고객을 세그먼테이션하자 [프로젝트]

사전 정보

- 비즈니스 도메인: 이커머스 플랫폼
- 목표: 고객별 맞춤 마케팅 전략 수립
- 고객 세그멘테이션 방법
  - RFM(Recency, Frequency, Monetary) 분석: 고객이 얼마나 최근에, 자주, 많은 금액을 지출했는지
- 사용 데이터: Kaggle E-Commerce Data
  - 2010~2011년 이커머스 데이터셋

  ▼ 데이터 컬럼

컬럼명	설명
InvoiceNo	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	각각의 제품에 할당된 고유 코드
Description	각 제품에 대한 설명
Quantity	거래에서 제품을 몇 개 구매했는지에 대한 단위 수
InvoiceDate	거래가 일어난 날짜와 시간
UnitPrice	제품 당 단위 가격(영국 파운드)
CustomerID	각 고객에게 할당된 고유 식별자 코드
Country	주문이 발생한 국가

- 수행 절차

11-2. 데이터 불러오기

데이터 살펴보기

- **테이블에 있는 10개의 행만 출력하기**

> 💡 데이터 처리에 시간이 오래 걸리는 것을 방지하기 위해 적당히 LIMIT을 넣어 먼저 데이터 일부를 확인해보기

```
SELECT *
FROM upbeat-aura-425501-u8.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 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**

> 현실 데이터는 결측치, 이상치가 있는 경우가 많으므로, 이를 파악하는 것이 중요하다.
> 먼저 전체 데이터의 개수를 파악해본다.
>
> count(*)은 NULL을 포함하여 개수를 세기 때문에 전체 데이터의 개수가 된다.

```
SELECT count(*) as row_count
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

행	row_count
1	541909

데이터 수 세기

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

> 컬럼별 데이터 개수와 전체 데이터 개수를 비교하여 결측치가 있는 컬럼 파악하기
>
> count(컬럼명)은 NULL을 제외하고 개수를 세기 때문에 전체 데이터 개수와의 차이를 비교해볼 수 있다.

```
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 upbeat-aura-425501-u8.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**

> 💡 결측치 비율을 계산한 결과,
>
> CustomerID 결측치 비율은 24.93%으로 높게 나왔다. 이 프로젝트의 목표는 고객별 마케팅 전략을 수립하는 것으로 고객 세그멘테이션이 필요하다. 따라서 고객 식별자 데이터는 정확해야 한다. CustomerID가 없는 행을 제거하는 것이 합리적이다.
>
> Description 결측치 비율은 0.27%로 비교적 적은 편이다. 삭제할지, 값을 대체할지 등 값을 보며 확인이 필요하다.

  - **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_perce
FROM upbeat-aura-425501-u8.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_perce
FROM upbeat-aura-425501-u8.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_per
FROM upbeat-aura-425501-u8.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_percen
FROM upbeat-aura-425501-u8.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_per
FROM upbeat-aura-425501-u8.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_perce
FROM upbeat-aura-425501-u8.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_perc
FROM upbeat-aura-425501-u8.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_percent
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

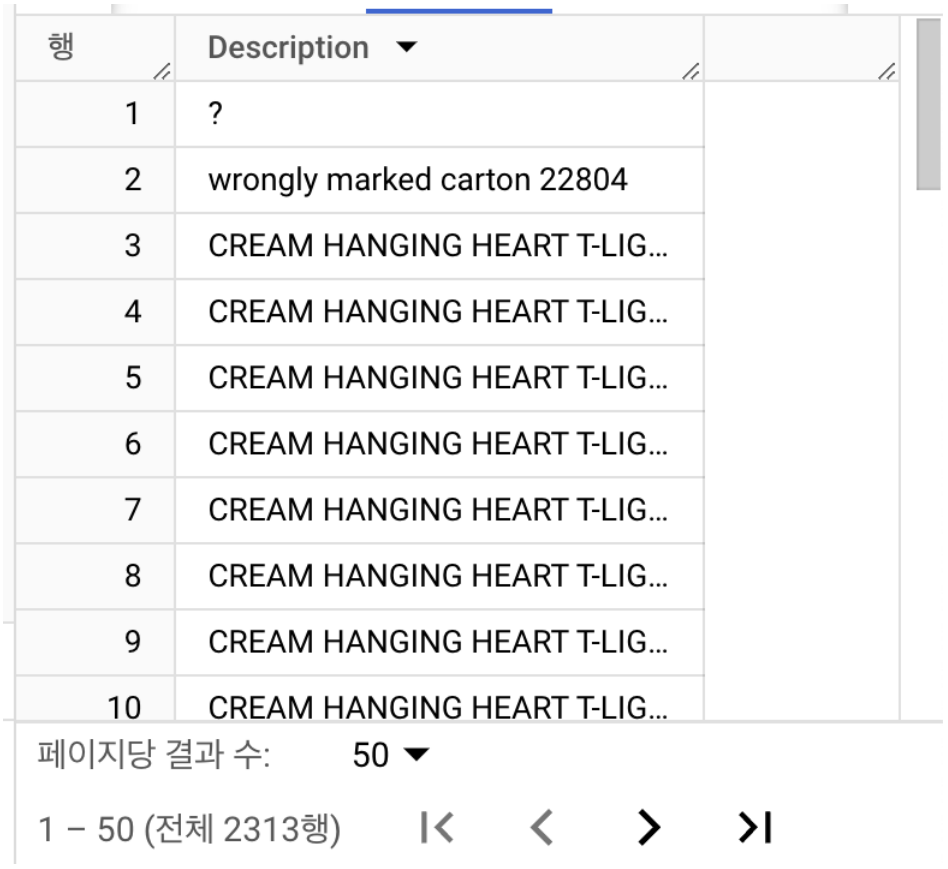
행	column_name	missing_percentage
1	Country	0.0
2	CustomerID	24.93
3	Quantity	0.0
4	Description	0.27
5	StockCode	0.0
6	InvoiceDate	0.0
7	UnitPrice	0.0
8	InvoiceNo	0.0

결측치 처리 전략

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

> 💡 Description 값을 보면 같은 제품(예: StockCode = ‘85123A’)임에도 항상 같은 상세 설명(Description) 값을 가지지 않는다는 문제가 있었다.
>
> 따라서, StockCode를 기반으로 누락된 Description 값을 대체하기에는 신뢰성이 떨어진다. 또한, 누락된 비율이 0.27%로 매우 낮기 때문에, 데이터 일관성 문제가 후속 분석 과정에 영향을 주지 않기 위해서 누락된 설명이 있는 행을 제거하는 것이 낫다.

