

## **DATA ENGINEERING - RECRUITMENT USE CASE**

## Resumo:

Este projeto apresenta o desenvolvimento de um pipeline de dados robusto e escalável para a consolidação de métricas de vendas, focado em fornecer inteligência competitiva para o departamento de **Martech**. A solução transforma dados brutos e heterogêneos em um ecossistema analítico baseado na **Arquitetura Medalhão (Lakehouse)**, garantindo a integridade dos dados desde a ingestão até o consumo final.

### Destaques da Implementação:

- **Escalabilidade e Performance:** Utilização de **PySpark** e **Delta Lake** em ambiente containerizado (**Docker**), assegurando que o pipeline suporte volumes de Big Data com alta performance de processamento.
- **Modelagem Star Schema:** Estruturação dos dados em tabelas Fato e Dimensões na camada **Gold**, otimizada para ferramentas de BI e visualização, reduzindo a latência em consultas complexas.
- **Enriquecimento de Dados (Master Data):** Implementação de uma dimensão de tempo customizada (com suporte a feriados internacionais e sazonalidade) e padronização de canais de venda via **Regex**, permitindo uma visão executiva clara sobre os principais agrupamentos de mercado (*Trade Groups*).
- **Analytics Avançado:** Aplicação de **Window Functions** para a geração de rankings regionais dinâmicos e análises de séries temporais para acompanhamento da evolução mensal por marca.

A entrega final não apenas responde aos requisitos técnicos do case, mas estabelece uma fundação de dados governada, pronta para suportar modelos de *Machine Learning* e dashboards de alta fidelidade para a tomada de decisão estratégica.

## Abstract:

This project demonstrates the development of a robust and scalable data pipeline for sales metrics consolidation, designed to provide competitive intelligence for the **Martech** department. The solution transforms raw, heterogeneous data into an analytical ecosystem based on the **Medallion Architecture (Lakehouse)**, ensuring data integrity from ingestion to end-user consumption.

### Technical Highlights:

- **Scalability & Performance:** Developed using **PySpark** and **Delta Lake** within a containerized environment (**Docker**), ensuring the pipeline supports Big Data volumes with high-performance processing.
- **Star Schema Modeling:** Data structured into Fact and Dimension tables within the **Gold layer**, optimized for BI tools and visualization, significantly reducing latency in complex analytical queries.
- **Master Data Enrichment:** Implementation of a customized Time Dimension (supporting international holidays and seasonality) and standardization of sales channels via **Regex**, enabling a clear executive view of strategic **Trade Groups**.
- **Advanced Analytics:** Application of **Window Functions** to generate dynamic regional rankings and time-series analysis for monitoring monthly brand performance evolution.

The final delivery not only fulfills the technical requirements of the business case but establishes a governed data foundation, ready to support Machine Learning models and high-fidelity dashboards for strategic decision-making.

# Sumário

|  |           |
|--|-----------|
| <b>1 . Do problema.....</b>                              | <b>5</b>  |
| 1.1 Das condições de resolução:.....                     | 6         |
| 1.1.1. Objetivo e Infraestrutura (The MVP Approach)..... | 6         |
| 1.1.2. Requisitos de Negócio.....                        | 6         |
| 1.1.3. Premissa de Arquitetura.....                      | 7         |
| 1.1.4. Arquitetura Aplicada e Infraestrutura.....        | 7         |
| 1.1.4.1 Do ambiente de desenvolvimento.....              | 9         |
| 1.2 Segurança.....                                       | 10        |
| 1.2.1 Segurança no Jupyter Lab.....                      | 11        |
| 1.2.2 Integridade de Dados.....                          | 11        |
| 1.2.3 O que seria feito em Produção.....                 | 11        |
| <b>2. Da Implementação.....</b>                          | <b>11</b> |
| 2.1 Lakehouse.....                                       | 13        |
| 2.1.1 Camada Bronze.....                                 | 14        |
| 2.1.1.1 Rotina de Channel Group.....                     | 15        |
| 2.1.1.2 Rotina de Sales.....                             | 16        |
| 2.1.2 Camada Silver.....                                 | 17        |
| 2.1.2.1 Rotina de Channel Group.....                     | 18        |
| 2.1.2.2 Rotina de Sales.....                             | 19        |
| 2.1.3 Camada Gold.....                                   | 20        |
| 2.1.3.1 Rotina de Channel Group.....                     | 20        |
| 2.1.3.2 Rotina de Sales.....                             | 21        |
| 2.2 Dos Conjuntos de Dados.....                          | 23        |
| 2.2.1 Camada Bronze (Landing to Raw).....                | 23        |
| 2.2.2 Camada Silver (Cleansing & Standard).....          | 25        |
| 2.2.3 Camada Gold.....                                   | 27        |
| 2.2.3.1 Modelo Entidade Relacionamento.....              | 31        |
| <b>3. Da solução do problema.....</b>                    | <b>32</b> |
| 3.1 Abordagem Analítica.....                             | 32        |
| 3.2 Implementação das Consultas de Negócio.....          | 34        |
| 3.1 Das Saídas e Exportação de Dados.....                | 38        |
| 3.2 Repositório GIT.....                                 | 39        |
| <b>4. Referências.....</b>                               | <b>41</b> |
| 4.1 Frameworks de Engenharia de Dados.....               | 41        |
| 4.2 Infraestrutura e Ambiente.....                       | 41        |
| 4.3 Bibliotecas e Utilitários.....                       | 41        |
| 4.4 Colaboração de IA.....                               | 41        |

# 1 . Do problema

O desafio central deste projeto reside na fragmentação e heterogeneidade dos dados de vendas provenientes de diferentes sistemas e regiões. Para o time de **Martech**, a ausência de uma padronização impossibilita uma visão clara do desempenho do mercado. Os principais pontos críticos identificados foram:

- **Dados Não Estruturados:** Arquivos em formatos brutos com problemas de *encoding* e falta de tipagem definida.
- **Sujeira no Master Data:** Canais de venda com nomes inconsistentes e prefixos numéricos que dificultam o agrupamento gerencial.
- **Falta de Contexto Temporal:** Ausência de uma base de tempo que correlacione vendas com feriados e sazonalidades, pontos vitais para a indústria de bebidas.
- **Necessidade de Escala:** A exigência de uma solução que não apenas processe o volume atual, mas que seja resiliente e escalável para cenários de Big Data.

## 1.1 Das condições de resolução:

A execução deste projeto seguiu rigorosamente as diretrizes estabelecidas nas "Main Instructions" e os requisitos específicos do "Business Case 1 – Beverage Sales", conforme detalhado abaixo:

### 1.1.1. Objetivo e Infraestrutura (The MVP Approach)

Conforme solicitado, o foco foi a construção de um **MVP para uma plataforma de analytics de vendas de bebidas**. A escolha da infraestrutura e stack tecnológica priorizou a demonstração de habilidades técnicas e escalabilidade:

- **Ambiente:** Stack *stand-alone* executada em ambiente local (Notebook/PC) via **WSL2** e **Docker**.
- **Stack:** Utilização de **Jupyter** e **PySpark** para o desenvolvimento do motor de processamento.
- **Data Resources:** Processamento e integração dos datasets `abi_bus_case1_beverage_sales.csv` e `abi_bus_case1_beverage_channel_group.csv`.

### 1.1.2. Requisitos de Negócio

A validação da arquitetura proposta deu-se através da criação de scripts/queries capazes de responder às seguintes questões de negócio, utilizando como métrica principal o volume de vendas (\$ Volume):

- **Item 4.1:** Identificação do Top 3 **Trade Groups** (**TRADE\_GROUP\_DESC**) por **Região** (**Bt1r\_Org\_LVL\_C\_Desc**).
- **Item 4.2:** Volume de vendas consolidado por **Marca** (**BRAND\_NM**) por **Mês**.
- **Item 4.3:** Mapeamento da **Marca** (**BRAND\_NM**) com menor volume de vendas para cada **Região** (**Bt1r\_Org\_LVL\_C\_Desc**).

