### ****Base Dataset Query****

sql

Copy code

-- Create a Base Dataset for All Analyses

WITH base\_data AS (

SELECT

-- Common Fields

DATE(request\_time) AS request\_date,

request\_time,

experiment\_limb\_id,

request\_section,

request\_channel,

request\_location,

add\_to\_cart\_flag,

product\_name,

-- Pricing Policy Fields

pricing\_tier,

pricing\_floor,

pricing\_ceiling,

target\_price,

-- Household and User Segmentation

household\_region,

household\_social\_grade,

household\_adults\_count,

household\_mrc\_avg,

-- Experimental Data

experiment\_id,

action\_id,

-- Conversion Metrics

SUM(CASE WHEN add\_to\_cart\_flag = 1 THEN 1 ELSE 0 END) OVER(PARTITION BY experiment\_limb\_id, request\_section) AS section\_add\_to\_cart\_count,

COUNT(\*) OVER(PARTITION BY experiment\_limb\_id, request\_section) AS section\_request\_count,

SAFE\_DIVIDE(

SUM(CASE WHEN add\_to\_cart\_flag = 1 THEN 1 ELSE 0 END) OVER(PARTITION BY experiment\_limb\_id, request\_section),

COUNT(\*) OVER(PARTITION BY experiment\_limb\_id, request\_section)

) \* 100 AS section\_conversion\_rate

FROM

`your\_project.your\_dataset.your\_table`

WHERE

request\_time BETWEEN '2024-01-01' AND '2024-01-31' -- Time Range Filter

AND experiment\_limb\_id IS NOT NULL

AND request\_section IS NOT NULL

AND pricing\_tier IS NOT NULL

)

SELECT

\*,

-- Derived Metrics for Cross-Sell Analysis

SAFE\_DIVIDE(

section\_add\_to\_cart\_count,

section\_request\_count

) AS overall\_conversion\_rate

FROM base\_data;

### ****Explanation of the Query****

1. base\_data **CTE**:
   * Combines all relevant fields from the raw table.
   * Precomputes essential metrics:
     + section\_add\_to\_cart\_count: Total number of add-to-cart actions per experiment limb and section.
     + section\_request\_count: Total number of requests per experiment limb and section.
     + section\_conversion\_rate: Conversion rate per section.
2. **Filters**:
   * **Time Range**: Filters data for a specific analysis window (2024-01-01 to 2024-01-31).
   * Ensures key columns (experiment\_limb\_id, request\_section, and pricing\_tier) are not null.
3. **Household and Segmentation**:
   * Includes fields like household\_region, household\_social\_grade, and household\_mrc\_avg to analyze confounding variables.
4. **Pricing Data**:
   * Includes pricing\_tier, pricing\_floor, pricing\_ceiling, and target\_price for pricing policy analysis.
5. **Derived Metrics**:
   * Adds pre-computed metrics (overall\_conversion\_rate, section\_conversion\_rate) to reduce the need for repetitive calculations in subsequent queries.

### ****Using the Base Dataset for Analysis****

This base dataset can now be used as the input for all subsequent tasks.

#### ****Task 1: Analyze Card Exposure****

sql

Copy code

SELECT

experiment\_limb\_id,

request\_section,

COUNT(\*) AS total\_requests

FROM base\_data

GROUP BY experiment\_limb\_id, request\_section

ORDER BY total\_requests DESC;

#### ****Task 2: Assess Limbs Assigned and Traceability****

sql

Copy code

SELECT

experiment\_limb\_id,

request\_section,

SUM(add\_to\_cart\_flag) AS add\_to\_cart\_count,

SAFE\_DIVIDE(SUM(add\_to\_cart\_flag), COUNT(\*)) \* 100 AS conversion\_rate

FROM base\_data

GROUP BY experiment\_limb\_id, request\_section;

#### ****Task 3: Evaluate Confounding Variables****

sql

Copy code

SELECT

household\_region,

household\_social\_grade,

experiment\_limb\_id,

SUM(add\_to\_cart\_flag) AS add\_to\_cart\_count,

COUNT(\*) AS total\_requests,

SAFE\_DIVIDE(SUM(add\_to\_cart\_flag), COUNT(\*)) \* 100 AS conversion\_rate

FROM base\_data

GROUP BY household\_region, household\_social\_grade, experiment\_limb\_id;

