유진국_고객을 세그먼테이션하자 [프로젝트]

11-2. 데이터 불러오기

데이터 살펴보기

- **테이블에 있는 10개의 행만 출력하기**

```
SELECT *
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
LIMIT 10
```

작업 정보 | 결과 | 시각화 | JSON | 실행 세부정보 | 실행 그래프

행	InvoiceNo	StockCode	Quantity	InvoiceDate
1	541431	23166	74215	2011-01-18 10:01:00
2	C541433	23166	-74215	2011-01-18 10:17:00
3	537626	85116	12	2010-12-07 14:57:00
4	537626	22195	12	2010-12-07 14:57:00
5	537626	21731	12	2010-12-07 14:57:00
6	537626	22375	4	2010-12-07 14:57:00
7	537626	84997C	6	2010-12-07 14:57:00
8	537626	85232D	3	2010-12-07 14:57:00

- **전체 데이터는 몇 행으로 구성되어 있는지 확인하기**

```
SELECT COUNT(*)
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	f0_
1	541909

데이터 수 세기

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

```
SELECT
    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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	COUNT_InvoiceNo	COUNT_StockCode	COUNT_Descripti...	COUNT_Quantity	COUNT_InvoiceD...	COUNT_UnitPrice	COUNT_Custome...	COUNT_Country
1	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,
   ROUND(SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`;
```

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

결측치 처리 전략

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

```
SELECT Description
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE StockCode = '85123A'
ORDER BY Description
```

행	Description
1	?
2	CREAM HANGING HEART T-LIG...
3	CREAM HANGING HEART T-LIG...
4	CREAM HANGING HEART T-LIG...
5	CREAM HANGING HEART T-LIG...
6	CREAM HANGING HEART T-LIG...
7	CREAM HANGING HEART T-LIG...
8	CREAM HANGING HEART T-LIG...

결측치 처리

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

```
DELETE FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE CustomerID IS NULL OR Description  IS NULL
```

이 문으로 data의 행 135,080개가 삭제되었습니다.

11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

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

```
SELECT COUNT(*) AS duplicated_row_count
FROM (
   SELECT
      InvoiceNo,
```

```
    StockCode,
    Description,
    Quantity,
    InvoiceDate,
    UnitPrice,
    CustomerID,
    Country
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY
    InvoiceNo,
    StockCode,
    Description,
    Quantity,
    InvoiceDate,
    UnitPrice,
    CustomerID,
    Country
  HAVING COUNT(*) > 1
)
```

행	duplicated_row_c...
1	4837

중복값 처리

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

```
CREATE OR REPLACE TABLE `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data` AS
SELECT DISTINCT *
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`;
```

이 문으로 이름이 data인 테이블이 교체되었습니다.

11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

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

```
SELECT COUNT(DISTINCT InvoiceNo)
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`;
```

행	f0_
1	22190

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

```
SELECT DISTINCT InvoiceNo
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
LIMIT 100
```

행	InvoiceNo
1	541431
2	C541433
3	537626
4	542237
5	549222
6	556201
7	562032
8	573511
9	581180
10	539318
11	541998
12	548955
13	568172

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

```
SELECT InvoiceNo
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE InvoiceNo LIKE 'C%'
LIMIT 100
```

행	InvoiceNo
1	C541433
2	C545329
3	C545329
4	C545330
5	C547388
6	C547388
7	C547388
8	C547388
9	C547388
10	C547388
11	C547388
12	C549955
13	C549955

- **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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	f0_
1	2.2

`StockCode` 살펴보기

- **고유한 `StockCode`의 개수를 출력하기**

```
SELECT COUNT(DISTINCT StockCode)
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	f0_
1	3684

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

```
SELECT StockCode, COUNT(*) AS sell_cnt
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
GROUP BY StockCode
ORDER BY sell_cnt DESC
LIMIT 10
```

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
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
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
)
WHERE number_count BETWEEN 0 AND 1
```

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

- **`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(SUM(
     CASE WHEN LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) IN (0, 1) THEN 1 ELSE 0 END) /
COUNT(*) * 100, 2) AS percentage
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	percentage
1	0.48

- **제품과 관련되지 않은 거래 기록을 제거하기**

```
DELETE FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE StockCode IN (
  SELECT DISTINCT StockCode
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  WHERE LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) IN (0, 1)
)
```

이 문으로 data의 행 1,915개가 삭제되었습니다.

Description 살펴보기

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

