컬럼 별 누락된 값의 비율 계산

그럼, 각 컬럼 별 누락된 값의 비율을 계산해보겠습니다.
각 컬럼에 대해서 누락 값을 계산한 후, 계산된 누락 값을 UNION ALL을 통해 합쳐주세요.

```sql
SELECT
  'InvoiceNo'      AS column_name,
  ROUND(
    SUM(CASE WHEN InvoiceNo    IS NULL THEN 1 ELSE 0 END)
    / COUNT(*) * 100
    , 2
  ) AS missing_percentage
FROM praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.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 praxis-beacon-464902-t2.modulabs_project.data;
```

← 쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세부정보

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

결측치 처리 전략

```
SELECT
  DISTINCT Description
FROM
  praxis-beacon-464902-t2.modulabs_project.data
WHERE
  StockCode = '85123A';
```

← 쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행

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

결측치 처리

```sql
DELETE FROM praxis-beacon-464902-t2.modulabs_project.data2
  WHERE

    Description IS NULL OR

    CustomerID IS NULL;
```

쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 data2의 행 135,080개가 삭제되었습니다.

11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

```sql
 SELECT
  COUNT(*) AS duplicate_group_count
FROM (
  SELECT
    COUNT(*) AS cnt
  FROM
    praxis-beacon-464902-t2.modulabs_project.data2
  GROUP BY
    InvoiceNo,
    StockCode,
    Description,
    Quantity,
    InvoiceDate,
    UnitPrice,
    CustomerID,
    Country
  HAVING
    cnt > 1
) t;
```

쿼리 결과

작업 정보 | 결과 | 차트

행	duplicate_group_...
1	4837

중복값 처리

```
CREATE OR REPLACE TABLE praxis-beacon-464902-t2.modulabs_project.data2 AS
SELECT
  DISTINCT *
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

← 쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 이름이 data2인 테이블이 교체되었습니다.

```
  SELECT
  COUNT(*) AS remaining_rows
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

← 쿼리 결과

작업 정보 | 결과 | 처

행	remaining_rows
1	401604

11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

고유(unique)한 InvoiceNo의 개수를 출력해 보세요.

```
SELECT
  COUNT(DISTINCT InvoiceNo) AS unique_invoiceno_count
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

Q. 유니크한 InvoiceNo**는 몇 개인가요?**

← 쿼리 결과

작업 정보 결과

행	unique_invoiceno...
1	22190

이번에는 고유한 InvoiceNo를 100개를 출력해 보세요. 아래와 같은 결과가 나와야 합니다.

```sql
WITH unique_invoices AS (
  SELECT
    DISTINCT InvoiceNo
  FROM
    praxis-beacon-464902-t2.modulabs_project.data2
)
SELECT
  InvoiceNo
FROM
  unique_invoices
LIMIT
  100;
```

쿼리 결과

작업 정보 결과

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
14	577609
15	543037
16	544156
17	545323

출력된 결과를 봤을 때, 특이한 점이 있는 값들이 있나요?

데이터를 자세히 살펴보면 몇 개의 InvoiceNo는 'C'로 시작한다는 것을 알 수 있습니다. 데이터 설명에도 나와 있던 것처럼, InvoiceNo가 'C'로 시작한다면 취소한 거래입니다.

고객 행동과 제품 선호도에 대한 이해를 높이기 위해서는 취소된 거래들도 고려해야 합니다.

먼저, InvoiceNo가 'C'로 시작하는 행을 필터링하여 취소된 거래들만 살펴봅시다. 이후에는 이 행들을 분석하여 공통적인 특성이나 패턴이 있는지 파악할 것입니다.

InvoiceNo가 **'C'로 시작하는 행을 필터링**할 수 있는 쿼리문을 작성해 주세요.

100행까지만 출력해 주세요. 아래와 같이 결과가 나와야 합니다.

```sql
SELECT
  InvoiceNo
FROM
  unique_invoices
LIMIT
  100;

SELECT *
FROM
  praxis-beacon-464902-t2.modulabs_project.data2
WHERE
  STARTS_WITH(InvoiceNo, 'C')
LIMIT
  100;
```

쿼리 결과

결과 저장 다음에서 열기

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	C541433	23166	MEDIUM CERAMIC TOP STORAGE JAR	-74215	2011-01-18 10:17:00 UTC	1.04	12346	United Kingdom
2	C545329	M	Manual	-1	2011-03-01 15:47:00 UTC	280.05	12352	Norway
3	C545329	M	Manual	-1	2011-03-01 15:47:00 UTC	183.75	12352	Norway
4	C545330	M	Manual	-1	2011-03-01 15:49:00 UTC	376.5	12352	Norway
5	C547388	37448	CERAMIC CAKE DESIGN SPOTTED MUG	-12	2011-03-22 16:07:00 UTC	1.49	12352	Norway
6	C547388	21914	BLUE HARMONICA IN BOX	-12	2011-03-22 16:07:00 UTC	1.25	12352	Norway
7	C547388	22413	METAL SIGN TAKE IT OR LEAVE IT	-6	2011-03-22 16:07:00 UTC	2.95	12352	Norway
8	C547388	84050	PINK HEART SHAPE EGG FRYING PAN	-12	2011-03-22 16:07:00 UTC	1.65	12352	Norway
9	C547388	22701	PINK DOG BOWL	-6	2011-03-22 16:07:00 UTC	2.95	12352	Norway
10	C547388	22784	LANTERN CREAM GAZEBO	-3	2011-03-22 16:07:00 UTC	4.95	12352	Norway
11	C547388	22645	CERAMIC HEART FAIRY CAKE MONEY...	-12	2011-03-22 16:07:00 UTC	1.45	12352	Norway
12	C549955	22666	RECIPE BOX PANTRY YELLOW DESIGN	-2	2011-04-13 13:38:00 UTC	2.95	12359	Cyprus
13	C549955	22839	3 TIER CAKE TIN GREEN AND CREAM	-2	2011-04-13 13:38:00 UTC	14.95	12359	Cyprus
14	C580165	22720	SET OF 3 CAKE TINS PANTRY DESIGN	-1	2011-12-02 11:21:00 UTC	4.95	12359	Cyprus
15	C580165	22797	CHEST OF DRAWERS GINGHAM HEART	-2	2011-12-02 11:21:00 UTC	16.95	12359	Cyprus
16	C580165	22826	LOVE SEAT ANTIQUE WHITE METAL	-1	2011-12-02 11:21:00 UTC	42.5	12359	Cyprus
17	C580165	23245	SET OF 3 REGENCY CAKE TINS	-2	2011-12-02 11:21:00 UTC	4.95	12359	Cyprus

데이터에 특이한 경향성이 보이나요?

Q. 취소된 거래 건들은 어떠한 특징이나 경향성이 있나요?

- Quantity가 음수입니다.

구매 건 상태가 Canceled 인 데이터의 비율(%)은 어떻게 되나요? 이를 계산할 수 있는 쿼리문을 작성하고, 취소 비율을 **소수점 첫번째 자리**까지 구해 주세요

