고객을 세그먼테이션하자 [프로젝트]

11-2. 데이터 불러오기

데이터 살펴보기

• 테이블에 있는 10개의 행만 출력하기

```
SELECT *
FROM avid-involution-439402-i8.modulabs_project.data
LIMIT 10;
```

뒝 ,	InvoiceNo ▼	StockCode ▼	Description ▼	Quantity •	InvoiceDate ▼	UnitPrice ▼	CustomerID ▼	Country -
1	574301	23511	EMBROIDERED RIBBON REEL E	6	2011-11-03 16:15:00 UTC	2.08	12544	Spain
2	574301	22144	CHRISTMAS CRAFT LITTLE FRI	6	2011-11-03 16:15:00 UTC	2.1	12544	Spain
3	574301	22910	PAPER CHAIN KIT VINTAGE CH	6	2011-11-03 16:15:00 UTC	2.95	12544	Spain
4	574301	22751	FELTCRAFT PRINCESS OLIVIA	4	2011-11-03 16:15:00 UTC	3.75	12544	Spain
5	574301	22077	6 RIBBONS RUSTIC CHARM	12	2011-11-03 16:15:00 UTC	1.95	12544	Spain
6	574301	23514	EMBROIDERED RIBBON REEL S	6	2011-11-03 16:15:00 UTC	2.08	12544	Spain
7	574301	20971	PINK BLUE FELT CRAFT TRINK	12	2011-11-03 16:15:00 UTC	1.25	12544	Spain
8	574301	22621	TRADITIONAL KNITTING NANCY	12	2011-11-03 16:15:00 UTC	1.65	12544	Spain
9	574301	84879	ASSORTED COLOUR BIRD ORN	8	2011-11-03 16:15:00 UTC	1.69	12544	Spain
10	574301	20749	ASSORTED COLOUR MINI CAS	4	2011-11-03 16:15:00 UTC	7.95	12544	Spain

• 전체 데이터는 몇 행으로 구성되어 있는지 확인하기

```
SELECT COUNT(*)
FROM avid-involution-439402-i8.modulabs_project.data
```



데이터 수 세기

• COUNT 함수를 사용해서, 각 컬럼 별 데이터 포인트의 수를 세어 보기

```
SELECT

COUNT(InvoiceNo) AS count_column1,
COUNT(StockCode) AS count_column2,
COUNT(Description) AS count_column3,
COUNT(Quantity) AS count_column4,
COUNT(InvoiceDate) AS count_column5,
COUNT(InvoiceDate) AS count_column6,
COUNT(UnitPrice) AS count_column7,
COUNT(CustomerID) AS count_column7,
COUNT(Country) AS count_column8
FROM warm-capsule-439401-j2.modulabs_project.data;
```

11-4. 데이터 전처리 방법(1): 결측치 제거

컬럼 별 누락된 값의 비율 계산

^{*}전처리 한 후에 확인한 데이터 수

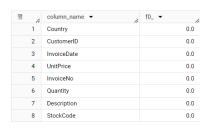
^{*}전처리가 끝나서 결측치가 제거 되어있다.. 그래서 컬럼별 데이터 수가 다 같다

- 각 컬럼 별 누락된 값의 비율을 계산
 - 각 컬럼에 대해서 누락 값을 계산한 후, 계산된 누락 값을 UNION ALL을 통해 합치기

```
SELECT column_name, ROUND((total - column_value) / total * 100, 2)

FROM
(

SELECT 'InvoiceNo' AS column_name, COUNT(InvoiceNo) AS column_value, COUNT(*) AS total FROM warm
SELECT 'StockCode' AS column_name, COUNT(StockCode) AS column_value, COUNT(*) AS total FROM warm
SELECT 'Description' AS column_name, COUNT(Description) AS column_value, COUNT(*) AS total FROM warm-c
SELECT 'Quantity' AS column_name, COUNT(Quantity) AS column_value, COUNT(*) AS total FROM warm-c
SELECT 'InvoiceDate' AS column_name, COUNT(InvoiceDate) AS column_value, COUNT(*) AS total FROM warm-
SELECT 'UnitPrice' AS column_name, COUNT(UnitPrice) AS column_value, COUNT(*) AS total FROM warm-
SELECT 'CustomerID' AS column_name, COUNT(CustomerID) AS column_value, COUNT(*) AS total FROM warm-cap
) AS column_data;
```



^{*}원래 pescription 0.27%, CustomerID 24.93% 결측치가 있어야 하는데 결측치가 제거된 상태라서 전부 0.0%가 되어있는 상태..

