PDA

고객을 세그먼테이션하자 [프로젝트]

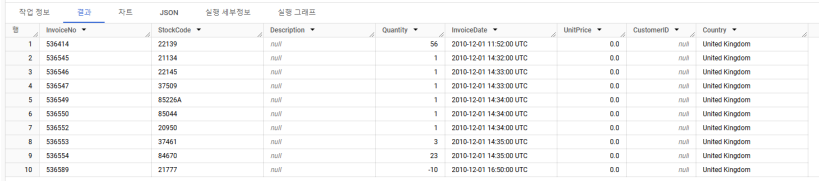
5-2. 데이터 불러오기

데이터 살펴보기

- **테이블에 있는 10개의 행만 출력하기**

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

[결과 이미지를 넣어주세요]



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**

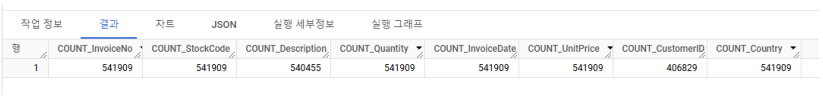
```
SELECT COUNT(*)
FROM modulabs_project.data
```

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 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

*** 데이터 셋 참조**

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

5-4. 데이터 전처리 방법(1): 결측치 제거

- 결측치 처리 방법
  - 결측치행 삭제, 다른 값들의 평균, 중앙값, 최빈값 등으로 대체
- 중복값 처리
  - 중복값을 삭제하는 것이 데이터의 일관성과 정확성을 유지하는데 필수지만
    특정 조건이나 규칙에 따라 중복된 데이터를 유지할 필요 있음
- 정규화 및 표준화
  - 데이터의 범위차이가 너무 크기 때문에 정규화 및 표준화를 통해 데이터 범위를 조정하거나, 데이터의 스케일을 변경
  - 두 데이터의 범위를 0과 1 사이의 숫자로 변경하거나, 로그 변환이나 제곱근 변환 등을 통해 데이터의 분포를 조정
- 명목형 데이터의 인코딩
  - 성별이나 구매 지역 등의 범주형 데이터를 변환. 지역이라면 수도권 1, 그외 지방을 0 등
- 이상치 분석 및 처리
  - 통계적인 방법이나 시각적인 도구를 활용해서 극단적으로 작은 이상치(outlier)들을 찾고, 필요에 따라 제거를 하거나 수정

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
```

작업 정보 | 결과 | 차트 | JSON | 실행 세

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

결측치 처리 전략

- CustomerID의 missing_percentage는 24.93%
  - 1/4이 결측치이기 때문에 제거하는 것은 상당한 편향과 노이즈가 될 수 있으나 RFM을 구하기 위해서 식별자 데이터는 정확해야하기 때문에, 제거하는 것이 가장 합리적인 접근
- Description의 경우, 결측치가 비교적 적으나 데이터의 일관성에서의 문제가 있음
  - 상세 설명이 일관적이지 않은 일관성의 결여의 문제. 누락된 비율이 매우 낮기 때문에 데이터 일관성 문제가 후속 분석 과정에 영향을 주지 않게 하기 위해 누락된 설명이 있는 행을 제거하는 것이 현명

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

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

작업 정보 | 결과 | 차트 | JSON | 실행 세부정보

행	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

결측치 처리

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

```
# 결측치 제거 전 조회
-- SELECT COUNT(*)
-- FROM modulabs_project.data
-- WHERE Description is null
-- OR CustomerID is null

DELETE FROM modulabs_project.data
WHERE Description is null
OR CustomerID is null
```

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 data의 행 135,080개가 삭제되었습니다.

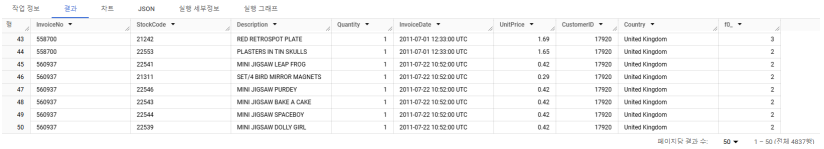
5-5. 데이터 전처리(2): 중복값 처리

중복값 확인

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

```
SELECT InvoiceNo,
    StockCode,
    Description,
    Quantity,
    InvoiceDate,
    UnitPrice,
    CustomerID,
    Country,
    COUNT(*)
FROM modulabs_project.data
```

