4437.23000000...
8	2011-12-09		4961	12423	115	1578.209999999...
9	2011-12-09		4961	15311	2475	59243.24000000...
10	2011-12-09		4961	14051	215	15477.33999999...
11	2011-12-09		4961	13777	217	25758.1
12	2011-12-09		4961	16558	468	8167.16000000...
13	2011-12-09		4961	17428	341	17041.00999999...
14	2011-12-09		4961	13113	278	10523.65000000...

```

263 -- 4-1. 고객별 거래 건수 계산
264 SELECT
265   · CustomerID,
266   · COUNT(DISTINCT InvoiceNo) AS purchase_cnt
267 FROM `thefirst-464902.modulabs_project3.data`
268 GROUP BY CustomerID
269 ORDER BY purchase_cnt DESC

```

✓ 쿼리 완료됨

쿼리 결과

작업 정보	결과	차트	JSON	실행 세부정보	실행
행	CustomerID		purchase_cnt		
1	14911		242		
2	12748		217		
3	17841		169		
4	14606		125		
5	13089		118		
6	15311		118		
7	12971		88		
8	13408		75		
9	14646		73		
10	16029		66		

유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장해 주겠습니다.
SELECT

```

CustomerID,
DATE_DIFF(CURRENT_DATE(),
DATE(DATE_TRUNC(MAX(InvoiceDate), DAY)), DAY) AS InvoiceDay
FROM `thefirst-464902.modulabs_project3.data`
GROUP BY CustomerID

```

```
ORDER BY InvoiceDay DESC;
```

```
9   ORDER BY InvoiceDay;  
10  -- 3-4. 유저별 가장 큰 InvoiceDay 계산 (CURRENT_DATE 기준)  
11  SELECT  
12    CustomerID,  
13    DATE_DIFF(CURRENT_DATE(), MAX(InvoiceDate), DAY) AS InvoiceDay  
14  FROM `thefirst-464902.modulabs_project3.data`  
15  GROUP BY CustomerID  
16  ORDER BY InvoiceDay DESC;  
17  
18  -- 3-5. 가장 최근 일자와 유저별 마지막 구매일 간의 차이 계산  
19  WITH MostRecent AS (  
20  
```

작리 완료됨

작리 결과

업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

	CustomerID	InvoiceDay	
1	17968	5334	
2	14142	5334	
3	17908	5334	
4	18011	5334	
5	14729	5334	
6	17643	5334	
7	15165	5334	
8	16274	5334	
9	13065	5334	
10	14237	5334	
11	18074	5334	
12	16583	5334	
13	15350	5334	
14	13747	5334	

마지막 결과

가장 최근 일자(most_recent_date)와 유저별 마지막 구매일(InvoiceDay)간의 차이를 계산

```
WITH MostRecent AS (  
  SELECT MAX(DATE(InvoiceDate)) AS most_recent_date  
  FROM `thefirst-464902.modulabs_project.data`  
)  
SELECT  
  CustomerID,  
  DATE_DIFF((SELECT most_recent_date FROM MostRecent),  
            MAX(DATE(InvoiceDate)), DAY) AS recency  
  FROM `thefirst-464902.modulabs_project.data`
```

```
GROUP BY CustomerID  
ORDER BY recency DESC;
```

```
CREATE OR REPLACE TABLE  
'thefirst-464902.modulabs_project.user_r' AS  
WITH MostRecent AS (  
    SELECT MAX(DATE(InvoiceDate)) AS most_recent_date  
    FROM `thefirst-464902.modulabs_project.data`  
)
```

```
274 GROUP BY CustomerID  
275 ORDER BY InvoiceDay DESC;  
276  
277 -- 3-5. 가장 최근 일자와 유저별 마지막 구매일 간의 차이 계산  
278 WITH MostRecent AS (  
279     SELECT MAX(InvoiceDate) AS most_recent_date  
280     FROM `thefirst-464902.modulabs_project3.data`  
281 )  
282 SELECT  
283     CustomerID,  
284     DATE_DIFF(SELECT most_recent_date FROM MostRecent), MAX(InvoiceDate), DAY AS  
285     FROM `thefirst-464902.modulabs_project3.data`  
286 GROUP BY CustomerID  
287 ORDER BY recency DESC;  
288  
289 -- 3-6. user_r 테이블로 저장 (데이터셋 내 상대적 recency)
```

▶ 실행 시 이 쿼리가 6.05MB를 처리합니다.

