고객을 세그먼테이션하자 [프로젝트]

11-2. 데이터 불러오기

데이터 살펴보기

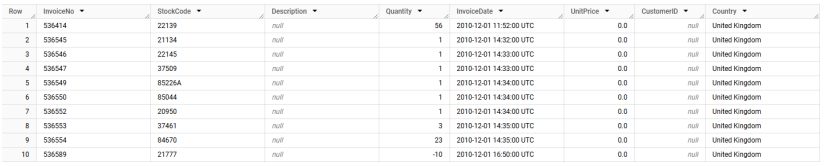
(사전 자료 파악) https://www.kaggle.com/datasets/carrie1/ecommerce-data data.csv 데이터 특징

컬럼명	설명
InvoiceNo	각각의 고유한 거래를 나타내는 코드.이 코드가 'C'라는 글자로 시작한다면, 취소를 나타냄..하나의 거래에 여러 개의 제품이 함께 구매되었다면, 1개의 InvoiceNo에는 여러 개의 StockCode가 연결되어 있음
StockCode	각각의 제품에 할당된 고유 코드
Description	각 제품에 대한 설명
Quantity	거래에서 제품을 몇 개 구매했는지에 대한 단위 수 (참고: - 값은 판매나 반품을 의미)
InvoiceDate	거래가 일어난 날짜와 시간
UnitPrice	제품 당 단위 가격(영국 파운드)
CustomerID	각 고객에게 할당된 고유 식별자 코드
Country	주문이 발생한 국가

- **테이블에 있는 10개의 행만 출력하기**

```
SELECT * FROM modulabs_project.data LIMIT 10;
```

[결과]



Row	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536414	22139	null	56	2010-12-01 11:52:00 UTC	0.0	null	United Kingdom
2	536545	21134	null	1	2010-12-01 14:32:00 UTC	0.0	null	United Kingdom
3	536546	22145	null	1	2010-12-01 14:33:00 UTC	0.0	null	United Kingdom
4	536547	37509	null	1	2010-12-01 14:33:00 UTC	0.0	null	United Kingdom
5	536549	85226A	null	1	2010-12-01 14:34:00 UTC	0.0	null	United Kingdom
6	536550	85044	null	1	2010-12-01 14:34:00 UTC	0.0	null	United Kingdom
7	536552	20950	null	1	2010-12-01 14:34:00 UTC	0.0	null	United Kingdom
8	536553	37461	null	3	2010-12-01 14:35:00 UTC	0.0	null	United Kingdom
9	536554	84670	null	23	2010-12-01 14:35:00 UTC	0.0	null	United Kingdom
10	536589	21777	null	-10	2010-12-01 16:50:00 UTC	0.0	null	United Kingdom

- **전체 데이터는 몇 행으로 구성되어 있는지 확인하기**

```
SELECT COUNT(*) FROM modulabs_project.data
```

[결과]

Row	f0_
1	541909

541,909 행으로 구성되어 있음

데이터 수 세기

- **COUNT 함수를 사용해서, 각 컬럼별 데이터 포인트의 수를 세어 보기**

```
-- SELECT COUNT(*), COUNT(InvoiceNo), COUNT(StockCode), COUNT(Description),COUNT(Quantity), COUNT(In
SELECT COUNT(*) AS total_rows, COUNT(InvoiceNo) AS InvoiceNo_count,COUNT(StockCode) AS StockCode_cou
  COUNT(Description) AS Description_count, COUNT(Quantity) AS Quantity_count,
  COUNT(InvoiceDate) AS InvoiceDate_count, COUNT(UnitPrice) AS UnitPrice_count,
  COUNT(CustomerID) AS CustomerID_count, COUNT(Country) AS Country_count
FROM `modulabs_project.data`;
```

[결과]

Row	total_rows	InvoiceNo_count	StockCode_count	Description_count	Quantity_count	InvoiceDate_count	UnitPrice_count	CustomerID_count	Country_count
1	541909	541909	541909	540455	541909	541909	541909	406829	541909