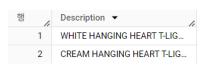
결측치 처리 전략

• StockCode = '85123A' 의 Description 을 추출하는 쿼리문을 작성하기

```
SELECT Description

FROM avid-involution-439402-i8.modulabs_project.data

WHERE StockCode = '85123A';
```



^{*}밑의 CustomerID 혹은 Description이 NULL값 일 경우 행을 삭제하는 과정을 거쳤기에 나온 결

결측치 처리

• DELETE 구문을 사용하며, WHERE 절을 통해 데이터를 제거할 조건을 제시

```
DELETE FROM warm-capsule-439401-j2.modulabs_project.data
WHERE CustomerID IS NULL OR Description IS NULL;
```

이 문으로 data의 행 0개가 삭제되었습니다.

11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

• 중복된 행의 수를 세어보기

^{*}원래라면 약 135000개 삭제되어야 하는데 이미 삭제가 다 되어 0개가 삭제되었다..

。 8개의 컬럼에 그룹 함수를 적용한 후, COUNT가 1보다 큰 데이터를 세어보기

SELECT InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Country, COUNT(*) FROM warm-capsule-439401-j2.modulabs_project.data
GROUP BY InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Country HAVING COUNT(*) > 1;
#지금보니 왜 이렇게 했는 모르겠다...



*원래 중복된 행이 50개가 나오는데 전처리 끝나고 실행해 본 거라 표시할 데이터가 없다고 나온다.. (다음부터는 실습 과정을 미리 제대로 기록하겠습니다)

중복값 처리

- 중복값을 제거하는 쿼리문 작성하기
 - CREATE OR REPLACE TABLE 구문을 활용하여 모든 컬럼(*)을 DISTINCT 한 데이터로 업데이트

CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.data AS SELECT DISTINCT *
FROM warm-capsule-439401-j2.modulabs_project.data;



11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

• 고유(unique)한 InvoiceNo 의 개수를 출력하기

SELECT COUNT(DISTINCT InvoiceNo)
FROM warm-capsule-439401-j2.modulabs_project.data



- *22190이 나와야 하나 **제품과 관련되지 않은 거래 기록을 제거 및 서비스 관련 정보를 포함하는 행들을 제거 작업이 진행된 상**
- 고유한 InvoiceNo 를 앞에서부터 100개를 출력하기

SELECT DISTINCT InvoiceNo
FROM warm-capsule-439401-j2.modulabs_project.data
LIMIT 100;

• InvoiceNo 가 'C'로 시작하는 행을 필터링 할 수 있는 쿼리문을 작성하기 (100행까지만 출력)

SELECT *
FROM warm-capsule-439401-j2.modulabs_project.data
WHERE InvoiceNo LIKE 'C%'
LIMIT 100;



• 구매 건 상태가 Canceled 인 데이터의 비율(%) - 소수점 첫번째 자리까지

SELECT ROUND(SUM(CASE WHEN InvoiceNo LIKE 'C%'THEN 1 ELSE 0 END)/COUNT(*)*100,1) FROM warm-capsule-439401-j2.modulabs_project.data

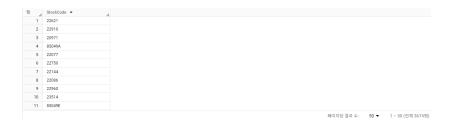


*전처리로 인해 값이 조금 다르다(원래2.2)