```
SELECT Description
FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE StockCode = '85123A'
;
```



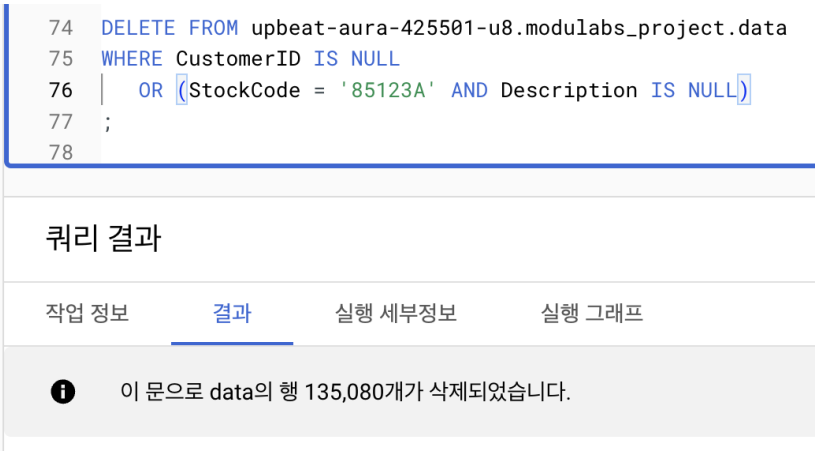
행	Description
1	?
2	wrongly marked carton 22804
3	CREAM HANGING HEART T-LIG...
4	CREAM HANGING HEART T-LIG...
5	CREAM HANGING HEART T-LIG...
6	CREAM HANGING HEART T-LIG...
7	CREAM HANGING HEART T-LIG...
8	CREAM HANGING HEART T-LIG...
9	CREAM HANGING HEART T-LIG...
10	CREAM HANGING HEART T-LIG...

결측치 처리

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

> CustomerID 결측값을 가진 데이터, 서로 다른 Description을 갖거나 결측값을 갖는 데이터는 삭제하여 더 정돈되고 신뢰할 수 있는 데이터셋을 구축한다.

```
DELETE FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE CustomerID IS NULL
   OR (StockCode = '85123A' AND Description IS NULL)
;
```



```
74  DELETE FROM upbeat-aura-425501-u8.modulabs_project.data
75  WHERE CustomerID IS NULL
76    OR (StockCode = '85123A' AND Description IS NULL)
77  ;
78
```

11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

- **중복된 행의 수를 세어보기**

> 💡 완전히 동일한 행은 데이터 오류일 가능성이 높다. 중복 행을 유지하면 분석 결과에 영향을 줄 수 있으니 중복행이 있는지 파악해본다.

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

```
WITH DCalc AS (
  SELECT *, COUNT(*)
  FROM upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY
      InvoiceNo, StockCode, Description, Quantity
    , InvoiceDate, UnitPrice, CustomerID, Country
  HAVING COUNT(*) > 1
)
SELECT COUNT(*) AS count_rows
FROM DCalc
;
```

행	count_rows
1	4837

중복값 처리

- **중복값을 제거하는 쿼리문 작성하기**

> 💡 중복 데이터를 제거하여 데이터셋을 정리한다.

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

```
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.modulabs_project.data
AS
SELECT DISTINCT *
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

```sql
-- 중복 제거하기
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.modulabs_project.data
AS
SELECT DISTINCT *
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 이름이 data인 테이블이 교체되었습니다.

```sql
SELECT COUNT(*)
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세부정보

행	f0_
1	401604

11-6. 데이터 전처리(3): 오류값 처리

컬럼 단위로 데이터를 살펴보면서 클렌징이 필요한 값들이 있는지 살펴본다.

`InvoiceNo` 살펴보기

`InvoiceNo` 데이터의 특징이나 경향성이 있는지 살펴본다.

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

```sql
SELECT COUNT(DISTINCT InvoiceNo) AS COUNT_Unique_InvoiceNo
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

행	COUNT_Unique_InvoiceNo
1	22190

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

```
SELECT DISTINCT InvoiceNo
FROM upbeat-aura-425501-u8.modulabs_project.data
LIMIT 100
;
```

행	InvoiceNo
1	574301
2	C575531
3	557305
4	543008
5	549735
6	554032
7	561387
8	574868
9	574827
10	546015
11	551859

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

> 'C'로 시작한다면 취소한 거래를 의미한다. 취소한 거래의 공통적인 특성이나 패턴이 있는지 파악해본다.
>
> 고객 행동과 제품 선호도에 대한 이해를 높이기 위해서는 취소된 거래들도 고려해야 한다.
> 본 프로젝트 목표가 고객을 세그멘테이션하는 것이므로, 고객의 취소 패턴을 이해하는 것도 중요하다. 취소된 거래에 공통점이 있는지 살펴보거나, 추천 가능성이 높은 제품을 찾아서 진단하는 방법도 있을 수 있다.
>
> 따라서, 취소된 거래 데이터는 유지하되, 명확하게 표시하여 추가 분석을 용이하게 만들어주는 것도 하나의 전략이다.
>
> 아래 결과에서는 Quantity가 음수라는 특징을 보인다.
> 취소를 많이 한 제품들의 가격대가 높았는지, 또는 거래 지역이 특정 지역에 몰려 있는지도 살펴보았지만, 그런 경향성은 크게 보이지 않는다.
> (Country는 취소 여부와 상관없이 United Kingdom에 밀집되어 있어서 경향성이 아님)

