Tutorial de BigQuery (con solucionario)

Herramientas de Procesamiento de Volúmenes Grandes de Datos

Enver G. Tarazona Vargas (peetaraz@upc.edu.pe)

Caso de Estudio: ElectroSmart - Optimizando el Marketing para Maximizar las Ventas

ElectroSmart es una empresa emergente en el sector del comercio electrónico, especializada en la venta de electrodomésticos y dispositivos electrónicos. Desde su fundación hace cinco años, la compañía ha experimentado un crecimiento constante, ampliando su catálogo de productos y su base de clientes en todo Perú. ElectroSmart busca ahora utilizar análisis de datos avanzados para optimizar sus estrategias de marketing y aumentar la eficiencia operativa.

Contexto

ElectroSmart se enfrenta al desafío de utilizar eficazmente la gran cantidad de datos acumulados para mejorar su marketing y operaciones. Los datos han sido recolectados a lo largo de varios años e incluyen información detallada sobre clientes, ventas y campañas de marketing.

Objetivos del Análisis

1. **Personalizar las campañas de marketing**, mejorando la respuesta de los clientes y aumentando el retorno de inversión (ROI).
2. **Identificar y capitalizar las tendencias de consumo**, asegurando que los productos más demandados estén disponibles y bien posicionados.
3. **Gestionar eficazmente el presupuesto de marketing**, asignando recursos de manera óptima entre los diferentes canales y campañas.
4. **Fomentar la lealtad del cliente**, desarrollando programas de fidelización que aumenten la retención y el valor de vida del cliente (CLV).

Base de Datos

Para realizar este análisis, se dispondrá de las siguientes tablas principales en BigQuery:

Tabla de Clientes

- `id_cliente`: ID único del cliente.
- `fecha_registro`: Fecha en que el cliente se registró.
- `ubicacion`: Ubicación geográfica del cliente en regiones de Perú como Lima, Cusco, Arequipa, y Trujillo.
- `edad`: Edad del cliente.
- `ingresos`: Rango de ingresos del cliente.

Tabla de Productos

- `id_producto`: ID único de producto.
- `nombre`: Nombre del producto.
- `categoria`: Categoría del producto (por ejemplo, electrodomésticos, electrónica, accesorios).
- `precio`: Precio del producto.
- `cantidad_stock`: Cantidad disponible en stock.

Tabla de Ventas

- `id_venta`: ID único de la venta.
- `id_cliente`: ID del cliente que realizó la compra.
- `id_producto`: ID del producto vendido.
- `fecha`: Fecha de la venta.
- `cantidad`: Cantidad de productos vendidos en esa transacción.
- `precio_total`: Precio total de la venta.

Tabla de Campañas de Marketing

- `id_campaña`: ID único de la campaña.
- `fecha_inicio`: Fecha de inicio de la campaña.
- `fecha_fin`: Fecha de finalización de la campaña.
- `presupuesto`: Presupuesto asignado a la campaña.
- `canal`: Canal de marketing utilizado (por ejemplo, Correo electrónico, Redes sociales, Televisión, Publicidad online).

Parte 1: Configuración inicial

1. Ir al sandbox de BigQuery: https://cloud.google.com/bigquery/docs/sandbox?hl=es-419
2. Crear un nuevo Proyecto: TutorialBQ
3. Crear conjunto de datos: electrosmart

4. Crear Tablas: Crear las tablas `ventas`, `productos`, `clientes` y `campañas` a partir de los archivos `ElectroSmart_Ventas.csv`,`ElectroSmart_Productos.csv`, `ElectroSmart_Clientes.csv` y `ElectroSmart_Campañas_Marketing.csv`.

Parte 2: Consultas Básicas en BigQuery

Tema 1: Selección y Filtrado de Datos

En SQL, la selección y el filtrado de datos se realizan utilizando la declaración `SELECT`. Esta declaración permite especificar exactamente qué columnas de una tabla deseas recuperar, así como las condiciones que deben cumplir los datos que quieres ver. Esto se facilita mediante el uso de `WHERE`, que filtra registros bajo condiciones específicas, y `ORDER BY`, que permite ordenar los resultados según una o más columnas.

Esquema de Consulta Básica

```
SELECT columna1, columna2
FROM tabla
WHERE condición
ORDER BY columna1 ASC|DESC
```

Este esquema muestra una consulta básica que selecciona `columna1` y `columna2` de `tabla`, filtra los registros donde la condición es verdadera y luego los ordena en orden ascendente o descendente según `columna1`.