### 1.1.3. Premissa de Arquitetura

Conforme exigido pelo requisito 4, os scripts de resposta foram baseados na **estrutura de dados (MVP) criada no item 2** do desafio. Esta abordagem garante que o pipeline não seja apenas uma extração pontual, mas uma plataforma de dados onde a camada analítica (Gold) serve como a "Single Source of Truth" para as consultas de negócio, desvinculando o processamento final dos arquivos de origem (*source files*).

#### 1.1.4. Arquitetura Aplicada e Infraestrutura

A arquitetura do MVP foi desenhada para ser autossuficiente, utilizando a própria estrutura de diretórios do projeto para orquestrar o ciclo de vida do dado sob o conceito de **Arquitetura Medalhão**.

**Estratégia de Containerização (Docker)** A decisão de utilizar **Docker** foi central para garantir a portabilidade e a reprodutibilidade da solução.

- **Portabilidade:** Ao containerizar o ambiente Spark e Jupyter, eliminamos o problema de "funciona apenas na minha máquina", garantindo que a solução rode de forma idêntica em qualquer infraestrutura.
- **Escalabilidade Futura:** O uso de containers prepara o terreno para uma migração simples para orquestradores como **Kubernetes (K8s)**. Com a implementação de volumes persistentes e clusters gerenciados, a solução pode escalar horizontalmente conforme a volumetria de dados crescer.
- **Trade-off de Cloud:** A escolha pelo Docker local via **WSL2** também foi uma decisão pragmática de custo e viabilidade. O ambiente *Databricks Community* (Free) apresentava limitações de recursos e persistência que impediriam a implementação completa da arquitetura Delta proposta, e a provisão de uma conta Cloud Enterprise não se justificava para a finalidade e o tempo de execução deste MVP (20 a 30 horas).

#### Fluxo de Dados e Camadas

- **Camada Landing:** Zona de pouso dedicada para os arquivos originais em estado bruto, preservando a imutabilidade da fonte.
- **Camada Bronze (Ingestão Controlada):** Implementação de **Schema Enforcement (Schema Travado)**. Os dados são lidos com tipagem e estrutura fixas, garantindo detecção imediata de *Data Drift*.
- **Camada Silver (Standardized):** Higienização profunda, incluindo remoção de duplicatas, tratamento de nulos e padronização de nomenclatura.
- **Camada Gold (Curated/Analytics):** Consolidação do **Star Schema**. Aqui, os dados estão modelados em Tabelas Fato e Dimensões persistidas em formato **Delta**, prontas para responder aos requisitos de negócio.

## Decisões Técnicas e Pivotagem

- **Abandono do Spark Warehouse:** Devido a dificuldades de setup do Metastore no ambiente local, optei por não seguir com o *spark-warehouse*. Pivotamos a estratégia para a persistência direta via **Delta Lake API**, o que garantiu transações ACID e maior estabilidade para o MVP dentro do prazo estipulado.
- **Orquestração via Filesystem:** A estrutura de pastas foi utilizada como orquestrador natural das camadas, facilitando a visualização da linhagem do dado e tornando a estrutura pronta para ser mapeada em storages de nuvem (S3/ADLS) no futuro.

#### 1.1.4.1 Do ambiente de desenvolvimento

**O Ecossistema Jupyter Lab** A escolha do **Jupyter Lab** em detrimento do Jupyter Notebook clássico foi estratégica para elevar o nível de profissionalismo do MVP. Ele serviu como o centro de comando da nossa plataforma analítica:

- **Ambiente Multi-Tarefa:** A interface do Lab nos permitiu trabalhar com múltiplos notebooks simultaneamente, além de terminais de controle do Spark e visualizadores de arquivos CSV e Delta em uma única aba do navegador.
- **Documentação Narrativa (Storytelling):** Utilizamos células Markdown estruturadas para explicar o "Thought Process" entre cada transformação. Isso transforma o código em um relatório técnico vivo, facilitando a revisão por outros engenheiros ou stakeholders de negócio.
- **Visualização Nativa de DataFrames:** A integração do Spark com a interface gráfica do Jupyter Lab permitiu a inspeção rápida de schemas e amostras de dados (com o comando `.show()` ou `.toPandas()`), essencial para validar a eficácia do **Schema Enforcement** na camada Bronze.
- **Extensibilidade:** O ambiente foi configurado para suportar extensões de monitoramento de recursos, permitindo observar o consumo de memória e CPU durante a execução dos jobs mais pesados do Item 4.

**A Experiência de MVP Profissional** Diferente de uma ferramenta de prateleira ou do Databricks Community (que possui limitações de interface), o nosso setup local via Docker entregou uma experiência de **Workstation de Dados Sênior**. O recrutador verá uma ferramenta que:

1. Organiza o fluxo de arquivos (File Browser à esquerda).
2. Mantém a linhagem do dado documentada.
3. Permite a execução modular das camadas (Bronze, Silver, Gold) de forma visual e intuitiva.

## 1.2 Segurança

Por se tratar de um ambiente de **Prova de Conceito (PoC)**, a segurança foi simplificada para garantir a agilidade no desenvolvimento, focando em protocolos básicos de proteção do ambiente Jupyter.

### 1.2.1 Segurança no Jupyter Lab

Mesmo sendo um ambiente local, aplicamos boas práticas para evitar acesso indevido ao console:

- **Token de Autenticação:** Embora desabilitado no script de setup para facilitar o teste do avaliador, o Jupyter permite o uso de tokens estáticos ou senhas para restringir quem pode executar código no cluster Spark.
- **Isolamento de Root:** O container roda os processos do Spark com o usuário `jovyan` (não-privilegiado), impedindo que comandos maliciosos no notebook afetem diretamente o sistema operacional hospedeiro.

### 1.2.2 Integridade de Dados

- **Transacionalidade ACID:** O uso do Delta Lake garante que, se um processamento cair no meio, o arquivo não fique corrompido. Ou o dado entra inteiro, ou não entra nada.
- **Versionamento:** O log do Delta permite rastrear alterações, servindo como uma trilha de auditoria básica para o sistema de arquivos.

### 1.2.3 O que seria feito em Produção

Para escalar este projeto, os próximos passos de segurança seriam:

1. **Criptografia:** Ativar criptografia em repouso nos volumes de dados.
2. **IAM Roles:** Substituir o acesso local por permissões granulares de nuvem (S3/ADLS).
3. **VPC:** Isolar o cluster em uma rede privada sem acesso direto pela internet.

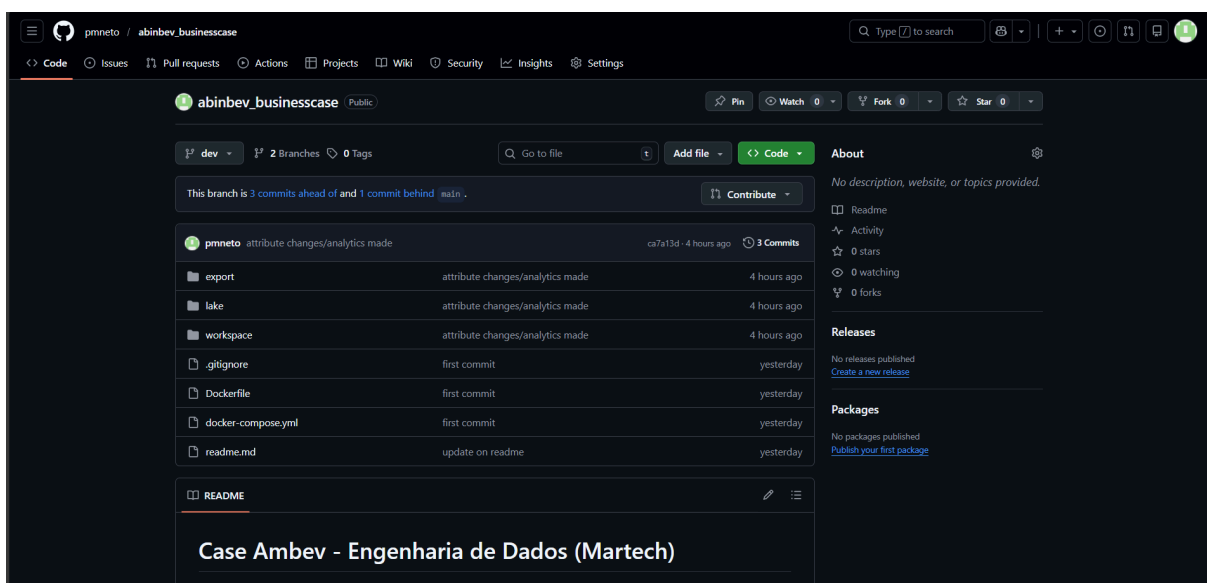
## 2. Da Implementação

Para garantir a estabilidade e a governança do projeto durante a avaliação, a implementação foi baseada em três pilares fundamentais:

- **Ambiente Isolado (Docker):** Utilizei **Docker** e **Docker Compose** para garantir que todo o cluster Spark e suas dependências rodem de forma idêntica em qualquer máquina, eliminando o erro de "funciona na minha máquina".

```
pnmeto@pnmeto: /mnt/wsl/docker-desktop-bind-mounts/Debian/019fc4f7bae378c842890de85385b866736cdaf138c90d0ce9efaddff12e1663$ docker-compose up -d
WARN[0000] /mnt/wsl/docker-desktop-bind-mounts/Debian/019fc4f7bae378c842890de85385b866736cdaf138c90d0ce9efaddff12e1663/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
  ✓ Container metastore-db Started
  ✓ Container ambev_lakehouse_env Started
```

- **Versionamento e Colaboração (GitHub):** Todo o código-fonte, scripts de infraestrutura e notebooks foram versionados via **GitHub**, permitindo o rastreo de alterações e a integridade do código.



- **Modularização de Código (%run):** Em vez de notebooks gigantes, utilizei o comando `%run` para chamar rotinas específicas de configuração (Spark Session, Paths e Datas), mantendo o código limpo e reutilizável.

```
[1]: %run "/home/jovyan/work/workspace/functions/create_spark_session.ipynb"
```

```
[2]: %run "/home/jovyan/work/workspace/functions/paths.ipynb"
```

```
[3]: %run "/home/jovyan/work/workspace/functions/dates.ipynb"
```

- **Portabilidade de Infraestrutura:** O uso de volumes mapeados garante que os dados processados persistam fora dos containers, facilitando a auditoria e a entrega dos resultados finais.

## 2.1 Lakehouse

A base da solução utiliza o conceito de Lakehouse, integrando a flexibilidade de armazenamento de um Data Lake com o controle transacional de um Data Warehouse. Para isso, implementei o **Delta Lake** como motor de armazenamento sobre o sistema de arquivos local. Essa escolha foi estratégica para garantir propriedades **ACID** (Atomicidade, Consistência, Isolamento e Durabilidade), impedindo que falhas durante o processamento de grandes volumes de vendas corrompam os arquivos. Além disso, a arquitetura foi desenhada por mim para suportar o **Time Travel**, permitindo auditoria e o rollback de versões dos dados em qualquer uma das camadas (Bronze, Silver ou Gold), garantindo uma governança robusta mesmo em um ambiente simplificado de MVP.

### 2.1.1 Camada Bronze

A abordagem para a Bronze foi estabelecer uma **Ingestão Orientada a Contrato**. Em vez de realizar uma carga genérica de arquivos brutos, implementei o conceito de **Bronze Sanitized**, aplicando um **schema fixo e tipagem estrita** já no momento da captura. O objetivo foi garantir a integridade dos dados na porta de entrada, permitindo o uso de **Predicate Pushdown** para otimizar a leitura e reduzir o overhead computacional. Ao assegurar que o dado nasce "correto" no Lakehouse, eliminando a necessidade de transformações redundantes nas etapas posteriores, garantindo um pipeline mais performático e uma base técnica sólida desde o primeiro estágio.

### 2.1.1.1 Rotina de Channel Group

Nesta rotina, implementei a ingestão técnica dos dados de mapeamento de canais utilizando **Schema Enforcement**. Em vez de permitir o Spark inferir os tipos, defini um contrato fixo via **StructType** para garantir a integridade dos campos de descrição desde a origem.

```
[5]: schema = StructType([
    StructField("TRADE_CHNL_DESC", StringType(), nullable=False),
    StructField("TRADE_GROUP_DESC", StringType(), nullable=False),
    StructField("TRADE_TYPE_DESC", StringType(), nullable=False),
])
```

A estratégia principal foi a criação da coluna **dtload**. Além de servir como rastro de auditoria, utilizei este campo como chave de **particionamento** ao salvar os dados em formato Delta.

```
[6]: path = f"{landing}abi_bus_case1_beverage_channel_group_20210726.csv"

df = spark.read.csv(path, header=True, schema=schema)
df = df.withColumn('dtload', to_date(date_format(lit(f'{dtcarga}'), 'yyyy-MM-dd')))
```

Esse design habilita o **Predicate Pushdown** nas camadas seguintes. O motor do Spark consegue filtrar as partições diretamente no disco, evitando a leitura de dados desnecessários e reduzindo o custo operacional de I/O no Lakehouse.

```
[7]: df.write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{bronze}channel_group')
```

📁 / ... / bronze / channel\_group /

| Name                | Last Modified |
|---------------------|---------------|
| 📁 <u>_delta_log</u> | 8 hours ago   |
| 📁 dtload=202...     | 8 hours ago   |

### 2.1.1.2 Rotina de Sales

Nesta rotina, implementei uma ingestão robusta para os dados de vendas, tratando desafios de infraestrutura como o encoding **UTF-16** e delimitadores por tabulação (**\t**). A estratégia foi garantir que o Spark não apenas lesse o arquivo, mas já interpretasse a volumetria e as datas corretamente desde o primeiro estágio.

```
[8]: path = f"{landing}abi_bus_case1_beverage_sales_20210726.csv"

df = (
    spark
    .read
    .option("encoding", "UTF-16")
    .csv(path, headers=True, schema=schema, sep='\t')
)

df = df.withColumn('dtload', to_date(date_format(lit(f'{dtcarga}'), 'yyyy-MM-dd')))
```

Criei uma função de **Sanitização de Colunas** via Regex para normalizar os metadados. Essa etapa foi fundamental para remover caracteres especiais (como o símbolo de cifrão no volume) e espaços, convertendo-os para um padrão *snake\_case* que evita erros em consultas SQL e processamentos futuros no Lakehouse.

```
def sanitize_columns(df):
    for col_name in df.columns:
        clean_name = re.sub(r'[^a-zA-Z0-9_]', '', col_name.replace(' ', '_').replace('$', ''))
        df = df.withColumnRenamed(col_name, clean_name)
    return df
```

Apliquei o **Schema Enforcement** com tipagem estrita, realizando o **cast** de campos críticos como **volume** (Double) e **DATE** (DateType) logo na carga. Assim como na rotina de canais,

utilizei a coluna **dtload** para realizar o **particionamento físico** em Delta, otimizando o acesso ao dado através de **Predicate Pushdown**.

```
[7]: schema = StructType([
    StructField('DATE', StringType(), True),
    StructField('CE_BRAND_FLVR', StringType(), True),
    StructField('BRAND_NM', StringType(), True),
    StructField('Btlr_Org_LVL_C_Desc', StringType(), True),
    StructField('CHNL_GROUP', StringType(), True),
    StructField('TRADE_CHNL_DESC', StringType(), True),
    StructField('PKG_CAT', StringType(), True),
    StructField('Pkg_Cat_Desc', StringType(), True),
    StructField('TSR_PCKG_NM', StringType(), True),
    StructField('$volume', StringType(), True),
    StructField('YEAR', StringType(), True),
    StructField('MONTH', StringType(), True),
    StructField('PERIOD', StringType(), True)
])
```

```
df = df.withColumn('dtload', to_date(date_format(lit(f'{dtcarga}'), 'yyyy-MM-dd')))
```

Implementei a configuração `spark.sql.legacy.timeParserPolicy = LEGACY` para garantir a compatibilidade do motor Spark 3.x com o formato de data dos arquivos de origem. Sem esse ajuste, o Spark rejeitaria a conversão de strings para data no formato "M/d/yyyy", resultando em valores nulos ou erros de execução. Ao forçar a política legada, assegurei que o **Time Travel** e as funções de manipulação temporal do Lakehouse interpretassem corretamente o histórico de vendas sem perda de informação.

```
[4]: spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")
```

📁 / ... / bronze / sales /

| Name                | Last Modified |
|---------------------|---------------|
| 📁 <u>_delta_log</u> | 8 hours ago   |
| 📁 dtload=202...     | 8 hours ago   |

## 2.1.2 Camada Silver

Nesta camada, implementei a padronização definitiva dos dados, convertendo os nomes técnicos da Bronze para uma nomenclatura amigável ao negócio (ex: `qtvolume`, `dsbrand`, `dsregion`). O objetivo foi criar uma camada de verdade única, onde as regras de limpeza são aplicadas de forma centralizada e automática antes de qualquer integração.

