전자 상거래 데이터

(E-Commerce Data)

- 목적 : 고객 데이터를 기반으로 고객별 구매 패턴을 파악하여 고객을 세그멘테이션
  - 세그멘테이션 : 고객의 다양한 특성에 따라 그룹으로 나누는 과정
    - 그룹에 맞춘 마케팅 전략이나 제품, 서비스를 통해 효과적으로 만족도를 높임
- 분석 기법 - RFM 분석
- 데이터

  https://www.kaggle.com/datasets/carrie1/ecommerce-data

11-2. 데이터 불러오기

데이터 살펴보기

- **테이블에 있는 10개의 행만 출력하기**

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

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

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

- **전체 데이터는 몇 행으로 구성되어 있는지 확인하기**

```
SELECT count(*) AS dataCount
FROM modulabs_project.data
```

[결과 이미지를 넣어주세요]

행	dataCount
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 modulabs_project.data
```

행	COUNT_InvoiceNo	COUNT_StockCode	COUNT_Description	COUNT_Quantity	COUNT_InvoiceDate	COUNT_UnitPrice	COUNT_CustomerID	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_percent
FROM 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_percent
FROM 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_perce
FROM 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_percenta
FROM 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_perce
FROM 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_percent
FROM 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_percen
FROM 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_percentag
FROM modulabs_project.data
```

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

👉 결측치가 있는 행은 CustomerID, Description이고
각각 24.93%, 0.27%의 결측치 비율 존재

결측치 처리 전략

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

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

행	Description
1	?
2	wrongly marked carton 22804
3	CREAM HANGING HEART T-LIG...
4	WHITE HANGING HEART T-LIG...

👉 같은 제품코드(85123A)를 가진 설명이 여러개로 나누어져 있음
⇒ 제품은 1개인데 다른 설명이 들어감(일관성 결여)
⇒ 누락된 비율이 낮아 DROP

결측치 처리

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

```
DELETE FROM modulabs_project.data
WHERE Description IS NULL
OR CustomerID 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 duplicate
FROM (
    SELECT COUNT(*) AS count
    FROM modulabs_project.data
    GROUP BY InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Countr
```

```
    HAVING COUNT(*) > 1
)
```

행	duplicate
1	4837

👉 완전히 동일한 행이거나 동일한 거래 시간을 포함한 행은
데이터 오류일 가능성이 존재함
⇒ 동일한 행 DROP

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
```

ⓘ 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) as CountInvoiceNo
FROM modulabs_project.data
```

행	CountInvoiceNo
1	22190

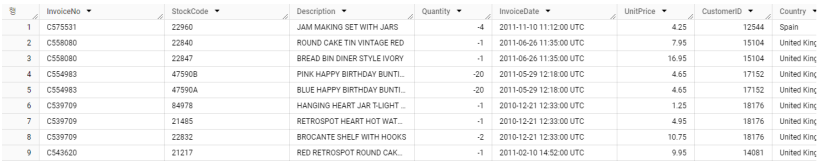
- **고유한 `InvoiceNo`를 앞에서부터 100개를 출력하기**

```
SELECT DISTINCT InvoiceNo as UniqueInvoiceNo
FROM modulabs_project.data
LIMIT 100
```

행	UniqueInvoiceNo
1	574301
2	C575531
3	557305
4	543008
5	549735
6	554032
7	561387
8	574868
9	574827
10	546015

- **`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
```



행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	C575531	22960	JAM MAKING SET WITH JARS	-4	2011-11-10 11:12:00 UTC	4.25	12544	Spain
2	C558080	22840	ROUND CAKE TIN VINTAGE RED	-1	2011-06-26 11:35:00 UTC	7.95	15104	United Kinc
3	C558080	22847	BREAD BIN DINER STYLE IVORY	-1	2011-06-26 11:35:00 UTC	16.95	15104	United Kinc
4	C554983	47590B	PINK HAPPY BIRTHDAY BUNTI..	-20	2011-05-29 12:18:00 UTC	4.65	17152	United Kinc
5	C554983	47590A	BLUE HAPPY BIRTHDAY BUNTI..	-20	2011-05-29 12:18:00 UTC	4.65	17152	United Kinc
6	C539709	84978	HANGING HEART JAR T-LIGHT ..	-1	2010-12-21 12:33:00 UTC	1.25	18176	United Kinc
7	C539709	21485	RETROSPOT HEART HOT WAT..	-1	2010-12-21 12:33:00 UTC	4.95	18176	United Kinc
8	C539709	22832	BROCANTE SHELF WITH HOOKS	-2	2010-12-21 12:33:00 UTC	10.75	18176	United Kinc
9	C543620	21217	RED RETROSPOT ROUND CAK..	-1	2011-02-10 14:52:00 UTC	9.95	14081	United Kinc

👉 InvoiceNo는 C로 시작하면 취소한 거래임

⇒ 고객 행동과 제품 선호도에 대한 이해를 높이기 위해 취소된 거래도 고려

*** Quantity가 음수임 ***

⇒ 취소를 많이한 제품들의 특이점은 보이지 않음

(오히려 취소 여부와 상관 없이 영국에 밀집되어 있음)

⇒ 그럼 추가 처리를 해야할까??

❗ 초기 목표 : 고객들의 구매 최신성(R), 구매 빈도(F), 구매 금액(M)에 따라 세그멘테이션 ⇒ 고객의 취소 패턴을 이해하는 것도 중요함

- **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 modulabs_project.data
```