```sql
SELECT
  ROUND(
    SUM(CASE
          WHEN STARTS_WITH(InvoiceNo, 'C') THEN 1
          ELSE 0
        END)
    / COUNT(*) * 100
  , 1) AS canceled_percentage
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

← 쿼리 결과

작업 정보 결과

행	canceled_percent...
1	2.2

데이터를 자세히 보면 InvoiceNo 컬럼에서 'C'가 붙은 거래 취소 건들은 Quantity가 음수인 것을 알 수 있습니다.

취소를 많이 한 제품들의 가격대가 높았는지, 또는 거래 지역이 특정 지역에 몰려 있는지도 살펴보았지만, 그런 경향성은 크게 보이지 않습니다. (Country는 취소 여부와 상관없이 United Kingdom에 밀집되어 있습니다.)

그럼 InvoiceNo에 따른 추가 처리를 해야 할까요?

실제 분석 과정에서는 분석가의 주관적 판단에 의해 전처리 여부와 그 방식을 결정 짓습니다. 위의 경우 프로젝트의 초기 목표가 **고객들의 구매 최신성, 구매 빈도, 구매 금액에 따라 세그멘테이션**하는 것이었기 때문에, 고객의 취소 패턴을 이해하는 것도 중요할 것입니다. 가령 취소된 거래에 공통점이 있는지 살펴볼 수도 있을 것이고, 추천될 가능성이 높은 제품을 찾아서 문제를 진단할 수도 있습니다.

따라서 취소된 거래 데이터는 유지하되, 명확하게 표시하여 추가 분석을 용이하게 만들어주는 것도 전략 중 하나입니다.

StockCode 살펴보기

이번엔 StockCode를 더 깊이 있게 살펴볼 차례입니다. 우선 고유한 StockCode의 개수를 출력해보겠습니다.

```sql
SELECT
  COUNT(DISTINCT StockCode) AS unique_stockcode_count
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

← 쿼리 결과

작업 정보 결과

행	unique_stockcod...
1	3684

이번에는 어떤 제품이 가장 많이 판매되었는지 보기 위하여 StockCode 별 **등장 빈도**를 출력해보겠습니다. 상위 10개의 제품들을 출력해 보세요.

```
SELECT
  StockCode,
  COUNT(*) AS sell_cnt
FROM
  praxis-beacon-464902-t2.modulabs_project.data2
GROUP BY
  StockCode
ORDER BY
  sell_cnt DESC
LIMIT
  10;
```

← 쿼리 결과

작업 정보 결과 차트 JSON 실행 세부정보

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

Q. StockCode **별 등장 빈도를 출력했을 때, 일정한 경향성이 보이나요? 이상치는 있었나요?**

- 대부분의 StockCode들은 5-6 자리 숫자입니다. StockCode 중 'POST'는 이상치 같아 보입니다.

가장 판매가 많이 일어난 상위 10개의 제품 코드를 자세히 살펴보면, 고객들이 자주 구매하는 인기 제품군이나 카테고리에 대한 인사이트도 얻을 수가 있습니다.

제품 코드는 대부분 5~6자리의 숫자와 문자 조합으로 구성되어 있는 반면, 'POST'와 같은 몇 가지 이상한 코드도 있습니다. 이러한 이상 현상은 실제 제품보다는 서비스나 배송비 같은 형태를 코드로 남긴 것일 수도 있습니다. 현재 진행하고 있는 프로젝트는 고객들의 **제품 구매**에 초점이 맞춰져 있기 때문에 이런 값들은 제거하는 것이 좋을 것 같습니다.

'POST'와 같은 이상치들이 몇 개나 있는지 확인하기 위하여 StockCode의 문자열 내 숫자의 길이를 살펴보겠습니다. 예를 들어 '22423'와 '85123A'는 모두 다 숫자가 5개씩 포함되어 있는 문자열이지만 'POST'와 같은 비정상적인 항목들은 숫자가 0개 포함되어 있습니다.

이처럼 각 StockCode 내 숫자의 개수를 살펴보면서 값의 특성에 대한 통찰력을 얻을 수 있습니다.

StockCode의 **문자열 내 숫자의 길이**를 구해 봅시다.