```
GROUP BY ALL
HAVING COUNT(*) > 1;
```



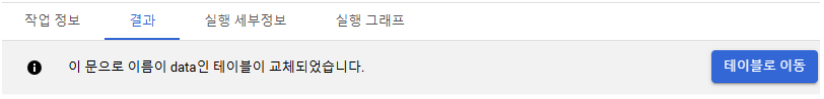
행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	f0_
43	558700	21242	RED RETROSPOT PLATE	1	2011-07-01 12:33:00 UTC	1.69	17920	United Kingdom	3
44	558700	22553	PLASTERS IN TIN SKULLS	1	2011-07-01 12:33:00 UTC	1.65	17920	United Kingdom	2
45	560937	22541	MINI JIGSAW LEAP FROG	1	2011-07-22 10:52:00 UTC	0.42	17920	United Kingdom	2
46	560937	21311	SET/4 BIRD MIRROR MAGNETS	1	2011-07-22 10:52:00 UTC	0.29	17920	United Kingdom	2
47	560937	22546	MINI JIGSAW PURDEY	1	2011-07-22 10:52:00 UTC	0.42	17920	United Kingdom	2
48	560937	22543	MINI JIGSAW BAKE A CAKE	1	2011-07-22 10:52:00 UTC	0.42	17920	United Kingdom	2
49	560937	22544	MINI JIGSAW SPACEBOY	1	2011-07-22 10:52:00 UTC	0.42	17920	United Kingdom	2
50	560937	22539	MINI JIGSAW DOLLY GIRL	1	2011-07-22 10:52:00 UTC	0.42	17920	United Kingdom	2

- COUNT(*)를 쓴 이유는 NULL값 중복까지 보기 위해서
- COUNT(컬럼이름)은 NULL값을 제외하고 카운트

중복값 처리

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

```
CREATE OR REPLACE TABLE `infrearn-bigquery-444412.modulabs_project.data` AS
SELECT DISTINCT InvoiceNo,
    StockCode,
    Description,
    Quantity,
    InvoiceDate,
    UnitPrice,
    CustomerID,
    Country
FROM infrearn-bigquery-444412.modulabs_project.data
```

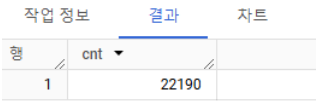


5-6. 데이터 전처리(3): 오류값 처리

`InvoiceNo` 살펴보기

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

```
SELECT COUNT(DISTINCT InvoiceNo) as cnt
FROM modulabs_project.data
```



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

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

작업 정보 결과 차트

행	InvoiceNo
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

- **`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
```

작업 정보 결과 차트

행	InvoiceNo
1	C575531
2	C558080
3	C558080
4	C554983
5	C554983
6	C539709
7	C539709

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

```
# Quantity 확인
SELECT MAX(Quantity), MIN(Quantity)
FROM modulabs_project.data
WHERE InvoiceNo LIKE 'C%'

SELECT ROUND(SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) / COUNT(InvoiceNo) * 100, 1) AS Ca
FROM modulabs_project.data
```

아래는 소수점 3자리까지

작업 정보 결과 차트

행	CanceldRate
1	2.2

- **취소를 많이 한 상품의 경향성 확인**
  - 취소를 많이 한 제품의 가격대가 높았는지
  - 거래 지역이 특정지역에 몰려있는지도 확인
  - RFM 분석이기 때문에 고객의 취소 패턴을 이해는 것도 굉장히 중요. 취소된 거래에 공통점을 찾아보는 것이 중요하다.

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

행	StockCode	SellCount
1	85123A	2065
2	22423	1894
3	85099B	1659
4	47566	1409
5	84879	1405
6	20725	1346
7	22720	1224

- **`StockCode`의 문자열 내 숫자의 길이를 구해보기**
  - REGEXP_REPLACE 라는 함수는 텍스트를 처리하는 정규 표현식(Regular Expression) 중 하나입니다. 'REGEXP'는 정규 표현식를 의미하며, 'REPLACE'는 텍스트를 대체한다는 의미로, REGEXP_REPLACE 함수는 특정 조건에 부합한 텍스트를 다른 텍스트로 대체
  - LENGTH 함수 내부에 있는 코드는 `REGEXP_REPLACE(StockCode, r'[0-9]', '')`라고 작성되어 있습니다. 이 코드는 StockCode 컬럼에 있던 값 중에서 0부터 9 사이의 숫자([0-9])를 비어 있는 값('')으로 대체하는 코드입니다. 이 코드를 통해 숫자를 제외한 문자만 남게 됩니다. 이후에 LENGTH 함수로 감싸주어서, 각 StockCode에 문자가 몇자리 수인지를 세어주기
  - 최종적으로 `LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', ''))` 연산을 통하여 StockCode 안에 있는 숫자의 수를 세어준 후, 이를 `number_count`라는 이름의 컬럼으로 저장

```
WITH UniqueStockCodes AS (
  SELECT DISTINCT StockCode
  FROM project_name.modulabs_project.data
)

SELECT
  LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count,
  COUNT(*) AS stock_cnt
FROM UniqueStockCodes
GROUP BY number_count
ORDER BY stock_cnt DESC;
```