11-4. 데이터 전처리 방법(1): 결측치 제거

컬럼 별 누락된 값의 비율 계산

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

```
-- SELECT 'InvoiceNo' AS column_name, COUNT(*) - COUNT(InvoiceNo) AS missing_values FROM modulabs_pr
-- UNION ALL
-- SELECT 'StockCode' AS column_name, COUNT(*) - COUNT(StockCode) AS missing_values FROM modulabs_pr
-- UNION ALL
-- SELECT 'Description' AS column_name, COUNT(*) - COUNT(Description) AS missing_values FROM modulab
-- UNION ALL
-- SELECT 'Quantity' AS column_name, COUNT(*) - COUNT(Quantity) AS missing_values FROM modulabs_proj
-- UNION ALL
-- SELECT 'InvoiceDate' AS column_name, COUNT(*) - COUNT(InvoiceDate) AS missing_values FROM modulab
-- UNION ALL
-- SELECT 'UnitPrice' AS column_name, COUNT(*) - COUNT(UnitPrice) AS missing_values FROM modulabs_pr
-- UNION ALL
-- SELECT 'CustomerID' AS column_name, COUNT(*) - COUNT(CustomerID) AS missing_values FROM modulabs_
-- UNION ALL
-- SELECT 'Country' AS column_name, COUNT(*) - COUNT(Country) AS missing_values FROM modulabs_projec
```

```
SELECT column_name,
  ROUND((total - column_value) / total * 100, 2) AS missing_percentage
FROM (
  SELECT
    CASE
      WHEN column_name = 'InvoiceNo' THEN COUNT(InvoiceNo)
      WHEN column_name = 'StockCode' THEN COUNT(StockCode)
      WHEN column_name = 'Description' THEN COUNT(Description)
      WHEN column_name = 'Quantity' THEN COUNT(Quantity)
      WHEN column_name = 'InvoiceDate' THEN COUNT(InvoiceDate)
      WHEN column_name = 'UnitPrice' THEN COUNT(UnitPrice)
      WHEN column_name = 'CustomerID' THEN COUNT(CustomerID)
      WHEN column_name = 'Country' THEN COUNT(Country)
    END AS column_value,
    COUNT(*) AS total,
    CASE
      WHEN column_name = 'InvoiceNo' THEN 'InvoiceNo'
      WHEN column_name = 'StockCode' THEN 'StockCode'
      WHEN column_name = 'Description' THEN 'Description'
      WHEN column_name = 'Quantity' THEN 'Quantity'
      WHEN column_name = 'InvoiceDate' THEN 'InvoiceDate'
      WHEN column_name = 'UnitPrice' THEN 'UnitPrice'
      WHEN column_name = 'CustomerID' THEN 'CustomerID'
      WHEN column_name = 'Country' THEN 'Country'
    END AS column_name
  FROM modulabs_project.data
  GROUP BY column_name
) AS column_data;
```

[결과]

Row	column_name	missing_values
1	CustomerID	135080
2	StockCode	0
3	Country	0
4	Quantity	0
5	InvoiceNo	0
6	UnitPrice	0
7	Description	1454
8	InvoiceDate	0

결측치 처리 전략

- **`StockCode = '85123A'`의 `Description`을 추출하는 쿼리문을 작성하기**

```
SELECT Description
FROM modulabs_project.data
WHERE StockCode = '85123A';
```

[결과]

Row	Description
1	?
2	wrongly marked carton 22804
3	CREAM HANGING HEART T-LIGHT HOLDER
4	CREAM HANGING HEART T-LIGHT HOLDER
5	CREAM HANGING HEART T-LIGHT HOLDER
6	CREAM HANGING HEART T-LIGHT HOLDER
7	CREAM HANGING HEART T-LIGHT HOLDER
8	CREAM HANGING HEART T-LIGHT HOLDER
9	CREAM HANGING HEART T-LIGHT HOLDER
10	CREAM HANGING HEART T-LIGHT HOLDER
11	CREAM HANGING HEART T-LIGHT HOLDER
12	WHITE HANGING HEART T-LIGHT HOLDER
13	WHITE HANGING HEART T-LIGHT HOLDER
14	WHITE HANGING HEART T-LIGHT HOLDER
15	WHITE HANGING HEART T-LIGHT HOLDER
16	WHITE HANGING HEART T-LIGHT HOLDER
17	WHITE HANGING HEART T-LIGHT HOLDER
18	WHITE HANGING HEART T-LIGHT HOLDER
19	WHITE HANGING HEART T-LIGHT HOLDER
20	WHITE HANGING HEART T-LIGHT HOLDER

(이하 결과 생략)

결측치 처리

- **DELETE 구문을 사용하며, WHERE 절을 통해 데이터를 제거할 조건을 제시**

```
-- CustomerID 또는 Description이 NULL인 데이터를 삭제 (결측치 처리해야 하므로)
DELETE FROM modulabs_project.data
WHERE CustomerID IS NULL OR Description IS NULL;
SELECT COUNT(*) FROM modulabs_project.data;
```

[결과]

Row	f0_
1	406829

11-5. 데이터 전처리(2): 중복값 처리

컬럼 별 누락된 값의 비율 계산

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

```
WITH TotalRows AS (
  SELECT COUNT(*) AS total_rows FROM modulabs_project.data
)
SELECT
  'InvoiceNo' AS column_name,
  (COUNT(*) - COUNT(InvoiceNo)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'StockCode' AS column_name,
  (COUNT(*) - COUNT(StockCode)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'Description' AS column_name,
  (COUNT(*) - COUNT(Description)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'Quantity' AS column_name,
  (COUNT(*) - COUNT(Quantity)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'InvoiceDate' AS column_name,
  (COUNT(*) - COUNT(InvoiceDate)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'UnitPrice' AS column_name,
  (COUNT(*) - COUNT(UnitPrice)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'CustomerID' AS column_name,
  (COUNT(*) - COUNT(CustomerID)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data
UNION ALL
SELECT
  'Country' AS column_name,
  (COUNT(*) - COUNT(Country)) / (SELECT total_rows FROM TotalRows) * 100 AS missing_percentage
FROM modulabs_project.data;
```