```
SELECT *
FROM upbeat-aura-425501-u8.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	22847	BREAD BIN DINER STYLE IVORY	-1	2011-06-26 11:35:00 UTC	16.95	15104	United Kingdom
3	C558080	22840	ROUND CAKE TIN VINTAGE RED	-1	2011-06-26 11:35:00 UTC	7.95	15104	United Kingdom
4	C554983	47590A	BLUE HAPPY BIRTHDAY BUNTI...	-20	2011-05-29 12:18:00 UTC	4.65	17152	United Kingdom
5	C554983	47590B	PINK HAPPY BIRTHDAY BUNTI...	-20	2011-05-29 12:18:00 UTC	4.65	17152	United Kingdom
6	C539709	21485	RETROSPOT HEART HOT WAT...	-1	2010-12-21 12:33:00 UTC	4.95	18176	United Kingdom
7	C539709	22832	BROCANTE SHELF WITH HOOKS	-2	2010-12-21 12:33:00 UTC	10.75	18176	United Kingdom
8	C539709	84978	HANGING HEART JAR T-LIGHT ...	-1	2010-12-21 12:33:00 UTC	1.25	18176	United Kingdom
9	C543620	21217	RED RETROSPOT ROUND CAK...	-1	2011-02-10 14:52:00 UTC	9.95	14081	United Kingdom
10	C546858	21534	DAIRY MAID LARGE MILK JUG	-1	2011-03-17 14:24:00 UTC	4.95	14081	United Kingdom
11	C546858	22839	3 TIER CAKE TIN GREEN AND ...	-1	2011-03-17 14:24:00 UTC	14.95	14081	United Kingdom

페이지당 결과 수: 50 1 – 50 (전체 100행)

- **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 upbeat-aura-425501-u8.modulabs_project.data
;
```

행	canceled_percentage
1	2.2

`StockCode` 살펴보기

💡 `StockCode` 데이터의 특징이나 경향성이 있는지 살펴본다.

- **고유한 `StockCode` 의 개수를 출력하기**

```
SELECT COUNT(DISTINCT StockCode) AS COUNT_Unique_StockCode
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

행	COUNT_Unique_StockCode
1	3684

- **어떤 제품이 가장 많이 판매되었는지 보기 위하여 `StockCode` 별 등장 빈도를 출력하기**

💡 가장 많이 판매된 제품을 살펴보면, 고객들이 자주 구매하는 인기 제품군이나 카테고리에 대한 인사이트도 얻을 수 있다.

  - 상위 10개의 제품들을 출력하기

```
SELECT
    StockCode
  , COUNT(*) AS sell_cnt
FROM upbeat-aura-425501-u8.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**

> 💡 위 결과에서 'POST'라는 다른 형태의 값이 발견되었다.
> 실제 제품보다는 서비스나 배송비 같은 형태를 코드로 남긴 것일 수 있다. 본 프로젝트는 고객들의 제품 구매에 초점이 맞춰져 있기 때문에 이런 값을 제거하는 것이 좋다.
>
> 이런 형태의 값들이 더 있는지, 어떤 모습인지 살펴본다.
> 'BANK CHARGES, POST' 등 제품과 관련되지 않은 거래 기록으로 보인다.

  - **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 project_name.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`의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고**

  💡 다른 형태의 StockCode 값의 전체 데이터 대비 비율은 어떠한지 파악한다.

  - **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]', ''))) <= 1
                                    THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS non_num_code_percen
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

행	percentage
1	0.48

- **제품과 관련되지 않은 거래 기록을 제거하기**

```
DELETE FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE StockCode IN (
  SELECT DISTINCT StockCode
  FROM (
    SELECT StockCode,
      LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
    FROM upbeat-aura-425501-u8.modulabs_project.data
  )
  WHERE number_count <= 1
);
```

```sql
DELETE FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE StockCode IN (
  SELECT DISTINCT StockCode
  FROM (
    SELECT StockCode,
      LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
    FROM upbeat-aura-425501-u8.modulabs_project.data
  )
  WHERE number_count <= 1
);
```

쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 data의 행 1,915개가 삭제되었습니다.

`Description` 살펴보기

`Description` 데이터의 특징이나 경향성이 있는지 살펴본다.

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

출력 결과를 보면 제품군들이 대부분 가정용품, 주방용품, 점심 도시락, 장식품과 관련된 것들로 보인다.
Description이 모두 대문자로 되어 있는데, 이는 데이터베이스에 제품 설명을 입력할 때 사용되는 표준화된 형식일 수도 있다. 그러나 혹시라도 대문자와 소문자가 혼합된 스타일로 입력된 설명이 있는지 확인해 보고, 혼합되어 있다면 표준화시켜줘야 한다.