StockCode 살펴보기

• 고유한 StockCode 의 개수를 출력하기

SELECT DISTINCT StockCode FROM warm-capsule-439401-j2.modulabs_project.data



• 어떤 제품이 가장 많이 판매되었는지 보기 위하여 StockCode 별 등장 빈도를 출력하기

。 상위 10개의 제품들을 출력하기

```
SELECT StockCode, COUNT(*) AS sell_cnt

FROM warm-capsule-439401-j2.modulabs_project.data

GROUP BY StockCode

ORDER BY sell_cnt DESC

LIMIT 10
```

행 //	StockCode ▼	sell_cnt ▼
1	85123A	2065
2	22423	1893
3	85099B	1659
4	47566	1408
5	84879	1405
6	20725	1346
7	22720	1224
8	22197	1110
9	23203	1108
10	20727	1099

- StockCode 의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고
 - 。 숫자가 **0~1개인 값**들에는 어떤 코드들이 들어가 있는지 출력하기

```
WITH uniqueStockCode AS(
    SELECT DISTINCT StockCode
    FROM warm-capsule-439401-j2.modulabs_project.data
)

SELECT LENGTH(StockCode)-LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS ssl,
    COUNT(*) AS Stock_cnt
    FROM uniqueStockCode
    GROUP BY ssl
    ORDER BY Stock_cnt DESC
```



- *숫자가 0~1 개, 즉 ssl이 0 또는 1 인 데이터를 이미 삭제해서 나온 결과(원래 0이 7개 1이 1개)
- StockCode 의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고
 - 。 숫자가 0~1개인 값들을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트인지 구하기 (소수점 두 번째 자리까지)

```
WITH uniqueStockCode AS (
    SELECT DISTINCT StockCode
    FROM warm-capsule-439401-j2.modulabs_project.data
),
sslCount AS (
    SELECT
        LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, '[0-9]', '')) AS ssl
    FROM uniqueStockCode
),
totalCount AS (
    SELECT COUNT(*) AS total_count
    FROM warm-capsule-439401-j2.modulabs_project.data
)
SELECT
ROUND(
    (SUM(CASE WHEN ssl IN (0, 1) THEN 1 ELSE 0 END) * 100.0) / (SELECT total_count FROM totalCount),
    2
```

```
) AS percentage FROM sslCount;
```

```
1 WITH uniqueStockCode AS (
2 SELECT DISTINCT StockCod
3 FROM warm-capsule-43940
     SELECT DISTINCT StockCode
FROM warm-capsule-439401-j2.modulabs_project.data
     sslCount AS (
     | SELECT | SELECT | LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, '[0-9]', '')) AS ssl
       FROM uniqueStockCode
10 totalCount AS (
11 SELECT COUNT(*) AS total_count
12 FROM warm-capsule-439401-j2.modulabs_project.data
11
12
13
14
       ROUND(

(SUM(CASE WHEN ssl IN (0, 1) THEN 1 ELSE 0 END) * 100.0) / (SELECT total_count FROM totalCount),

2
쿼리 결과
             결과
                           차트
                                          JSON
작업 정보
                                                         실행 세부정보
                                                                                실행 그래프
                       0.0
```

*원래 0.48 (숫자가 0~1개인 값 제거된 상태라서 0.0)

• 제품과 관련되지 않은 거래 기록을 제거하기

```
DELETE FROM warm-capsule-439401-j2.modulabs_project.data
WHERE StockCode IN (
    SELECT StockCode
FROM (
    SELECT
        StockCode,
        LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, '[0-9]', '')) AS ssl
FROM (
        SELECT DISTINCT StockCode
        FROM warm-capsule-439401-j2.modulabs_project.data
    )
    ) AS sslCount
WHERE ssl IN (0, 1)
);
```

```
🔾 제목 없는 쿼리
                     DELETE FROM warm-capsule-439401-j2.modulabs_project.data
    WHERE StockCode IN (
  3
      SELECT StockCode
  4
      FROM (
      SELECT
  5
        StockCode, LENGTH(REGEXP_REPLACE(StockCode, '[0-9]', '')) AS ssl
  8
       FROM (
       SELECT DISTINCT StockCode
FROM warm-capsule-439401-j2.modulabs_project.data
 10
 12
      ) AS sslCount
 13
      WHERE ssl IN (0, 1)
 14
 쿼리 결과
 작업 정보
            결과
                    실행 세부정보
                                  실행 그래프
     이 문으로 data의 행 0개가 삭제되었습니다.
```

