Documentação Rota01 etl

Links:

- **Apresentação:**https://docs.google.com/presentation/d/1sb17TfKWQTtP8vbcLa2wXxQdc9UlyuI9GrTtcHCD9eI/edit?usp=sharing
- **Google Colab:** https://colab.research.google.com/drive/1zI8AtCC3zh90I97AvsaJmxHYXO7Tujmv?usp=sharing
- **Dashboard:** https://app.powerbi.com/view?r=eyJrIjoiODMyYjdjMjMtOTFkNi00NDkwLWI2ZTUtYmNjZGM1MTU4YTlhIiwidCI6IjhiZTBkOTY1LWE1NTktNDYyNC1iN

Guia de Projeto de Aprofundamento: Estrutura de Dados

Rota 1 - Exploração em ETL

Neste projeto você construirá um sistema de tabelas para consultar informações de forma organizada e relacionada que permite unir dados de diferentes maneiras dependendo do objetivo da análise.

O processo de ETL acontece antes da fase de análise. ETL é o processo que garante aos analistas que os dados estejam disponíveis e prontos para serem consultados.

> Critérios de aceitação:
>
> - Você criou as tabelas fatos e dimensões para organizar os dados.
> - Você criou uma Ficha Técnica para o seu projeto, com o seu processo de tomada de decisão e informações úteis (inclua as consultas utilizadas, e outros arquivos e links nos quais você trabalhou durante o projeto, onde podemos ver claramente todas as etapas que você seguiu e os resultados que você obteve).
> - Você subiu este projeto no seu repositório/portfólio no GitHub.
> - Você compartilhou o link da sua ficha técnica ou repositório no GitHub com o desenvolvimento do seu projeto através da plataforma.
> - Você gravou e compartilhou um vídeo de no máximo 5 minutos, explicando como você organizou os dados em tabelas de fatos e dimensões.

Caso

A gestão eficiente de dados tornou-se crucial para o sucesso das organizações. A Super Store, líder em seu setor, enfrenta o desafio de gerenciar grandes volumes de dados dispersos e desestruturados. Para resolver esse problema e melhorar a tomada de decisões, propõe-se a implementação de um sistema ETL robusto (Extract, Transform, Load ou em português: Extrair, Transformar e Carregar) com tabelas de fatos e dimensões.

Este projeto foca na criação de um sistema abrangente que permita extrair dados de diversas fontes, transformá-los de acordo com as necessidades específicas da Super Store e carregá-los de forma eficiente em um **data warehouse** organizado.

O objetivo deste projeto não é apenas otimizar o armazenamento de dados, mas também aumentar a capacidade da Super Store de identificar padrões, tendências e oportunidades de mercado. Com um sistema ETL bem projetado, a empresa estará melhor preparada para se adaptar rapidamente às mudanças na demanda do consumidor, melhorar a eficiência operacional e manter uma vantagem competitiva em um mercado em constante evolução.

Ao longo deste projeto, exploraremos em detalhes os benefícios esperados, custos associados, riscos potenciais e retorno sobre o investimento projetado para apoiar a decisão estratégica de implementar este sistema ETL com tabelas de fatos e dimensões na Super Store.

1. Ferramentas, linguagens e insumos

1.1 Ferramentas e/ou plataformas

Neste projeto você utilizará diversas ferramentas, em alguns objetivos você poderá escolher com qual ferramenta irá trabalhar:

- Para entender os dados e definir tabela você usará o Google BigQuery
- Para pesquisar dados de outras fontes você pode usar o Planilhas Google ou Python e Google Colab

1.2 Linguagens

Você usará a linguagem SQL no BigQuery e Python no Google Colab (se você escolher usá-lo neste projeto).

1.3 Insumos

Neste projeto você terá uma tabela com dados de vendas da Super Store e uma segunda tabela que você vai construir com os dados dos concorrentes. Você usará os conceitos apresentados neste projeto para transformar os dados em um conjunto de dados estruturados.

O dataset da SuperStore está disponível para download neste **link**.

Abaixo, você pode consultar a descrição das variáveis que compõem a tabela:

> category: Representa as categorias de produtos vendidos no hipermercado.
>
> **city:** Representa a cidade onde o pedido foi feito.
>
> **country:** Representa o país onde o hipermercado está localizado.
>
> **customer_id:** Representa um identificador único para cada cliente.
>
> **customer_name:** Representa o nome do cliente que fez o pedido.
>
> **discount:** Representa o desconto aplicado no pedido.
>
> **market:** Representa o mercado ou região onde o hipermercado atua.
>
> **unknown:** Uma coluna desconhecida ou não especificada.
>
> **order_date:** Representa a data em que o pedido foi feito.
>
> **order_id:** Um identificador único para cada pedido.
>
> **order_priority:** Representa o nível de prioridade do pedido.
>
> **product_id:** Representa um identificador exclusivo para cada produto.
>
> **product_name:** Representa o nome do produto.
>
> **profit:** Representa o lucro gerado pelo pedido.
>
> **quantity:** Representa a quantidade de produtos encomendados.
>
> **region:** Representa a região onde o pedido foi feito.
>
> **row_id:** Representa um identificador exclusivo para cada linha do conjunto de dados.
>
> **sales:** Representa o valor total de vendas do produto no pedido.
>
> **segment:** Representa o segmento do cliente (por exemplo, consumidores, empresas ou escritórios domésticos).
>
> **ship_date:** Representa a data em que o pedido foi enviado.