작업 정보 | 결과 | 차트 | JSON

행	number_count	stock_cnt
1	5	3676
2	0	7
3	1	1

- `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 in (0,1);
```

작업 정보 | 결과 | 차트 | JSON | 실행

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**
  - **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 number_count in (0,1) THEN 1 ELSE 0 END) / COUNT(*) *100, 2 ) AS StockPar
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM modulabs_project.data
)
```

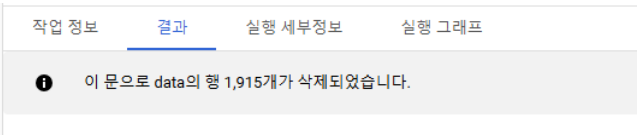
작업 정보 | 결과 | 차.

행	StockPartRate
1	0.48

- **제품과 관련되지 않은 거래 기록을 제거하기**

```
DELETE FROM 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 modulabs_project.data
  )
```

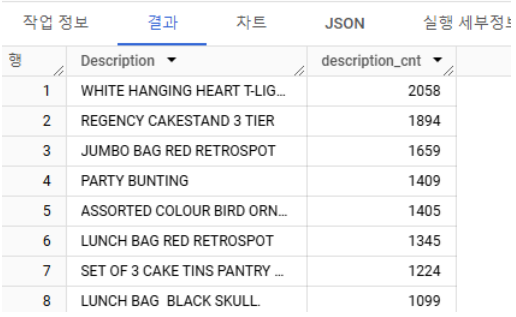
```
  WHERE number_count in (0,1)
);
```



`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 2 DESC
LIMIT 30
```

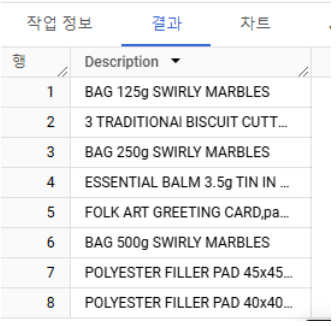


행	Description	description_cnt
1	WHITE HANGING HEART T-LIG...	2058
2	REGENCY CAKESTAND 3 TIER	1894
3	JUMBO BAG RED RETROSPOT	1659
4	PARTY BUNTING	1409
5	ASSORTED COLOUR BIRD ORN...	1405
6	LUNCH BAG RED RETROSPOT	1345
7	SET OF 3 CAKE TINS PANTRY ...	1224
8	LUNCH BAG BLACK SKULL.	1099

  - 출력결과를 보면 가정용품, 주방용품, 도시락, 장식품과 관련된 것
  - 모두 대문자로 되어있다. 혹시 혼용으로 된 곳도 있나 확인해보는 것도 필요

- **대소문자가 혼합된 Description이 있는지 확인하기**

```
SELECT DISTINCT Description
FROM project_name.modulabs_project.data
WHERE REGEXP_CONTAINS(Description, r'[a-z]');
```



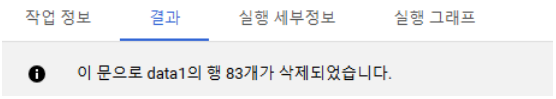
행	Description
1	BAG 125g SWIRLY MARBLES
2	3 TRADITIONAI BISCUIT CUTT...
3	BAG 250g SWIRLY MARBLES
4	ESSENTIAL BALM 3.5g TIN IN ...
5	FOLK ART GREETING CARD,pa...
6	BAG 500g SWIRLY MARBLES
7	POLYESTER FILLER PAD 45x45...
8	POLYESTER FILLER PAD 40x40...

  - REGEXP_CONTAINS 함수는 특정 패턴이 문자열에 포함되어 있는지 여부를 확인하는 데에 유용한 함수입니다. 특정 패턴이 문자열에 포함되어 있으면 True를, 포함되어 있지 않으면 False를 반환
  - `REGEXP_CONTAINS(Description, r'[a-z]')` 코드는 Description 컬럼에 있는 문자열에서 소문자 알파벳(`[a-z]`)이 포함되어 있는지를 확인하는 조건문입니다. 만약 `r'[a-z]'` 대신 `r'[A-Z]'` 를 사용했다면 대문자 알파벳이 포함되어 있는지를 확인하는 조건문

- **서비스 관련 정보를 포함하는 행들을 제거하기**