[결과 이미지]

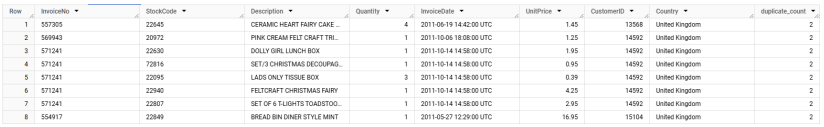
Row	column_name	missing_percentage
1	UnitPrice	0.0
2	CustomerID	0.0
3	Quantity	0.0
4	Country	0.0
5	StockCode	0.0
6	Description	0.0
7	InvoiceDate	0.0
8	InvoiceNo	0.0

중복값 확인

- **중복된 행의 수를 세어보기**
  - **8개의 컬럼에 그룹 함수를 적용한 후, COUNT가 1보다 큰 데이터를 세어보기**

```
SELECT *, COUNT(*) AS duplicate_count
FROM modulabs_project.data
GROUP BY InvoiceNo, StockCode, Description, Quantity,
 InvoiceDate, UnitPrice, CustomerID, Country
HAVING COUNT(*) > 1;
```

[결과]



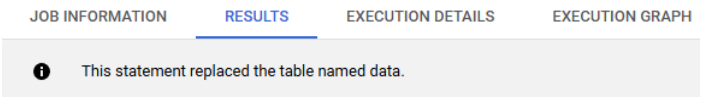
Row	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	duplicate_count
1	557305	22645	CERAMIC HEART FAIRY CAKE...	4	2011-06-19 14:42:00 UTC	1.45	13568	United Kingdom	2
2	569943	20972	PINK CREAM FELT CRAFT TRI...	1	2011-10-06 18:08:00 UTC	1.25	14592	United Kingdom	2
3	571241	22630	DOLLY GIRL LUNCH BOX	1	2011-10-14 14:58:00 UTC	1.95	14592	United Kingdom	2
4	571241	72816	SET/3 CHRISTMAS DECOUPAG...	1	2011-10-14 14:58:00 UTC	0.95	14592	United Kingdom	2
5	571241	22095	LADS ONLY TISSUE BOX	3	2011-10-14 14:58:00 UTC	0.39	14592	United Kingdom	2
6	571241	22940	FELTCRAFT CHRISTMAS FAIRY	1	2011-10-14 14:58:00 UTC	4.25	14592	United Kingdom	2
7	571241	22807	SET OF 6 T-LIGHTS TOADSTOO...	1	2011-10-14 14:58:00 UTC	2.95	14592	United Kingdom	2
8	554917	22849	BREAD BIN DINER STYLE MINT	1	2011-05-27 12:29:00 UTC	16.95	15104	United Kingdom	2

중복값 처리

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

```
CREATE OR REPLACE TABLE modulabs_project.data AS
SELECT DISTINCT *
FROM modulabs_project.data;
```

[결과]



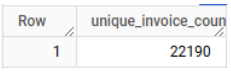
11-6. 데이터 전처리(3): 오류값 처리

`InvoiceNo` 살펴보기

- **고유(unique)한 `InvoiceNo` 의 개수를 출력하기**

```
SELECT COUNT(DISTINCT InvoiceNo) AS unique_invoice_count
FROM modulabs_project.data;
```

[결과]



Row	unique_invoice_coun
1	22190

- **고유한 `InvoiceNo` 를 앞에서부터 100개를 출력하기**

```
SELECT DISTINCT InvoiceNo AS unique_invoice_count
FROM modulabs_project.data LIMIT 100;
```

[결과]

Row	unique_invoice_count
1	574301
2	C575531
3	557305
4	543008
5	549735
6	554032
7	561387
8	574868
9	574827
10	546015
11	551859
12	554665
13	578187
14	569943
15	571241
16	574573
17	545419

Load more

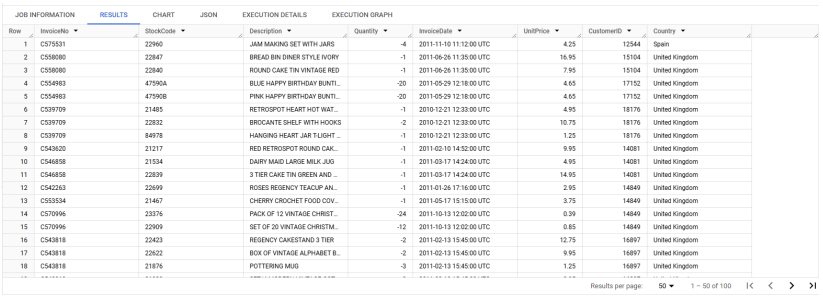
...