```sql
WITH UniqueStockCodes AS (
  SELECT DISTINCT StockCode
  FROM praxis-beacon-464902-t2.modulabs_project.data2
)
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;
```

위의 코드를 더 자세히 살펴보도록 하겠습니다.

우선 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 대체하는 함수입니다.

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 몇 자리 수인지를 세어주었습니다.

최종적으로 LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '') 연산을 통하여 StockCode 안에 있는 숫자의 수를 세어준 후, 이를 number_count라는 이름의 컬럼으로 저장해 주었습니다.

쿼리문을 실행시키면 아래와 같은 결과가 나옵니다.

← 쿼리 결과

작업 정보 | 결과 | 차트 | JSON

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

출력 결과를 보면, 8개를 제외하곤 StockCode에 **5개의 숫자들이 포함**되어 있는 것을 알 수 있습니다. 숫자가 0개인 코드는 7개, 숫자가 1개인 코드는 1개입니다.

숫자가 0~1개인 값들에는 어떤 코드들이 들어가 있는지를 확인해 봅시다.

```
SELECT DISTINCT StockCode, number_count
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM praxis-beacon-464902-t2.modulabs_project.data2
)
WHERE  number_count BETWEEN 0 AND 1;
```

Q. 숫자가 0~1개인 값들에는 어떤 코드들이 들어가 있나요?

- POST, D, C2, M, BANK CHARGES, PADS, DOT, CRUK

← 쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세부정

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

해당 코드 값들을 가지고 있는 **데이터 수는 전체 데이터 수 대비 몇 퍼센트**일까요? **소수점 두번째 자리**까지 구해 주세요.

```sql
SELECT
  ROUND(
    COUNTIF(
      -- StockCode 내 숫자 개수 계산 후 0~1 사이인지 판별
      LENGTH(StockCode)
        - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', ''))
      BETWEEN 0 AND 1
    )
    / COUNT(*) * 100
  , 2) AS percent_zero_or_one_digits
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

Q. 해당 코드 값들을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트인가요?

- 0.48%

← 쿼리 결과

작업 정보 | 결과

행	percent_zero_or_...
1	0.48

분석에 따르면 전체 데이터 중 매우 작은 비율인 0.48%의 데이터가 일반적인 형식에서 벗어난 값을 가지고 있었습니다.

이 코드들은 'BANK CHARGES, POST' 등 제품과 관련되지 않은 거래 기록으로 보입니다. 진행하고 있는 프로젝트의 목표는 고객들의 '제품 구매'에 기반하여 세그먼테이션을 하는 것이므로, 이런 StockCode가 포함된 기록은 데이터셋에서 제외하도록 하겠습니다.

제품과 관련되지 않은 거래 기록을 제거하는 쿼리문을 작성해 주세요.

```sql
DELETE FROM praxis-beacon-464902-t2.modulabs_project.data2
WHERE StockCode IN (
  SELECT DISTINCT StockCode
  FROM  (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM praxis-beacon-464902-t2.modulabs_project.data2)
  where number_count between 0 and 1);
```

Description 살펴보기

먼저 데이터셋에서 고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력해 주세요.

```sql
SELECT
  Description,
  COUNT(*) AS description_count
FROM
  praxis-beacon-464902-t2.modulabs_project.data2
GROUP BY
  Description
ORDER BY
  description_count DESC
LIMIT
  30;
```

출력 결과를 보니 제품군들이 대부분 가정용품, 주방용품, 점심 도시락, 장식품과 관련된 것들임을 알 수 있습니다.

흥미롭게도 모든 Description이 대문자로 되어 있는데, 이는 데이터베이스에 제품 설명을 입력할 때 사용되는 표준화된 형식일 수도 있습니다. 그러나 혹시라도 대문자와 소문자가 혼합된 스타일로 입력된 설명이 있는지 확인해 보는 것이 현명할 것입니다.

대소문자가 혼합된 Description이 있는지 확인해 봅시다.

```sql
SELECT DISTINCT Description
FROM praxis-beacon-464902-t2.modulabs_project.data2
WHERE REGEXP_CONTAINS(Description, r'[a-z]');
```

위의 코드를 더 자세히 살펴보도록 하겠습니다.

WHERE 절의 REGEXP_CONTAINS 함수는 특정 패턴이 문자열에 포함되어 있는지 여부를 확인하는 데에 유용한 함수입니다. 특정 패턴이 문자열에 포함되어 있으면 True를, 포함되어 있지 않으면 False를 반환합니다. 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 조건문이 됩니다.

결과는 아래와 같이 나옵니다. 빅쿼리에서 실행 결과의 스크롤을 내리면 총 19개의 Description이 대소문자를 혼합하고 있다는 것을 알 수 있습니다.

'"3 TRADITIONAl BISCUIT CUTTERS SET'는 모두 대문자로 작성된 거 아냐?"라고 생각할 수 있지만 TRADITIONA'l'에서 'l'은 대문자 I(아이)가 아니라 소문자 l(엘)입니다.