> **ship_mode:** Representa o modo de envio usado para o pedido.
>
> **shipping_cost:** Representa o custo de envio do pedido.
>
> **state:** Representa o estado ou região do país.
>
> **sub_category:** Representa a subcategoria de produtos dentro da categoria principal.
>
> **year:** Representa o ano em que o pedido foi feito.
>
> **market2:** Outra coluna relacionada a informações de mercado.
>
> **weeknum:** Representa o número da semana em que o pedido foi feito.

2. Mãos à obra!

ETL refere-se a um processo de três fases: **E**xtração (**E**xtraction), **T**ransformação (**T**ransformation) e **C**arga (**L**oad). Esse processo é comumente usado na área de gerenciamento e análise de dados, especialmente no contexto de *data warehouses* e *business intelligence*.

Aqui está uma breve descrição de cada fase do processo ETL:

Extração (Extraction)

Nesta fase, os dados são extraídos de uma ou várias fontes, que podem ser bancos de dados, arquivos simples, serviços web ou outras fontes. A extração envolve a coleta das informações necessárias para o processamento posterior.

Transformação (Transformation)

Nesta etapa, os dados extraídos são transformados de acordo com os requisitos do sistema de destino. As transformações podem incluir limpeza dos dados, conversão de formatos, combinação de dados de múltiplas fontes, filtragem e outras operações que garantem que os dados sejam consistentes e úteis para a análise.

Carga (Load)

A fase final envolve carregar os dados transformados no sistema de destino, que geralmente é um *data warehouse* ou um banco de dados projetado para análise de negócios. Os dados estão agora prontos para serem consultados e analisados de forma eficiente.

🟦 2.1 Processar e Preparar a Base de Dados

Tempo estimado: 8 a 12 horas

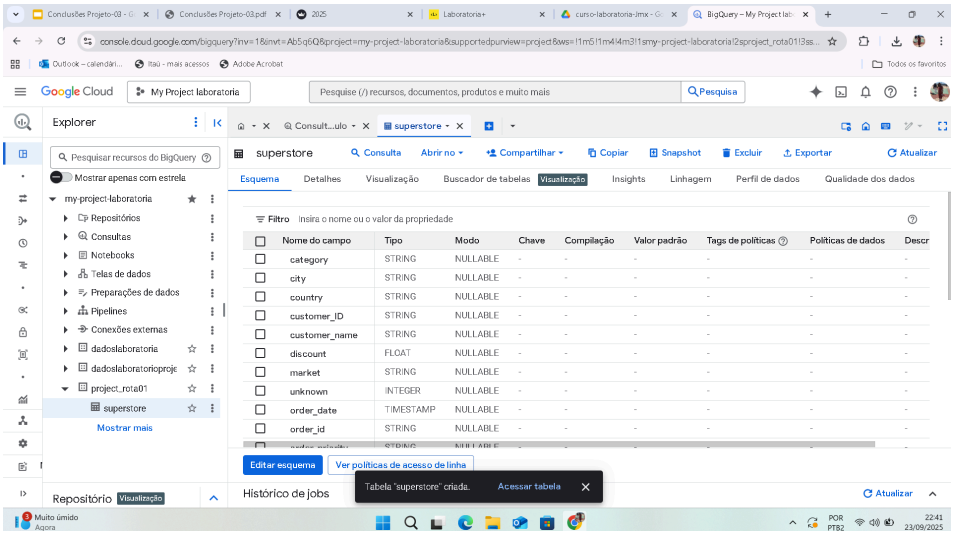
Utilize o conhecimento e as habilidades desenvolvidas em outros projetos para garantir a consistência dos dados antes de avançar para as novas metas.