83	557118
84	560918
85	563110
86	563112
87	565610
88	565611
89	569766
90	569767
91	574239
92	574241
93	574245
94	574246
95	577173
96	577175
97	C542263
98	C553534
99	C570996
100	561542

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

```
SELECT *
FROM modulabs_project.data
WHERE InvoiceNo LIKE 'C%'
LIMIT 100;
```

[결과]



- **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) AS canceled_p
FROM modulabs_project.data;
```

[결과]

Row	canceled_percentage
1	2.2

`StockCode` 살펴보기

- **고유한 `StockCode` 의 개수를 출력하기**

```
SELECT COUNT(DISTINCT StockCode) AS unique_stockcode_count
FROM modulabs_project.data;
```

[결과]

Row	unique_stockcode_count
1	3684

- **어떤 제품이 가장 많이 판매되었는지 보기 위하여 `StockCode` 별 등장 빈도를 출력하기**
  - 상위 10개의 제품들을 출력하기

```
SELECT StockCode, COUNT(*) AS sell_count
FROM modulabs_project.data
GROUP BY StockCode
ORDER BY sell_count DESC
LIMIT 10;
```

[결과]

Row	StockCode	sell_count
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
10	23203	1108

- **`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

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

[결과]

Row	StockCode	number_count
1	POST	0
2	M	0
3	PADS	0
4	D	0
5	BANK CHARGES	0
6	DOT	0
7	CRUK	0
8	C2	1

Tip. `LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', ''))`: `StockCode`의 전체 길이에서 숫자를 제거한 후의 길이를 뺌으로써 숫자의 개수를 계산

- **`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 퍼센트**인지 구하기 (소수점 두 번째 자리까지)

```
SELECT
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM modulabs_project.data), 2) AS percentage
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM modulabs_project.data
) AS subquery
WHERE number_count <= 1;
```

[결과]

Row	percentage
1	0.48

- **제품과 관련되지 않은 거래 기록을 제거하기**

```
DELETE FROM modulabs_project.data
WHERE StockCode IN (
  SELECT DISTINCT StockCode
  FROM (
    SELECT StockCode
    FROM (
        SELECT StockCode,
        LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
        FROM modulabs_project.data
         )
  WHERE number_count <= 1
    )
)
```

[결과]

This statement removed 1,915 rows from data.

`Description` 살펴보기

- **고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기**

```
SELECT Description, COUNT(*) AS description_cnt
FROM modulabs_project.data
GROUP BY Description ORDER BY description_cnt DESC LIMIT 30;
```

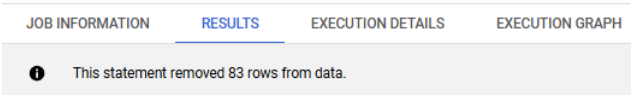
[결과]

Row	Description	description_cnt
1	WHITE HANGING HEART T-LIG...	2058

- **서비스 관련 정보를 포함하는 행들을 제거하기**

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

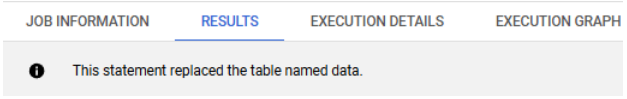
[결과]



- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

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

[결과]



`UnitPrice` 살펴보기

- `UnitPrice`**의 최솟값, 최댓값, 평균을 구하기**

```
SELECT MIN(UnitPrice) AS min_price, MAX(UnitPrice) AS max_price, AVG(UnitPrice) AS avg_price
FROM modulabs_project.data;
```

[결과]

Row	min_price	max_price	avg_price
1	0.0	649.5	2.9049567574060102

- **단가가 0원인 거래의 개수, 구매 수량(** `Quantity` **)의 최솟값, 최댓값, 평균 구하기**

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

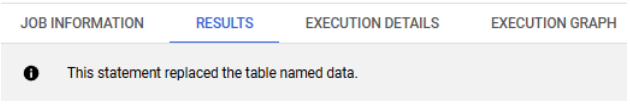
[결과]

Row	cnt_quantity	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.5151515151515

- `UnitPrice = 0` **를 제거하고 일관된 데이터셋을 유지하기**

