Sección 4: Estrategia

1.Estrategias.

Web Scrapping

La tarea consistía en obtener la población para cada uno de los países mencionados en el archivo. Lo primero que hice fue obtener los países únicos, pues el archivo contenía información sobre los estados/provincias y el país. Después de eso busque un sitio web público que tuviera la población de todos los países en el mundo, antes de decidir que sitio use revisé las políticas de cada uno de los sitios sobre el uso de web crawlers y seleccioné uno que sí permitía su uso. Una vez seleccionado el sitio web revisé el código fuente para entender la estructura y programar el crawler.

Para el crawler usé BeautifulSoup, lo que se hace es seleccionar la tabla donde estaba la información con el código soup.find('table', {'class': 'wikitable'}), después extraer el nombre del país y su población usando su posición en la tabla (el índice de la columna).

Finalmente transformamos tanto el nombre y la población a los formatos deseados, para el nombre del país se quitaron todos los caracteres especiales y duplicaciones. Para la población se quitaron todos los espacios en blanco, "," o "." Y se convirtió a numérico, todo esto se guardé en un diccionario para convertir en un data frame y guardarlo en el formato deseado (csv)

Transformación de Datos

Todo el proceso de limpieza y transformación de datos se realizó usando la API Pyspark, lo primero que hice fue iniciar una sesión de Spark. Definí un esquema (schema) para la tabla de primera ingesta y para el catalogo de relación geográfica use una función que infiriera el esquema pues era muy sencillo.

En el esquema para la tabla de primera ingesta definí los tipos de datos y los nombres de las columnas; después define el formato de la columna de tipo Timestamp.

El primer paso, después de leer mis dos bases fue transformar la columna de Timestamp de UTC a UTC-6.

Luego cree una llave primaria para definir mis registros únicos. Para esta llave primaria use la combinación de timestamp, estado/provincia y ID del ticket. Esta llave primaria

toma en cuenta los falsos Duplicados, pues al tener el mismo ID, timestamp y ubicación se considera un solo ticket (mismo valor para la llave primaria), sin importar el monto de la venta.

Una vez definí la llave primaria, agregué mi base para obtener el total de ventas por llave primaria única, es decir, se sumaron las ventas de aquellos registros que tenían misma llave primare (y por lo tanto mismo timestamp, ID del ticket y estado/provincia).

Finalmente hice un left-join con la tabla "catalogo de localización geográfica" y con la tabla generada con el crawler, la cual incluye la población para los países seleccionados. Una vez hice el left-join agregué las ventas por país, cree la variable de ventas por habitante (ventas/población) y cree la variable de ventas por un millón.

Para la tabla de la segunda ingesta, use el mismo esquema que para la primera, además hice un anti left-join para quedarme únicamente con registros nuevos (sin importar que estuvieran atrasados), cree una llave primaria y agregué los datos de la misma forma que para la primera ingesta, para finalmente unir estas nuevas ventas con las ventas agregadas de la primera ingesta.

Una vez junté las ventas de ambas ingestas, volví a agregar los datos para sumar los montos de aquellas ventas que llegaron retrasadas y repetí el proceso para sumar por país, calcular las ventas por habitante y crear la variable de ventas por un millón.

Carga en la nube

Para cargar los datos a un bucket en la nube decidí usar AWS, para eso utilicé el modulo boto3 que sirve para conectar Python con AWS. En el código solo se hace el import, pues al principio se hace la instalación y se asume que el sistema ya tiene las credenciales definidas para hacer la conexión.

Con boto3 se define el nombre del bucket y se da la instrucción que lo cree en caso de que no exista, para finalmente guardar el archivo creado.

También incluí la sección de Azure, la idea es la misma que AWS S3, pero en lugar de usar boto3 se usa BlobServiceClient del modulo azure.storage.blob.

2. Administración de costos asociados con la ejecución continua del pipeline

Lo primero es tener en cuenta los factores de costo del pipeline, sin embargo en general consideraría:

- Configurar instancias escalables que ajusten los recursos automáticamente según la carga de trabajo, para reducir el uso de recursos en momentos de menor actividad.
- Si el pipeline no necesita ejecutarse en tiempo real, se podría programar para ejecutarse en horas de menor demanda.
- Implementar políticas que muevan automáticamente los datos menos utilizados a clases de almacenamiento más económicas (por ejemplo, de S3 Standard a S3 Glacier o en una capa de archivo en Azure), los datos de acceso frecuente pueden permanecer en almacenamiento regular.
- Para reducir costos en la transferencia y procesamiento, considerar comprimir o particionar datos antes de subirlos al bucket y hacer transformaciones ligeras en la ingesta.
- Usar la misma región para los recursos de computación y almacenamiento para minimizar las tarifas de transferencia de datos.
- Configurar alertas que notifiquen cuando los gastos se acerquen a los límites presupuestados.

3. Mejoras del Pipeline

Asumiendo que el primer paso es la obtención de los datos poblacionales por país, considero que como lo definí ahorita es óptimo.

Definiría un segundo paso en el que se extraiga de un bucket, Data warehouse o fuente deseada todos los datos de las ingestas disponibles, para hacerles la limpieza y transformación a todos a la vez y finalmente solo hacer una vez en todo el proceso el join con la población, agregar por país y creación de nuevas variables.

También se podrían hacer pruebas automatizadas de validación de datos y monitorear la calidad en cada paso para asegurar que las transformaciones se realicen correctamente. Además de almacenar versiones del pipeline y datos históricos en caso de que se necesite retroceder o comparar con versiones previas en caso de errores.

Finalmente, agregar un sistema de cacheo para datos que se consultan frecuentemente. Esto evitaría llamar a APIs externas o regenerar datos en cada ejecución.