쿼리 결과

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프
	// CustomerID ▾ // recency ▾ //				
1	17968		373		
2	14142		373		
3	17908		373		
4	18011		373		
5	14729		373		
6	17643		373		
7	15165		373		
8	16274		373		
9	13065		373		
10	14237		373		

지금까지의 결과를 user_r이라는 이름의 테이블로 저장

```
-- 3-6. `user_r` 테이블로 저장 (데이터셋 내 상대적 recency)
CREATE OR REPLACE TABLE `thefirst-464902.modulabs_project3.user_r` AS
WITH MostRecent AS (
  SELECT MAX(InvoiceDate) AS most_recent_date
  FROM `thefirst-464902.modulabs_project3.data`
)
SELECT
  CustomerID,
  DATE_DIFF((SELECT most_recent_date FROM MostRecent), MAX(InvoiceDate), DAY) AS recency
FROM `thefirst-464902.modulabs_project3.data`
GROUP BY CustomerID;
-- =====
-- 4. RFM 분석 - Frequency 계산
```

| 완료됨

! 결과

결과 저장 다음에서 열기 ▾

정보 결과 실행 세부정보 실행 그래프

이 문으로 이름이 user_r인 테이블이 교체되었습니다.

테이블로 이동

SELECT

```
CustomerID,
DATE_DIFF((SELECT most_recent_date FROM MostRecent),
MAX(DATE(InvoiceDate)), DAY) AS recency
FROM `thefirst-464902.modulabs_project.data`'
GROUP BY CustomerID;
```

Frequency

전체 거래 건수 계산

행	CustomerID	purchase_cnt
1	14911	242
2	12748	217
3	17841	169
4	14606	125
5	13089	118
6	15311	118
7	12971	88
8	13408	75
9	14646	73
10	16029	66

#우선 각 고객의 거래 건수를 세어 봅시다. 거래 건은 InvoiceNo를 기준으로 파악하면 되기 때문에, 고객마다 고유한 InvoiceNo의 수를 세어 주겠습니다.

```

SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM `thefirst-464902.modulabs_project.data`
GROUP BY CustomerID;

```

구매한 아이템의 총 수량 계산

```

315     CustomerID,
316     SUM(Quantity) AS item_cnt
317 FROM `thefirst-464902.modulabs_project3.data`
318 GROUP BY CustomerID
319 ORDER BY item_cnt DESC
320 LIMIT 10;
321
322 -- 4-3. Recency + Frequency 통합 테이블 생성
323 CREATE OR REPLACE TABLE `thefirst-464902.modulabs_project3.user_rf` AS
324 WITH purchase_cnt AS (
325     SELECT
326         CustomerID,
327         COUNT(DISTINCT InvoiceNo) AS purchase_cnt
328     FROM `thefirst-464902.modulabs_project3.data`
329     GROUP BY CustomerID
330 ),

```

쿼리 완료됨

쿼리 결과

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프
	// CustomerID //	// item_cnt //			
1	14646	196279			
2	14911	76395			
3	12415	76334			
4	17450	66912			
5	17511	62655			
6	18102	61724			
7	13694	61520			
8	14298	57691			
9	14156	56033			
10	16684	48635			

-- 5-1. 고객별 총 지출액 계산

```

SELECT
    CustomerID,
    ROUND(SUM(UnitPrice * Quantity), 1) AS

```

```

total_expenditure
FROM `thefirst-464902.modulabs_project3.data`
GROUP BY CustomerID
ORDER BY total_expenditure DESC
LIMIT 10;

```

#1. 전체 거래 건수 계산'과 '2. 구매한 아이템의 총 수량 계산'의 결과를 합쳐서 user_rf라는 이름의 테이블에 저장

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project.user_rf` AS



행	CustomerID	purchase_cnt	item_cnt	recency
1	12713	1	505	0
2	13298	1	96	1
3	14569	1	79	1
4	17436	1	12	1
5	13436	1	76	1
6	15520	1	314	1
7	15195	1	1404	2
8	14204	1	72	2
9	15471	1	249	2
10	16560	1	02	2