```
CREATE OR REPLACE TABLE modulabs_project.data AS
SELECT *
FROM modulabs_project.data
WHERE UnitPrice > 0;
```

[결과]



11-7. RFM 스코어

RFM 스코어 복습

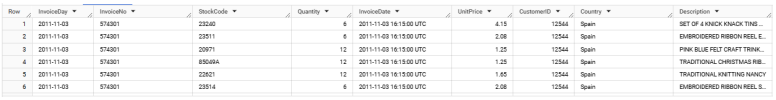
- **Recency**: **고객이 마지막으로 구매한 시점**. 최근에 구매한 고객들은 더 자주 구매할 가능성이 높기 때문에, 최신성 점수가 높은지를 고려
- **Frequency**: **특정 기간 동안 고객이 얼마나 자주 우리의 제품이나 서비스를 구매하는지**. 빈번하게 구매하는 고객은 충성도가 높은 고객일 확률이 높기 때문에, 빈도 점수가 높은지를 고려
- **Monetary**: **고객이 지출한 총 금액**. 많은 금액을 지불한 고객일수록 더 가치가 높은 충성 고객일 수 있음. 앞으로도 우리의 제품과 사이트에 많은 돈을 지불할 수 있는 고객이므로, 가치 점수가 높은지를 함께 고려

Recency

- `InvoiceDate` **컬럼을 연월일 자료형으로 변경하기**

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

[결과]



...

- **가장 최근 구매 일자를 MAX() 함수로 찾아보기**

```
SELECT DATE(MAX(InvoiceDate)) AS most_recent_date,
FROM modulabs_project.data;
```

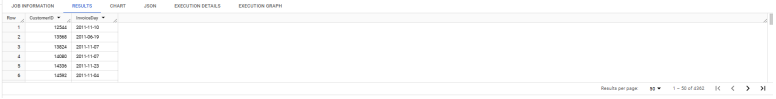
[결과]

Row	most_recent_date
1	2011-12-09

- **유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기**

```
SELECT
    CustomerID,
    MAX(CAST(InvoiceDate AS DATE)) AS InvoiceDay
FROM modulabs_project.data
GROUP BY CustomerID;
```

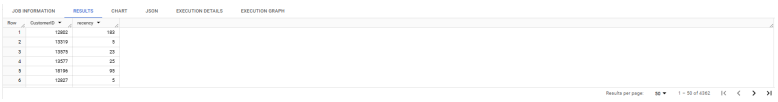
[결과]



- **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 modulabs_project.data
  GROUP BY CustomerID
);
```

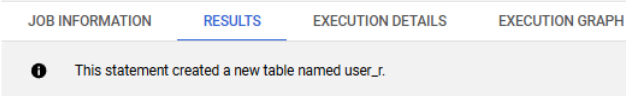
[결과]



- **최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이고 지금까지의 결과를 `user_r`이라는 이름의 테이블로 저장하기**
  - TIPS: 쿼리문에서 EXTRACT 함수의 사용 방법
    - `MAX(InvoiceDay) OVER () - InvoiceDay` : 각 고객(CustomerID)의 각 구매일(InvoiceDay)과 전체 데이터셋에서의 마지막 구매일(MAX(InvoiceDay)) 간의 차이를 계산
    - `EXTRACT(DAY FROM …)` : 여기에서 EXTRACT 함수는 위에서 계산된 날짜 차이에서 일(DAY) 부분만을 추출. 즉 각 고객의 최근 구매일로부터 해당 구매 건의 구매일부터의 날짜 차이를 계산하는 함수

```
CREATE OR REPLACE TABLE modulabs_project.user_r AS
SELECT
  CustomerID,
  EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
  FROM modulabs_project.data
  GROUP BY CustomerID
);
```

[결과]

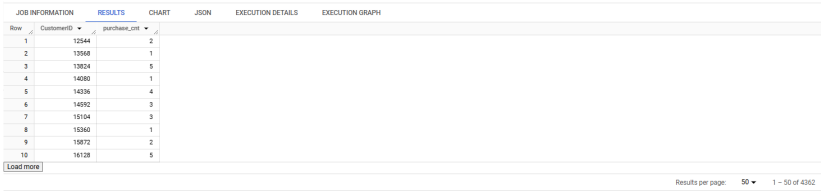


Frequency

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

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

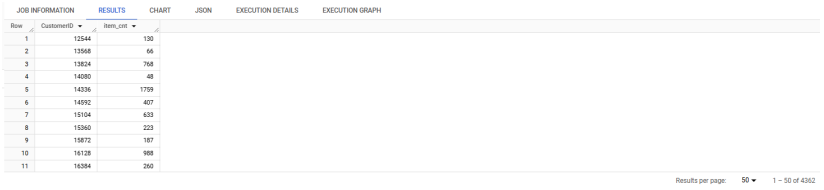
[결과]



- **각 고객 별로 구매한 아이템의 총 수량 더하기**

```
SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
FROM modulabs_project.data
GROUP BY CustomerID;
```

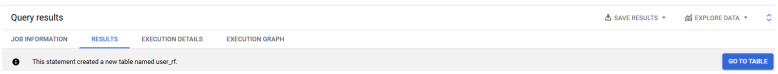
[결과]