```sql
SELECT Description, COUNT(*) AS description_cnt
FROM upbeat-aura-425501-u8.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
11	PAPER CHAIN KIT 50'S CHRIST…	1013

페이지당 결과 수: 50 1 – 30 (전체 30행)

- **서비스 관련 정보를 포함하는 행들을 제거하기**

실제 제품 정보와 관련 없는, 서비스 관련 정보로 보이는 'Next Day Carriage'나 'High Resolution Image' 포함 행들을 삭제한다.

```
DELETE
FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE Description IN ('Next Day Carriage', 'High Resolution Image')
;
```

```
130  DELETE
131  FROM upbeat-aura-425501-u8.modulabs_project.data
132  WHERE Description IN ('Next Day Carriage', 'High Resolution Image')
133  ;
134
```

쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 data의 행 83개가 삭제되었습니다.

- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

대소문자 혼합값을 대문자로 표준화하여 데이터셋의 품질을 향상시킨다.

`* EXCEPT (Description)` 는 테이블의 모든 컬럼들을 가져오되, Description은 제외하라는 의미이다. 포함하려고 하는 컬럼명을 일일이 쓰는 것보다 제거하고자 하는 몇 개 컬럼을 적을 때 자주 활용한다.

```
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.modulabs_project.data AS
SELECT
  * EXCEPT (Description),
  UPPER(Description) AS Description
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

```
138  CREATE OR REPLACE TABLE upbeat-aura-425501-u8.modulabs_project.data AS
139  SELECT
140    * EXCEPT (Description),
141    UPPER(Description) AS Description
142  FROM upbeat-aura-425501-u8.modulabs_project.data
143  ;
144
```

쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 이름이 data인 테이블이 교체되었습니다.

`UnitPrice` 살펴보기

`UnitPrice` 데이터의 특징이나 경향성이 있는지 살펴본다.

`Quantity`와 `InvoiceDate`에 대한 데이터를 분석했을 때에는 크게 문제가 되는 이상 데이터가 없어서 넘어간다. 데이터를 분석할 때 전처리하면 좋을 것 같은 데이터가 나타난다면, 분석가/과학자의 판단에 따라 처리 방법을 선택할 수 있다.

- **`UnitPrice`의 최솟값, 최댓값, 평균을 구하기**

  최솟값, 최댓값, 평균 데이터를 확인해 봄으로써 이상치를 살펴본다.
  최솟값이 0으로 나온 것으로 보아, 단가가 0원인 제품이 존재한다는 것이며, 이는 무료 제품이거나 데이터 오류일 수도 있다는 의미이다.

```
SELECT
    MIN(UnitPrice) AS min_price
  , MAX(UnitPrice) AS max_price
  , AVG(UnitPrice) AS avg_price
FROM upbeat-aura-425501-u8.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**

> 단가가 0원인 거래의 성격을 제대로 이해하기 위해서는 데이터를 더 자세히 살펴볼 필요가 있다. 단가가 0원인 제품을 상세하게 분석해보면서 특정한 패턴이 있는지 살펴본다.
>
> 단가가 0인 데이터의 개수는 33개로 비교적 적다. 데이터의 수가 적은 걸 보니 무료 제품이라기보다 데이터 오류일 가능성이 더 높으므로, 이 데이터들을 삭제하기로 한다.
> 구매 수량(Quantity)은 최소 1개부터 최대 12,540개에 이르기까지 굉장히 큰 편차를 가진다.

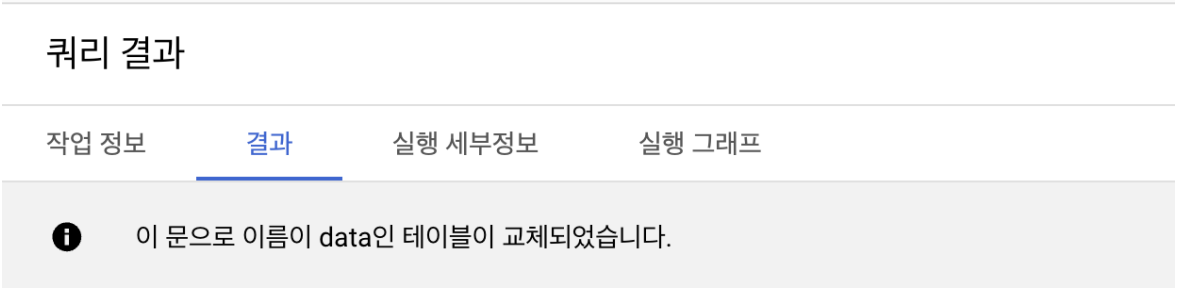
```
SELECT
    COUNT(*) AS cnt_quantity
  , MIN(Quantity) AS min_quantity
  , MAX(Quantity) AS max_quantity
  , AVG(Quantity) AS avg_quantity
FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE UnitPrice = 0
;
```

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 upbeat-aura-425501-u8.modulabs_project.data AS
SELECT *
FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE UnitPrice != 0
;
```