← 쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세부정보 | 실

행	Description	description_count
1	WHITE HANGING HEART T-LIGHT HOLDER	2058
2	REGENCY CAKESTAND 3 TIER	1894
3	JUMBO BAG RED RETROSPOT	1659
4	PARTY BUNTING	1409
5	ASSORTED COLOUR BIRD ORNAMENT	1405
6	LUNCH BAG RED RETROSPOT	1345
7	SET OF 3 CAKE TINS PANTRY DESIGN	1224
8	LUNCH BAG BLACK SKULL.	1099
9	PACK OF 72 RETROSPOT CAKE CASES	1062
10	SPOTTY BUNTING	1026
11	PAPER CHAIN KIT 50'S CHRISTMAS	1013
12	LUNCH BAG SPACEBOY DESIGN	1006
13	LUNCH BAG CARS BLUE	1000
14	HEART OF WICKER SMALL	990
15	NATURAL SLATE HEART CHALKBOARD	989
16	JAM MAKING SET WITH JARS	966
17	LUNCH BAG PINK POLKADOT	961
18	LUNCH BAG SUKI DESIGN	932

출력 결과를 보면 사이즈(cm)나 무게(g) 등의 단위를 나타내는 설명이 포함되어 있고, 'Next Day Carriage'나 'High Resolution Image'와 같은 일부 항목들처럼 실제 제품에 대한 Description이 아닌 것도 있는 것을 알 수 있습니다. 이 데이터들은 실제 제품 정보와 관련이 없어 보입니다. 아마도 다른 유형의 정보나 서비스 세부사항을 나타내는 것일수도 있습니다. 이를 처리하기 위한 몇가지 전략들을 생각해볼 수 있습니다.

1. **'Next Day Carriage'와 'High Resolution Image'와 같은 서비스 관련 정보**를 포함하는 행들을 제거합니다.
2. 대소문자를 혼합해서 사용하는 경우, 대문자로 표준화하여 데이터셋 전체에서 일관성을 유지할 수 있습니다. 이는 대소문자에 의한 중복 항목의 가능성을 줄이는 데에도 도움이 될 것입니다.

이러한 전략을 선택함으로써, 데이터셋의 품질을 향상시킬 수 있고 프로젝트의 분석 단계에 더 적합한 데이터셋을 완성할 수 있습니다.

우선 **서비스 관련 정보를 포함하는 행들을 제거**하는 쿼리문을 작성해 보세요.

```sql
DELETE
FROM praxis-beacon-464902-t2.modulabs_project.data2
WHERE
  -- Description에 서비스 관련 키워드가 포함된 행 제거
  REGEXP_CONTAINS(
    LOWER(Description),
    r'service|delivery|postage|shipping|charges'
  );
```

이번에는 **대소문자를 혼합하고 있는 데이터를 대문자로 표준화**하는 쿼리문을 작성해 보세요.

```sql
CREATE OR REPLACE TABLE praxis-beacon-464902-t2.modulabs_project.data2 AS
SELECT
  * EXCEPT (Description),
  UPPER(Description) AS Description
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

SELECT 절에 EXCEPT (Description)이라는 구문은 처음 보셨을 겁니다. * EXCEPT (Description)는 테이블의 모든 컬럼들을 가져오되, Description은 제외하라는 의미입니다. 포함하려고 하는 컬럼명을 일일이 쓰는 것보다 제거하고자 하는 몇 개 컬럼을 적을 때 자주 활용합니다.

Quantity와 InvoiceDate에 대한 데이터를 분석했을 때에는 크게 문제가 되는 이상 데이터가 없어서 넘어가겠습니다. 데이터를 분석할 때 전처리하면 좋을 것 같은 데이터가 나타난다면, 분석가/과학자의 판단에 따라 처리 방법을 선택할 수 있습니다.

UnitPrice 살펴보기

이번에는 UnitPrice에서 이상치를 찾아봅시다. 최솟값, 최댓값, 평균 데이터를 확인해 봄으로써, 단위 가격의 요약 통계량을 살펴보겠습니다.

✳ 단위 가격(UnitPrice)이란?

상품 1개당 가격을 말합니다. 즉 "단위 당 가격"을 의미합니다.

UnitPrice의 **최솟값, 최댓값, 평균**을 구해 보세요.

```sql
SELECT
  MIN(UnitPrice)                          AS min_unitprice,
  MAX(UnitPrice)                          AS max_unitprice,
  ROUND(AVG(UnitPrice), 2)                AS avg_unitprice
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세부정보

행	min_unitprice	max_unitprice	avg_unitprice
1	0.0	649.5	2.91

Q. UnitPrice **데이터에 특이한 값이 있나요?**

- UnitPrice가 0인 데이터가 있습니다.

단위 가격의 요약 통계량을 보면 최소 단위 가격(min_price)이 0인 것을 알 수 있습니다. 이는 단가가 0원인 데이터가 존재한다는 것을 의미하고, 이는 이 제품이 무료 제품이거나 데이터 오류일 수도 있다는 의미입니다.

단가가 0원인 거래의 성격을 제대로 이해하기 위해서는 데이터를 더 자세히 살펴볼 필요가 있습니다. 단가가 0원인 제품을 상세하게 분석해보면서 특정한 패턴이 있는지 살펴보겠습니다.

단가가 0원인 거래의 개수, 구매 수량(Quantity**)의 최솟값, 최댓값, 평균**을 구해 보세요.

```
SELECT
  COUNT(*)                                AS zero_price_count,
  MIN(Quantity)                           AS min_quantity,
  MAX(Quantity)                           AS max_quantity,
  ROUND(AVG(Quantity), 2)                 AS avg_quantity
FROM
  praxis-beacon-464902-t2.modulabs_project.data2
WHERE
  UnitPrice = 0;
```

쿼리 결과

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

행	zero_price_count	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.52

UnitPrice가 0인 행의 수는 33개로 비교적 적음을 알 수 있습니다. 구매 수량(Quantity)은 최소 1개부터 최대 12,540개에 이르기까지 굉장히 큰 편차를 가집니다.