Ejemplo 1: Selección simple y filtrado

Utilizaremos la tabla `Clientes` para este ejemplo. La consulta muestra cómo seleccionar clientes de una ubicación específica y ordenar los resultados por edad.

```
SELECT id_cliente, ingresos, edad
FROM `electrosmart.clientes`
WHERE ubicacion = 'Lima'
ORDER BY edad DESC
```

Ejercicios

1. Selecciona los nombres de los clientes que tienen entre 30 y 40 años.

- **Solución**:

```
SELECT id_cliente
FROM `electrosmart.clientes`
WHERE edad BETWEEN 30 AND 40
```

2. Selecciona todos los datos de los clientes cuya ubicación sea Trujillo y tengan un ingreso bajo, ordenados por edad de manera ascendente.

- **Solución**:

```
SELECT *
FROM `electrosmart.clientes`
WHERE ubicacion = 'Trujillo' AND ingresos = 'Bajo'
ORDER BY edad ASC
```

3. Selecciona todos los datos de los clientes mayores de 50 años que tengan un ingreso alto, ordenados por fecha de registro de manera descendente.

- **Solución**:

```
SELECT *
FROM `electrosmart.clientes`
WHERE edad > 50 AND ingresos = 'Alto'
ORDER BY fecha_registro DESC
```

Tema 2: Funciones de Agregación

Las funciones de agregación son herramientas poderosas en SQL que permiten realizar cálculos sobre un conjunto de valores, devolviendo un único valor. Estas funciones son esenciales para realizar análisis estadísticos como promedios, sumas, máximos y mínimos. Algunas de las funciones de agregación más comunes incluyen `AVG()`, `SUM()`, `MAX()`, y `MIN()`. Estas funciones son frecuentemente usadas con `GROUP BY`, que agrupa filas que tienen los mismos valores en columnas especificadas, permitiendo realizar cálculos agregados por grupo.

Ejemplo de uso de funciones de agregación

```
SELECT AVG(columna) as Promedio,
       MAX(columna) as Máximo,
       MIN(columna) as Mínimo,
       SUM(columna) as Suma
FROM tabla
WHERE condición
GROUP BY columna_agrupadora
```

Este esquema muestra una consulta que calcula el promedio, el máximo, el mínimo y la suma de `columna` en `tabla`, donde los registros cumplen una condición y luego agrupa los resultados por `columna_agrupadora`

Ejemplo 2: Uso de funciones de agregación

Calcularemos la edad promedio de todos los clientes en la base de datos utilizando la tabla `Clientes`.

```sql
SELECT AVG(edad) as edad_promedio
FROM `electrosmart.clientes`
```

Ejercicios:

1. Calcula el número total de clientes.
   - **Solución**:

     ```sql
     SELECT COUNT(*) as total_clientes
     FROM `electrosmart.clientes`
     ```

2. Encuentra la edad máxima y mínima de los clientes.
   - **Solución**:

     ```sql
     SELECT MAX(edad) as edad_maxima, MIN(edad) as edad_minima
     FROM `electrosmart.clientes`
     ```

3. Calcula el total de ingresos generados por las ventas.
   - **Solución**:

     ```sql
     SELECT SUM(precio_total) as ingresos_totales
     FROM `electrosmart.ventas`
     ```

Parte 3: Consultas intermedias

Tema 3: Agrupación de Datos

Agrupación de Datos con `GROUP BY`

La agrupación de datos en SQL se realiza utilizando la cláusula `GROUP BY`. Esta cláusula es esencial cuando se utilizan funciones de agregación para obtener resultados por grupos específicos. Permite segmentar los datos en conjuntos únicos, que pueden ser sumados, promediados o evaluados de otra manera. Por ejemplo, podría agrupar las ventas por producto o por fecha, y luego calcular totales o promedios para cada grupo.

Esquema de Consulta con Agrupación de Datos

La cláusula `GROUP BY` se utiliza en SQL para agrupar filas que tienen los mismos valores en columnas especificadas. Esto es especialmente útil cuando se combinan con funciones de agregación para calcular, por ejemplo, sumas o promedios para cada grupo.