* 'BANK CHARGES, POST' 등 제품과 관련되지 않은 거래 기록이 삭제가 되어 있는 상

고객을 세그먼테이션하자 [프로젝트]

6

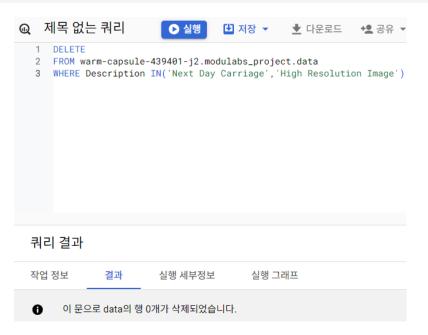
Description 살펴보기

• 고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기

페이지당 결과 수: 50 ▼ 1 - 30 (전체 30행)

• 서비스 관련 정보를 포함하는 행들을 제거하기

```
DELETE
FROM warm-capsule-439401-j2.modulabs_project.data
WHERE Description IN('Next Day Carriage','High Resolution Image')
```



*이미 삭제 했

• 대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.data AS
SELECT

*,
UPPER(Description) AS Description_Upper -- 대문자로 표준화된 Description 컬럼 추가
FROM warm-capsule-439401-j2.modulabs_project.data;
```



UnitPrice 살펴보기

• UnitPrice 의 최솟값, 최댓값, 평균을 구하기

```
SELECT
MIN(UnitPrice) AS min_price,
MAX(UnitPrice) AS max_price,
AVG(UnitPrice) AS average_price
FROM warm-capsule-439401-j2.modulabs_project.data;
```



- 원래는 min_price가 0 이지만 UnitPrice = 0 인 데이터 제거 작업이 수행된 상태라 위와 같은 결과임
- 단가가 0원인 거래의 개수, 구매 수량(Quantity)의 최솟값, 최댓값, 평균 구하기





단가 0원 거래 데이터 전부 제거 되어 나오는 결

• UnitPrice = 0 를 제거하고 일관된 데이터셋을 유지하기

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.data AS
SELECT *
FROM warm-capsule-439401-j2.modulabs_project.data
WHERE UnitPrice != 0
```

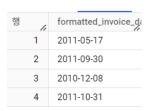
① 문으로 이름이 data인 테이블이 교체되었습니다.

11-7. RFM 스코어

Recency

• InvoiceDate 컬럼을 연월일 자료형으로 변경하기

```
SELECT
  DATE(InvoiceDate) AS formatted_invoice_date,
  *
FROM project_name.modulabs_project.data;
```



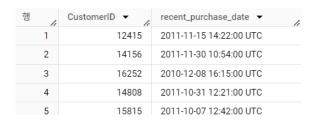
• 가장 최근 구매 일자를 MAX() 함수로 찾아보기

```
SELECT
DATE(MAX(InvoiceDate)) AS recent_purchase_date
FROM warm-capsule-439401-j2.modulabs_project.data;
```

1 2011-12-09

• 유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기

```
SELECT
CustomerID,
MAX(InvoiceDate) AS recent_purchase_date
FROM warm-capsule-439401-j2.modulabs_project.data
GROUP BY CustomerID;
```



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

```
SELECT
CustomerID,
EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
SELECT
CustomerID,
MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM warm-capsule-439401-j2.modulabs_project.data
GROUP BY CustomerID
);
```

행 //	CustomerID ▼	recency ▼
1	15389	172
2	12375	2
3	12939	64
4	12708	29
5	13952	211
6	16909	64

• 최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이고 지금까지의 결과를 user_r 이라는 이름의 테이블로 저장하기

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.user_r AS

SELECT
    CustomerID,
    EXTRACT(DAY FROM CURRENT_DATE - InvoiceDay) AS recency