행	f0_
1	2.2

`StockCode` 살펴보기

- **고유한 `StockCode`의 개수를 출력하기**

```
SELECT COUNT(DISTINCT StockCode)
FROM 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 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

👉 제품 코드는 대부분 5 ~ 6자리로 구성되어 있음
POST의 경우 제품에 관련한 설명이 아닌
서비스나 배송비 같은 형태를 코드로 남긴 것 일수도 있음
⇒ 제품 구매에 초점이 맞춰져 있기에 제거해야할듯
⇒ 비정상적인 값들의 경우 숫자가 0 ~ 1개 정도 포함되어 있음

- `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;
```

행	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

```
# StockCode 컬럼에 있던 값 중 0 부터 9사이 숫자를 공백으로 대체하는 코드
REGEXP_REPLACE(StockCode, r'[0-9]', '')

# 총 문자열 길이 - 숫자를 제외한 문자 개수
LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
```

- `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 StockCode IN ('POST', 'D', 'C2', 'M', 'BANK CHARGES', 'PADS', 'DOT', 'CRU
FROM modulabs_project.data
```

행	ErrorRating
1	0.48

👉 **제품 구매에 기반하여 세그먼테이션 하는 것이 목적이므로 DROP**

- **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
    )
)
```

ⓘ 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 modulabs_project.data
GROUP BY Description
ORDER BY description_cnt DESC
LIMIT 30
```

행	Description	description_cnt
1	WHITE HANGING HEART T-LIG...	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 ...	1224
8	LUNCH BAG BLACK SKULL.	1099
9	PACK OF 72 RETROSPOT CAKE...	1062
10	SPOTTY BUNTING	1026

👉 **'Next Day Carriage'나 'High Resolution Image' 경우 제품에 대한 Descrition이 아님**
⇒ DROP

대문자로 표준화 하여 데이터셋 일관성 유지

- **서비스 관련 정보를 포함하는 행들을 제거하기**

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

이 문으로 data의 행 83개가 삭제되었습니다.

- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

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

이 문으로 이름이 data인 테이블이 교체되었습니다.

```
* EXCEPT (Description) -> 모든 컬럼들을 가져오되, (Description)은 제외
- 제거하고자 하는 몇 개의 컬럼을 적을 때 활용함
```

`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;
```

행	min_price	max_price	avg_price
1	0.0	649.5	2.904956757406...

👉 상품 1개의 최소 가격이 0인 경우가 존재함

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

```
SELECT COUNT(*) AS cnt_quantity, MIN(UnitPrice) AS min_price, MAX(UnitPrice) AS max_price, AVG(unitP
FROM modulabs_project.data
WHERE UnitPrice = 0
```

행	cnt_quantity	min_price	max_price	avg_price
1	33	0.0	0.0	0.0

👉 UnitPrice가 0인 경우인 행 개수가 33개로 적은 것을 알수 있음

⇒ 별 다른 특징이 있는 것이 아닌 오류일 가능성이 존재함

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

```
CREATE OR REPLACE TABLE modulabs_project.data AS
SELECT *
```

```
FROM modulabs_project.data
WHERE UnitPrice != 0
```

이 문으로 이름이 data인 테이블이 교체되었습니다.

11-7. RFM 스코어

Recency(고객이 마지막으로 구매한 시점)

- `InvoiceDate` 컬럼을 연월일 자료형으로 변경하기

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

행	InvoiceDay	InvoiceNo	StockCode	Quantity	InvoiceDate
1	2010-12-21	539722	22423	10	2010-12-21 13:45:00 UTC
2	2011-08-11	562973	23157	240	2011-08-11 11:42:00 UTC
3	2011-07-28	561669	22960	11	2011-07-28 17:09:00 UTC
4	2011-11-18	577314	23407	2	2011-11-18 13:23:00 UTC
5	2010-12-05	537197	22841	1	2010-12-05 14:02:00 UTC
6	2011-11-04	574469	22385	12	2011-11-04 11:55:00 UTC
7	2011-05-20	554037	22619	80	2011-05-20 14:13:00 UTC
8	2011-11-03	574138	23234	216	2011-11-03 11:26:00 UTC
9	2011-08-26	564651	23270	96	2011-08-26 14:19:00 UTC