Estructura Básica de una Consulta con `GROUP BY`

```
SELECT columna1, función_agregación(columna2)
FROM nombre_tabla
WHERE condición
GROUP BY columna1
```

Ejemplo 3: Agrupación de Datos

Supongamos que queremos agrupar ventas por producto y calcular el total de ventas para cada producto. La consulta sería:

```
SELECT categoria, COUNT(*) as total_productos
FROM `electrosmart.productos`
GROUP BY categoria
```

Ejercicios: Agrupación de Datos

1. Agrupa las ventas por producto y calcula el total de ventas para cada producto.
   - **Solución**:

     ```
     SELECT id_producto, SUM(precio_total) as total_ventas
     FROM `electrosmart.ventas`
     GROUP BY id_producto
     ```

2. Encuentra el número promedio de ventas por cliente.
   - **Solución**:

     ```
     SELECT id_cliente, AVG(precio_total) as promedio_ventas
     FROM `electrosmart.ventas`
     GROUP BY id_cliente
     ```

3. Calcula el número total de ventas y el promedio de ventas para cada categoría de producto.
   - **Solución**:

     ```
     SELECT categoría, COUNT(*) as total_ventas, AVG(precio_total) as promedio_venta
     FROM `electrosmart.ventas`
     JOIN `electrosmart.productos` ON `electrosmart.ventas`.id_producto = `electrosm
     GROUP BY categoría
     ```

Agrupación de Datos con `GROUP BY` y `HAVING`

La cláusula `GROUP BY` se utiliza para agrupar filas que tienen los mismos valores en columnas especificadas, lo cual es útil cuando se combinan con funciones de agregación como `SUM()`, `AVG()`, `MAX()`, y `MIN()`. La cláusula `HAVING` se utiliza para filtrar los registros que resultan

de las agregaciones. Mientras que `WHERE` filtra filas antes de la agrupación, `HAVING` filtra los resultados después de agrupar los datos.

Esquema de Consulta con `GROUP BY` y `HAVING`

```
SELECT columna1, función_agregación(columna2)
FROM nombre_tabla
WHERE condición
GROUP BY columna1
HAVING condición_agregación
```

Ejemplo con `GROUP BY` y `HAVING`

Supongamos que queremos agrupar ventas por producto y calcular el total de ventas para cada producto, pero solo queremos mostrar aquellos productos que tienen un total de ventas superior a un cierto umbral. La consulta sería:

```
SELECT id_producto, SUM(precio_total) as total_ventas
FROM electrosmart.ventas
GROUP BY id_producto
HAVING SUM(precio_total) > 10000
```

- `GROUP BY id_producto` agrupa las ventas por cada producto. Esto es fundamental para aplicar la función de agregación al conjunto de datos resultante.
- `SUM(precio_total)` calcula el total de ventas para cada grupo. Esta función de agregación suma todos los valores de `precio_total` dentro de cada grupo definido por `id_producto`.
- `HAVING SUM(precio_total) > 10000` filtra y muestra solo aquellos grupos cuyas ventas totales superan los 10,000. `HAVING` se utiliza aquí para aplicar un filtro sobre el resultado de la función de agregación, permitiendo que solo los grupos que cumplen esta condición sean incluidos en los resultados finales.

Este enfoque permite enfocarse en productos de alto rendimiento, excluyendo aquellos que no alcanzan el umbral de ventas establecido, lo cual es útil para análisis dirigidos y decisiones estratégicas.

Ejercicios del Tema 3: Agrupación de Datos

1. Agrupa las ventas por producto y calcula el total de ventas para cada producto. Solo muestra aquellos productos cuyas ventas totales superen los 5,000.
   - **Solución**:

     ```
     SELECT id_producto, SUM(precio_total) as total_ventas
     FROM `electrosmart.ventas`
     GROUP BY id_producto
     ```

```
HAVING SUM(precio_total) > 5000
```

2. Encuentra el número promedio de ventas por cliente, pero solo para aquellos clientes que han gastado más de 1,000 en total.

   - **Solución**:

```
SELECT id_cliente, AVG(precio_total) as promedio_ventas
FROM `electrosmart.ventas`
GROUP BY id_cliente
HAVING SUM(precio_total) > 1000
```

3. Calcula el total y el promedio de las ventas diarias, solo incluye los días donde el promedio de ventas supera los 200.