Metas de Aprendizagem

Objetivo	Recurso
🔵 **Conectar/importar dados para outras ferramentas** Carregar os dados para o BigQuery.	-
🔵 **Identificar e lidar com valores nulos** Identificar os valores nulos.	💡 Neste projeto, o objetivo é construir uma estrutura de tabelas que permita consultar e armazenar dados de forma mais eficiente. Por essa razão, apenas identificaremos onde há valores nulos, sem realizar nenhuma alteração. Mais tarde, quando começarmos a criar as tabelas de dimensão, garantiremos que não haja valores nulos nessas tabelas.
🔵 **Identificar e lidar com valores duplicados** Identificar os valores duplicados.	💡 Iremos identificar os duplicados para considerar no momento de criar as tabelas de dimensão e de fatos.
🔵 **Identificar e lidar com dados discrepantes em**	💡 Pela mesma razão que não inseriremos dados para valores nulos, também não o faremos para variáveis categóricas. Apenas

variáveis categóricas Identificar e lidar com inconsistências em variáveis categóricas.	garantiremos que não haja inconsistências e padronizaremos os dados se necessário (por exemplo, deixando todos os dados de uma mesma variável em minúsculas ou maiúsculas).
🔵 **Identificar e lidar com dados discrepantes em variáveis numéricas** Identificar inconsistências em variáveis numéricas.	💡 Pela mesma razão que não inseriremos dados para valores nulos, também não o faremos para variáveis numéricas. Apenas garantiremos que não haja inconsistências, como valores de texto ou de outro tipo que não seja numérico.
🔵 **Verificar e alterar o tipo de dado** Identificar se é necessário alterar o tipo de dados de alguma variável.	-
🔵 **Buscar dados de outras fontes** Fazer *Web scraping*.	💡 Use `IMPORTHTML` no Google Sheets ou o pacote `Beautiful Soup` no Python para extrair os dados da tabela "multinacional" desta página da Wikipédia, a fim de incluir os concorrentes na estrutura de dados da empresa Super Store.
🔵 **Projetar a estrutura do banco de dados (tabelas de fatos e dimensões)** Projetar tabelas de fatos e de dimensões.	• 📄 Conceito de tabelas de fatos e de dimensão • 📹 Conceito de relacionamento entre tabelas e criação de tabelas no Access. ⚠️ Use este recurso para entender a relação entre tabelas e a construção das tabelas de fatos e dimensões, pois não usaremos a mesma ferramenta neste projeto. • 💡 Esta meta visa projetar a estrutura das tabelas antes de criá-las. Você pode usar ferramentas como esta para projetar seu esquema.
🔵 **Criar a estrutura do banco de dados (tabelas de fatos e dimensões)** Criar no BigQuery tabelas de fatos e tabelas de dimensões.	💡 Use os identificadores (IDs) existentes na tabela; se necessário, gere IDs para as tabelas de dimensão onde a informação não tiver um, como no caso de 'ship_mode'. Ao criar tabelas de dimensão, você obterá várias tabelas com informações únicas. Por exemplo, você pode ter uma tabela que contém todos os dados do cliente sem duplicatas. Consequentemente, ao trabalhar com uma tabela de dimensão, não é necessário trazer a totalidade dos dados do cliente; apenas o ID do cliente e as métricas de venda pertinentes para compor essa tabela. Esta estratégia otimiza o processo, pois a informação do cliente pode ser vinculada eficientemente a outras consultas através do uso dos IDs.
🔵 **Programar atualizações de tabela** Projetar um *pipeline* de atualização de dados.	• 📹 Conceito de *Data Pipeline* • 💡 Um *pipeline* de dados é o processo automatizado de atualizar sua estrutura de dados. Imagine que todos os dias chegam dados sobre novos pedidos, como podemos garantir que esses dados sejam atualizados em nossas tabelas de fatos e dimensões? Existem vários serviços na nuvem para resolver esse problema. Neste projeto, vamos determinar qual seria a sequência de tabelas a serem atualizadas, considerando as relações entre as tabelas. • ⚠️ Por enquanto, não vamos usar nenhuma ferramenta para fazer o *pipeline*. O objetivo desta meta é projetar o fluxo de atualização das tabelas que você criou. • 🤸 Apenas por diversão: Se você se animar, pode pesquisar conteúdos sobre o Google Cloud.
🔵 **Unir tabelas** Criar a partir das tabelas estruturadas uma tabela para análise exploratória.	💡 Selecione as variáveis que você usará para realizar a análise exploratória e conecte esses dados a uma ferramenta de visualização de dados.

> 🔵 **Conectar/importar dados para outras ferramentas**
> Carregar os dados para o BigQuery.