데이터의 수가 적은 걸 보니 무료 제품이라기보다 데이터 오류일 가능성이 더 높을 것 같습니다. 그래서 **이 데이터(**UnitPrice = 0**)를 제거**하고 일관된 데이터셋을 유지하도록 하겠습니다.

```sql
CREATE OR REPLACE TABLE praxis-beacon-464902-t2.modulabs_project.data2 AS
SELECT *
FROM
  praxis-beacon-464902-t2.modulabs_project.data2
WHERE
  UnitPrice != 0;
```

11-7. RFM 스코어

이번 시간에는 본격적으로 RFM 분석을 해보도록 하겠습니다. 다시 한번 RFM 분석에 대해 배운 내용을 복습해 봅시다.

RFM분석은 구매 최신성(Recency), 구매 빈도(Frequency), 구매 가치(Monetary)에 따라 고객들을 여러 그룹으로 나누는 세그먼테이션(segmentation) 방법이라고 하였습니다.

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

Recency

Recency 단계에서는 고객이 얼마나 최근에 구매를 했는지에 중점을 둡니다. 그러므로 '마지막 구매일로부터 현재까지 경과한 일수'를 계산해야 합니다. 낮은 값일수록 고객이 최근에 구매를 했음을 의미하며, 제품이나 서비스에 더 관심을 보인다고 예측할 수 있습니다. Recency를 통해 오랜 시간 동안 구매를 하지 않았던 고객을 발견하고, 다시 제품과 서비스로 불러들이기 위한 마케팅 전략을 맞춤화해볼 수도 있습니다.

우선 InvoiceDate를 '2010-12-01 08:26:00'와 같은 'YYYY-MM-DD HH:MM:SS' 형태에서 'YYYY-MM-DD' 형태로 날짜에 해당하는 부분만 남겨놓고 싶습니다. 이를 위해 DATE 함수를 활용하여 InvoiceDate 컬럼을 **연월일 자료형으로 변경**해 주세요.

```sql
SELECT
  DATE(InvoiceDate) AS InvoiceDay,
  *
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

실제 회사에서 다루는 데이터라면 오늘 날짜를 기준으로 최종 구매일이 몇 일 지났는지 계산하겠지만, 여러분이 다루고 있는 데이터는 2010년~2011년 사이의 데이터이므로, 최종 구매일로부터 꽤 오랜 시간이 지난 데이터라는 특성이 있습니다. 그래서 이번 프로젝트에서는 모든 고객들을 통틀어 가장 최근 구매 일자를 기준으로 Recency를 구하려고 합니다.

우선 가장 **최근 구매 일자**를 MAX() 함수로 찾아보겠습니다.

```sql
SELECT
  MAX(InvoiceDate) OVER()        AS most_recent_date,
  DATE(InvoiceDate)              AS InvoiceDay,
  *
FROM
  praxis-beacon-464902-t2.modulabs_project.data2;
```

쿼리 결과

결과 저장 ▾ 다음에서 열기 ▾

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

	most_recent_date	InvoiceDay	InvoiceNo	StockCode	Quantity	InvoiceDate
1	2011-12-09 12:50:00 UTC	2011-10-13	571034	23094	4	2011-10-13 12:4
2	2011-12-09 12:50:00 UTC	2011-12-02	C580165	22826	-1	2011-12-02 11:2
3	2011-12-09 12:50:00 UTC	2011-08-19	563852	84816	1	2011-08-19 15:1
4	2011-12-09 12:50:00 UTC	2011-06-17	557147	47518F	24	2011-06-17 10:5
5	2011-12-09 12:50:00 UTC	2011-03-03	545475	21915	240	2011-03-03 10:5
6	2011-12-09 12:50:00 UTC	2011-08-18	563614	23203	100	2011-08-18 08:5
7	2011-12-09 12:50:00 UTC	2011-08-18	563614	23343	200	2011-08-18 08:5
8	2011-12-09 12:50:00 UTC	2011-10-05	569650	22728	50	2011-10-05 12:4
9	2011-12-09 12:50:00 UTC	2011-06-24	558041	21391	24	2011-06-24 13:2
10	2011-12-09 12:50:00 UTC	2011-06-20	557472	35933	30	2011-06-20 13:2
11	2011-12-09 12:50:00 UTC	2011-12-09	581476	23167	96	2011-12-09 08:4
12	2011-12-09 12:50:00 UTC	2011-09-21	567526	21524	50	2011-09-21 09:0
13	2011-12-09 12:50:00 UTC	2011-10-26	572886	21084	24	2011-10-26 13:4
14	2011-12-09 12:50:00 UTC	2011-10-05	569653	21376	4	2011-10-05 12:5
15	2011-12-09 12:50:00 UTC	2011-07-07	559298	23319	12	2011-07-07 12:3

Q. most_recent_date**의 값 중 가장 최근 구매일은 몇 일인가요?**

- 2011-12-09

이번에는 유저 별로 가장 최근에 일어난 구매 정보를 정리해 봅시다. **유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장**해 주겠습니다.

```sql
SELECT
  CustomerID,
  MAX(DATE(InvoiceDate)) AS most_recent_purchase_date
FROM
  praxis-beacon-464902-t2.modulabs_project.data2
GROUP BY
  CustomerID;
```

쿼리 결과

작업 정보 | 결과 | 차트 | JSON

행	CustomerID	most_recent_pur...
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

다음에는 **가장 최근 일자(**most_recent_date**)와 유저별 마지막 구매일(**InvoiceDay**)간의 차이를 계산**하겠습니다.