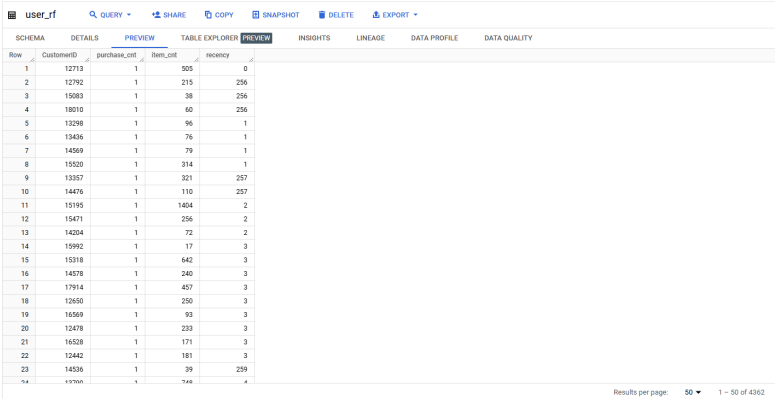


- **전체 거래 건수 계산와 구매한 아이템의 총 수량 계산의 결과를 합쳐서 `user_rf` 라는 이름의 테이블에 저장하기**

```
CREATE OR REPLACE TABLE modulabs_project.user_rf AS

-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt
  FROM modulabs_project.data
  GROUP BY CustomerID
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
    CustomerID,
    SUM(Quantity) AS item_cnt
  FROM modulabs_project.data
  GROUP BY CustomerID
)

-- 기존의 user_r에 (1)과 (2)를 통합
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 modulabs_project.user_r AS ur
  ON pc.CustomerID = ur.CustomerID;
```

[결과]



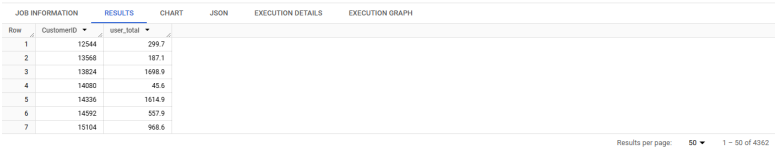


Monetary

- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

```
SELECT
  CustomerID,
  ROUND(SUM(UnitPrice * Quantity), 1) AS user_total
FROM modulabs_project.data
GROUP BY CustomerID;
```

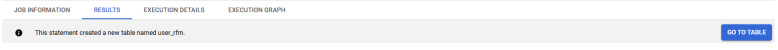
[결과]

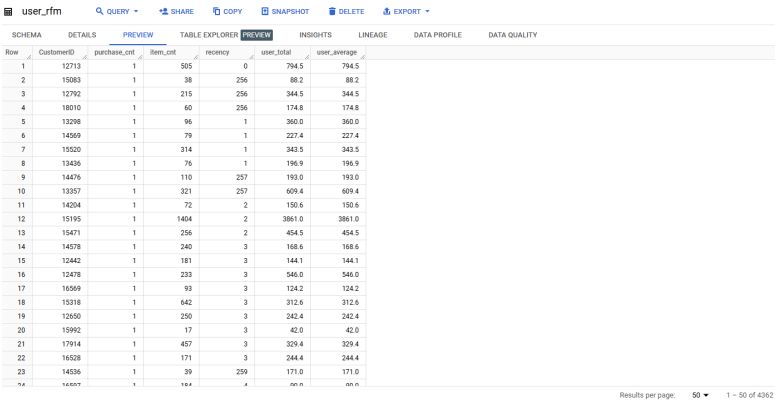


- **고객별 평균 거래 금액 계산**
  - **고객별 평균 거래 금액을 구하기 위해 1) `data` 테이블을 `user_rf` 테이블과 조인(LEFT JOIN) 한 후, 2) `purchase_cnt`로 나누어서 3) `user_rfm` 테이블로 저장하기**

```
CREATE OR REPLACE TABLE modulabs_project.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 modulabs_project.user_rf rf
LEFT JOIN (
  -- 고객 별 총 지출액
  SELECT
    CustomerID,
    ROUND(SUM(UnitPrice * Quantity), 1) AS user_total
  FROM modulabs_project.data
  GROUP BY CustomerID
) ut
ON rf.CustomerID = ut.CustomerID;
```

[결과]



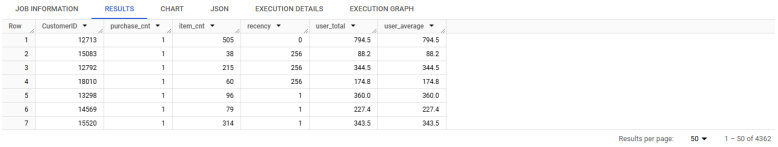


RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

```
SELECT *
FROM modulabs_project.user_rfm;
```

[결과]