2.1.1 🔵 Identificar e tratar valores nulos

Objetivo: Identificar Nulos

Neste projeto o objetivo é construir uma estrutura de tabela que nos permita consultar e armazenar dados de forma mais eficiente. Por esse motivo, identificamos apenas onde temos nulos, sem fazer nenhuma alteração. Posteriormente, quando começarmos a criar tabelas de dimensões, que seguem uma característica específica, vamos nos certificar de que não teremos nenhum valor nulo nessas tabelas.

```
SELECT
 COUNT(*) AS total_linhas,
 COUNTIF(category IS NULL) AS category_nulos,
 COUNTIF(city IS NULL) AS city_nulos,
 COUNTIF(country IS NULL) AS country_nulos,
 COUNTIF(customer_id IS NULL) AS customer_id_nulos,
 COUNTIF(customer_name IS NULL) AS customer_name_nulos,
 COUNTIF(discount IS NULL) AS discount_nulos,
 COUNTIF(market IS NULL) AS market_nulos,
 COUNTIF(unknown IS NULL) AS unknown_nulos,
 COUNTIF(order_date IS NULL) AS order_date_nulos,
 COUNTIF(order_id IS NULL) AS order_id_nulos,
 COUNTIF(order_priority IS NULL) AS order_priority_nulos,
 COUNTIF(product_id IS NULL) AS product_id_nulos,
 COUNTIF(product_name IS NULL) AS product_name_nulos,
 COUNTIF(profit IS NULL) AS profit_nulos,
 COUNTIF(quantity IS NULL) AS quantity_nulos,
 COUNTIF(region IS NULL) AS region_nulos,
 COUNTIF(row_id IS NULL) AS row_id_nulos,
 COUNTIF(sales IS NULL) AS sales_nulos,
 COUNTIF(segment IS NULL) AS segment_nulos,
 COUNTIF(ship_date IS NULL) AS ship_date_nulos,
 COUNTIF(ship_mode IS NULL) AS ship_mode_nulos,
 COUNTIF(shipping_cost IS NULL) AS shipping_cost_nulos,
 COUNTIF(state IS NULL) AS state_nulos,
 COUNTIF(sub_category IS NULL) AS sub_category_nulos,
 COUNTIF(year IS NULL) AS year_nulos,
 COUNTIF(market2 IS NULL) AS market2_nulos,
 COUNTIF(weeknum IS NULL) AS weeknum_nulos
FROM
  `my-project-laboratoria.project_rota01.superstore`
```

Nenhum valor nulo encontrado.

> 2.1.2 🔵 Identificar e tratar valores duplicados
>
> **Objetivo**: Identificar duplicatas
>
> Identifique duplicados a serem levados em consideração ao crie tabelas de dimensões e fatos.

```
  #Consulta para identificar valores duplicados na tabela superstore
SELECT
  category,
  city,
  country,
  customer_id,
  customer_name,
  discount,
  market,
  unknown,
  order_date,
  order_id,
  order_priority,
  product_id,
  product_name,
  profit,
  quantity,
  region,
  sales,
  segment,
  ship_date,
  ship_mode,
  shipping_cost,
  state,
  sub_category,
  year ,
  market2,
  weeknum,
  COUNT(*) AS num_duplicates
FROM
  `my-project-laboratoria.project_rota01.superstore`
GROUP BY
  category,
  city,
  country,
  customer_id,
  customer_name,
  discount,
  market,
  unknown,
  order_date,
  order_id,
  order_priority,
  product_id,
  product_name,
  profit,
  quantity,
  region,
  sales,
  segment,
  ship_date,
```

```
  ship_mode,
  shipping_cost,
  state,
  sub_category,
  year ,
  market2,
  weeknum
HAVING
  COUNT(*) > 1
```

Nenhum valor duplicado encontrado.

> 2.1.3 🔵 Identificar e tratar dados discrepantes em variáveis categóricas
>
> **Objetivo**: Identificar e tratar inconsistências em variáveis categórica
>
> > Oráculo: Dicas, Glossário e Conselhos da/do Coach
> >
> > Pelo mesmo motivo que não vamos inserir nenhum dado para valores nulos, também não vamos inserir dados para variáveis categóricas, simplesmente garantimos que não haja inconsistência e padronizamos os dados se necessário, por exemplo, para que os dados de uma mesma variável estejam todos em letras minúsculas ou maiúsculas.