   - **Solución**:

```
SELECT DATE(fecha) as venta_dia, COUNT(*) as total_ventas, AVG(precio_total) as
FROM `electrosmart.ventas`
GROUP BY DATE(fecha)
HAVING AVG(precio_total) > 200
```

4. Lista todos los clientes que han realizado más de 5 compras.

   - **Solución**:

```
SELECT id_cliente, COUNT(*) as numero_compras
FROM `electrosmart.ventas`
GROUP BY id_cliente
HAVING COUNT(*) > 5
```

Tema 4: Unión de Tablas

La unión de tablas es una operación clave en SQL que permite combinar filas de dos o más tablas basadas en una columna común entre ellas. A menudo se refiere a `JOIN` en su forma más básica como `INNER JOIN`, que es uno de los varios tipos de uniones disponibles:

- `JOIN` o `INNER JOIN`: Devuelve filas cuando hay una coincidencia en las columnas especificadas de ambas tablas. Si se usa `JOIN` sin un prefijo específico, generalmente se asume que es un `INNER JOIN`.
- `LEFT JOIN` o `LEFT OUTER JOIN`: Devuelve todas las filas de la tabla izquierda y las filas coincidentes de la tabla derecha. Si no hay coincidencia, los resultados tendrán `NULL` en las columnas de la tabla derecha.

- `RIGHT JOIN` o `RIGHT OUTER JOIN`: Es el opuesto al `LEFT JOIN`, devolviendo todas las filas de la tabla derecha y las coincidencias de la izquierda. Si no hay coincidencia, los resultados tendrán `NULL` en las columnas de la tabla izquierda.
- `FULL OUTER JOIN`: Combina los resultados de `LEFT JOIN` y `RIGHT JOIN`. Devuelve filas cuando hay una coincidencia en una de las tablas y también filas con `NULL` en las columnas de la tabla donde no hay coincidencias.

Estos diferentes tipos de uniones permiten a los analistas elegir cómo manejar los casos donde puede o no haber correspondencias entre las tablas, lo cual es crucial para la integridad del análisis y la presentación de datos completos y contextuales.

Ejemplo Práctico con `INNER JOIN`

Para comprender mejor cómo los productos son adquiridos por diferentes clientes, podemos obtener una lista de todos los clientes junto con los detalles de sus compras. Esto se logra mediante un `INNER JOIN`, que combina las filas de las tablas `clientes` y `ventas` basándose en una columna común. Supongamos que queremos ver los clientes con las cantidades de productos comprados y los montos totales. La consulta podría ser:

```
SELECT C.id_cliente, V.id_producto, V.cantidad, V.precio_total
FROM `electrosmart.clientes` C
INNER JOIN `electrosmart.ventas` V ON C.id_cliente = V.id_cliente
```

Supongamos ahora que quisiéramos mostrar en vez del código del producto el nombre. Esto implicaría realizar una unión más con la tabla `producto`:

```
SELECT C.id_cliente, P.nombre AS nombre_producto, V.cantidad, V.precio_total
FROM `electrosmart.clientes` C
INNER JOIN `electrosmart.ventas` V ON C.id_cliente = V.id_cliente
INNER JOIN `electrosmart.productos` P ON V.id_producto = P.id_producto
```

Ejercicios del Tema 4: Unión de Tablas

1. Encuentra el ID del cliente junto con el total de unidades compradas y el total gastado, agrupados por cliente.

   - **Solución**:

     ```
     SELECT C.id_cliente, SUM(V.cantidad) AS total_unidades, SUM(V.precio_total) AS
     FROM `electrosmart.clientes` C
     INNER JOIN `electrosmart.ventas` V ON C.id_cliente = V.id_cliente
     GROUP BY C.id_cliente
     ```

2. Lista todos los productos comprados por cada cliente junto con el total gastado por cada producto, solo para aquellos clientes que han gastado más de 500 en total.

   - **Solución**:

```
SELECT C.id_cliente, V.id_producto, SUM(V.precio_total) AS total_gastado
FROM `electrosmart.clientes` C
INNER JOIN `electrosmart.ventas` V ON C.id_cliente = V.id_cliente
GROUP BY C.id_cliente, V.id_producto
HAVING SUM(V.precio_total) > 500
```

3. Determina cuántos clientes han comprado cada producto y el total de ingresos generados por cada producto, pero solo para productos que generaron más de 1,000 en total.