```sql
SELECT
  CustomerID,
  EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
  FROM praxis-beacon-464902-t2.modulabs_project.data2
  GROUP BY CustomerID
);
```

쿼리 결과

작업 정보 결과 차트 JSON 실

행	CustomerID	recency
1	12410	301
2	12445	22
3	12515	353
4	12793	334
5	13045	99
6	13075	129
7	13106	128
8	13149	14
9	13165	46

위의 코드를 더 자세히 살펴보도록 하겠습니다.

EXTRACT 함수는 날짜 또는 시간 데이터 타입에서 특정 부분을 추출하는 데 사용됩니다. 이 함수는 주로 SQL에서 날짜와 시간 연산을 수행할 때 사용되며, 날짜의 연도, 월, 일 또는 시간의 시, 분, 초 등을 추출할 수 있습니다.

쿼리문에서 EXTRACT 함수의 사용 방법을 자세히 살펴보겠습니다.

1. **MAX(InvoiceDay) OVER () - InvoiceDay** : 각 고객(CustomerID)의 각 구매일(InvoiceDay)과 전체 데이터셋에서의 마지막 구매일(MAX(InvoiceDay)) 간의 차이를 계산합니다.
2. **EXTRACT(DAY FROM ...)** : 여기에서 EXTRACT 함수는 위에서 계산된 날짜 차이에서 일(DAY) 부분만을 추출합니다. 즉 각 고객의 최근 구매일로부터 해당 구매 건의 구매일부터의 날짜 차이를 계산하는 함수입니다.

이제 **최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이도록** 하겠습니다. **지금까지의 결과를** user_r**이라는 이름의 테이블로 저장**해 주세요.

```sql
CREATE OR REPLACE TABLE `praxis-beacon-464902-t2.modulabs_project.user_r` AS

WITH last_purchase AS (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS last_invoice_day
  FROM
    `praxis-beacon-464902-t2.modulabs_project.data2`
  GROUP BY
    CustomerID
),

most_recent AS (
  SELECT
    MAX(last_invoice_day) AS overall_last_day
  FROM
    last_purchase
)

SELECT
  lp.CustomerID,
  DATE_DIFF(mr.overall_last_day, lp.last_invoice_day, DAY) AS recency
FROM
  last_purchase lp
CROSS JOIN
  most_recent mr;
```

user_r 쿼리 다음

스키마 세부정보 미리보기

행	CustomerID	recency
1	15311	0
2	15694	0
3	16626	0
4	12713	0
5	15910	0
6	17315	0
7	14441	0
8	17428	0
9	12423	0
10	14422	0
11	12433	0
12	12526	0
13	17754	0
14	12985	0
15	16446	0
16	13069	0
17	16954	0
18	12662	0
19	15344	0
20	17364	0

Frequency

Frequency를 계산하는 단계에서는 **고객의 구매 빈도 또는 참여 빈도**에 초점을 맞춥니다. 전체 거래 건수로 계산을 할 수도 있고, 구매한 아이템의 수량을 합하여 계산할 수도 있습니다. 예를 들어 한 명의 고객이 구매를 2번 했는데 각각 아이템을 4개씩 구매한 경우, 해당 고객의 거래 건수는 2회겠지만 실제로 구매한 수량은 8개가 됩니다. 이 두가지 측면을 모두 포착하기 위해 두 개를 모두 계산해 봅시다.

1. 전체 거래 건수 계산

우선 각 고객의 거래 건수를 세어 봅시다. 거래 건은 InvoiceNo를 기준으로 파악하면 되기 때문에, **고객마다 고유한 InvoiceNo의 수**를 세어 주겠습니다.

```sql
SELECT
  CustomerID,
  COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM
  `praxis-beacon-464902-t2.modulabs_project.data2`
GROUP BY
  CustomerID;
```

쿼리 결과

작업 정보 | 결과 | 차트 | JSON

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

2. 구매한 아이템의 총 수량 계산

그 다음으로는 **각 고객 별로 구매한 아이템의 총 수량**을 더해주겠습니다.

```sql
SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
FROM
  `praxis-beacon-464902-t2.modulabs_project.data2`
GROUP BY
  CustomerID;
```

쿼리 결과

작업 정보 | 결과 | 차트 | JSON

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

이제 위에서 구한 **'1. 전체 거래 건수 계산'과 '2. 구매한 아이템의 총 수량 계산'의 결과를 합쳐서** user_rf**라는 이름의 테이블에 저장**해 주겠습니다.

```sql
CREATE OR REPLACE TABLE `praxis-beacon-464902-t2.modulabs_project.user_rf` AS

WITH purchase_cnt AS (
  -- (1) 고객별 고유 거래 건수 계산
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt
  FROM
    `praxis-beacon-464902-t2.modulabs_project.data2`
  GROUP BY
    CustomerID
),

item_cnt AS (
  -- (2) 고객별 구매 아이템 총 수량 계산
  SELECT
    CustomerID,
    SUM(Quantity) AS item_cnt
  FROM
    `praxis-beacon-464902-t2.modulabs_project.data2`
  GROUP BY
    CustomerID
)

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
  `praxis-beacon-464902-t2.modulabs_project.user_r` AS ur
  ON pc.CustomerID = ur.CustomerID;
```

user_rf 쿼리 다음에서 열기 공유

스키마 세부정보 미리보기 테이블 탐색기 프리뷰

행	CustomerID	purchase_cnt	item_cnt	recency
1	12713	1	505	0
2	14569	1	79	1
3	13298	1	96	1
4	13436	1	76	1
5	15520	1	314	1
6	15471	1	256	2
7	15195	1	1404	2
8	14204	1	72	2
9	16528	1	171	3
10	14578	1	240	3

Monetary

Monetary를 계산하는 단계에서는 **고객이 지불한 총 금액**에 초점을 맞춥니다. 이 때 총 지출액을 계산할수도 있고, 거래당 평균 거래 금액을 계산할 수도 있습니다. 예를 들어, 한 명의 고객이 총 2번의 구매를 했고, 그 합산 금액이 10만원인 경우, 총 지출액은 10만원, 거래당 평균 거래 금액은 5만원이 되는 것입니다.

결제한 총 금액이 높은 고객을 찾는 것도 좋지만, 한번에 많이 구매하는 고객들을 찾는 것도 굉장히 중요합니다. 결제 금액은 낮지만 구매를 자주 하는 고객과, 한번 결제할 때 큰 금액을 결제하는 고객은 분명 특성이나 행동 패턴이 다를테니까요.

이 두 가지 측면을 모두 포착하기 위해 고객별 총 지출액과 평균 거래 금액을 모두 계산해 봅시다.

1. 고객별 총 지출액 계산

고객별 총 지출액을 계산해 보세요. **소수점 첫째 자리에서 반올림**하세요.