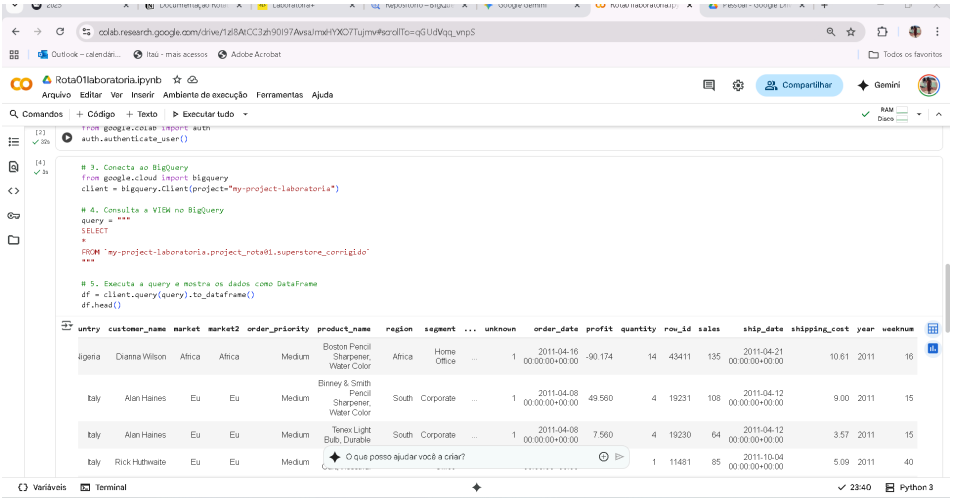
```
CREATE OR REPLACE VIEW `my-project-laboratoria.project_rota01.superstore_corrigido` AS
SELECT
  -- Colunas que devem ser capitalizadas
  INITCAP(category) AS category,
  INITCAP(city) AS city,
  INITCAP(country) AS country,
  INITCAP(customer_name) AS customer_name,
  INITCAP(market) AS market,
  INITCAP(market2) AS market2,
  INITCAP(order_priority) AS order_priority,
  INITCAP(product_name) AS product_name,
  INITCAP(region) AS region,
  INITCAP(segment) AS segment,
  INITCAP(ship_mode) AS ship_mode,
  INITCAP(state) AS state,
  INITCAP(sub_category) AS sub_category,

  -- Colunas que não devem ser capitalizadas (numéricas, IDs, e datas)
  customer_ID,
  order_id,
  product_id,
  discount,
  unknown,
  order_date,
  profit,
  quantity,
  row_id,
  sales,
  ship_date,
  shipping_cost,
  year,
  weeknum
```

```
FROM
  `my-project-laboratoria.project_rota01.superstore`
```

2.1.4 🔵 Identificar e tratar dados discrepantes em variáveis numéricas

> **Objetivo**: Identificar inconsistências em variáveis numéricas
>
> Oráculo: Dicas, Glossário e Conselhos da/do Coach
>
> > Pela mesma razão que não vamos inserir nenhum dado para valores nulos, também não vamos inserir dados para variáveis numéricas, simplesmente nos certificarmos de que não haja inconsistências, como valores de texto ou outros tipos não numéricos.

Nesta etapa criei um caderno colab conectando ao big query




```
import pandas as pd
import numpy as np

# A sua linha de código para carregar o DataFrame
# Certifique-se de que a sua query seleciona as variáveis numéricas necessárias
df = client.query(query).to_dataframe()

# Lista das variáveis numéricas a serem analisadas
# Adapte esta lista se as suas colunas numéricas forem diferentes
variaveis_numericas = [
    'discount', 'profit', 'quantity', 'unknown', 'row_id',
    'sales', 'shipping_cost', 'year', 'weeknum'
]

print("Identificando valores discrepantes (outliers) usando o método IQR:")
print("-" * 70)

# Loop para identificar outliers em cada variável
for col in variaveis_numericas:
    # Verifica se a coluna existe no DataFrame
    if col not in df.columns:
        print(f"Aviso: A coluna '{col}' não foi encontrada no DataFrame. Pulando...")
        continue

    Q1 = df[col].quantile(0.25)
```

```
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    limite_inferior = Q1 - 1.5 * IQR
    limite_superior = Q3 + 1.5 * IQR

    # Encontra os valores que estão fora dos limites
    outliers = df[(df[col] < limite_inferior) | (df[col] > limite_superior)]

    print(f"Variável: {col}")
    print(f"  Limites (IQR): [{limite_inferior:.2f}, {limite_superior:.2f}]")
    print(f"  Total de outliers encontrados: {len(outliers)}")

    if not outliers.empty:
        print("  Valores discrepantes:")
        print(outliers[[col]])
    else:
        print("  Nenhum outlier encontrado.")

    print("-" * 70)
```

Resumo da Análise de Outliers

A análise de valores discrepantes (outliers) usando o método IQR (Intervalo Interquartil) revelou os seguintes pontos:

Variáveis com um número significativo de outliers:

- `profit`: Com 9.755 outliers, esta variável tem a maior quantidade de valores discrepantes. Os limites aceitáveis para o lucro estão entre **55,22 e 92,03**. Há uma grande quantidade de lucros (ou prejuízos) muito altos ou muito baixos que precisam ser investigados.
- `shipping_cost`: Foram encontrados 5.909 outliers. Os custos de envio considerados normais estão entre **30,15 e 57,21**. Valores acima disso são atípicos.
- `sales`: Com 5.655 outliers, esta variável mostra que a maioria das vendas está na faixa entre **299,00 e 581,00**. Vendas acima desse valor são raras.
- `discount`: Foram identificados 4.172 outliers. A maior parte dos descontos se concentra entre **0,30 e 0,50**, indicando que valores como `0.7` estão fora do padrão.