```sql
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.modulabs_project.data AS
SELECT *
FROM upbeat-aura-425501-u8.modulabs_project.data
WHERE UnitPrice != 0
;
```



다른 컬럼들은 크게 오류값 처리를 할 게 없어 오류값 처리는 여기서 마친다. 이렇게 다양한 방식으로 데이터를 파고 들어 이상치나 오류값들을 식별해서 처리할 수 있다.

11-7. RFM 스코어

Recency: 고객이 마지막으로 구매한 시점.
최근에 구매한 고객들은 더 자주 구매할 가능성이 높기 때문에, 최신성 점수가 높은지를 고려한다.

Frequency: 특정 기간 동안 고객이 얼마나 자주 우리의 제품이나 서비스를 구매하는지를 나타냄.
빈번하게 구매하는 고객은 충성도가 높은 고객일 확률이 높기 때문에, 빈도 점수가 높은지를 고려한다.

Monetary: 고객이 지출한 총 금액.
많은 금액을 지불한 고객일수록 더 가치가 높은 충성 고객일 수 있다. 앞으로도 우리의 제품과 사이트에 많은 돈을 지불할 수 있는 고객이므로, 가치 점수가 높은지를 함께 고려한다.

Recency

Recency 단계에서는 고객이 얼마나 최근에 구매를 했는지에 중점을 둔다.

그러므로 '마지막 구매일로부터 현재까지 경과한 일수'를 계산해야 한다. 낮은 값일수록 고객이 최근에 구매를 했음을 의미하며, 제품이나 서비스에 더 관심을 보인다고 예측할 수 있다.

Recency를 통해 오랜 시간 동안 구매를 하지 않았던 고객을 발견하고, 다시 제품과 서비스로 불러들이기 위한 마케팅 전략을 맞춤화해볼 수도 있다.

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

💡 연월일 자료형으로 변경하여 날짜에 해당하는 부분만 남겨둔다.

```
SELECT DATE(InvoiceDate) AS InvoiceDay, *
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

행	InvoiceDay	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	2011-11-03	574301	23511	6	2011-11-03 16:15:00 UTC	2.08	12544	Spain	EMBROIDERED RIBBON REE...
2	2011-11-03	574301	22144	6	2011-11-03 16:15:00 UTC	2.1	12544	Spain	CHRISTMAS CRAFT LITTLE ...
3	2011-11-03	574301	22910	6	2011-11-03 16:15:00 UTC	2.95	12544	Spain	PAPER CHAIN KIT VINTAGE ...
4	2011-11-03	574301	85049E	12	2011-11-03 16:15:00 UTC	1.25	12544	Spain	SCANDINAVIAN REDS RIBB...
5	2011-11-03	574301	22751	4	2011-11-03 16:15:00 UTC	3.75	12544	Spain	FELTCRAFT PRINCESS OLIV...
6	2011-11-03	574301	23240	6	2011-11-03 16:15:00 UTC	4.15	12544	Spain	SET OF 4 KNICK KNACK TIN...
7	2011-11-03	574301	20749	4	2011-11-03 16:15:00 UTC	7.95	12544	Spain	ASSORTED COLOUR MINI C...
8	2011-11-03	574301	85049A	12	2011-11-03 16:15:00 UTC	1.25	12544	Spain	TRADITIONAL CHRISTMAS ...
9	2011-11-03	574301	22960	6	2011-11-03 16:15:00 UTC	4.25	12544	Spain	JAM MAKING SET WITH JA...
10	2011-11-03	574301	22077	12	2011-11-03 16:15:00 UTC	1.95	12544	Spain	6 RIBBONS RUSTIC CHARM
11	2011-11-03	574301	22621	12	2011-11-03 16:15:00 UTC	1.65	12544	Spain	TRADITIONAL KNITTING NA...

- **가장 최근 구매 일자를 MAX() 함수로 찾아보기**

💡 실제 회사에서는 오늘 날짜를 기준으로 최종 구매일이 며칠 지났는지 계산하지만, 이 데이터는 2010~2011년 사이의 데이터이므로 오늘로부터 꽤 오랜 시간이 지난 상태다.
따라서 여기에서는 오늘 날짜 대신 전체 고객을 통틀어 가장 최근 구매 일자를 기준으로 Recency를 구한다.

```
SELECT
    MAX(DATE(InvoiceDate)) OVER () AS most_recent_date
  , DATE(InvoiceDate) AS InvoiceDay
  , *
FROM upbeat-aura-425501-u8.modulabs_project.data
;
```

행	most_recent_date	InvoiceDay	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	2011-12-09	2010-12-10	538260	22835	4	2010-12-10 12:42:00 UTC	4.65	18178	United Kingdom	HOT WATER BOTTLE I AM SO ...
2	2011-12-09	2011-08-30	564842	22624	12	2011-08-30 13:47:00 UTC	7.65	16133	United Kingdom	IVORY KITCHEN SCALES
3	2011-12-09	2011-11-10	575681	21034	1	2011-11-10 15:32:00 UTC	0.95	16903	United Kingdom	REX CASH+CARRY JUMBO SH...
4	2011-12-09	2011-11-02	574067	85048	60	2011-11-02 14:50:00 UTC	6.95	14088	United Kingdom	15CM CHRISTMAS GLASS BAL...
5	2011-12-09	2011-11-23	578305	23284	58	2011-11-23 15:44:00 UTC	7.08	14088	United Kingdom	DOORMAT KEEP CALM AND C...
6	2011-12-09	2011-11-08	575091	23375	10	2011-11-08 14:05:00 UTC	0.82	17160	United Kingdom	50'S CHRISTMAS PAPER GIFT ...
7	2011-12-09	2011-10-20	572082	23314	4	2011-10-20 14:18:00 UTC	12.5	17672	United Kingdom	VINTAGE CHRISTMAS TABLEC...
8	2011-12-09	2011-10-28	573270	23132	3	2011-10-28 13:00:00 UTC	5.75	17418	United Kingdom	SMALL IVORY HEART WALL O...
9	2011-12-09	2011-06-30	558632	21137	24	2011-06-30 18:44:00 UTC	3.39	17930	United Kingdom	BLACK RECORD COVER FRAME
10	2011-12-09	2011-02-22	544666	22832	1	2011-02-22 15:00:00 UTC	10.75	16907	United Kingdom	BROCANTE SHELF WITH HOOKS
11	2011-12-09	2011-07-14	560098	23132	1	2011-07-14 17:44:00 UTC	5.75	16907	United Kingdom	SMALL IVORY HEART WALL O...