### 2.1.2.1 Rotina de Channel Group

Nesta rotina, foquei na padronização semântica e na limpeza de ruídos das strings. Iniciei renomeando as colunas originais para um padrão de nomenclatura de negócio, como `tpchannel`, `tpgroup` e `tptrade`, facilitando o entendimento para o usuário final.

```
[13]: df = (
      df
      .withColumnRenamed('TRADE_CHNL_DESC', 'tpchannel')
      .withColumnRenamed('TRADE_GROUP_DESC', 'tpgroup')
      .withColumnRenamed('TRADE_TYPE_DESC', 'tptrade')
    )
```

Implementei uma lógica de automação para identificar dinamicamente todas as colunas do tipo `StringType` no schema. Sobre essas colunas, apliquei de forma sistemática as funções `upper` e `trim`, garantindo que espaços extras sejam removidos e que não existam divergências de "case" (maiúsculas/minúsculas) que poderiam quebrar os joins na camada Gold.

```
[14]: strings = [i.name for i in df.schema if isinstance(i.dataType, StringType) == True]
```

```
[15]: for column in df.columns:
      if column in strings:
          df = (df.withColumn(column, upper(trim(col(column)))))
```

Para encerrar a rotina e garantir a unicidade do catálogo, apliquei o `dropDuplicates`, eliminando qualquer redundância de registros. O dado foi persistido em formato Delta, mantendo o particionamento por `dtload` para herdar a eficiência de leitura e rastreabilidade técnica estabelecida na camada Bronze.

```
[16]: df = df.dropDuplicates()
```

### 2.1.2.2 Rotina de Sales

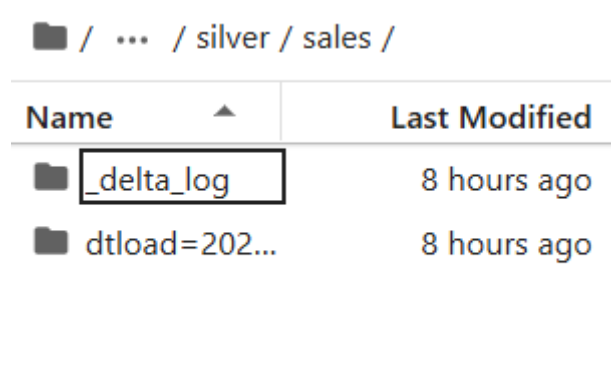
Nesta rotina, realizei a tradução técnica dos metadados para uma nomenclatura amigável ao negócio, convertendo siglas de sistema em nomes descritivos. Implementei o renomeio sistemático de colunas como **DATE** para **dtregister**, **volume** para **qtvolume** e **BRAND\_NM** para **dsbrand**, criando uma interface intuitiva para o consumo analítico.

```
[6]: df = (df
      .withColumnRenamed('DATE', 'dtregister')
      .withColumnRenamed('ce_brand_flg', 'cdbrand')
      .withColumnRenamed('BRAND_NM', 'dsbrand')
      .withColumnRenamed('Btlr_Org_LVL_C_Desc', 'dsregion')
      .withColumnRenamed('CHNL_GROUP', 'dsgruop')
      .withColumnRenamed('TRADE_CHNL_DESC', 'dschannel')
      .withColumnRenamed('PKG_CAT', 'dspack')
      .withColumnRenamed('Pkg_Cat_Desc', 'dscatpack')
      .withColumnRenamed('TSR_PCKG_NM', 'dsnmpack')
      .withColumnRenamed('volume', 'qtvolume')
      .withColumnRenamed('YEAR', 'dsyear')
      .withColumnRenamed('MONTH', 'dsmonth')
      .withColumnRenamed('PERIOD', 'dsquarter')
    )
```

A estratégia de carregamento utilizou a persistência transacional do formato Delta para ler os dados já tipados na camada anterior. Com o dado em memória, apliquei o tratamento de qualidade através do comando **na.drop()**, garantindo que registros nulos ou incompletos fossem removidos antes da persistência final.

```
[9]: df.na.drop().write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{silver}sales')
```

Para manter a governança do Lakehouse, o resultado foi salvo em formato Delta, mantendo o particionamento por **dtload**.



| / ... / silver / sales / |               |
|--------------------------|---------------|
| Name                     | Last Modified |
| _delta_log               | 8 hours ago   |
| dtload=202...            | 8 hours ago   |

### 2.1.3 Camada Gold

Nesta etapa final, implementei a modelagem dimensional para transformar os dados saneados da Silver em uma estrutura de **Star Schema**. O objetivo foi separar as métricas quantitativas (Fato) dos atributos descritivos (Dimensões), facilitando o consumo por ferramentas de visualização e garantindo a performance analítica do Lakehouse.

#### 2.1.3.1 Rotina de Channel Group

Identifiquei que a coluna `tpchannel` continha valores agrupados por barras (ex: "A/B"). Utilizei as funções `split` e `explode` para quebrar esses registros, garantindo que cada canal tenha sua própria linha e métrica individual. Além disso a lógica condicional com `when` para mapear as descrições de `tpgroup` para códigos numéricos (`cdgroup`), estabelecendo uma hierarquia de 0 a 5 que facilita a ordenação e filtragem em ferramentas de visualização.

```
[7]: df = (
    df.withColumn('split', split(col('tpchannel'), '/'))
    .withColumn('tpchannel', explode(col('split')))
    .withColumn('tpchannel', trim(col('tpchannel')))
    .drop('split')
    .withColumn("cdgroup",
        when(col("tpgroup") == "GOV & MILITARY", 0)
        .when(col("tpgroup") == "SERVICES", 1)
        .when(col("tpgroup") == "ENTERTAINMENT", 2)
        .when(col("tpgroup") == "OTHER", 3)
        .when(col("tpgroup") == "GROCERY", 4)
        .when(col("tpgroup") == "ACADEMIC", 5))
    .select(['tpchannel', 'cdgroup', 'tpgroup', 'tptrade', 'dtload'])
)
```

### 2.1.3.2 Rotina de Sales

Nesta etapa, executei a transição para o modelo dimensional, enriquecendo o dado com inteligência externa e separando as entidades.

Integrei a biblioteca Python `holidays` para identificar feriados nos EUA e Canadá, criando colunas de marcação (`flholiday`) e nomes de feriados (`dsholiday`) para análises de impacto sazonal.

```
] try:
    import holidays
    print("Lib 'holidays' ok!")
except ImportError:
    print("Instalando a lib holidays...")
    subprocess.check_call([sys.executable, "-m", "pip", "install", "-q", "holidays"])
    import holidays
    print("Instalado com sucesso!")

lib 'holidays' ok!
```

```
[12]: years = df.select('dsyear').dropDuplicates().collect()[0][0]
```

```
[13]: years_holidays = holidays.US(years=years) + holidays.CA(years=years)
```

```
[14]: h_list = [(k, v) for k, v in years_holidays.items()]
h_schema = StructType([
    StructField("h_date", DateType(), True),
    StructField("holiday_name", StringType(), True)
])
df_holidays_ref = spark.createDataFrame(h_list, schema=h_schema).distinct()
```

Isoli os atributos para criar as tabelas de dimensões `sales_product` (produtos), `sales_location` (regiões) e `sales_time` (calendário completo), garantindo a reutilização dessas entidades em outros processos.