- 가장 최근 구매 일자를 MAX() 함수로 찾아보기

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

행	most_recent_date
1	2011-11-25

- 유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기

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

행	CustomerID	most_recent_date
1	14911	2011-08-11
2	12507	2011-07-28
3	12444	2011-11-18
4	12647	2010-12-05
5	12431	2011-11-04
6	12415	2011-11-03
7	14646	2011-08-26
8	12457	2011-04-14
9	16818	2011-07-26
10	13985	2011-11-07

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

행	CustomerID	recency
1	12457	225
2	15107	316
3	12415	22
4	15804	50
5	12431	21
6	15602	8
7	17667	197
8	13256	0
9	18059	30
10	16560	344

```
MAX(InvoiceDay) OVER () - InvoiceDay -> 고객의 구매일과 마지막 구매일 간의 차이
EXTRACT(DAY FROM …) -> 계산된 날짜 차이에서 일 부분만 추출함
```

👉 마지막 구매일로부터 현재까지 경과한 일수 → 낮은 값일 수록 최근에 구매했음

⇒ 해당 데이터를 통해 오랜 시간 동안 구매하지 않은 고객들에 대한 마케팅 전략 수립

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

```
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
);
```

행	CustomerID	recency
1	13256	0
2	12444	7
3	15602	8
4	13081	15
5	13014	18
6	13985	18
7	12431	21
8	12415	22
9	14110	22
10	18059	30

Frequency(특정 기간 동안 고객의 구매 빈도 수)

👉 고객의 구매 빈도 또는 참여 빈도에 초점을 맞추며
전체 거래 건수, 구매한 아이템의 수량 건수로 계산 가능함
EX) 1명의 고객이 구매를 2번 했으나 아이템을 4개씩 구매함
⇒ 거래 건수는 2회지만 실제 구매 수량은 8개임

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

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

행	CustomerID	purchase_cnt
1	14911	2
2	12507	1
3	12444	1
4	12647	1
5	12431	1
6	12415	2
7	14646	1
8	12457	1
9	16818	1
10	13985	1

- **각 고객 별로 구매한 아이템의 총 수량 더하기**

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

행	CustomerID	item_cnt
1	14911	250
2	12507	11
3	12444	2
4	12647	1
5	12431	12
6	12415	296
7	14646	576
8	12457	1
9	16818	1
10	13985	2

- **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;
```

Monetary(고객이 지출한 총 금액)

👉 **고객이 지불한 총 금액에 초점을 맞추며**
총 지출액, 거래당 평균 거래 금액을 계산 가능
EX) 1명의 고객이 총 2번 구매를 했고, 합산 금액이 10만원인 경우,
총 지출액은 10만원, 거래당 평균 거래 금액은 5만원임
➔ 결제 금액은 낮으나 구매를 자주하는 고객과, 한번에 결제할 때 큰 금액을 결제하는 고객 데이터도 중요

- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

```
SELECT
  CustomerID,
  ROUND(SUM(UnitPrice * Quantity)) AS user_total
FROM modulabs_project.data
WHERE CustomerID = 12713
GROUP BY CustomerID
```

행	CustomerID	user_total
1	12544	54.3
2	13568	130.6
3	13824	126.6
4	14080	3.8
5	14336	145.4
6	14592	323.6
7	15104	365.4
8	15360	30.6
9	15872	225.7
10	16128	215.8

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

이 문으로 이름이 user_rfm인 테이블이 교체되었습니다.

RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

```
SELECT *
FROM 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	15083	1	38	256	88.0	88.0
3	12792	1	215	256	345.0	345.0
4	18010	1	60	256	175.0	175.0
5	14569	1	79	1	227.0	227.0
6	15520	1	314	1	343.0	343.0
7	13436	1	76	1	197.0	197.0
8	13298	1	96	1	360.0	360.0
9	13357	1	321	257	609.0	609.0
10	14476	1	110	257	193.0	193.0

11-8. 추가 Feature 추출

👉 **RFM 분석방법에는 허점이 존재함**

⇒ 사이트에 방문한 횟수가 동일하고, 비슷한 금액을 지출했으나
구매 패턴이 다른사람을 분류하지 못함

EX) 10개의 서로 다른 제품을 구매한 사람과 1개 상품을 10개 구매한 사람은 분명
다른 쇼핑 패턴을 가지고 있지만 RFM 분석으로는 같은 그룹으로 구분

- **커머스에서 구매하는 제품의 폭이 넓은 사람일수록, 장기적으로 봤을 때 온라인 커머스 사이트에서 구매를 더 많이 할 가능성이 높음**
- **유저의 구매 패턴 속에서 뽑아낼 수 있는 추가적인 특징이 필요할 수도 있음**
  1. 구매하는 제품의 다양성
  2. 평균 구매 주기
  3. 구매 취소 경향성