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

💡 Recency는 고객이 얼마나 최근에 구매를 했는지를 알아보려는 것이므로 사용자별 가장 최근 구매일을 찾아 저장한다.
다음 단계에서 최근 일자(일반 회사 데이터라면 오늘)와 사용자별 마지막 구매일(InvoiceDay)간의 차이를 구하게 된다.

```
SELECT
    CustomerID
  , MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM upbeat-aura-425501-u8.modulabs_project.data
GROUP BY CustomerID
;
```

행	CustomerID	InvoiceDay
1	12544	2011-11-10
2	13568	2011-06-19
3	13824	2011-11-07
4	14080	2011-11-07
5	14336	2011-11-23
6	14592	2011-11-04
7	15104	2011-06-26
8	15360	2011-10-31
9	15872	2011-11-25
10	16128	2011-11-22
11	16384	2011-09-11

- **가장 최근 일자(`most_recent_date`)와 유저별 마지막 구매일(`InvoiceDay`)간의 차이를 계산하기**

  💡 최근 일자(일반 회사 데이터라면 오늘)와 사용자별 마지막 구매일(InvoiceDay)간의 차이로 recency를 구한다.

```
SELECT
  CustomerID,
  EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
  FROM upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID
);
```

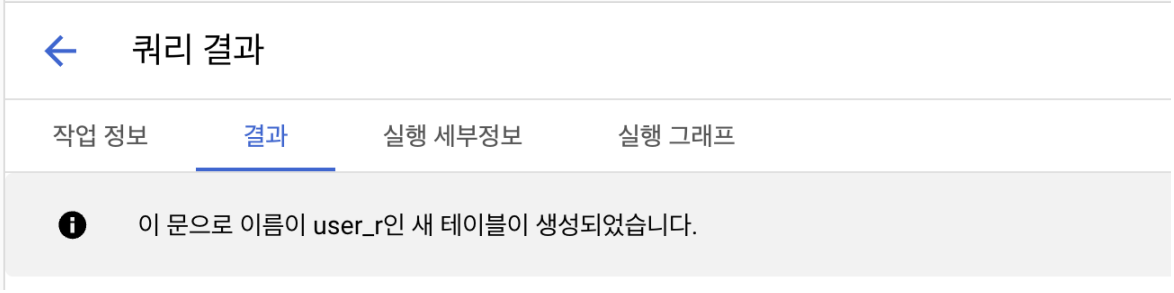
행	CustomerID	recency
1	14593	21
2	16387	322
3	17940	49
4	15153	31
5	17457	126
6	17974	24
7	13631	99
8	12362	3
9	15434	8
10	13649	256
11	16215	80

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

  💡 추후 최종적으로 사용할 데이터를 만들기 위해 위해 테이블로 저장한다.

```
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.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 upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID
);
```

```sql
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.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 upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID
);
```



Frequency

Frequency를 계산하는 단계에서는 고객의 구매 빈도 또는 참여 빈도에 초점을 맞춘다.

전체 거래 건수로 계산을 할 수도 있고, 구매한 아이템의 수량을 합하여 계산할 수도 있다.
예를 들어 한 명의 고객이 구매를 2번 했는데 각각 아이템을 4개씩 구매한 경우, 해당 고객의 거래 건수는 2회겠지만 실제로 구매한 수량은 8개가 된다.

이 두가지 측면을 모두 포착하기 위해 두 개를 모두 계산해 본다.

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

  거래 건은 InvoiceNo를 기준으로 파악하면 되기 때문에, 고객마다 고유한 InvoiceNo의 수를 세어준다.

```sql
SELECT
    CustomerID
  , COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM upbeat-aura-425501-u8.modulabs_project.data
GROUP BY CustomerID
;
```

행	CustomerID	purchase_cnt
1	12544	2
2	13568	1
3	13824	5
4	14080	1
5	14336	4
6	14592	3
7	15104	3
8	15360	1
9	15872	2
10	16128	5
11	16384	2

- **각 고객 별로 구매한 아이템의 총 수량 더하기**

  💡 고객별로 구매한 아이템의 총 수량을 계산한다.

```
SELECT
    CustomerID
  , SUM(Quantity) AS item_cnt
FROM upbeat-aura-425501-u8.modulabs_project.data
GROUP BY CustomerID
;
```

행	CustomerID	item_cnt
1	12544	130
2	13568	66
3	13824	768
4	14080	48
5	14336	1759
6	14592	407
7	15104	633
8	15360	223
9	15872	187
10	16128	988
11	16384	260

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

  💡 추후 최종적으로 사용할 데이터를 만들기 위해 위해 테이블로 저장한다.

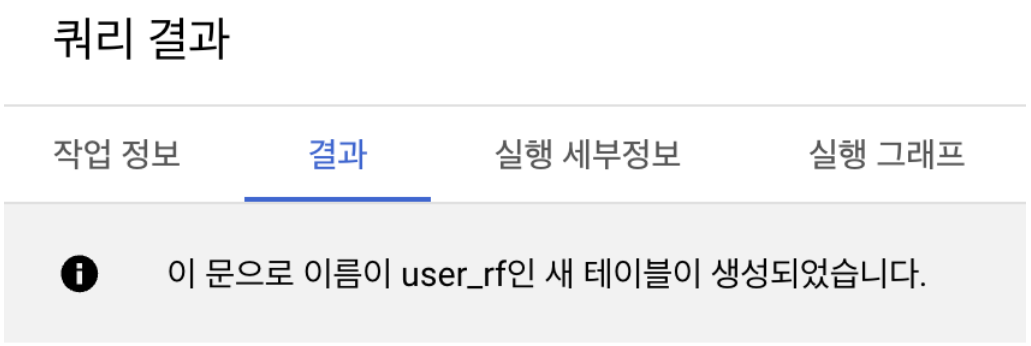
```
CREATE OR REPLACE TABLE upbeat-aura-425501-u8.modulabs_project.user_rf AS

-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
  SELECT
      CustomerID
    , COUNT(DISTINCT InvoiceNo) AS purchase_cnt
  FROM upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
      CustomerID
    , SUM(Quantity) AS item_cnt
  FROM upbeat-aura-425501-u8.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 upbeat-aura-425501-u8.modulabs_project.user_r AS ur
  ON pc.CustomerID = ur.CustomerID;
```