```
DELETE
FROM modulabs_project.data1
```

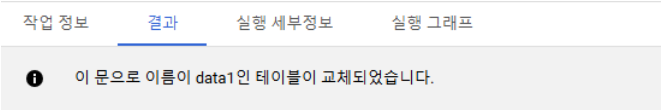
```
WHERE Description in ('Next Day Carriage', 'High Resolution Image')
```



- 대소문자 있는 걸 잘못 delete해서 data1으로 다시 임포트 하여 진행합니다.

- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

```
CREATE OR REPLACE TABLE modulabs_project.data1 AS
SELECT
  * EXCEPT (Description),
  REGEXP_REPLACE(Description, r'[a-z]', r'[A-Z]') AS Description
FROM modulabs_project.data1;

# 값이 대문자, 소문자 혼용있는지 확인
-- 표시할 결과 없음

-- SELECT DISTINCT Description
-- FROM modulabs_project.data1
-- WHERE REGEXP_CONTAINS(Description, r'[a-z]');
```



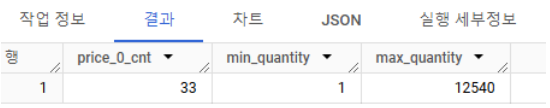
`UnitPrice` 살펴보기

- `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.data1
```

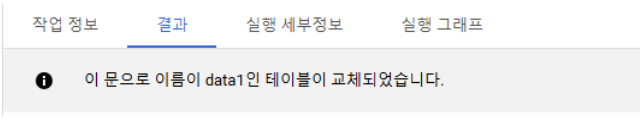
- **단가가 0원인 거래의 개수, 구매 수량(**`Quantity`**)의 최솟값, 최댓값, 평균 구하기**

```
SELECT COUNT(UnitPrice) AS price_0_cnt ,
     MIN(Quantity) AS min_quantity,
     MAX(Quantity) AS max_quantity
FROM modulabs_project.data1
WHERE UnitPrice = 0
```



- `UnitPrice = 0` 를 제거하고 일관된 데이터셋을 유지하기

```
CREATE OR REPLACE TABLE modulabs_project.data1 AS
SELECT *
FROM modulabs_project.data1
WHERE UnitPrice <> 0
```



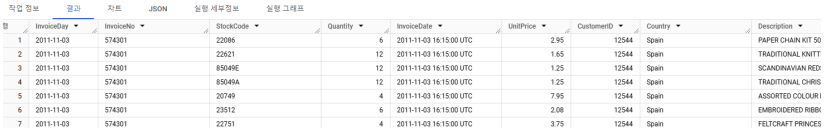
  - **=0 값으로 잘못 delete해서 data2으로 다시 임포트 하여 진행합니다.**

5-7. RFM 스코어

Recency

- `InvoiceDate` 컬럼을 연월일 자료형으로 변경하기

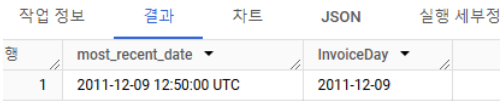
```
SELECT DATE(InvoiceDate) AS InvoiceDay, *
FROM modulabs_project.data2;
```



행	InvoiceDay	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	2011-11-03	574301	22086	6	2011-11-03 16:15:00 UTC	2.95	12544	Spain	PAPER CHAIN KIT 50
2	2011-11-03	574301	22621	12	2011-11-03 16:15:00 UTC	1.65	12544	Spain	TRADITIONAL KNITT
3	2011-11-03	574301	85049E	12	2011-11-03 16:15:00 UTC	1.25	12544	Spain	SCANDINAVIAN RED
4	2011-11-03	574301	85049A	12	2011-11-03 16:15:00 UTC	1.25	12544	Spain	TRADITIONAL CHRIS
5	2011-11-03	574301	20749	4	2011-11-03 16:15:00 UTC	7.95	12544	Spain	ASSORTED COLOUR I
6	2011-11-03	574301	23512	6	2011-11-03 16:15:00 UTC	2.08	12544	Spain	EMBROIDERED RIBB
7	2011-11-03	574301	22751	4	2011-11-03 16:15:00 UTC	3.75	12544	Spain	FELTCRAFT PRINCES