FROM (
    SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay

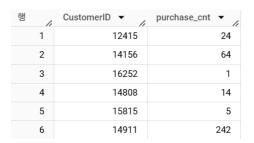
FROM warm-capsule-439401-j2.modulabs_project.data
    GROUP BY CustomerID
) AS subquery;
```

❶ 이 문으로 이름이 user_r인 테이블이 교체되었습니다.

Frequency

• 고객마다 고유한 InvoiceNo의 수를 세어보기

```
SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS purchase_cnt FROM warm-capsule-439401-j2.modulabs_project.data GROUP BY CustomerID;
```



• 각 고객 별로 구매한 아이템의 총 수량 더하기

```
SELECT CustomerID, SUM(quantity) AS item_cnt
FROM warm-capsule-439401-j2.modulabs_project.data
GROUP BY CustomerID;
```

행 //	CustomerID ▼	item_cnt ▼
1	12415	76946
2	14156	56896
3	16252	-158
4	14808	2028
5	15815	1856
6	14911	76823

• 전체 거래 건수 계산와 구매한 아이템의 총 수량 계산의 결과를 합쳐서 user_rf 라는 이름의 테이블에 저장하기

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.user_rf AS
-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
 SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS purchase_cnt
 FROM warm-capsule-439401-j2.modulabs_project.data
 GROUP BY CustomerID
),
-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
 SELECT CustomerID, SUM(quantity) AS item_cnt
 FROM warm-capsule-439401-j2.modulabs_project.data
 GROUP BY CustomerID
-- (3) (1)과 (2)를 통합하고 user_r에서 recency 값을 가져옴
SELECT
 pc.CustomerID,
 pc.purchase_cnt,
 ic.item_cnt,
 ur.recency
FROM purchase_cnt AS pc
JOIN item_cnt AS ic
 ON pc.CustomerID = ic.CustomerID
JOIN warm-capsule-439401-j2.modulabs_project.user_r AS ur
 ON pc.CustomerID = ur.CustomerID;
```

● 이 문으로 이름이 user_rf인 테이블이 교체되었습니다.

Monetary

• 고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)

```
SELECT
CustomerID,
ROUND(SUM(UnitPrice), 1) AS user_total
FROM warm-capsule-439401-j2.modulabs_project.data
GROUP BY CustomerID;
```

행 //	CustomerID ▼	user_total ▼
1	12415	1887.1
2	14156	4764.2
3	16252	67.1
4	14808	1098.9
5	15815	105.9
6	14911	19617.1

- 고객별 평균 거래 금액 계산
 - 고객별 평균 거래 금액을 구하기 위해 1) data 테이블을 user_rf 테이블과 조인(LEFT JOIN) 한 후, 2) purchase_cnt 로 나누어서 3) user_rfm 테이블로 저장하기

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.user_rfm AS
SELECT
 rf.CustomerID,
 rf.purchase_cnt,
 rf.item_cnt,
 rf.recency,
 ROUND(SUM(d.UnitPrice * d.quantity), 1) AS user_total,
 {\tt ROUND(SUM(d.UnitPrice * d.quantity) / NULLIF(rf.purchase\_cnt, 0), 2) \ AS \ user\_average}
FROM warm-capsule-439401-j2.modulabs_project.user_rf rf
LEFT JOIN warm-capsule-439401-j2.modulabs_project.data d
 ON rf.CustomerID = d.CustomerID
GROUP BY
 rf.CustomerID,
 rf.purchase_cnt,
 rf.item_cnt,
 rf.recency;
```

❶ 이 문으로 이름이 user_rfm인 테이블이 교체되었습니다.

RFM 통합 테이블 출력하기

• 최종 user_rfm 테이블을 출력하기

```
SELECT *
FROM warm-capsule-439401-j2.modulabs_project.user_rfm;
```



11-8. 추가 Feature 추출

1. 구매하는 제품의 다양성