Monetary

- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

```
SELECT
    CustomerID
  , ROUND(SUM(UnitPrice * Quantity), 0) AS user_total
from upbeat-aura-425501-u8.modulabs_project.data
GROUP BY CustomerID
;
```

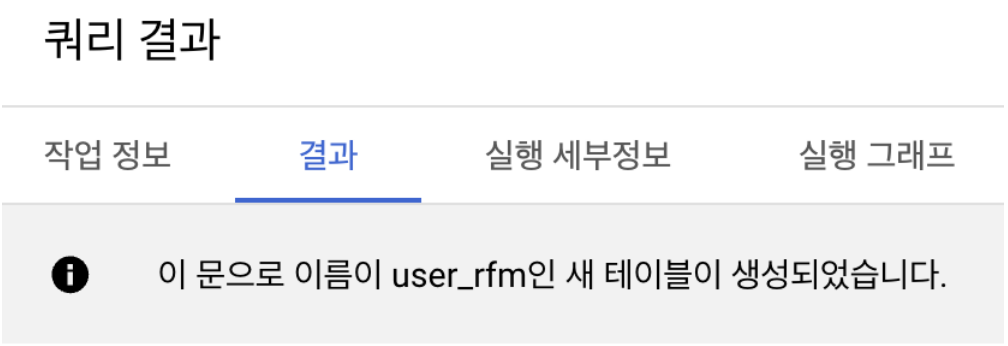
행	CustomerID	user_total
1	12544	300.0
2	13568	187.0
3	13824	1699.0
4	14080	46.0
5	14336	1615.0
6	14592	558.0
7	15104	969.0
8	15360	428.0
9	15872	316.0
10	16128	1880.0
11	16384	584.0

- **고객별 평균 거래 금액 계산**

  💡 최종적으로 사용할 데이터 테이블 user_rfm을 만든다.

  - **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 upbeat-aura-425501-u8.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 upbeat-aura-425501-u8.modulabs_project.user_rf rf
LEFT JOIN (
  -- 고객 별 총 지출액
  SELECT
      CustomerID
    , ROUND(SUM(UnitPrice * Quantity), 0) AS user_total
  from upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID
) ut
ON rf.CustomerID = ut.CustomerID
;
```



RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

```
SELECT *
FROM upbeat-aura-425501-u8.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	12792	1	215	256	345.0	345.0
3	18010	1	60	256	175.0	175.0
4	15083	1	38	256	88.0	88.0
5	13298	1	96	1	360.0	360.0
6	15520	1	314	1	343.0	343.0
7	14569	1	79	1	227.0	227.0
8	13436	1	76	1	197.0	197.0
9	13357	1	321	257	609.0	609.0
10	14476	1	110	257	193.0	193.0
11	15471	1	256	2	454.0	454.0

11-8. 추가 Feature 추출

RFM 스코어 외, 유저의 구매 패턴 속에서 뽑아낼 수 있는 추가적인 특징을 찾아본다.

1. 구매하는 제품의 다양성
2. 평균 구매 주기
3. 구매 취소 경향성

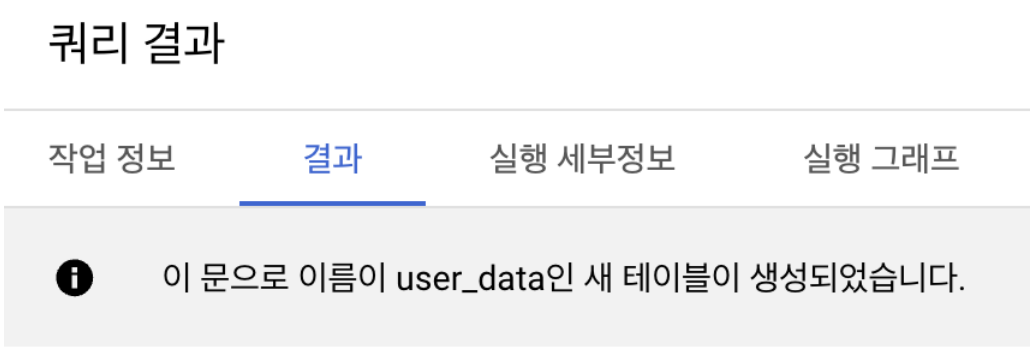
RFM은 허점이 있다.
RFM 분석으로는 사이트에 방문한 횟수가 동일하고 비슷한 금액을 지출했지만 구매 패턴이 다른 사람을 분류하지 못한다. 예를 들어 10개의 서로 다른 제품을 구매한 사람과 1개 상품을 10개 구매한 사람은 분명 다른 쇼핑 패턴을 가지고 있지만 RFM 분석으로는 같은 그룹으로 구분한다.

1. 구매하는 제품의 다양성

구매하는 제품의 폭이 넓은 사람일수록, 장기적으로 봤을 때 온라인 커머스 사이트에서 구매를 더 많이 할 가능성이 높아진다.
디지털 제품만을 구매하는 사람이라면 새로 산 전자 기기가 고장날 때까지 재방문을 하지 않을 수도 있지만, 디지털 제품, 식품, 패션 상품을 고루 사던 사람이라면 다른 제품을 사러 재방문할 확률이 높다.
또한, 개인 맞춤형 마케팅 전략과 추천 서비스를 계획하는 데에도 큰 도움이 될 수 있다.