Variáveis com poucos ou nenhum outlier:

- `quantity`: Apenas 877 outliers foram encontrados, com os valores discrepantes sendo principalmente quantidades de itens maiores que `9`.
- `unknown`, `row_id`, `year` e `weeknum`: Nenhuma dessas variáveis apresentou outliers. Isso é esperado para `row_id`, `year` e `weeknum`, pois são dados que geralmente seguem um padrão regular e previsível.

Em resumo, as variáveis `profit`, `shipping_cost`, `sales` e `discount` contêm a maioria dos valores atípicos. Essas variáveis merecem uma atenção especial para entender a causa desses dados discrepantes, que podem ser erros de entrada, eventos incomuns ou observações que realmente representam valores extremos.

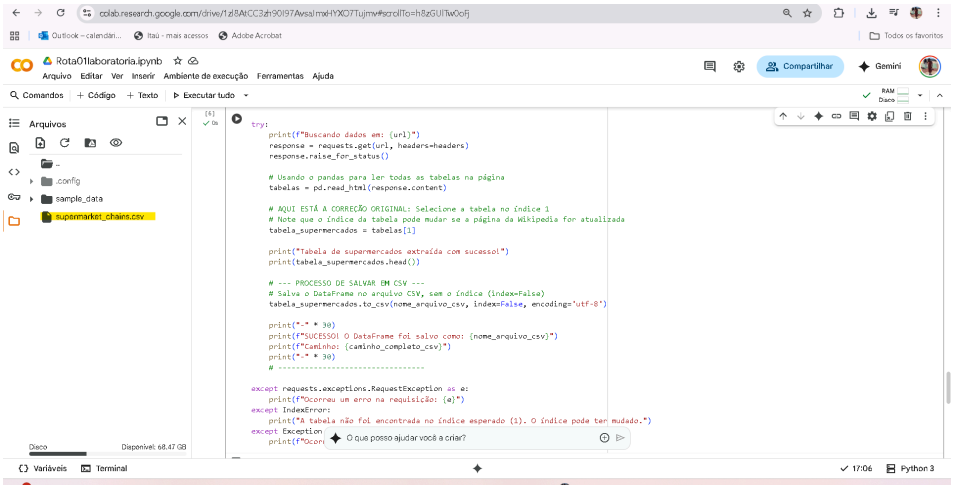
2.1.5 🔵 Pesquisar dados de outras fontes

Objetivo: Faça web scraping

💡 Use **IMPORTHTML** no Planilhas Google ou **Beautiful Soup** em Python para extrair os dados da tabela "multinacional" nesta página da **wikipidia**, e assim incluir os concorrentes da empresa Super Store na estrutura de dados.

```
import pandas as pd
import requests
import os

url = 'https://en.wikipedia.org/wiki/List_of_supermarket_chains'
```

```
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.
0.4472.124 Safari/537.36'
}

# --- Configurações de Salvar CSV ---
# Nome do arquivo CSV de saída
nome_arquivo_csv = 'supermarket_chains.csv'
# Onde o arquivo será salvo (o diretório atual do script)
caminho_completo_csv = os.path.join(os.getcwd(), nome_arquivo_csv)
# ----------------------------------

try:
    print(f"Buscando dados em: {url}")
    response = requests.get(url, headers=headers)
    response.raise_for_status()

    # Usando o pandas para ler todas as tabelas na página
    tabelas = pd.read_html(response.content)

    # AQUI ESTÁ A CORREÇÃO ORIGINAL: Selecione a tabela no índice 1
    # Note que o índice da tabela pode mudar se a página da Wikipedia for atualizada
    tabela_supermercados = tabelas[1]

    print("Tabela de supermercados extraída com sucesso!")
    print(tabela_supermercados.head())

    # --- PROCESSO DE SALVAR EM CSV ---
    # Salva o DataFrame no arquivo CSV, sem o índice (index=False)
    tabela_supermercados.to_csv(nome_arquivo_csv, index=False, encoding='utf-8')

    print("-" * 30)
    print(f"SUCESSO! O DataFrame foi salvo como: {nome_arquivo_csv}")
    print(f"Caminho: {caminho_completo_csv}")
    print("-" * 30)
    # ---------------------------------

except requests.exceptions.RequestException as e:
    print(f"Ocorreu um erro na requisição: {e}")
except IndexError:
    print("A tabela não foi encontrada no índice esperado (1). O índice pode ter mudado.")
except Exception as e:
    print(f"Ocorreu um erro inesperado: {e}")
```



supermarket_chains.csv

2.1.6 🔵 Projetar estrutura de base de dados (tabelas de fatos e dimensões)

Objetivo: Projetar tabelas de fatos e tabelas de dimensões

1. Proposta de Modelo Estrela

A modelagem de dados para análise de vendas como a do `superstore.csv` geralmente se baseia em uma **Tabela Fato** central, chamada **Fato_Vendas**, e várias **Tabelas Dimensão** que a descrevem.