```sql
SELECT
  CustomerID,
  ROUND(SUM(Quantity * UnitPrice), 1) AS user_total
FROM
  `praxis-beacon-464902-t2.modulabs_project.data2`
GROUP BY
  CustomerID;
```

쿼리 결과

작업 정보 | 결과 | 차트 | JSON

행	CustomerID	user_total
1	12346	0.0

2. 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 나누어**서 3) user_rfm **테이블로 저장**해 봅시다.

```
CREATE OR REPLACE TABLE `praxis-beacon-464902-t2.modulabs_project.user_rfm` AS

SELECT
  rf.CustomerID                          AS CustomerID,
  rf.purchase_cnt,
  rf.item_cnt,
  rf.recency,
  ut.user_total,
  ROUND(ut.user_total / rf.purchase_cnt, 1) AS user_average
FROM
  `praxis-beacon-464902-t2.modulabs_project.user_rf` AS rf
LEFT JOIN (
  -- (1) 고객별 총 지출액 계산
  SELECT
    CustomerID,
    ROUND(SUM(Quantity * UnitPrice), 1) AS user_total
  FROM
    `praxis-beacon-464902-t2.modulabs_project.data`
  GROUP BY
    CustomerID
) AS ut
  ON rf.CustomerID = ut.CustomerID;
```

user_rfm 쿼리 다음에서 열기 공유 복사 스냅샷

스키마 세부정보 미리보기 테이블 탐색기 프리뷰 통계 계보 데이터 프로필

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	848.5	848.5
2	13436	1	76	1	196.9	196.9
3	14569	1	79	1	227.4	227.4
4	15520	1	314	1	343.5	343.5
5	13298	1	96	1	360.0	360.0
6	15471	1	256	2	469.5	469.5
7	14204	1	72	2	161.0	161.0
8	15195	1	1404	2	3861.0	3861.0
9	15318	1	642	3	312.6	312.6
10	12478	1	233	3	681.0	681.0

RFM 통합 테이블 출력하기

위와 같은 과정을 거쳐서 생성된 **최종 user_rfm 테이블을 출력**해주겠습니다.

```
SELECT
  *
FROM
  `praxis-beacon-464902-t2.modulabs_project.user_rfm`;
```

쿼리 결과 결과 저

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	848.5	848.5
2	13436	1	76	1	196.9	196.9
3	14569	1	79	1	227.4	227.4
4	15520	1	314	1	343.5	343.5
5	13298	1	96	1	360.0	360.0
6	15471	1	256	2	469.5	469.5

Q. 고유한 유저의 수는 몇 명인가요?

4,362명

11-8. 추가 Feature 추출

1. 구매하는 제품의 다양성

이 단계에서는 고객들의 제품 구매 행동 속 구매 제품의 다양성을 살펴보려고 합니다. 고객이 얼마나 다양한 제품들에 관심 있는 사람인지를 알게 되면, 개인 맞춤형 마케팅 전략과 추천 서비스를 계획하는 데에도 큰 도움이 될 수 있습니다.

우선 1) **고객 별로 구매한 상품들의 고유한 수를 계산**합니다. 높은 숫자가 나오는 것은 해당 고객이 다양한 제품들을 구매한다는 의미이며, 낮은 값이 나오는 경우 소수의 제품들만 구매한다는 것을 의미합니다.

이후 2) user_rfm **테이블과 결과를 합치고**, 이를 3) user_data**라는 이름의 테이블에 저장**하겠습니다.

```sql
CREATE OR REPLACE TABLE praxis-beacon-464902-t2.modulabs_project.user_data AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM praxis-beacon-464902-t2.modulabs_project.data2
  GROUP BY CustomerID
)
SELECT ur.*, up.* EXCEPT (CustomerID)
FROM praxis-beacon-464902-t2.modulabs_project.user_rfm AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

user_data 쿼리 다음에서 열기 공유 복사 스냅샷 삭제 내보

스키마 세부정보 미리보기 테이블 탐색기 프리뷰 통계 계보 데이터 프로필 데이터 품질

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	17956	1	1	249	12.8	12.8	1
2	17291	1	72	308	550.8	550.8	1
3	17331	1	16	123	175.2	175.2	1
4	16953	1	10	30	20.8	20.8	1
5	14090	1	72	324	76.3	76.3	1
6	18068	1	6	289	101.7	101.7	1
7	18141	1	-12	360	-35.4	-35.4	1
8	16061	1	-1	269	-29.9	-29.9	1
9	13366	1	144	50	56.2	56.2	1
10	16881	1	600	66	432.0	432.0	1