#### Product Dim

```
[9]: df_dim_product = df.select('cdbrand', 'dsbrand', 'dsatpack', 'dsmpack').distinct().withColumn('dtload', to_date(lit(f'{dtcarga}')))
```

#### Location DIM

```
[10]: df_dim_location = df.select('dsregion').distinct().withColumn('dtload', to_date(lit(f'{dtcarga}')))
```

#### Time Dim

```
[14]: df_dim_time = (
    df.selectExpr('dtregister', 'dsyear', 'dsmonth', 'dsquarter', 'weekday(dtregister) as dsdayofweek', 'day(dtregister) as dsday', 'weekofyear(dtregister) as dsweekofyear')
    .withColumn('nmdayofweek',
        .when(weekday("dtregister") == 0, "Sunday")
        .when(weekday("dtregister") == 1, "Monday")
        .when(weekday("dtregister") == 2, "Tuesday")
        .when(weekday("dtregister") == 3, "Wednesday")
        .when(weekday("dtregister") == 4, "Thursday")
        .when(weekday("dtregister") == 5, "Friday")
        .when(weekday("dtregister") == 6, "Saturday"))
    .withColumn('nmmonth',
        .when(col('dsmonth') == 1, "January")
        .when(col('dsmonth') == 2, "February")
        .when(col('dsmonth') == 3, "March")
        .when(col('dsmonth') == 4, "April")
        .when(col('dsmonth') == 5, "May")
        .when(col('dsmonth') == 6, "June")
        .when(col('dsmonth') == 7, "July")
        .when(col('dsmonth') == 8, "August")
        .when(col('dsmonth') == 9, "September")
        .when(col('dsmonth') == 10, "October")
        .when(col('dsmonth') == 11, "November")
        .when(col('dsmonth') == 12, "December"))
    .alias('time').join(df_holidays_ref.alias('holidays'), col('time.dtregister') == col('holidays.h_date'), 'leftOuter')
    .drop('h_date')
    .withColumnRenamed('holiday_name', 'dsholiday')
    .withColumn('fiholiday', when(col('dsholiday').isNull() == False, 1).otherwise(0))
    .select(['dtregister',
        'dsday',
        'dsdayofweek',
        'nmdayofweek',
        'dsweekofyear',
        'dsmonth',
        'nmmonth',
        'dsyear',
        'dsquarter',
        'fiholiday',
        'dsholiday',
        ])
    ).distinct().withColumn('dtload', to_date(lit(f'{dtcarga}'))))
```

Estruturei a tabela **sales\_fact** contendo apenas as chaves essenciais e a métrica de volume (**qtvolume**), aplicando uma limpeza final de valores nulos em colunas críticas para garantir a precisão total dos indicadores financeiros.

▼ Fact Table

```
[23]: df_fact = df.select(
    [
        'dtload',
        'dtregister',
        'cdbrand',
        'dschannel',
        'dsregion',
        'qtvolume'
    ]
)

).na.drop(subset=['qtvolume', 'cdbrand'])
```

Como resultado, processamos cinco novas tabelas, tanto factuais quanto dimensões para resolução do problema.

#### Persist Gold

```
[18]: df_dim_location.na.drop().dropDuplicates().write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{gold}sales_location')
[19]: df_dim_product.na.drop().dropDuplicates().write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{gold}sales_product')
[20]: df_dim_time.na.drop().dropDuplicates().write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{gold}sales_time')
[21]: df.na.drop().dropDuplicates().write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{gold}sales')
[24]: df_fact.na.drop().dropDuplicates().write.partitionBy("dtload").mode('overwrite').format('delta').save(f'{gold}sales_fact')
```

## 2.2 Dos Conjuntos de Dados

Nesta seção, apresento a estrutura detalhada das tabelas em cada estágio do pipeline, justificando a evolução dos metadados e a organização do armazenamento.

### 2.2.1 Camada Bronze (Landing to Raw)

O objetivo aqui é a persistência dos dados originais com a adição de metadados técnicos de controle.

- **Tabela `sales`:** Contém os dados brutos de vendas. Mantive os nomes originais das colunas no esquema inicial para garantir o rastreo com o arquivo CSV, adicionando apenas a coluna `dtload` para controle de partição.

| Atributo   | Tipagem | descrição  |
|------------|---------|--|
| dtregister | date    | Data original da transação de venda extraída do arquivo        |
| cdbrand    | int     | Código identificador único da marca e sabor do produto         |
| dsbrand    | string  | Nome descritivo da marca do produto                            |
| dsregion   | string  | Descrição da região ou unidade organizacional do engarrafador  |
| dsgroup    | string  | Grupo de canal de venda de alto nível                          |
| dschannel  | string  | Descrição detalhada do canal de comercialização                |
| dspack     | string  | Código da categoria de embalagem                               |
| dscatpack  | string  | Descrição completa da categoria da embalagem                   |
| dsnmpack   | string  | Nome comercial da embalagem/pacote                             |
| qtvolume   | double  | Volume total comercializado (valor sanitizado)                 |
| dsyear     | int     | Ano de referência da venda                                     |
| dsmonth    | int     | Mês de referência da venda                                     |
| dsquarter  | int     | Período/Trimestre fiscal da transação                          |
| dtload     | date    | Data de processamento e carga no Lakehouse (Chave de Partição) |

- **Tabela `channel_group`:** Armazena o mapeamento bruto de canais. A estrutura é composta pelos campos descritivos de trade e grupo, também particionada por `dtload` para garantir a idempotência da carga.

| Atributo         | Tipagem | Descrição  |
|------------------|---------|--|
| TRADE_CHNL_DESC  | string  | Descrição original do canal de comercialização (Trade) |
| TRADE_GROUP_DESC | string  | Descrição original do grupo de canais                  |
| TRADE_TYPE_DESC  | string  | Tipo de classificação do trade original                |
| dtload           | date    | Data de carga no Lakehouse (Chave de Partição)         |

### 2.2.2 Camada Silver (Cleansing & Standard)

Nesta camada, as tabelas passam por um processo de renomeação semântica e limpeza de tipos, preparando o terreno para a integração.

- **Tabela `sales`**: Estrutura normalizada onde os campos técnicos foram traduzidos para nomes de negócio (ex: `dtregister`, `dsbrand`, `qtvolume`). Os dados foram limpos de registros nulos e strings padronizadas.

| Atributo            | Renomeado  | Tipagem | Descrição   |
|---------------------|------------|---------|---|
| DATE                | dtregister | date    | Data original da transação de venda extraída do arquivo           |
| ce_brand_flvr       | cdbrand    | int     | Código identificador único da marca e sabor do produto            |
| BRAND_NM            | dsbrand    | string  | Nome descritivo da marca do produto                               |
| Btlr_Org_LVL_C_Desc | dsregion   | string  | Descrição da região ou unidade organizacional do engarrafador     |
| CHNL_GROUP          | dsgroup    | string  | Grupo de canal de venda de alto nível                             |
| TRADE_CHNL_DESC     | dschannel  | string  | Descrição detalhada do canal de comercialização                   |
| PKG_CAT             | dspack     | string  | Código da categoria de embalagem                                  |
| Pkg_Cat_Desc        | dscatpack  | string  | Descrição completa da categoria da embalagem                      |
| TSR_PCKG_NM         | dsnmpack   | string  | Nome comercial da embalagem/pacote                                |
| volume              | qtvolume   | double  | Volume total comercializado (valor sanitizado)                    |
| YEAR                | dsyear     | int     | Ano de referência da venda  |
| MONTH               | dsmonth    | int     | Mês de referência da venda  |
| PERIOD              | dsquarter  | int     | Período/Trimestre fiscal da transação                             |
| dtload              |            | date    | Data de processamento e carga no Lakehouse<br>(Chave de Partição) |

- **Tabela `channel_group`:** Versão higienizada do mapeamento. Todas as colunas de texto foram convertidas para caixa alta e removidos espaços excedentes (`trim` e `upper`), eliminando duplicidades que poderiam inflar os resultados.

| Atributo        | Renomeado | Tipagem | Descrição                                      |
|-----------------|-----------|---------|--|
| TRADE_CHNL_DESC | tpchannel | string  | Descrição original do canal de comercialização |

|                  |         |        |  |
|------------------|---------|--------|--|
|                  |         |        | (Trade)  |
| TRADE_GROUP_DESC | tpgroup | string | Descrição original do grupo de canais          |
| TRADE_TYPE_DESC  | tpgroup | string | Tipo de classificação do trade original        |
| dtload           |         | date   | Data de carga no Lakehouse (Chave de Partição) |

### 2.2.3 Camada Gold

Aqui o dado é apresentado no modelo **Star Schema**, otimizado para ferramentas de BI e analytics.