```
    1) 고객 별로 구매한 상품들의 고유한 수를 계산하기
    2)
        user_rfm 테이블과 결과를 합치기
    3)
        user_data 라는 이름의 테이블에 저장하기
```

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.user_data AS

WITH unique_products AS (
    SELECT
        CustomerID,
        COUNT(DISTINCT StockCode) AS unique_products
        FROM warm-capsule-439401-j2.modulabs_project.data
        GROUP BY CustomerID
)

SELECT ur.*, up.* EXCEPT (CustomerID)

FROM warm-capsule-439401-j2.modulabs_project.user_rfm AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

이 문으로 이름이 user_data인 테이블이 교체되었습니다.

2. 평균 구매 주기

- 고객들의 쇼핑 패턴을 이해하는 것을 목표 (고객 별 재방문 주기 살펴보기)
 - 균 구매 소요 일수를 계산하고, 그 결과를 user_data 에 통합

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.user_data AS
WITH purchase_intervals AS (
  -- (2) 고객 별 구매와 구매 사이의 평균 소요 일수
  SELECT
    CustomerID,
     {\it CASE~WHEN~ROUND(AVG(interval\_),~2)~IS~NULL~THEN~0~ELSE~ROUND(AVG(interval\_),~2)~END~AS~average\_interval\_),~2) {\it END~AS~average\_interval\_} \\
    -- (1) 구매와 구매 사이에 소요된 일수
    SELECT
      DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID ORDER BY InvoiceDate), DAY)
      warm-capsule-439401-j2.modulabs_project.data
    WHERE CustomerID IS NOT NULL
  GROUP BY CustomerID
SELECT u.*, pi.* EXCEPT (CustomerID)
FROM warm-capsule-439401-j2.modulabs_project.user_data AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

이 문으로 이름이 user_data인 테이블이 교체되었습니다.

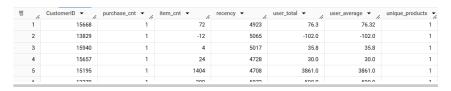
3. 구매 취소 경향성

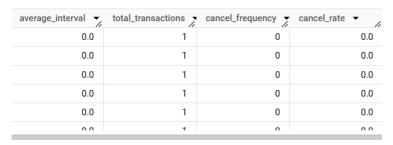
- 고객의 취소 패턴 파악하기
 - 1) 취소 빈도(cancel_frequency) : 고객 별로 취소한 거래의 총 횟수
 - 2) 취소 비율(cancel_rate): 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율
 - 취소 빈도와 취소 비율을 계산하고 그 결과를 user_data 에 통합하기 (취소 비율은 소수점 두번째 자리)

```
CREATE OR REPLACE TABLE warm-capsule-439401-j2.modulabs_project.user_data AS
WITH TransactionInfo AS (
 SELECT
    CustomerID,
    COUNT(*) AS total_transactions,
    COUNT(CASE WHEN InvoiceNo = 'Cancelled' THEN 1 END) AS cancel_frequency
  FROM warm-capsule-439401-j2.modulabs_project.data
  GROUP BY CustomerID
SELECT
 u.*,
 t.total_transactions,
 t.cancel_frequency,
 {\tt ROUND(t.cancel\_frequency~^*~1.0~/~NULLIF(t.total\_transactions,~0),~2)~AS~cancel\_rate}
FROM warm-capsule-439401-j2.modulabs_project.user_data AS u
LEFT JOIN TransactionInfo AS t
ON u.CustomerID = t.CustomerID;
```

• 다양한 컬럼들을 활용하여 고객의 구매 패턴과 선호도를 보다 심층적으로 이해할 수 있도록 최종적으로 user_data 를 출력하기

SELECT *
FROM warm-capsule-439401-j2.modulabs_project.user_data





회고

SQL의 기본적인 문법만 알고 간단하지만 직접 쿼리를 짜며 과제를 하는 게 처음이라 굉장히 어려웠고 도무지 주어진 시간 안에 다 할 수 없었지만 꼭 수업 시간 외에 별도로 학습해서 공부한 내용이 나중에 쉽게 느껴질 수 있도록 노력하겠습니다