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.user_rfm` AS
SELECT
rf.CustomerID AS CustomerID,
rf.purchase_cnt,
rf.item_cnt,
rf.recency,
ut.user_total,
ROUND(ut.user_total / rf.purchase_cnt, 1) AS
user_average

```

FROM `thefirst-464902.modulabs_project3.user_rf` rf
LEFT JOIN (
    SELECT
        CustomerID,
        ROUND(SUM(UnitPrice * Quantity), 1) AS user_total
    FROM `thefirst-464902.modulabs_project3.data`
    GROUP BY CustomerID
) ut
ON rf.CustomerID = ut.CustomerID;

```

Monetary

고객별 총 지출액 계산

행	CustomerID	total_expenditure
1	14646	277837.1
2	18102	256424.5
3	17450	179008.8
4	14911	127841.8
5	12415	121540.0
6	14156	111611.1
7	17511	87450.1
8	16684	64332.5
9	13694	62712.9
10	16029	60168.3

고객별 총 지출액을 계산해 보세요. 소수점 첫째 자리에서 반올림

```

-- SELECT
-- CustomerID,
-- ROUND(SUM(UnitPrice * Quantity), 1) AS total_expenditure --
Total expenditure rounded to 1 decimal
-- FROM `thefirst-464902.modulabs_project3.data`
-- GROUP BY CustomerID
-- ORDER BY total_expenditure;

```

고객별 평균 거래 금액 계산

고객별 평균 거래 금액을 구하기 위해 1) data 테이블을 user_rf 테이블과 조인 (**LEFT JOIN**) 한 후, 2) purchase_cnt로 나누어서 3) user_rfm 테이블로 저장해 봅시다.

```
CREATE OR REPLACE TABLE
project_name.modulabs_project.user_rfm AS
SELECT
    rf.CustomerID AS CustomerID,
    rf.purchase_cnt,
    rf.item_cnt,
    rf.recency,
    ut.user_total,
    # [[YOUR QUERY]] AS user_average
FROM project_name.modulabs_project.user_rf rf
LEFT JOIN (
    -- 고객 별 총 지출액
    SELECT
        # [[YOUR QUERY]]
    ) ut
ON rf.CustomerID = ut.CustomerID;
#계산: 총 지출액 ÷ 총 거래 건수
#의미: 거래당 평균 지출 금액
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	794.6	794.6
2	15520	1	314	1	343.5	343.5
3	17436	1	12	1	17.4	17.4
4	14569	1	79	1	227.4	227.4
5	13298	1	96	1	360.0	360.0
6	13436	1	76	1	196.9	196.9
7	14204	1	72	2	150.6	150.6
8	15195	1	1404	2	3861.0	3861.0
9	15471	1	249	2	442.1	442.1
10	15992	1	17	3	42.0	42.0
11	12442	1	181	3	144.1	144.1
12	16528	1	171	3	244.4	244.4
13	17914	1	457	3	329.4	329.4
14	12478	1	233	3	546.0	546.0
15	12650	1	250	3	242.4	242.4
16	15318	1	642	3	312.6	312.6
17	14578	1	240	3	168.6	168.6
18	16569	1	93	3	124.2	124.2
19	17383	1	148	4	193.4	193.4
20	14219	1	78	4	89.9	89.9
21	16597	1	184	4	90.0	90.0
22	13790	1	748	4	348.8	348.8
23	18015	1	157	4	120.0	120.0
24	12367	1	172	4	150.9	150.9

RFM 통합 테이블 출력하기

추가 Feature 추출

- 구매하는 제품의 다양성
- 평균 구매 주기
- 구매 취소 경향성

1. 구매하는 제품의 다양성

이 단계에서는 고객들의 제품 구매 행동 속 구매 제품의 다양성을 살펴보려고 합니다. 고객이 얼마나 다양한 제품들에 관심 있는 사람인지를 알게 되면, 개인 맞춤형 마케팅 전략과 추천 서비스를 계획하는 데에도 큰 도움이 될 수 있습니다.

우선 1) 고객 별로 구매한 상품들의 고유한 수를 계산합니다. 높은 숫자가 나오는 것은 해당 고객이 다양한 제품들을 구매한다는 의미이며, 낮은 값이 나오는 경우 소수의 제품들만 구매한다는 것을 의미합니다.

이후 2) user_rfm 테이블과 결과를 합치고, 이를 3) user_data라는 이름의 테이블에 저장하겠습니다.

- customer_rfm_enhanced 테이블 생성

-- =====