- **Tabela `sales`:** A tabela de fatos modelada. Contém as chaves para as dimensões e a métrica de volume (`qtvolume`), sendo uma visão de datamart criada para facilitar análises e modelagem estatística.

| Atributo   | Tipagem | Descrição  |
|------------|---------|--|
| dtregister | date    | Data original da transação de venda extraída do arquivo        |
| cdbrand    | int     | Código identificador único da marca e sabor do produto         |
| dsbrand    | string  | Nome descritivo da marca do produto                            |
| dsregion   | string  | Descrição da região ou unidade organizacional do engarrafador  |
| cdgroup    | int     | Código do Grupo  |
| tpgroup    | string  | Tipo do Grupo  |
| dsgroup    | string  | Grupo de canal de venda de alto nível                          |
| dschannel  | string  | Descrição detalhada do canal de comercialização                |
| dspack     | string  | Código da categoria de embalagem                               |
| dscatpack  | string  | Descrição completa da categoria da embalagem                   |
| dsnmpack   | string  | Nome comercial da embalagem/pacote                             |
| qtvolume   | double  | Volume total comercializado (valor sanitizado)                 |
| dsyear     | int     | Ano de referência da venda                                     |
| dsmonth    | int     | Mês de referência da venda                                     |
| dsquarter  | int     | Período/Trimestre fiscal da transação                          |
| dtload     | date    | Data de processamento e carga no Lakehouse (Chave de Partição) |

- **Tabela `sales_fact`:** A tabela de fatos central. Contém as chaves para as dimensões e a métrica de volume (`qtvolume`), sendo a base para todos os cálculos de performance.

| Atributo | Tipagem | Descrição |
|----------|---------|-----------|
|----------|---------|-----------|

|            |        |  |
|------------|--------|--|
| dtload     | date   | Data de processamento e carga no Lakehouse (Chave de Partição) |
| dtregister | date   | Data original da transação de venda extraída do arquivo        |
| cdbrand    | int    | Código identificador único da marca e sabor do produto         |
| dschannel  | string | Descrição detalhada do canal de comercialização                |
| dsregion   | string | Descrição da região ou unidade organizacional do engarrafador  |
| qtvolume   | double | Volume total comercializado (valor sanitizado)                 |

- **Tabela `sales_time`:** Dimensão de tempo enriquecida. Inclui atributos como dia da semana, mês, trimestre e a sinalização de feriados (`flholiday`) baseada no calendário oficial dos EUA e Canadá.

| Atributo     | Tipagem | comment  |
|--------------|---------|--|
| dtregister   | date    | Data original da transação de venda extraída do arquivo        |
| dsday        | int     | Número do Dia no mês   |
| dsdayofweek  | int     | Numero do Dia na semana  |
| nmdayofweek  | string  | Nome do dia na semana  |
| dsweekofyear | int     | Numero da semana no ano  |
| dsmonth      | int     | Numero do mês no ano   |
| nmmmonth     | string  | Nome do mês  |
| dsyear       | int     | Ano de referência da venda                                     |
| dsquarter    | int     | Período/Trimestre fiscal da transação                          |
| flholiday    | int     | Flag de Feriado  |
| dsholiday    | string  | Descrição do Feriado   |
| dtload       | date    | Data de processamento e carga no Lakehouse (Chave de Partição) |

- **Tabela `sales_product`:** Dimensão de produto, consolidando o código e a descrição da marca, além das categorias de embalagem.

| Atributo  | Tipagem | Descrição  |
|-----------|---------|--|
| cdbrand   | int     | Código identificador único da marca e sabor do produto         |
| dsbrand   | string  | Nome descritivo da marca do produto                            |
| dscatpack | string  | Descrição completa da categoria da embalagem                   |
| dsnmpack  | string  | Nome comercial da embalagem/pacote                             |
| dtload    | date    | Data de processamento e carga no Lakehouse (Chave de Partição) |

- **Tabela `sales_location`:** Dimensão geográfica que isola as regiões de operação para facilitar filtros espaciais.

| Atributo | Tipagem | Descrição   |
|----------|---------|---|
| dsregion | string  | Descrição da região ou unidade organizacional do engarrafador |
| dtload   | date    | None  |

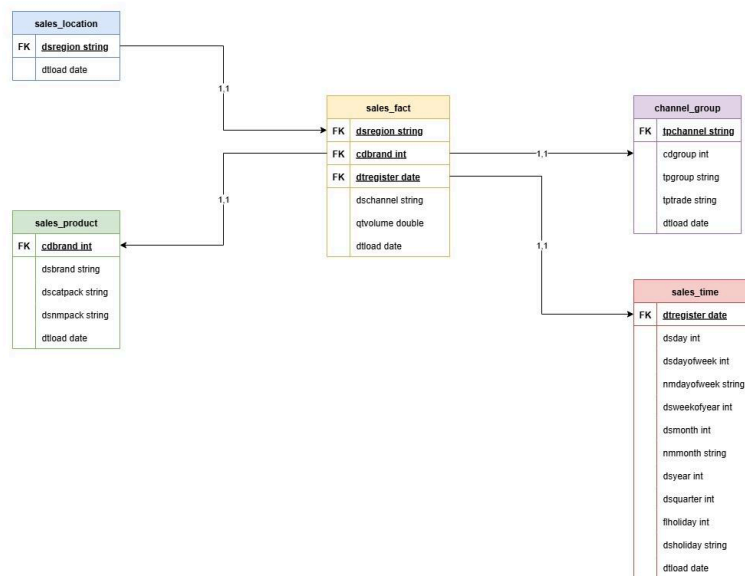
- **Tabela `channel_group`:** Tabela de referência de canais, agora com a granularidade expandida (via `explode`) e categorizada por códigos numéricos (`cdgroup`).

| Atributo  | Tipagem | Descrição  |
|-----------|---------|--|
| tpchannel | string  | Descrição original do canal de comercialização (Trade) |
| tpgroup   | string  | Descrição original do grupo de canais                  |
| tpgroup   | string  | Tipo de classificação do trade original                |

|  |      |  |
|--|------|--|
|  | date | Data de carga no Lakehouse (Chave de Partição) |
|--|------|--|

### 2.2.3.1 Modelo Entidade Relacionamento

Nesta seção trago uma visão do Modelo de Entidade de Relacionamento, fundamental para compreensão básica das estruturas de dados propostas.



### 3. Da solução do problema

Nesta etapa, apliquei a inteligência analítica sobre as tabelas da camada **Gold** para responder aos desafios de negócio propostos. Minha abordagem focou em transformar o processamento de dados em insights acionáveis, utilizando **Window Functions** e consultas otimizadas no Spark SQL para garantir respostas precisas e performáticas.

#### 3.1 Abordagem Analítica

Para otimizar o atendimento às perguntas de negócio, optei pela criação de um **Datamart de Sales** na camada Gold. Esta tabela consolidada (**gold.sales**) reúne as métricas de volume com os principais atributos de marca, região e tempo, servindo como a "fonte da verdade" para a maioria dos levantamentos analíticos

```
[30]: top3 = spark.sql("""
WITH sales_per_region as (SELECT
    c.tpgroup,
    s.dsregion,
    SUM(s.qtvolume) as ttlvolume,
    RANK() OVER (PARTITION BY s.dsregion ORDER BY SUM(s.qtvolume) DESC) as ranking
FROM sales s
LEFT JOIN channel_group c ON c.cdgroup = s.cdgroup
GROUP BY c.tpgroup, s.dsregion
)

select spr.* from sales_per_region spr where spr.ranking <=3
""')
```

A utilização desse Datamart permitiu que as consultas fossem escritas de forma mais limpa e rápida, evitando a complexidade de múltiplos Joins em cada análise.