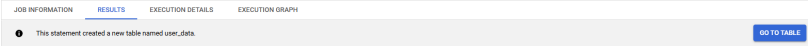
11-8. 추가 Feature 추출

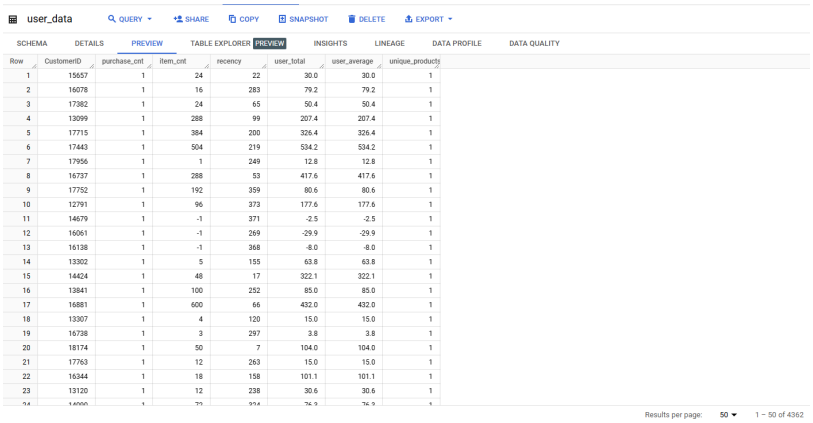
1. 구매하는 제품의 다양성

- **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 modulabs_project.user_data AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM modulabs_project.data
  GROUP BY CustomerID
)
SELECT ur.*, up.* EXCEPT (CustomerID)
FROM modulabs_project.user_rfm AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

[결과]





user_data QUERY SHARE COPY SNAPSHOT DELETE EXPORT

SCHEMA DETAILS PREVIEW TABLE EXPLORER PREVIEW INSIGHTS LINEAGE DATA PROFILE DATA QUALITY

Row	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	15657	1	24	22	30.0	30.0	1
2	16078	1	16	283	79.2	79.2	1
3	17382	1	24	65	50.4	50.4	1
4	13099	1	288	99	207.4	207.4	1
5	17715	1	384	200	326.4	326.4	1
6	17443	1	504	219	534.2	534.2	1
7	17956	1	1	249	12.8	12.8	1
8	16737	1	288	53	417.6	417.6	1
9	17752	1	192	359	80.6	80.6	1
10	12791	1	96	373	177.6	177.6	1
11	14679	1	-1	371	-2.5	-2.5	1
12	16061	1	-1	269	-29.9	-29.9	1
13	16138	1	-1	368	-8.0	-8.0	1
14	13302	1	5	155	63.8	63.8	1
15	14424	1	48	17	322.1	322.1	1
16	13841	1	100	252	85.0	85.0	1
17	16881	1	600	66	432.0	432.0	1
18	13307	1	4	120	15.0	15.0	1
19	16738	1	3	297	3.8	3.8	1
20	18174	1	50	7	104.0	104.0	1
21	17763	1	12	263	15.0	15.0	1
22	16344	1	18	158	101.1	101.1	1
23	13120	1	12	238	30.6	30.6	1

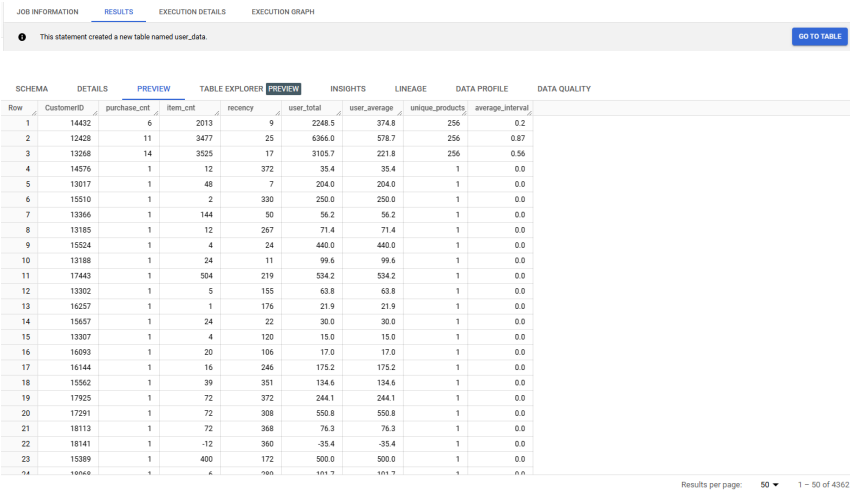
Results per page: 50 1 – 50 of 4362

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 modulabs_project.user_data AS
WITH purchase_intervals AS (
  -- (2) 고객 별 구매와 구매 사이의 평균 소요 일수
  SELECT
    CustomerID,
    CASE WHEN ROUND(AVG(interval_), 2) IS NULL THEN 0 ELSE ROUND(AVG(interval_), 2) END AS average_inte
  FROM (
    -- (1) 구매와 구매 사이에 소요된 일수
    SELECT
      CustomerID,
      DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID ORDER BY InvoiceDate), DAY)
    FROM
      modulabs_project.data
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM modulabs_project.user_data AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

[결과]