```
CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.customer_rfm_enha
nced` AS
SELECT
    rfm.CustomerID,
    rfm.recency,
    rfm.purchase_cnt,
    rfm.item_cnt,
    rfm.user_total,
    rfm.user_average,
    rfm.unique_products,
-- 재방문 주기 정보
    COALESCE(rc.avg_days_between_purchases, NULL) AS
avg_revisit_days,
    COALESCE(rc.total_intervals, 0) AS
purchase_intervals_count,
    COALESCE(rc.min_days_between_purchases, NULL) AS
min_revisit_days,
    COALESCE(rc.max_days_between_purchases, NULL) AS
max_revisit_days,
    COALESCE(rc.stddev_days_between_purchases, NULL)
AS stddev_revisit_days,
-- 취소 정보
    COALESCE(ca.cancel_frequency, 0) AS
cancel_frequency,
    COALESCE(ca.cancel_rate, 0.00) AS cancel_rate,
    COALESCE(ca.total_cancelled_quantity, 0) AS
total_cancelled_quantity,
    COALESCE(ca.total_cancelled_amount, 0.00) AS
total_cancelled_amount,
    COALESCE(ca.cancel_behavior_type, '취소 없음') AS
cancel_behavior_type,
-- 재방문 패턴 분류
CASE
    WHEN rc.avg_days_between_purchases IS NULL THEN
'일회성 고객'
    WHEN rc.avg_days_between_purchases <= 7 THEN '주
간 고객'
```

```

        WHEN rc.avg_days_between_purchases <= 30 THEN
'월간 고객'
        WHEN rc.avg_days_between_purchases <= 90 THEN
'계절 고객'
        ELSE '비정기 고객'
    END AS revisit_pattern
FROM `thefirst-464902.modulabs_project3.user_rfm` rfm
LEFT JOIN
`thefirst-464902.modulabs_project3.customer_revisit_cycle` rc
    ON rfm.CustomerID = rc.CustomerID
LEFT JOIN
`thefirst-464902.modulabs_project3.customer_cancel_analysis` ca
    ON rfm.CustomerID = ca.CustomerID;

```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	13185	1	12	267	71.4	71.4	1
2	17923	1	50	282	207.5	207.5	1
3	14090	1	72	324	76.3	76.3	1
4	14679	1	-1	371	-2.5	-2.5	1
5	17948	1	144	147	358.6	358.6	1
6	16738	1	3	297	3.8	3.8	1
7	13270	1	200	366	590.0	590.0	1
8	18174	1	50	7	104.0	104.0	1
9	17956	1	1	249	12.8	12.8	1
10	17291	1	72	308	550.8	550.8	1
11	16881	1	600	66	432.0	432.0	1
12	15940	1	4	311	35.8	35.8	1
13	15070	1	36	372	106.2	106.2	1

2. 평균 구매 주기

이 단계에서는 **고객들의 쇼핑 패턴을 이해하는 것을 목표로 합니다.** 그 중에서도 고객 별 재방문 주기를 살펴볼 것입니다. 고객들의 구매와 구매 사이의 기간이 평균적으로 몇 일인지를 보여주는 평균 일수를 계산하면, 고객이 다음 구매를 언제 할지 예측하는 데에도 큰 도움이 됩니다. **평균 구매 소요 일수를 계산하고, 그 결과를 user_data에 통합해 줍니다.**

≡ 필터 속성 이름 또는 값 입력

<input type="checkbox"/> 필드 이름	유형	모드
<input type="checkbox"/> CustomerID	INTEGER	NULLABLE
<input type="checkbox"/> total_transactions	INTEGER	NULLABLE
<input type="checkbox"/> cancel_frequency	INTEGER	NULLABLE
<input type="checkbox"/> normal_transactions	INTEGER	NULLABLE
<input type="checkbox"/> total_cancelled_quantity	INTEGER	NULLABLE
<input type="checkbox"/> total_cancelled_amount	FLOAT	NULLABLE
<input type="checkbox"/> cancel_rate	FLOAT	NULLABLE
<input type="checkbox"/> cancel_behavior_type	STRING	NULLABLE

customer_cancel_behavior								
스키마		세부정보		미리보기		테이블 탐색기		미리보기
행	CustomerID	total_transactions	cancel_frequency	normal_transactions	total_cancelled_quantity	total_cancelled_amount	cancel_rate	cancel_behavior_type
1	18043	115	0	115	0	0.0	0.0	취소 없음
2	12524	135	0	135	0	0.0	0.0	취소 없음
3	16241	653	0	653	0	0.0	0.0	취소 없음
4	14419	201	0	201	0	0.0	0.0	취소 없음
5	13158	79	0	79	0	0.0	0.0	취소 없음
6	13744	63	0	63	0	0.0	0.0	취소 없음
7	16464	230	0	230	0	0.0	0.0	취소 없음
8	14524	194	0	194	0	0.0	0.0	취소 없음
9	12432	107	0	107	0	0.0	0.0	취소 없음

구매 취소 경향성

취소 빈도와 취소 비율을 계산하고 그 결과를 user_data에 통합해 줍시다. 취소 비율은 소수점 두번째 자리까지 구한다.