#### ****Task 4: Analyze Pricing Policy Effects****

sql

Copy code

SELECT

pricing\_tier,

experiment\_limb\_id,

AVG(pricing\_floor) AS avg\_pricing\_floor,

AVG(pricing\_ceiling) AS avg\_pricing\_ceiling,

SAFE\_DIVIDE(SUM(add\_to\_cart\_flag), COUNT(\*)) \* 100 AS conversion\_rate

FROM base\_data

GROUP BY pricing\_tier, experiment\_limb\_id;

#### ****Task 5: Link Digital Touchpoints****

sql

Copy code

SELECT

request\_channel,

request\_location,

experiment\_limb\_id,

SUM(add\_to\_cart\_flag) AS add\_to\_cart\_count,

SAFE\_DIVIDE(SUM(add\_to\_cart\_flag), COUNT(\*)) \* 100 AS conversion\_rate

FROM base\_data

GROUP BY request\_channel, request\_location, experiment\_limb\_id;

### ****Advantages of Using a Base Dataset****

****import pandas as pd****

****# Load your dataset****

****df = pd.read\_csv("your\_dataset.csv") # Replace with your dataset file path****

****# Filter non-numerical columns excluding date columns****

****non\_numerical\_columns = df.select\_dtypes(include=['object']).columns****

****# Extract distinct values from each column and store them in a DataFrame****

****distinct\_values\_data = {****

****"Column": [],****

****"Distinct Values": []****

****}****

****for col in non\_numerical\_columns:****

****distinct\_values\_data["Column"].append(col)****

****distinct\_values\_data["Distinct Values"].append(", ".join(map(str, df[col].dropna().unique())))****

****# Convert to DataFrame****

****distinct\_values\_df = pd.DataFrame(distinct\_values\_data)****

****# Save to CSV****

****output\_file = "distinct\_non\_numerical\_values.csv"****

****distinct\_values\_df.to\_csv(output\_file, index=False)****

****print(f"Distinct values saved to {output\_file}")****

****import pandas as pd****

****# Load your dataset****

****df = pd.read\_csv("your\_dataset.csv") # Replace with your dataset file path****

****# Identify non-numerical columns excluding date columns****

****non\_numerical\_columns = df.select\_dtypes(include=['object']).columns****

****date\_columns = df.select\_dtypes(include=['datetime']).columns****

****# Exclude date columns from the list of non-numerical columns****

****columns\_to\_process = [col for col in non\_numerical\_columns if col not in date\_columns]****

****# Extract distinct values from each column and store them in a DataFrame****

****distinct\_values\_data = {****

****"Column": [],****

****"Distinct Values": []****

****}****

****for col in columns\_to\_process:****

****distinct\_values\_data["Column"].append(col)****

****distinct\_values\_data["Distinct Values"].append(", ".join(map(str, df[col].dropna().unique())))****

****# Convert to DataFrame****

****distinct\_values\_df = pd.DataFrame(distinct\_values\_data)****

****# Save to CSV****

****output\_file = "distinct\_non\_numerical\_values.csv"****

****distinct\_values\_df.to\_csv(output\_file, index=False)****

****print(f"Distinct values saved to {output\_file}")****

### ****1. Check the Context of the**** curr\_trading\_activity ****Column****

In the query, the field curr\_trading\_activity is used to identify the activity type, such as **cross-sell** or **regrade**. This column must originate from one of the tables in your dataset. To verify:

* Look at the **table definitions** or **data dictionary** for the column curr\_trading\_activity.
* Use a simple query to inspect the table columns:

sql

Copy code

SELECT \*

FROM bt-con.findcmi-dp-prod.com.dp\_cdm\_tread\_rw.cdm\_dly\_trad

LIMIT 10;

If curr\_trading\_activity exists in this table, it likely stores the activity type for transactions, including cross-sell events.

### ****2. Trace the Source of Joins****

In your query, multiple tables are joined. Each table enriches the data with additional attributes. You need to:

* Check which table contains fields related to **cross-sell-specific attributes** (e.g., product codes, promotion groups, activity types).
* Simplify the joins to isolate the source of cross-sell data:

sql

Copy code

SELECT DISTINCT curr\_trading\_activity, curr\_service\_product\_code, prev\_service\_product\_code

FROM bt-con.findcmi-dp-prod.com.dp\_cdm\_tread\_rw.cdm\_dly\_trad

WHERE LOWER(curr\_trading\_activity) = 'cross-sell';

### ****3. Look for Related Product Information****

Cross-sell typically involves **new product codes** or **promotion groups** added to a customer's existing portfolio. Check the product-related tables (e.g., bvp\_data\_product\_ro):