- **가장 최근 구매 일자를 MAX() 함수로 찾아보기**

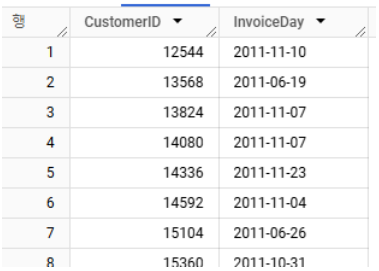
```
SELECT
    MAX(InvoiceDate) AS most_recent_date,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM modulabs_project.data2
```



행	most_recent_date	InvoiceDay
1	2011-12-09 12:50:00 UTC	2011-12-09

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

```
SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM modulabs_project.data2
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

- **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
);
```

작업 정보 | 결과 | 차트 | JSON

행	CustomerID	recency
1	12353	204
2	12480	28
3	12690	205
4	12953	9
5	13182	45
6	13513	126
7	13539	85
8	14178	8

- **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.data1
  GROUP BY CustomerID
  ORDER BY CustomerID
);
```

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 이름이 user_r인 새 테이블이 생성되었습니다.

Frequency

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

```
SELECT
  CustomerID,
  COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM modulabs_project.data2
GROUP BY CustomerID;
```

작업 정보 | 결과 | 차트 | JSON

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

- **각 고객 별로 구매한 아이템의 총 수량 더하기**

```
SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
FROM modulabs_project.data2
GROUP BY CustomerID;
```

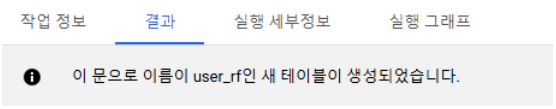
작업 정보 | 결과 | 차트 | JSON

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

- **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.data2
  GROUP BY CustomerID
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
  FROM modulabs_project.data2
  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
```

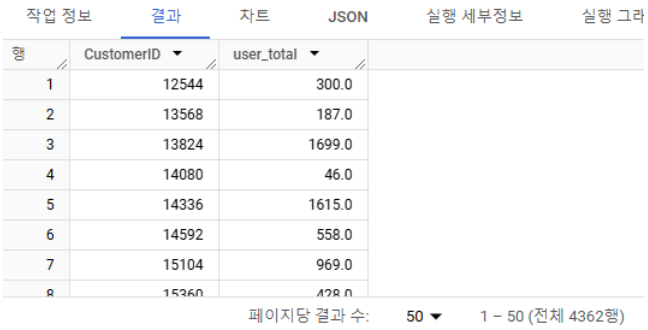
```
  ON pc.CustomerID = ic.CustomerID
JOIN modulabs_project.user_r AS ur
  ON pc.CustomerID = ur.CustomerID;
```



Monetary

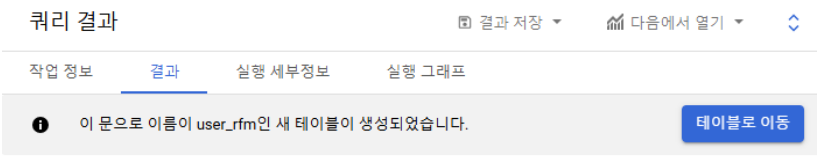
- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

```
SELECT
  CustomerID,
  ROUND(SUM(Quantity * UnitPrice),0) as user_total
FROM modulabs_project.data2
GROUP BY CustomerID
```



- **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_name.modulabs_project.user_rfm AS
SELECT
  rf.CustomerID AS CustomerID,
  rf.purchase_cnt,
  rf.item_cnt,
  rf.recency,
  ut.user_total,
  # [[YOUR QUERY]] AS user_average
FROM project_name.modulabs_project.user_rf rf
LEFT JOIN (
  -- 고객 별 총 지출액
  SELECT
    # [[YOUR QUERY]]
) ut
ON rf.CustomerID = ut.CustomerID
```



RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

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

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

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	15083	1	38	256	88.0	88.0
4	18010	1	60	256	175.0	175.0
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

5-8. 추가 Feature 추출

- RFM 분석의 허점은 1개를 10개 구매한 고객과 10개를 1번 구매한 고객을 같은 그룹으로 보는 것에 있다. 분명 다른 구매 패턴이 있을 것.
- 커머스에서 구매한 제품의 폭이 넓을 수록, 장기적으로 봤을 때 온라인 커머스에서 구매할 가능성이 높을 것
- 유저 구매 패턴 속에서 어떤 추가적인 특징들을 추가할 예정

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

쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

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_inte
  FROM (
    -- (1) 구매와 구매 사이에 소요된 일수
    SELECT
      CustomerID,
      DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID ORDER BY InvoiceDate), DAY)
    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;
```