~~~~~  
~~~~~  
~

-- 2. 재방문 주기 계산

customer_rfm_enhanced

쿼리 다음에서 열기 공

스키마 세부정보 미리보기 테이블 탐색기 미리보기 통계 계보

필터 속성 이름 또는 값 입력

<input type="checkbox"/> 필드 이름	유형	모드	키
<input type="checkbox"/> CustomerID	INTEGER	NULLABLE	-
<input type="checkbox"/> purchase_cnt	INTEGER	NULLABLE	-
<input type="checkbox"/> item_cnt	INTEGER	NULLABLE	-
<input type="checkbox"/> recency	INTEGER	NULLABLE	-
<input type="checkbox"/> user_total	FLOAT	NULLABLE	-
<input type="checkbox"/> user_average	FLOAT	NULLABLE	-
<input type="checkbox"/> avg_revisit_days	FLOAT	NULLABLE	-
<input type="checkbox"/> purchase_intervals_count	INTEGER	NULLABLE	-
<input type="checkbox"/> min_revisit_days	INTEGER	NULLABLE	-
<input type="checkbox"/> max_revisit_days	INTEGER	NULLABLE	-
<input type="checkbox"/> stddev_revisit_days	FLOAT	NULLABLE	-
<input type="checkbox"/> cancel_frequency	INTEGER	NULLABLE	-
<input type="checkbox"/> cancel_rate	FLOAT	NULLABLE	-
<input type="checkbox"/> total_cancelled_quantity	INTEGER	NULLABLE	-
<input type="checkbox"/> total_cancelled_amount	FLOAT	NULLABLE	-
<input type="checkbox"/> cancel_behavior_type	STRING	NULLABLE	-
<input type="checkbox"/> revisit_pattern	STRING	NULLABLE	-

```

CREATE OR REPLACE TABLE
`thefirst-464902.modulabs_project3.customer_rfm_enha
nced` AS
SELECT
    rfm.*,
    -- 재방문 주기 정보
    COALESCE(rc.avg_days_between_purchases, NULL) AS
    avg_revisit_days,
    COALESCE(rc.total_intervals, 0) AS
    purchase_intervals_count,
    COALESCE(rc.min_days_between_purchases, NULL) AS
    min_revisit_days,

```