* Investigate which product categories or service codes represent cross-sell events:

sql

Copy code

SELECT DISTINCT curr\_service\_product\_code, prev\_service\_product\_code, curr\_service\_product\_group\_lv12, prev\_service\_product\_group\_lv12

FROM bt-dp-dataprod-dp-prod.com.dp\_bvp\_data\_product\_ro.vw.bvp\_dp\_202\_bq\_bt\_product\_master\_raw

WHERE curr\_service\_product\_code IS NOT NULL;

### ****4. Validate Using Business Rules****

Ask the following:

* Does cross-sell involve a specific **product type or combination** (e.g., mobile + broadband)?
* Does the dataset record **changes in products** from one category to another? For instance:

sql

Copy code

SELECT \*

FROM bt-con.findcmi-dp-prod.com.dp\_cdm\_tread\_rw.cdm\_dly\_trad

WHERE prev\_service\_product\_code IS NOT NULL

AND curr\_service\_product\_code IS NOT NULL

AND prev\_service\_product\_code != curr\_service\_product\_code;

### ****5. Perform a Data Inspection****

If the dataset contains sample data, inspect for patterns:

* Run queries on each table to see where **"cross-sell"** activity appears.
* Example queries:
  + **Transaction Table**:

sql

Copy code

SELECT DISTINCT curr\_trading\_activity

FROM bt-con.findcmi-dp-prod.com.dp\_cdm\_tread\_rw.cdm\_dly\_trad;

* + **Product Master Table**:

sql

Copy code

SELECT DISTINCT curr\_service\_product\_code, curr\_service\_product\_group\_lv12

FROM bt-dp-dataprod-dp-prod.com.dp\_bvp\_data\_product\_ro.vw.bvp\_dp\_202\_bq\_bt\_product\_master\_raw;

### ****6. Business Logic Check****

Cross-sell often comes from a combination of:

* **Transactional data**: Which records customer interactions and new product purchases.
* **Product data**: Which categorizes services/products to identify new additions.

If curr\_trading\_activity alone does not help, investigate:

* Promotion codes or groupings (e.g., curr\_promotion\_group\_lv12).
* Product codes (curr\_service\_product\_code).

### ****7. Test Individual Tables****

To isolate the **exact table**, test each join in the query:

1. Comment out all joins except the one you suspect.
2. Check if the cross-sell data still appears:

sql

Copy code

SELECT \*

FROM bt-con.findcmi-dp-prod.com.dp\_cdm\_tread\_rw.cdm\_dly\_trad

WHERE LOWER(curr\_trading\_activity) = 'cross-sell';

By isolating tables and tracing relevant fields, you can determine which one is contributing the cross-sell information. Let me know if you'd like help writing queries for further validation!

****SELECT 'column\_1' AS column\_name, column\_1 AS distinct\_value****

****FROM your\_table\_name****

****WHERE column\_1 IS NOT NULL****

****GROUP BY column\_1****

****UNION ALL****

****SELECT 'column\_2' AS column\_name, column\_2 AS distinct\_value****

****FROM your\_table\_name****

****WHERE column\_2 IS NOT NULL****

****GROUP BY column\_2****

****UNION ALL****

****SELECT 'column\_3' AS column\_name, column\_3 AS distinct\_value****

****FROM your\_table\_name****

****WHERE column\_3 IS NOT NULL****

****GROUP BY column\_3;****

****Here’s a rewritten version of the text from the image in different words:****

### ****Current State****:

The datasets contain numerous fields, all of which are potentially relevant to the analysis or specific tasks. However, there is a need to focus on identifying the most impactful ones.

### ****Proposed Change****:

Conduct a thorough analysis of the dataset to determine and extract the key fields essential for the objectives. This includes evaluating field importance, creating new derived fields if necessary, and eliminating redundant or irrelevant data.

### ****Benefit/Goal****:

This change aims to streamline the data analysis process by concentrating on the most valuable fields, ensuring that the generated insights align closely with the project goals.

### ****Acceptance Criteria****:

* A detailed list of relevant fields is compiled, with explanations provided for each if required.
* A final review of the selected fields is conducted to ensure completeness and alignment with objectives.