A. Tabela Fato: Fato_Vendas (Fact_Sales)

Esta tabela registra os eventos mensuráveis (as vendas em si). Cada linha representa um item vendido em um pedido.

Coluna	Tipo (Conceitual)	Descrição	Origem (Colunas do CSV)
chave_cliente	Chave Estrangeira (FK)	Chave para a Dimensão Cliente.	`customer_ID`
chave_produto	Chave Estrangeira (FK)	Chave para a Dimensão Produto.	`product_id`
chave_localidade	Chave Estrangeira (FK)	Chave para a Dimensão Localidade.	`state` , `city` , `country` , `region` , `market`
chave_tempo_pedido	Chave Estrangeira (FK)	Data do pedido.	`order_date`
chave_tempo_envio	Chave Estrangeira (FK)	Data de envio.	`ship_date`
sales	Métrica	Valor da venda.	`sales`
quantity	Métrica	Quantidade vendida.	`quantity`
profit	Métrica	Lucro obtido.	`profit`
discount	Métrica	Valor do desconto aplicado.	`discount`
shipping_cost	Métrica	Custo de envio.	`shipping_cost`

B. Tabelas Dimensão (Dimension Tables)

As tabelas de dimensão fornecem o *contexto* (quem, o quê, onde e quando) sobre as vendas.

1. Dim_Cliente (Dim_Customer)

Captura informações sobre quem comprou.

Coluna	Tipo (Conceitual)	Descrição	Origem (Colunas do CSV)
chave_cliente	**Chave Primária (PK)**	Chave substituta (Surrogate Key).	(Gerada)
customer_ID	Chave Natural	ID original do cliente no sistema transacional.	`customer_ID`
customer_name	Atributo	Nome completo do cliente.	`customer_name`
segment	Atributo	Segmento do cliente.	`segment`

versao_atual	SCD Tipo 2	Indica se esta é a linha de registro atual do cliente.	(Gerada - Booleana)

2. Dim_Produto (Dim_Product)

Detalha o que foi vendido.

Coluna	Tipo (Conceitual)	Descrição	Origem (Colunas do CSV)
chave_produto	**Chave Primária (PK)**	Chave substituta (Surrogate Key).	(Gerada)
product_id	Chave Natural	ID original do produto.	`product_id`
product_name	Atributo	Nome do produto.	`product_name`
category	Atributo	Categoria principal.	`category`
sub_category	Atributo	Subcategoria.	`sub\_category`

3. Dim_Localidade (Dim_Geography)

Detém o contexto geográfico (Onde). É importante para permitir análises por diferentes níveis de hierarquia (Cidade → Estado → País → Região).

Coluna	Tipo (Conceitual)	Descrição	Origem (Colunas do CSV)
chave_localidade	**Chave Primária (PK)**	Chave substituta (Surrogate Key).	(Gerada)
city	Atributo	Cidade.	`city`
state	Atributo	Estado.	`state`
country	Atributo	País.	`country`
region	Atributo	Região (geográfica).	`region`
market	Atributo	Mercado (Global/Customizado).	`market` ou `market2`

4. Dim_Tempo (Dim_Time)

Estrutura o tempo para permitir fatiamento e agregação de dados por diferentes períodos (Quando). Será criada a partir das colunas `order_date` e `ship_date`.

Coluna	Tipo (Conceitual)	Descrição	Origem (Colunas do CSV)
chave_tempo	**Chave Primária (PK)**	Chave baseada em data (formato `AAAAMMDD`).	(Gerada)
data_completa	Atributo	Data completa.	`order_date` ou `ship_date`
ano	Atributo	Ano.	`year`
trimestre	Atributo	Trimestre (Q1, Q2, etc.).	(Derivada)
mes	Atributo	Mês.	(Derivada)
semana_do_ano	Atributo	Número da semana no ano.	`weeknum`

---

2. Abordagem para Slowly Changing Dimensions (SCD)

Dimensões Cliente e Produto são as mais prováveis de precisarem de um histórico de alterações.

- **SCD Tipo 1 (Sobrescrita):** Se o `customer_name` (nome do cliente) mudasse e você decidisse que apenas o nome atual é importante, você **sobrescreveria** o registro na `Dim_Cliente`.
- **SCD Tipo 2 (Histórico):** Se o `segment` (segmento do cliente) mudasse e fosse crucial manter o registro das vendas feitas antes e depois da mudança (para análises históricas), você usaria o **SCD Tipo 2**. Isso exigiria a adição de colunas como:
  - `data_inicio_vigencia`
  - `data_fim_vigencia`
  - `versao_atual` (Booleano: Sim/Não)

Recomendação para a Fase de ETL: Implemente o **SCD Tipo 2** para a **Dim_Cliente** para rastrear mudanças no `segment` e a **Dim_Produto** para rastrear mudanças em `category` ou `sub_category`.