- **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 upbeat-aura-425501-u8.modulabs_project.user_data AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID
)
SELECT ur.*, up.* EXCEPT (CustomerID)
FROM upbeat-aura-425501-u8.modulabs_project.user_rfm AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```



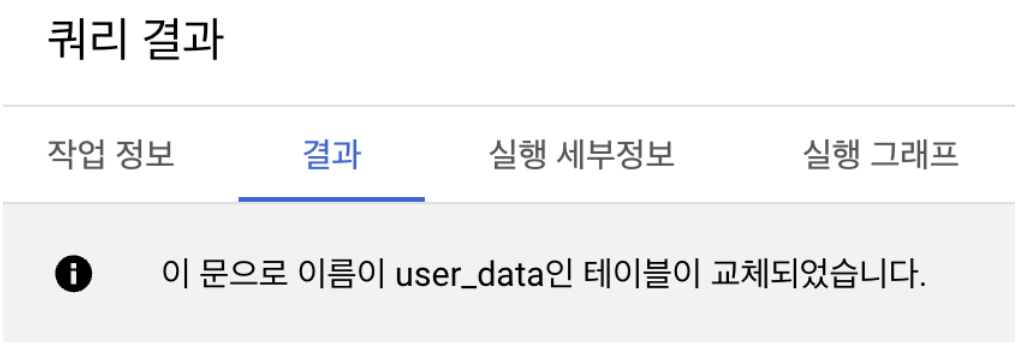
2. 평균 구매 주기

고객들의 쇼핑 패턴 이해를 위해 고객별 재방문 주기를 살펴본다. 고객들의 구매와 구매 사이의 기간이 평균적으로 몇 일인지를 보여주는 평균 일수를 계산하면, 고객이 다음 구매를 언제할지 예측하는 데에도 큰 도움이 된다.

- **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 upbeat-aura-425501-u8.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
      upbeat-aura-425501-u8.modulabs_project.data
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM upbeat-aura-425501-u8.modulabs_project.user_data AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```



3. 구매 취소 경향성

고객의 취소 패턴을 더 깊이 있게 파고 들어 고객을 세그먼테이션할 때 사용한다.

취소 빈도: 고객 별로 취소한 거래의 총 횟수
취소 빈도를 이해하면 거래를 취소할 가능성이 높은 고객을 식별할 수 있다. 취소 빈도는 불만족의 정도이거나 다른 문제에 대한 지표일 수 있다. 따라서 취소 빈도를 이해함으로써 거래 취소 횟수를 줄이고 고객 만족도를 높이기 위한 전략을 세울 수 있다.

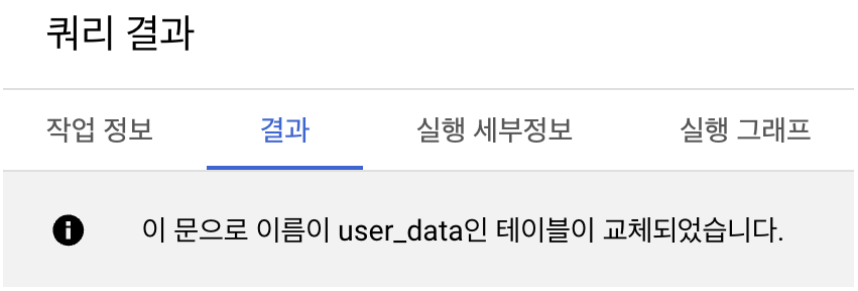
취소 비율: 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율
취소 비율은 특정 고객이 원래 취소를 잘 하는 고객인지와 같은 고객의 특징을 잡아내기 위한 지표다. 이런 특성을 식별함으로써 고객의 쇼핑 경험을 개선하고 취소 비율을 감소시키기 위해 어떤 고객 대상군을 공략해야 할지에 대한 실마리를 얻을 수 있다.

- **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 upbeat-aura-425501-u8.modulabs_project.user_data AS

WITH TransactionInfo AS (
  SELECT
      DISTINCT CustomerID
    , COUNT(InvoiceNo) OVER (PARTITION BY CustomerID) AS total_transactions
    , SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) OVER (PARTITION BY CustomerID) AS cancel_fre
  FROM upbeat-aura-425501-u8.modulabs_project.data
  GROUP BY CustomerID, InvoiceNo
)

SELECT
    u.*, t.* EXCEPT(CustomerID)
  , ROUND(t.cancel_frequency / total_transactions * 100, 2) AS cancel_rate
FROM upbeat-aura-425501-u8.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 upbeat-aura-425501-u8.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	17752	1	192	359	81.0	81.0	1	0.0	1	0	0.0
2	14821	1	20	366	49.0	49.0	2	0.0	1	0	0.0
3	13682	1	10	187	60.0	60.0	3	0.0	1	0	0.0
4	13216	1	297	267	334.0	334.0	5	0.0	1	0	0.0
5	14967	1	140	49	464.0	464.0	6	0.0	1	0	0.0
6	13686	1	70	182	72.0	72.0	8	0.0	1	0	0.0
7	12588	1	52	39	175.0	175.0	9	0.0	1	0	0.0
8	17538	1	134	25	98.0	98.0	9	0.0	1	0	0.0
9	12618	1	95	21	137.0	137.0	10	0.0	1	0	0.0
10	13348	1	136	64	234.0	234.0	14	0.0	1	0	0.0
11	17176	1	170	201	306.0	306.0	15	0.0	1	0	0.0