****-- Replace 'your\_table\_name' with the actual table name****

****-- Replace 'your\_schema\_name' if applicable, depending on the database****

****WITH non\_timestamp\_columns AS (****

****SELECT COLUMN\_NAME****

****FROM INFORMATION\_SCHEMA.COLUMNS****

****WHERE TABLE\_NAME = 'your\_table\_name' -- Specify your table name here****

****AND DATA\_TYPE NOT IN ('TIMESTAMP', 'DATETIME', 'DATE') -- Exclude timestamp-related columns****

****),****

****distinct\_values\_query AS (****

****SELECT STRING\_AGG(****

****'SELECT ''' || COLUMN\_NAME || ''' AS column\_name, CAST(' || COLUMN\_NAME || ' AS STRING) AS distinct\_value FROM your\_table\_name'****

****, ' UNION ALL '****

****) AS query\_text****

****FROM non\_timestamp\_columns****

****)****

****-- Execute the generated dynamic query****

****EXECUTE IMMEDIATE (****

****SELECT query\_text FROM distinct\_values\_query****

****);****

****-- Step 1: Create a dynamic SQL query****

****DECLARE dynamic\_query STRING;****

****-- Step 2: Build the query dynamically by selecting non-datetime columns****

****SET dynamic\_query = (****

****SELECT STRING\_AGG(****

****FORMAT(****

****"SELECT '%s' AS column\_name, %s AS distinct\_value FROM `your\_project\_id.your\_dataset.your\_table` GROUP BY %s",****

****column\_name, column\_name, column\_name****

****),****

****" UNION ALL "****

****)****

****FROM `your\_project\_id.your\_dataset.INFORMATION\_SCHEMA.COLUMNS`****

****WHERE table\_name = 'your\_table'****

****AND data\_type NOT IN ('DATETIME', 'TIMESTAMP', 'DATE') -- Exclude datetime columns****

****);****

****-- Step 3: Execute the generated query****

****EXECUTE IMMEDIATE dynamic\_query;****

****import pandas as pd****

****import numpy as np****

****import matplotlib.pyplot as plt****

****# Load your data into a DataFrame****

****# Replace 'your\_data.csv' with the actual dataset file path****

****df = pd.read\_csv('your\_data.csv')****

****# 1. Basic Schema Analysis****

****print("Dataset Overview:")****

****print(df.info()) # Prints column names, data types, and non-null counts****

****# 2. Descriptive Statistics****

****print("\nNumerical Columns Summary:")****

****print(df.describe()) # Summary for numerical columns****

****print("\nCategorical Columns Summary:")****

****categorical\_columns = df.select\_dtypes(include=['object']).columns****

****for col in categorical\_columns:****

****print(f"Column: {col}")****

****print(df[col].value\_counts())****

****print()****

****# 3. Missing Values Analysis****

****print("\nMissing Values Analysis:")****

****missing\_values = df.isnull().sum()****

****missing\_percentage = (missing\_values / len(df)) \* 100****

****print(pd.DataFrame({'Missing Values': missing\_values, 'Percentage': missing\_percentage}))****

****# 4. Correlation Matrix (Numerical Columns Only)****

****print("\nCorrelation Matrix:")****

****numerical\_columns = df.select\_dtypes(include=['float64', 'int64']).columns****

****correlation\_matrix = df[numerical\_columns].corr()****

****print(correlation\_matrix)****

****# Optional: Plot Heatmap for Correlation Matrix****

****plt.figure(figsize=(10, 8))****

****plt.title("Correlation Matrix Heatmap")****

****plt.imshow(correlation\_matrix, cmap='coolwarm', interpolation='nearest')****

****plt.colorbar()****

****plt.show()****

****# 5. Unique Value Analysis for Categorical Columns****

****print("\nUnique Values in Categorical Columns:")****

****for col in categorical\_columns:****

****print(f"{col}: {df[col].nunique()} unique values")****

****# 6. Data Quality Checks****

****# Check for duplicates****

****duplicates = df.duplicated().sum()****

****print(f"\nNumber of duplicate rows: {duplicates}")****

****# 7. Visualization Examples****

****# Distribution Plot for Numerical Columns****

****for col in numerical\_columns:****

****df[col].hist(bins=20, figsize=(5, 4))****

****plt.title(f"Distribution of {col}")****

****plt.xlabel(col)****

****plt.ylabel("Frequency")****

****plt.show()****

****# Bar Chart for Categorical Columns****

****for col in categorical\_columns:****

****df[col].value\_counts().plot(kind='bar', figsize=(8, 5))****

****plt.title(f"Value Counts of {col}")****

****plt.xlabel(col)****

****plt.ylabel("Frequency")****

****plt.show()****

****# Save processed data for further analysis****

****df.to\_csv('processed\_data.csv', index=False)****

****print("Processed data saved to 'processed\_data.csv'")****