   - **Solución**:

```
SELECT V.id_producto, COUNT(DISTINCT C.id_cliente) AS total_clientes, SUM(V.pre
FROM `electrosmart.ventas` V
INNER JOIN `electrosmart.clientes` C ON V.id_cliente = C.id_cliente
GROUP BY V.id_producto
HAVING SUM(V.precio_total) > 1000
```

4. Encuentra el total de compras y el número de productos comprados por cada cliente.

   - **Solución**:

```
SELECT C.id_cliente, COUNT(DISTINCT V.id_producto) AS numero_productos, SUM(V.p
FROM `electrosmart.clientes` C
INNER JOIN `electrosmart.ventas` V ON C.id_cliente = V.id_cliente
GROUP BY C.id_cliente
```

Parte 4: Consultas Avanzadas

Tema 5: Subconsultas y Consultas Anidadas

Las subconsultas y consultas anidadas permiten realizar operaciones más complejas dentro de una consulta SQL. Estas técnicas son cruciales para situaciones donde la ejecución de una consulta depende de los resultados de otra. Las subconsultas pueden aparecer en las cláusulas `SELECT`, `FROM`, o `WHERE`:

- **Subconsultas en `SELECT`**: Permiten realizar cálculos dinámicos sobre los datos seleccionados.
- **Subconsultas en `FROM`**: Se utilizan para tratar el resultado de una consulta como si fuera una tabla temporal.
- **Subconsultas en `WHERE`**: Ayudan a filtrar datos basados en un criterio que depende de otra consulta.

Ejemplo Práctico con Subconsulta en `WHERE`

Supongamos que queremos identificar los clientes que han gastado más que el promedio de gasto de todos los clientes. Para lograr esto, debemos calcular el promedio de gasto dentro de una subconsulta y luego usar este resultado en la cláusula `WHERE` de la consulta principal. Aquí está cómo podemos estructurar la consulta:

```
SELECT id_cliente, total_gastado
FROM (
    SELECT id_cliente, SUM(precio_total) AS total_gastado
    FROM electrosmart.ventas
    GROUP BY id_cliente
) AS gastos
WHERE total_gastado > (
    SELECT AVG(total_gastado) FROM (
        SELECT id_cliente, SUM(precio_total) AS total_gastado
        FROM electrosmart.ventas
        GROUP BY id_cliente
    ) AS promedios
)
```

En este ejemplo, utilizamos subconsultas para identificar a los clientes que han gastado más que el promedio de gasto de todos los clientes. La consulta se estructura de la siguiente manera:

1. **Subconsulta Interna en `FROM`**: Esta subconsulta crea una tabla temporal llamada `gastos` que contiene el `id_cliente` y el `total_gastado` por cada cliente. Esto se logra agrupando los datos de ventas por `id_cliente` y sumando el `precio_total` para cada grupo.
2. **Subconsulta en `WHERE` para calcular el promedio**: Aquí, calculamos el promedio de gasto de todos los clientes. Esta subconsulta opera sobre la tabla temporal `gastos`, sumando el `total_gastado` de todos los clientes y luego calculando el promedio.
3. **Consulta Principal**: En la consulta principal, seleccionamos los clientes cuyo `total_gastado` supera el promedio calculado. Esto se logra comparando el `total_gastado` de cada cliente en la tabla `gastos` con el promedio obtenido de la subconsulta.

Este enfoque permite realizar comparaciones dinámicas dentro de una base de datos, facilitando análisis detallados basados en comportamientos agregados de los usuarios o clientes. Al usar subconsultas, garantizamos que todos los cálculos necesarios para el filtro se realizan antes de producir el resultado final, asegurando eficiencia y precisión en la consulta.

Ejercicios del Tema 5: Subconsultas y Consultas Anidadas

1. Encuentra los clientes que han gastado más en un solo pedido que el promedio de gastos de todos los pedidos.

- **Solución**:

```
SELECT id_cliente, MAX(precio_total) AS max_gasto
FROM electrosmart.ventas
GROUP BY id_cliente
HAVING MAX(precio_total) > (
    SELECT AVG(precio_total) FROM electrosmart.ventas
)
```

2. Lista los productos que han sido comprados en una cantidad total que supera el promedio de cantidad vendida por producto.

- **Solución**:

```
SELECT id_producto, SUM(cantidad) AS total_vendida
FROM electrosmart.ventas
GROUP BY id_producto
HAVING SUM(cantidad) > (
    SELECT AVG(total_vendida) FROM (
        SELECT SUM(cantidad) AS total_vendida
        FROM electrosmart.ventas
        GROUP BY id_producto
    ) AS promedios
)
```

3. Determina los días en los que el total de ventas fue superior al promedio diario de ventas.

- **Solución**:

```
SELECT fecha, SUM(precio_total) AS total_ventas
FROM electrosmart.ventas
GROUP BY fecha
HAVING SUM(precio_total) > (
    SELECT AVG(total_ventas) FROM (
        SELECT fecha, SUM(precio_total) AS total_ventas
        FROM electrosmart.ventas
        GROUP BY fecha
    ) AS ventas_diarias
)
```