```
SELECT Description, COUNT(*) AS description_cnt
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
GROUP BY Description
ORDER BY description_cnt DESC
LIMIT 30
```

행	Description	description_cnt
1	WHITE HANGING HEART T-LIGH...	2058
2	REGENCY CAKESTAND 3 TIER	1894
3	JUMBO BAG RED RETROSPOT	1659
4	PARTY BUNTING	1409
5	ASSORTED COLOUR BIRD ORN...	1405
6	LUNCH BAG RED RETROSPOT	1345
7	SET OF 3 CAKE TINS PANTRY D...	1224
8	LUNCH BAG BLACK SKULL.	1099
9	PACK OF 72 RETROSPOT CAKE ...	1062
10	SPOTTY BUNTING	1026
11	PAPER CHAIN KIT 50'S CHRIST...	1013
12	LUNCH BAG SPACEBOY DESIGN	1006
13	LUNCH BAG CARS BLUE	1000

- **서비스 관련 정보를 포함하는 행들을 제거하기**

```
DELETE
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE Description = 'High Resolution Image' OR Description = 'Next Day Carriage'
```

이 문으로 data의 행 83개가 삭제되었습니다.

- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

```
CREATE OR REPLACE TABLE `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data` AS
SELECT
  * EXCEPT (Description),
  UPPER(Description) AS Description
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

이 문으로 이름이 data인 테이블이 교체되었습니다.

UnitPrice 살펴보기

- **UnitPrice의 최솟값, 최댓값, 평균을 구하기**

```
SELECT MIN(UnitPrice) AS min_price, MAX(UnitPrice) AS max_price, AVG(UnitPrice) AS avg_price
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	min_price	max_price	avg_price
1	0.0	649.5	2.904956757406...

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

```
SELECT COUNT(Quantity) AS cnt_quantity, MIN(Quantity) AS min_quantity, MAX(Quantity) AS max_quantity, AVG(Quantity)
AS avg_quantity
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE UnitPrice = 0
```

행	cnt_quantity	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.5151515151...

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

```
CREATE OR REPLACE TABLE `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data` AS
SELECT *
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
WHERE UnitPrice != 0
```

이 문으로 이름이 data인 테이블이 교체되었습니다.

11-7. RFM 스코어

Recency

- **InvoiceDate 컬럼을 연월일 자료형으로 변경하기**

```
SELECT DATE(InvoiceDate) AS InvoiceDay
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
```

행	InvoiceDay
1	2011-01-18
2	2011-01-18
3	2010-12-07
4	2010-12-07
5	2010-12-07
6	2010-12-07
7	2010-12-07
8	2010-12-07
9	2010-12-07
10	2010-12-07
11	2010-12-07
12	2010-12-07
13	2010-12-07

- **가장 최근 구매 일자를 MAX() 함수로 찾아보기**

```
SELECT MAX(DATE(InvoiceDate)) AS most_recent_date
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`;
```

행	most_recent_date
1	2011-12-09

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

```
SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
GROUP BY CustomerID
```

행	CustomerID	InvoiceDay
1	12346	2011-01-18
2	12347	2011-12-07
3	12348	2011-09-25
4	12349	2011-11-21
5	12350	2011-02-02
6	12352	2011-11-03
7	12353	2011-05-19
8	12354	2011-04-21
9	12355	2011-05-09
10	12356	2011-11-17
11	12357	2011-11-06
12	12358	2011-12-08
13	12359	2011-12-02

- **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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY CustomerID
);
```

행	CustomerID	recency
1	12407	49
2	12489	336
3	12577	35
4	12578	21
5	12581	39
6	12684	7
7	12712	22
8	12715	106
9	12818	178
10	12847	22
11	13052	212
12	13138	22
13	13475	191

- **최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이고 지금까지의 결과를 `user_r` 이라는 이름의 테이블로 저장하기**

```
CREATE OR REPLACE TABLE `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY CustomerID
);
```

이 문으로 이름이 user_r인 새 테이블이 생성되었습니다.

Frequency

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

```
SELECT
  CustomerID,
  COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
GROUP BY CustomerID
```

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
10	12356	3
11	12357	1
12	12358	2
13	12359	6

- **각 고객 별로 구매한 아이템의 총 수량 더하기**

```
SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
GROUP BY CustomerID
```

행	CustomerID	item_cnt
1	12346	0
2	12347	2458
3	12348	2332
4	12349	630
5	12350	196
6	12352	463
7	12353	20
8	12354	530
9	12355	240
10	12356	1573
11	12357	2708
12	12358	242
13	12359	1599

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

```
CREATE OR REPLACE TABLE `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_rf` AS

-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY CustomerID
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
    CustomerID,
    SUM(Quantity) AS item_cnt
  FROM `project-3fa5ea36-1f76-4d6b-9e3.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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_r` AS ur
  ON pc.CustomerID = ur.CustomerID;
```

이 문으로 이름이 user_rf인 새 테이블이 생성되었습니다.