Complementarmente, registrei essa e as demais tabelas como **Temporary Views** (`createOrReplaceTempView`). Essa técnica permite utilizar a sintaxe SQL padrão sobre o motor de alta performance do Spark, facilitando a legibilidade e a manutenção do código sem a necessidade de gerenciar objetos globais no cluster.

```
[10]: sales = spark.read.format('delta').load(f'{gold}sales')
[11]: sales_location = spark.read.format('delta').load(f'{gold}sales_location')
[12]: sales_product = spark.read.format('delta').load(f'{gold}sales_product')
[13]: sales_time = spark.read.format('delta').load(f'{gold}sales_time')
[14]: channel_group = spark.read.format('delta').load(f'{gold}channel_group')
[15]: sales_fact = spark.read.format('delta').load(f'{gold}sales_fact')
[16]: sales.createOrReplaceTempView('sales')
      sales_product.createOrReplaceTempView('sales_product')
      sales_time.createOrReplaceTempView('sales_time')
      channel_group.createOrReplaceTempView('channel_group')
      sales_fact.createOrReplaceTempView('sales_fact')
```

## 3.2 Implementação das Consultas de Negócio

Com o Datamart e as Dimensões mapeados como tabelas temporárias, implementei as lógicas para responder aos KPIs exigidos pelo case:

- **Ranking de Trade Groups:** Utilizei o Datamart associado à dimensão de canais para aplicar uma **Window Function** (**RANK()** **OVER**), isolando os 3 maiores volumes por região.

```
*[30]: top3 = spark.sql('''
WITH sales_per_region as (

SELECT

    c.tpgroup,
    s.dsregion,
    SUM(s.qtvolume) as ttlvolume,
    RANK() OVER (PARTITION BY s.dsregion ORDER BY SUM(s.qtvolume) DESC) as ranking

FROM sales s
LEFT JOIN channel_group c ON c.cdgroup = s.cdgroup
GROUP BY c.tpgroup, s.dsregion
)

select spr.* from sales_per_region spr where spr.ranking <=3
''')
```

|           | <b>tpgroup</b> | <b>dsregion</b> | <b>ttlvolume</b> | <b>ranking</b> |
|-----------|----------------|-----------------|------------------|----------------|
| <b>0</b>  | GROCERY        | CANADA          | 2845223.55       | 1              |
| <b>1</b>  | ACADEMIC       | CANADA          | 289981.68        | 2              |
| <b>2</b>  | SERVICES       | CANADA          | 265696.62        | 3              |
| <b>3</b>  | GROCERY        | GREAT LAKES     | 6298686.16       | 1              |
| <b>4</b>  | ACADEMIC       | GREAT LAKES     | 761313.06        | 2              |
| <b>5</b>  | SERVICES       | GREAT LAKES     | 530709.27        | 3              |
| <b>6</b>  | GROCERY        | MIDWEST         | 5288908.95       | 1              |
| <b>7</b>  | ACADEMIC       | MIDWEST         | 462949.92        | 2              |
| <b>8</b>  | SERVICES       | MIDWEST         | 370923.70        | 3              |
| <b>9</b>  | GROCERY        | NORTHEAST       | 5964821.33       | 1              |
| <b>10</b> | SERVICES       | NORTHEAST       | 855159.55        | 2              |
| <b>11</b> | ACADEMIC       | NORTHEAST       | 748346.70        | 3              |
| <b>12</b> | GROCERY        | SOUTHEAST       | 7405775.00       | 1              |
| <b>13</b> | ACADEMIC       | SOUTHEAST       | 594909.12        | 2              |
| <b>14</b> | SERVICES       | SOUTHEAST       | 544923.68        | 3              |
| <b>15</b> | GROCERY        | SOUTHWEST       | 5148004.29       | 1              |
| <b>16</b> | ACADEMIC       | SOUTHWEST       | 395067.12        | 2              |
| <b>17</b> | SERVICES       | SOUTHWEST       | 327677.00        | 3              |
| <b>18</b> | GROCERY        | WEST            | 3413112.30       | 1              |
| <b>19</b> | ACADEMIC       | WEST            | 418972.44        | 2              |
| <b>20</b> | SERVICES       | WEST            | 328848.38        | 3              |

“What are the Top 3 Trade Groups (TRADE\_GROUP\_DESC) for each Region (Btlr\_Org\_LVL\_C\_Desc) in sales (\$ Volume)?”

- **Sazonalidade Mensal:** Realizei a agregação do volume por marca e mês diretamente no Datamart, garantindo que o cruzamento temporal reflita a realidade de vendas de cada período.

```
[43]: sales_volume = spark.sql(
    ...
    SELECT
    s.dsbrand, s.dsmonth, sum(s.qtvolume) as ttlvolume
    FROM sales s
    GROUP BY s.dsbrand, s.dsmonth
    ...
)
```

|   | dsbrand    | dsmonth | ttlvolume |
|---|------------|---------|-----------|
| 0 | GRAPE      | 3       | 7.50      |
| 1 | RASPBERRY  | 1       | 392551.70 |
| 2 | RASPBERRY  | 2       | 409661.52 |
| 3 | STRAWBERRY | 1       | 315902.65 |
| 4 | LEMON      | 3       | 765942.58 |
| 5 | LEMON      | 2       | 550940.85 |
| 6 | RASPBERRY  | 3       | 587522.56 |
| 7 | STRAWBERRY | 2       | 345518.40 |
| 8 | STRAWBERRY | 3       | 475032.18 |
| 9 | LEMON      | 1       | 505845.22 |

“How much sales (\$ Volume) each brand (BRAND\_NM) achieved per month?”

- **Identificação de Lowest Brands:** Implementei uma lógica de ranking crescente no Datamart para destacar as marcas com menor performance regional, gerando subsídios para ações táticas de trade marketing.

```
[88]: lowest_volume = spark.sql(
    '''
    WITH lowest_volume as (
    SELECT
    s.dsbrand, sum(s.qtvolume) as ttlvolume, s.dsregion,
    RANK() OVER (PARTITION BY s.dsregion ORDER BY sum(s.qtvolume) ASC) as ranking
    FROM sales s
    GROUP BY s.dsbrand, s.dsregion)

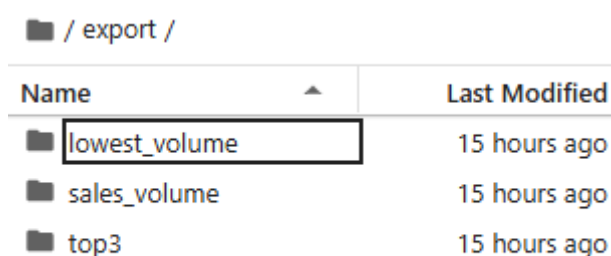
    SELECT l.dsbrand, l.ttlvolume as minvolume, l.dsregion FROM lowest_volume l
    WHERE l.ranking = 1
    ORDER BY l.dsregion asc
    ''')
```

|   | dsbrand    | minvolume | dsregion    |
|---|------------|-----------|-------------|
| 0 | STRAWBERRY | 74009.61  | CANADA      |
| 1 | STRAWBERRY | 208747.90 | GREAT LAKES |
| 2 | STRAWBERRY | 149206.78 | MIDWEST     |
| 3 | STRAWBERRY | 195581.49 | NORTHEAST   |
| 4 | RASPBERRY  | 223958.50 | SOUTHEAST   |
| 5 | GRAPE      | 7.50      | SOUTHWEST   |
| 6 | STRAWBERRY | 134511.18 | WEST        |

“Which are the lowest brand (BRAND\_NM) in sales (\$ Volume) for each region (Btlr\_Org\_LVL\_C\_Desc)?”