```

        COALESCE(rc.max_days_between_purchases, NULL) AS
max_revisit_days,
        COALESCE(rc.stddev_days_between_purchases, NULL)
AS stddev_revisit_days,
-- 취소 정보
        COALESCE(ca.cancel_frequency, 0) AS
cancel_frequency,
        COALESCE(ca.cancel_rate, 0.00) AS cancel_rate,
        COALESCE(ca.total_cancelled_quantity, 0) AS
total_cancelled_quantity,
        COALESCE(ca.total_cancelled_amount, 0.00) AS
total_cancelled_amount,
        COALESCE(ca.cancel_behavior_type, '취소 없음') AS
cancel_behavior_type,
-- 재방문 패턴 분류
CASE
    WHEN rc.avg_days_between_purchases IS NULL THEN
'일회성 고객'
    WHEN rc.avg_days_between_purchases <= 7 THEN '주
간 고객'
    WHEN rc.avg_days_between_purchases <= 30 THEN
'월간 고객'
    WHEN rc.avg_days_between_purchases <= 90 THEN
'계절 고객'
    ELSE '비정기 고객'
END AS revisit_pattern
FROM `thefirst-464902.modulabs_project3.user_rfm` rfm
LEFT JOIN
`thefirst-464902.modulabs_project3.customer_revisit_
cycle` rc
    ON rfm.CustomerID = rc.CustomerID
LEFT JOIN
`thefirst-464902.modulabs_project3.customer_cancel_a
nalysis` ca
    ON rfm.CustomerID = ca.CustomerID;

```

-- 4. 최종 user_data 테이블 생성 (취소 정보 + 재방문 주기 모두 포함)

[

cancel_behavior_type

revisit_pattern

] 문자형 컬럼 추가 분석

customer_rfm_enhanced

쿼리 다음에서 열기 공유

스키마 세부정보 미리보기 테이블 탐색기 미리보기 통계 계보

필터 속성 이름 또는 값 입력

필드 이름	유형	모드	키	대조
CustomerID	INTEGER	NULLABLE	-	-
purchase_cnt	INTEGER	NULLABLE	-	-
item_cnt	INTEGER	NULLABLE	-	-
recency	INTEGER	NULLABLE	-	-
user_total	FLOAT	NULLABLE	-	-
user_average	FLOAT	NULLABLE	-	-
unique_products	INTEGER	NULLABLE	-	-
avg_revisit_days	FLOAT	NULLABLE	-	-
purchase_intervals_count	INTEGER	NULLABLE	-	-
min_revisit_days	INTEGER	NULLABLE	-	-
max_revisit_days	INTEGER	NULLABLE	-	-
stddev_revisit_days	FLOAT	NULLABLE	-	-
cancel_frequency	INTEGER	NULLABLE	-	-
cancel_rate	FLOAT	NULLABLE	-	-
total_cancelled_quantity	INTEGER	NULLABLE	-	-
total_cancelled_amount	FLOAT	NULLABLE	-	-
cancel_behavior_type	STRING	NULLABLE	-	-
revisit_pattern	STRING	NULLABLE	-	-

9-1. 최종 user_data 테이블 샘플 확인

SELECT

```
CustomerID,  
customer_segment,  
recency,  
frequency,  
monetary,
```

```
    avg_revisit_days,
    cancel_frequency,
    cancel_rate,
    risk_level,
    revisit_pattern,
    cancel_behavior_type
FROM `thefirst-464902.modulabs_project3.user_data`'
ORDER BY monetary DESC
LIMIT 10;
```

-- 9-2. user_data 테이블 전체 요약

```
SELECT
    COUNT(*) AS total_customers,
    COUNT(CASE WHEN cancel_frequency > 0 THEN 1 END)
AS customers_with_cancellations,
    ROUND(
        COUNT(CASE WHEN cancel_frequency > 0 THEN 1 END)
* 100.0 / COUNT(*), 2
    ) AS cancellation_customer_percentage,
    ROUND(AVG(cancel_frequency), 1) AS
avg_cancel_frequency,
    ROUND(AVG(cancel_rate), 2) AS avg_cancel_rate,
    ROUND(SUM(total_cancelled_amount), 2) AS
total_revenue_lost,
    COUNT(CASE WHEN avg_revisit_days IS NOT NULL THEN
1 END) AS customers_with_revisit_data
FROM `thefirst-464902.modulabs_project3.user_data`;
```

-- 9-3. 고객 세그먼트별 종합 통계