Monetary

- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

```
SELECT
  CustomerID,
  ROUND(SUM(Quantity * UnitPrice), 0) AS total_spent
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
GROUP BY CustomerID;
```

행	CustomerID	total_spent
1	12346	0.0
2	12347	4310.0
3	12348	1437.0
4	12349	1458.0
5	12350	294.0
6	12352	1265.0
7	12353	89.0
8	12354	1079.0
9	12355	459.0
10	12356	2487.0
11	12357	6208.0
12	12358	928.0
13	12359	6183.0

- **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 `project-3fa5ea36-1f76-4d6b-9e3.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, 0) AS user_average
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_rf` rf
```

```
LEFT JOIN (
  -- 고객 별 총 지출액
  SELECT
    CustomerID,
    ROUND(SUM(Quantity * UnitPrice), 0) AS user_total
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY CustomerID
) ut
ON rf.CustomerID = ut.CustomerID;
```

이 문으로 이름이 user_rfm인 새 테이블이 생성되었습니다.

RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

```
SELECT *
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_rfm`
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	795.0	795.0
2	13436	1	76	1	197.0	197.0
3	15520	1	314	1	343.0	343.0
4	13298	1	96	1	360.0	360.0
5	14569	1	79	1	227.0	227.0
6	15195	1	1404	2	3861.0	3861.0
7	15471	1	256	2	454.0	454.0
8	14204	1	72	2	151.0	151.0
9	15318	1	642	3	313.0	313.0
10	12478	1	233	3	546.0	546.0
11	16528	1	171	3	244.0	244.0
12	12442	1	181	3	144.0	144.0
13	15992	1	17	3	42.0	42.0

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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_data` AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY CustomerID
)
SELECT
  ur.*,
  up.* EXCEPT (CustomerID)
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_rfm` AS ur
```

```
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

이 문으로 이름이 user_data인 새 테이블이 생성되었습니다.

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 project_name.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_interval
  FROM (
    -- (1) 구매와 구매 사이에 소요된 일수
    SELECT
      CustomerID,
      DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID ORDER BY InvoiceDate), DAY) AS interval_
    FROM
      project_name.modulabs_project.data
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM project_name.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 `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_data` AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS total_transactions,
    COUNT(DISTINCT CASE WHEN InvoiceNo LIKE 'C%' THEN InvoiceNo END) AS cancel_frequency
  FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.data`
  GROUP BY CustomerID
)
```

```
SELECT
  u.*,
  t.* EXCEPT(CustomerID),
  ROUND(t.cancel_frequency / t.total_transactions, 2) AS cancel_rate
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_data` AS u
LEFT JOIN TransactionInfo AS t
ON u.CustomerID = t.CustomerID;
```

ℹ 이 문으로 이름이 user_data인 테이블이 교체되었습니다.

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

```
SELECT *
FROM `project-3fa5ea36-1f76-4d6b-9e3.modulabs_project.user_data`
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval	total_transactions	cancel_frequency	cancel_rate
1	13307	1	4	120	15.0	15.0	1	0.0	1	0	0.0
2	17347	1	216	86	229.0	229.0	1	0.0	1	0	0.0
3	16093	1	20	106	17.0	17.0	1	0.0	1	0	0.0
4	17896	1	288	23	256.0	256.0	2	0.0	1	0	0.0
5	12548	1	48	166	67.0	67.0	4	0.0	1	0	0.0
6	14027	1	62	60	105.0	105.0	6	0.0	1	0	0.0
7	17879	1	45	173	179.0	179.0	6	0.0	1	0	0.0
8	17958	1	84	116	508.0	508.0	7	0.0	1	0	0.0
9	14890	1	51	253	126.0	126.0	7	0.0	1	0	0.0
10	15739	1	472	66	449.0	449.0	8	0.0	1	0	0.0
11	16457	1	140	218	208.0	208.0	9	0.0	1	0	0.0
12	12430	1	116	43	216.0	216.0	10	0.0	1	0	0.0
13	16829	1	99	156	197.0	197.0	11	0.0	1	0	0.0

회고

Keep :

- SQL을 사용해서 데이터를 전처리하고 RFM 분석에 알맞는 feature를 생성하여 저장하는 파이프라인을 구축했음
- 결측치/중복값/이상값을 처리할 때에 raw 데이터를 직접 확인하고 비율을 점검하는 것으로 논리적 근거를 세워 전처리의 신뢰도를 높임

Problem :

- SQL 문법에 대한 이해가 부족하여 문법 오류를 여러번 수정하고 점검함
- Quantitiy등의 컬럼에는 아직 양이 음수인 행이 있는 등 이해할 수 없는 이상값이 존재함을 확인함
- 전처리 과정에서 raw데이터를 덮어쓰는 방식으로 진행되어 전처리 과정에서 발생할 실수를 바로잡기 어렵다는 점을 확인함

Try :

- 컬럼별 전처리 기준을 미리 문서화한 뒤 쿼리를 작성해 불필요한 수정 과정을 줄이고자 한다
- 전처리가 완료된 데이터를 시각화할 수 있는 프로세스가 추가된다면 더 명확히 전처리 이후의 분석 방향을 정할 수 있을 것으로 예상된다