### 3.1 Das Saídas e Exportação de Dados

Para garantir que os resultados das análises de negócio sejam consumíveis por diferentes áreas (como Trade Marketing e Controladoria), implementei uma rotina de exportação automática para a pasta `/export`. Os resultados das queries de **Top 3 Trade Groups**, **Volume por Marca** e **Lowest Sales** foram persistidos fisicamente em arquivos CSV.



| / export /    |               |
|---------------|---------------|
| Name          | Last Modified |
| lowest_volume | 15 hours ago  |
| sales_volume  | 15 hours ago  |
| top3          | 15 hours ago  |

Utilizei a função `coalesce(1)` do Spark antes da escrita para consolidar o processamento distribuído em arquivos únicos. Essa abordagem facilita o consumo direto por ferramentas que não suportam leitura particionada, garantindo que o usuário final receba um arquivo pronto para uso, sem a necessidade de manipulação manual.

```
[95]: top3.coalesce(1).write.format('csv').save(workdir+"/export/top3")
[96]: sales_volume.coalesce(1).write.format('csv').save(workdir+"/export/sales_volume")
[97]: lowest_volume.coalesce(1).write.format('csv').save(workdir+"/export/lowest_volume")
[98]: !ls -a /home/jovyan/work/export
. . . lowest_volume sales_volume top3
```

Todo o processo de saída é **auditável e transparente**. Como o export é o estágio final de um pipeline que passou pelas camadas Bronze, Silver e Gold, cada linha nos arquivos de saída possui linhagem garantida. Isso assegura que qualquer número apresentado nos relatórios finais possa ser rastreado até sua origem bruta, mantendo a integridade e a governança exigidas pela **ABInBev** no Business Case.

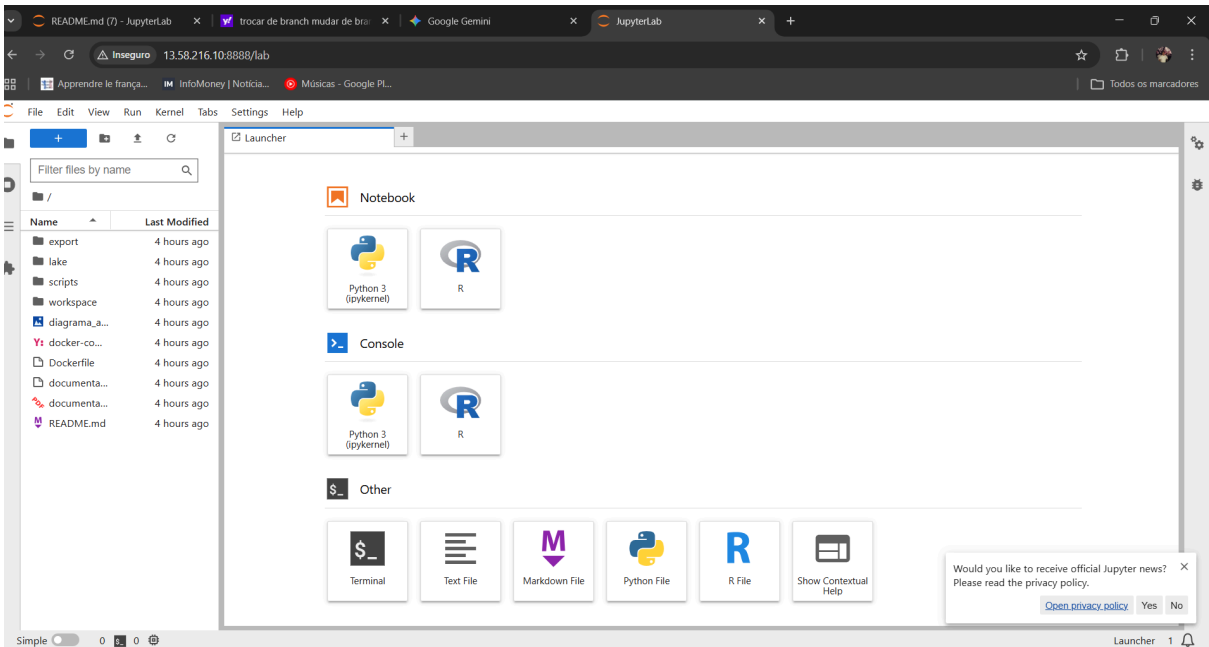
### 3.2 Repositório GIT

O resultado é possível ser conferido em repositório GIT, persistente e acessível publicamente para todas as partes interessadas em:

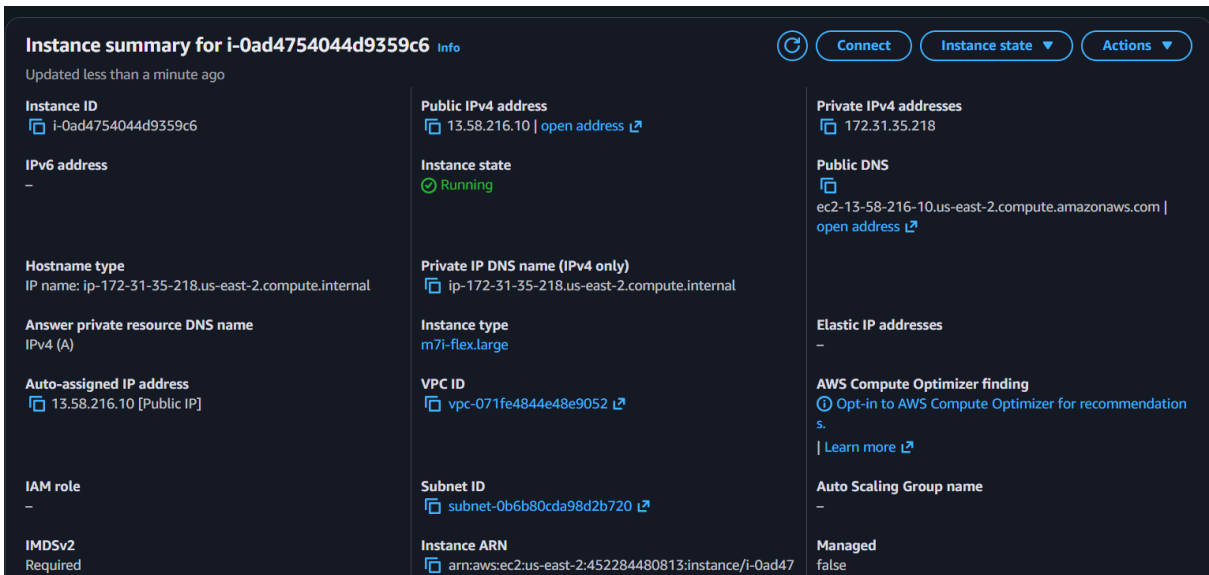
- [pmneto/abinbev\\_businesscase](https://github.com/pmneto/abinbev_businesscase)

### 3.3 Testagem do Container

Realizei a testagem do container em uma Virtual Machine EC2 da AWS, o propósito era ver se o container performou o provisionamento de ambiente de desenvolvimento e execução.



Jupyter Lab corretamente provisionado no IP efêmero



Informações do da Virtual Machine EC2, recurso já descontinuado

## 4. Referências

Para garantir a robustez e o padrão de mercado exigido pela **ABInBev**, utilizei as seguintes documentações oficiais e bibliotecas como base:

### 4.1 Frameworks de Engenharia de Dados

- [Apache Spark Python API \(PySpark\)](#): Documentação base para todas as transformações, uso de `Window Functions` e `Temporary Views`.
- [Delta Lake Documentation](#): Referência crucial para implementação das camadas **Bronze, Silver e Gold**, garantindo transações ACID e persistência física eficiente.
- [PyGWalker \(Graphic Walker\)](#): Biblioteca utilizada para a análise exploratória visual e validação dos dados da camada Gold através de interface drag-and-drop.

### 4.2 Infraestrutura e Ambiente

- [Docker & Docker Compose](#): Documentação utilizada para refatorar o ambiente, removendo o metastore externo e garantindo a portabilidade total do Lakehouse.
- [Jupyter Docker Stacks](#): Base para a imagem `all-spark-notebook` utilizada no nosso Dockerfile.

### 4.3 Bibliotecas e Utilitários

- [Python Holidays Lib](#): Documentação da biblioteca instalada via `pip` para enriquecer a dimensão de tempo com feriados dos EUA e Canadá.
- [Pandas Styler](#): Referência para o uso do `.style.hide(axis='index')`, garantindo que os prints do `toPandas()` fiquem limpos no documento.
- [Diagrams.net \(draw.io\) SQL Import](#): Guia utilizado para criar o script SQL DDL que gera automaticamente o MER no padrão Star Schema.