JOB INFORMATION RESULTS EXECUTION DETAILS EXECUTION GRAPH

This statement created a new table named user_data. GO TO TABLE

SCHEMA DETAILS PREVIEW TABLE EXPLORER PREVIEW INSIGHTS LINEAGE DATA PROFILE DATA QUALITY

Row	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval
1	14432	6	2013	9	2248.5	374.8	256	0.2
2	12428	11	3477	25	6366.0	578.7	256	0.87
3	13268	14	3525	17	3105.7	221.8	256	0.56
4	14576	1	12	372	35.4	35.4	1	0.0
5	13017	1	48	7	204.0	204.0	1	0.0
6	15510	1	2	330	250.0	250.0	1	0.0
7	13366	1	144	50	56.2	56.2	1	0.0
8	13185	1	12	267	71.4	71.4	1	0.0
9	15524	1	4	24	440.0	440.0	1	0.0
10	13188	1	24	11	99.6	99.6	1	0.0
11	17443	1	504	219	534.2	534.2	1	0.0
12	13302	1	5	155	63.8	63.8	1	0.0
13	16257	1	1	176	21.9	21.9	1	0.0
14	15657	1	24	22	30.0	30.0	1	0.0
15	13307	1	4	120	15.0	15.0	1	0.0
16	16093	1	20	106	17.0	17.0	1	0.0
17	16144	1	16	246	175.2	175.2	1	0.0
18	15562	1	39	351	134.6	134.6	1	0.0
19	17925	1	72	372	244.1	244.1	1	0.0
20	17291	1	72	308	550.8	550.8	1	0.0
21	18113	1	72	368	76.3	76.3	1	0.0
22	18141	1	-12	360	-35.4	-35.4	1	0.0
23	15389	1	400	172	500.0	500.0	1	0.0

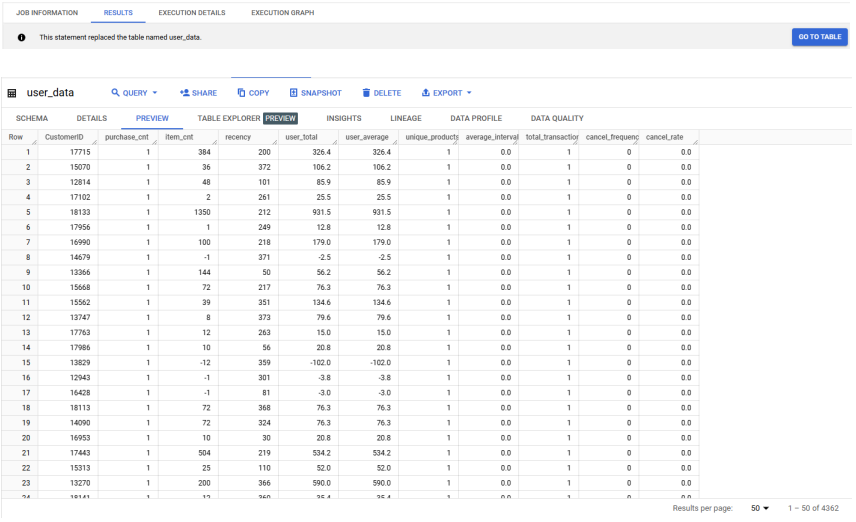
Results per page: 50 1 – 50 of 4362

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 bright-meridian-439401-g6.modulabs_project1.user_data AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(InvoiceNo) AS total_transactions,
    SUM(CASE WHEN stockcode LIKE 'C%'THEN 1 ELSE 0 END) AS cancel_frequency
  FROM bright-meridian-439401-g6.modulabs_project1.data
  GROUP BY customerID
)

SELECT u.*, t.* EXCEPT(CustomerID),
      ROUND(cancel_frequency * 100.0 / total_transactions, 2)
      AS cancel_rate
FROM bright-meridian-439401-g6.modulabs_project1.user_data AS u
LEFT JOIN TransactionInfo AS t
ON  U.customerID = t.customerID
```

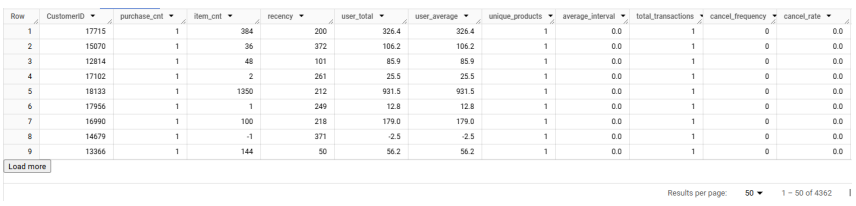
[결과]



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

```
SELECT * FROM modulabs_project.user_data
```

[결과]



회고

하루에 완수하기 다수 버거웠으나, 지금까지 배운 SQL과 빅쿼리, 그리고 고객 데이터 분석을 익히는데 많은 도움이 되었습니다.