```
SELECT
    customer_segment,
    COUNT(*) AS customer_count,
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(), 2)
AS segment_percentage,
    ROUND(AVG(recency), 1) AS avg_recency,
    ROUND(AVG(frequency), 1) AS avg_frequency,
    ROUND(AVG(monetary), 1) AS avg_monetary,
    ROUND(AVG(avg_revisit_days), 1) AS
avg_revisit_days,
    ROUND(AVG(cancel_rate), 2) AS avg_cancel_rate,
    ROUND(SUM(monetary), 2) AS total_revenue
FROM `thefirst-464902.modulabs_project3.user_data`
```

```
GROUP BY customer_segment  
ORDER BY total_revenue DESC;
```

-- 9-4. 재방문 패턴별 고객 분포

```
SELECT  
    revisit_pattern,  
    COUNT(*) AS customer_count,  
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(), 2)  
AS percentage,  
    ROUND(AVG(avg_revisit_days), 1) AS avg_days,  
    ROUND(AVG(monetary), 1) AS avg_monetary  
FROM `thefirst-464902.modulabs_project3.user_data`  
GROUP BY revisit_pattern  
ORDER BY customer_count DESC;
```

-- 9-5. 취소 행동 유형별 고객 분포

```
SELECT  
    cancel_behavior_type,  
    COUNT(*) AS customer_count,  
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(), 2)  
AS percentage,  
    ROUND(AVG(cancel_frequency), 1) AS  
avg_cancel_frequency,  
    ROUND(AVG(cancel_rate), 2) AS avg_cancel_rate,  
    ROUND(AVG(monetary), 1) AS avg_monetary  
FROM `thefirst-464902.modulabs_project3.user_data`  
GROUP BY cancel_behavior_type  
ORDER BY customer_count DESC;
```

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프	
행	CustomerID	customer_segment	customer_value_tier	recency	avg_revisit_days	predicted_next_p...
1	14646	휴면 우수 고객	최고가치 고객	4962	8.0	2011-12-16
2	18102	휴면 우수 고객	최고가치 고객	4961	14.7	2011-12-24
3	17450	휴면 우수 고객	최고가치 고객	4969	13.8	2011-12-15
4	14911	휴면 우수 고객	최고가치 고객	4962	2.9	2011-12-11
5	12415	휴면 우수 고객	최고가치 고객	4985	22.4	2011-12-07
6	14156	휴면 우수 고객	최고가치 고객	4970	8.6	2011-12-09
7	17511	휴면 우수 고객	최고가치 고객	4963	13.7	2011-12-21
8	16029	휴면 우수 고객	최고가치 고객	4999	9.1	2011-11-10
9	16684	휴면 우수 고객	최고가치 고객	4965	23.6	2011-12-29
10	13694	휴면 우수 고객	최고가치 고객	4964	10.9	2011-12-17
11	15311	휴면 우수 고객	최고가치 고객	4961	4.2	2011-12-13
12	13089	휴면 우수 고객	최고가치 고객	4963	5.6	2011-12-13
13	17949	휴면 우수 고객	최고가치 고객	4962	13.7	2011-12-22
14	15769	휴면 우수 고객	최고가치 고객	4968	17.3	2011-12-19
15	15061	휴면 우수 고객	최고가치 고객	4964	13.2	2011-12-19
16	14096	휴면 우수 고객	최고가치 고객	4965	6.1	2011-12-11
17	14298	휴면 우수 고객	최고가치 고객	4969	14.7	2011-12-16
18	14088	휴면 우수 고객	최고가치 고객	4971	28.4	2011-12-27

최종결과 오류 확인 :

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프	
행	status	total_customers	segments_count	value_tiers_count	customers_with_product_data	customers_with_revisit_data
1	Analysis Complete - 모든 오류 수정 완료	4333	3	4	4333	2775

작업 정보	결과	차트	JSON	실행 세부정보	실행 그래프	
행	customers_with_product_data	customers_with_revisit_data	customers_with_cancellations	min_recency	max_recency	avg_monetary_value
	4333	2775	1490	4961	5334	1997.54

status total_customers segments_count value_tiers_count
 customers_with_product_data customers_with_revisit_data
 customers_with_cancellations min_recency max_recency
 avg_monetary_value
 Analysis Complete - 모든 오류 수정 완료 4333 3 4 4333
 2775 1490 4961 5334 1997.54