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;
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
5	18113	1	72	368	76.0	76.0	1
6	17331	1	16	123	175.0	175.0	1
7	16323	1	50	196	208.0	208.0	1
8	18184	1	60	15	50.0	50.0	1
9	14119	1	-2	354	-20.0	-20.0	1
10	13120	1	12	238	31.0	31.0	1
11	16995	1	-1	372	-1.0	-1.0	1
12	13703	1	10	318	100.0	100.0	1
13	14705	1	100	198	179.0	179.0	1
14	13017	1	48	7	204.0	204.0	1

👉 높은 숫자가 나오는 것은 해당 고객이 다양한 제품들을 구매한다는 의미

2. 평균 구매 주기

- **고객들의 쇼핑 패턴을 이해하는 것을 목표 (고객 별 재방문 주기 살펴보기)**

- **평균 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
CASE WHEN ROUND(AVG(interval_), 2) IS NULL THEN 0 ELSE ROUND(AVG(interval_), 2) END AS average_interval
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval
1	14432	6	2013	9	2248.0	374.6666666666…	256	0.2
2	12428	11	3477	25	6366.0	578.7272727272…	256	0.87
3	13268	14	3525	17	3106.0	221.8571428571…	256	0.56
4	13270	1	200	366	590.0	590.0	1	0.0
5	17752	1	192	359	81.0	81.0	1	0.0
6	16995	1	-1	372	-1.0	-1.0	1	0.0
7	13017	1	48	7	204.0	204.0	1	0.0
8	17443	1	504	219	534.0	534.0	1	0.0
9	16148	1	72	296	76.0	76.0	1	0.0
10	13120	1	12	238	31.0	31.0	1	0.0

```
# 과거 구매 기록이 있는 경우 재방문까지 걸린 일자의 평균을 계산하여 넣고,
# 만약 과거 구매 기록이 없는 경우 0을 넣음
CASE WHEN ROUND(AVG(interval_), 2) IS NULL THEN 0 ELSE ROUND(AVG(interval_), 2) END AS average_interval

# 단 1개의 구매 건만 있어 과거의 구매일이 없는 경우임
ROUND(AVG(interval_), 2) IS NULL
```

👉 **고객들의 구매와 구매 사이의 기간이 평균적으로 몇 일인지를 보여주는 평균 일수를 계산하면, 고객이 다음 구매를 언제할지 예측하는 데 도움이 됨**

3. 구매 취소 경향성

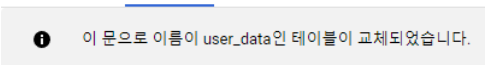
- **고객의 취소 패턴 파악하기**
  1) 취소 빈도(cancel_frequency) : 고객 별로 취소한 거래의 총 횟수
  2) 취소 비율(cancel_rate) : 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율
  - **취소 빈도와 취소 비율을 계산하고 그 결과를 `user_data` 에 통합하기**
    (취소 비율은 소수점 두번째 자리)

```
CREATE OR REPLACE TABLE modulabs_project.user_data AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(*) AS total_transactions,
```

```
    COUNT(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 END) AS cancel_frequency
  FROM modulabs_project.data
GROUP BY CustomerID
)

SELECT u.*, t.* EXCEPT(CustomerID), ROUND(t.cancel_frequency / t.total_transactions * 100, 2) cancel_ra
FROM 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 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	16406	1	116	18	154.0	154.0	54	0.0	58	0	0.0
2	16340	1	486	107	534.0	534.0	124	0.0	141	0	0.0
3	16222	1	325	318	774.0	774.0	120	0.0	122	0	0.0
4	16274	1	151	373	332.0	332.0	63	0.0	63	0	0.0
5	12534	1	645	130	981.0	981.0	62	0.0	62	0	0.0
6	12988	1	126	292	506.0	506.0	75	0.0	83	0	0.0
7	14382	1	1231	26	615.0	615.0	121	0.0	126	0	0.0
8	12371	1	582	59	1528.0	1528.0	62	0.0	62	0	0.0
9	13635	1	677	67	1071.0	1071.0	62	0.0	62	0	0.0
10	14953	1	224	25	286.0	286.0	54	0.0	54	0	0.0

회고

기존에서 파이썬을 이용하여 간단한 데이터 분석은 해본적이 있었지만,

자세한 분석과 SQL을 사용해서 분석하는 것은 처음 경험해서 굉장히 재미있었다.

또한 전자 상거래 데이터 분석 관련 기초적인 도메인 지식도 같이 공부할 수 있어서 의미가 있었다.

원래도 기본개념을 가지고 이리저리 만지며 문제를 푸는 것을 좋아하는데 시간가는줄 모르고

풀었다.