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

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval	average_interval_1
1	14432	6	2013	9	2248.0	375.0	256	0.2	0.2
2	12428	11	3477	25	6366.0	579.0	256	0.87	0.87
3	13268	14	3525	17	3106.0	222.0	256	0.56	0.56
4	16144	1	16	246	175.0	175.0	1	0.0	0.0
5	13366	1	144	50	56.0	56.0	1	0.0	0.0
6	18233	1	4	325	440.0	440.0	1	0.0	0.0
7	18113	1	72	368	76.0	76.0	1	0.0	0.0
8	17752	1	192	359	81.0	81.0	1	0.0	0.0
9	17956	1	1	249	13.0	13.0	1	0.0	0.0

3. 구매 취소 경향성

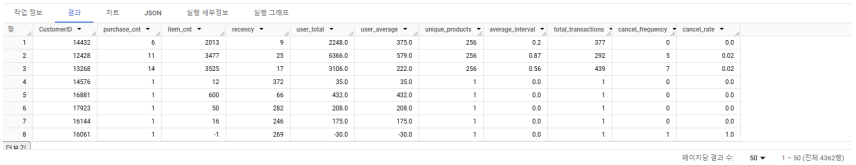
- **고객의 취소 패턴 파악하기**
  1) 취소 빈도(cancel_frequency) : 고객 별로 취소한 거래의 총 횟수
  2) 취소 비율(cancel_rate) : 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율
  - **취소 빈도와 취소 비율을 계산하고 그 결과를 `user_data` 에 통합하기**
    (취소 비율은 소수점 두번째 자리)

```
CREATE OR REPLACE TABLE modulabs_project.user_data AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(*) AS total_transactions,
    SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) AS cancel_frequency
  FROM modulabs_project.data2
  GROUP BY CustomerID
)

SELECT u.*, t.* EXCEPT(CustomerID), ROUND(cancel_frequency / t.total_transactions, 2) AS cancel_rate
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
```



APPENDIX. 회고

정말 하나하나 한땀한땀 이해하며 내려갔다. 매커니즘은 이해했으니 이제 내 것으로 만들어야겠다.

- 수작업 RFM , 백분위 5개로 나눠 1~5 값을 할당. recency(R), RFM_Score, user_average(M) 사용

```
WITH pre_RFM AS (
SELECT *
  , recency,
  NTILE(5) OVER(ORDER BY recency DESC) AS Recency_Score,
  user_average,
  NTILE(5) OVER(ORDER BY user_average ASC) AS Monetary_Score,
  purchase_cnt,
  NTILE(5) OVER(ORDER BY purchase_cnt ASC) AS Frequecy_Score
FROM modulabs_project.user_data
), RFM_Score_caculate AS (
SELECT *, Recency_Score + Monetary_Score + Frequecy_Score AS RFM_Score
FROM pre_RFM
)
SELECT RFM_Score, COUNT(*) AS cnt
FROM RFM_Score_caculate
GROUP BY RFM_Score
ORDER BY 1 ASC
```

RFM 값이 8이 그룹이 가장 많음. 평균..

RFM 값이 하위인 그룹은 이탈 가능성 높은 고객.

RFM 값이 상위인 그룹은 충성고객임으로 해당 고객의 행동 패턴 분석 필요.

작업 정보 | 결과 | 차트 | JSON

행	RFM_Score	cnt
1	3	109
2	4	194
3	5	287
4	6	374
5	7	470
6	8	512
7	9	501
8	10	472
9	11	457
10	12	387
11	13	303
12	14	198
13	15	98

상위, 하위 그룹의 취소율, 구매다양성, 구매주기 등을 분석해보면 더 재밌는 결과가 나올 수 있을 것 같다.

그리고 다양한 Feature들을 더 많이 알 수 있다면 더 도움이 될 것 같다.