2. 평균 구매 주기

이 단계에서는 **고객들의 쇼핑 패턴을 이해**하는 것을 목표로 합니다. 그 중에서도 고객 별 재방문 주기를 살펴볼 것입니다. 고객들의 구매와 구매 사이의 기간이 평균적으로 몇 일인지를 보여주는 평균 일수를 계산하면, 고객이 다음 구매를 언제할지 예측하는 데에도 큰 도움이 됩니다. **평균 구매 소요 일수를 계산**하고, 그 **결과를** user_data **에 통합**해 줍시다.

```sql
CREATE OR REPLACE TABLE praxis-beacon-464902-t2.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
      praxis-beacon-464902-t2.modulabs_project.data2
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM praxis-beacon-464902-t2.modulabs_project.user_data AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

user_data 쿼리 다음에서 열기 공유 복사 스냅샷 삭제 내보내기

스키마 세부정보 미리보기 테이블 탐색기 프리뷰 통계 계보 데이터 프로필 데이터 품질

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_prod...	average_interval
1	16953	1	10	30	20.8	20.8	1	0.0
2	13270	1	200	366	590.0	590.0	1	0.0
3	16138	1	-1	368	-8.0	-8.0	1	0.0
4	12603	1	56	21	739.2	739.2	1	0.0
5	18113	1	72	368	76.3	76.3	1	0.0
6	16737	1	288	53	417.6	417.6	1	0.0
7	14090	1	72	324	76.3	76.3	1	0.0
8	14119	1	-2	354	-19.9	-19.9	1	0.0
9	13120	1	12	238	30.6	30.6	1	0.0
10	13135	1	4300	196	3096.0	3096.0	1	0.0

3. 구매 취소 경향성

이 단계에서는 **고객의 취소 패턴**을 더 깊이 있게 파고 들어 고객을 세그먼테이션할 때 사용하려고 합니다. 아래와 같은 특징들을 추가해 보겠습니다.

1. 취소 빈도(cancel_frequency)

취소 빈도는 **고객 별로 취소한 거래의 총 횟수**입니다.

취소 빈도를 이해하면 거래를 취소할 가능성이 높은 고객을 식별할 수 있습니다. 취소 빈도는 불만족의 정도이거나 다른 문제에 대한 지표일 수 있습니다. 따라서 취소 빈도를 이해함으로써 거래 취소 횟수를 줄이고 고객 만족도를 높이기 위한 전략을 세울 수 있습니다.

2. 취소 비율(cancel_rate)

취소 비율은 **각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율**입니다.

취소 비율은 특정 고객이 원래 취소를 잘 하는 고객인지와 같은 고객의 특징을 잡아내기 위한 지표입니다. 이런 특성을 식별함으로써 고객의 쇼핑 경험을 개선하고 취소 비율을 감소시키기 위해 어떤 고객 대상군을 공략해야 할지에 대한 실마리를 얻을 수 있습니다.

취소 빈도와 취소 비율을 계산하고 그 **결과를** user_data**에 통합**해 줍시다. **취소 비율은 소수점 두번째 자리**까지 구해 주세요.

```
CREATE OR REPLACE TABLE `praxis-beacon-464902-t2.modulabs_project.user_data` AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo)                    AS total_transactions,
    COUNTIF(STARTS_WITH(InvoiceNo, 'C'))          AS cancel_frequency
  FROM
    `praxis-beacon-464902-t2.modulabs_project.data`
  GROUP BY
    CustomerID
)

SELECT
  rf.*,
  t.total_transactions,
  t.cancel_frequency,
  ROUND(t.cancel_frequency / t.total_transactions * 100, 2) AS cancel_rate
FROM
  `praxis-beacon-464902-t2.modulabs_project.user_rfm` AS rf
LEFT JOIN
  TransactionInfo AS t
ON
  rf.CustomerID = t.CustomerID;
```

user_data 쿼리 다음에서 열기 공유 복사 스냅샷 삭제 내보내기

스키마 세부정보 미리보기 테이블 탐색기 프리뷰 통계 계보 데이터 프로필 데이터 품질

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	total_transac...	cancel_frequ...	cancel_rate
1	13844	1	196	11	361.6	361.6	1	0	0.0
2	15043	1	300	31	537.0	537.0	1	0	0.0
3	16318	1	199	35	328.1	328.1	1	0	0.0
4	18104	1	362	37	754.6	754.6	1	0	0.0
5	13103	1	82	39	243.9	243.9	1	0	0.0
6	15385	1	161	40	316.9	316.9	1	0	0.0
7	13845	1	182	64	313.0	313.0	1	0	0.0
8	13514	1	40	73	152.2	152.2	1	0	0.0
9	15325	1	231	103	162.3	162.3	1	0	0.0
10	15007	1	69	159	156.9	156.9	1	0	0.0
11	12922	1	370	161	405.4	405.4	1	0	0.0

지금까지 추가적인 feature를 추출하기 위한 작업을 진행하였습니다.

다양한 컬럼들을 활용하여 고객의 구매 패턴과 선호도를 보다 심층적으로 이해할 수 있도록 최종적으로 user_data를 출력해보겠습니다.

```
SELECT
  *
FROM
  `praxis-beacon-464902-t2.modulabs_project.user_data`;
```

쿼리 결과

결과 저장 다음에서 열기

작업 정보 결과 차트 JSON 실행 세부정보 실행 그래프

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	total_transactions	cancel_frequency	cancel_rate
1	13844	1	196	11	361.6	361.6	1	0	0.0
2	15043	1	300	31	537.0	537.0	1	0	0.0
3	16318	1	199	35	328.1	328.1	1	0	0.0
4	18104	1	362	37	754.6	754.6	1	0	0.0
5	13103	1	82	39	243.9	243.9	1	0	0.0
6	15385	1	161	40	316.9	316.9	1	0	0.0
7	13845	1	182	64	313.0	313.0	1	0	0.0

페이지당 결과 수: 50 1 – 50 (전체 4362행)