---

3. Próximos Passos (Estrutura do ETL)

ETL (Extração, Transformação e Carga). A estrutura que você precisa seguir é:

1. **Extração (E):** Ler o `superstore.csv`.
2. **Transformação (T):**
   - **Limpeza/Padronização:** Corrigir erros de formatação ou valores ausentes.
   - **Criação de Dimensões:** Para cada linha do `superstore.csv`, extrair e criar os registros das tabelas dimensão (Dim_Cliente, Dim_Produto, Dim_Localidade, Dim_Tempo), garantindo que as chaves substitutas (PKs) sejam geradas e que o SCD Tipo 2 seja aplicado onde necessário.
   - **Montagem da Tabela Fato:** Usar as chaves substitutas geradas (PKs) e as métricas para compor a tabela **Fato_Vendas**.
3. **Carga (L):** Inserir os dados limpos e modelados nas tabelas de destino (no seu Data Warehouse).

2.1.7 🔵 Criar estrutura de base de dados (tabelas de fatos e dimensões)

```
# 0.3 Configuração do BigQuery
# --- CONFIGURAÇÕES DE DESTINO ---
# SUBSTITUA PELO SEU PROJECT ID do Google Cloud
PROJECT_ID = "my-project-laboratoria"
# SUBSTITUA PELO NOME DO SEU DATASET no BigQuery
DATASET_ID = "project_rota01"
# --------------------------------

client = bigquery.Client(project=PROJECT_ID)
print(f"Conexão com BigQuery configurada para o Projeto: {PROJECT_ID}")
```

```
# 0.4 Extração (E) - Leitura da VIEW no BigQuery (CORRIGIDO)
# --- EXTRAÇÃO (E) - Lendo dados da VIEW Corrigida no BigQuery ---

query = """
SELECT
  *
FROM `my-project-laboratoria.project_rota01.superstore_corrigido`
"""

try:
  # 5. Executa a query e carrega os dados no DataFrame
  df_superstore = client.query(query).to_dataframe()

  print(f"Dados extraídos do BigQuery com sucesso. {len(df_superstore)} linhas.")

  # Exibe as primeiras linhas para confirmação
  print("Primeiras linhas do DataFrame 'df_superstore':")
  print(df_superstore.head(2).to_markdown(index=False))

except Exception as e:
  print(f"Ocorreu um erro ao consultar o BigQuery: {e}")
  # Cria um DataFrame vazio em caso de erro para evitar quebra do script
  df_superstore = pd.DataFrame()
```

Passo 1: Transformação (T) - Limpeza e Dimensão Tempo

```


# --- TRANSFORMAÇÃO (T) ---

# 1. TRATAMENTO INICIAL DE DATAS
# Converte as colunas de data para o tipo datetime
```

```
df_superstore['order_date'] = pd.to_datetime(df_superstore['order_date'], errors='coerce')
df_superstore['ship_date'] = pd.to_datetime(df_superstore['ship_date'], errors='coerce')

# 2. CRIAÇÃO DA DIMENSÃO TEMPO (DIM_TIME)
# Obtém todas as datas únicas de pedido e envio
datas_unicas = pd.concat([df_superstore['order_date'], df_superstore['ship_date']]).unique()
datas_unicas = pd.Series(datas_unicas).dropna().sort_values().reset_index(drop=True)

# Cria o DataFrame da Dimensão Tempo
df_time = pd.DataFrame({'data_completa': datas_unicas})

# Cria atributos de tempo
df_time['ano'] = df_time['data_completa'].dt.year
df_time['mes'] = df_time['data_completa'].dt.month
df_time['nome_mes'] = df_time['data_completa'].dt.strftime('%B')
df_time['trimestre'] = 'Q' + ((df_time['mes'] - 1) // 3 + 1).astype(str)
df_time['dia_da_semana'] = df_time['data_completa'].dt.dayofweek + 1 # 1=Segunda, 7=Domingo
df_time['semana_do_ano'] = df_time['data_completa'].dt.isocalendar().week.astype(int)

# Geração da CHAVE SUBSTITUTA (PK) no formato AAAAMMDD
df_time['chave_tempo'] = df_time['data_completa'].dt.strftime('%Y%m%d').astype(int)

# Seleciona as colunas finais
df_dim_tempo = df_time[['chave_tempo', 'data_completa', 'ano', 'mes', 'nome_mes', 'trimestre', 'dia_da_semana', 'se
mana_do_ano']]

print(f"Dim_Tempo criada com sucesso. {len(df_dim_tempo)} datas únicas.")
df_dim_tempo.head(2)
```

Passo 2: Transformação (T) - Dimensões Geográficas, Produto e Cliente

```


# 2.1 Dimensões Geográfica e Produto
# 3. CRIAÇÃO DA DIMENSÃO LOCALIDADE (DIM_GEOGRAPHY)
cols_geo = ['city', 'state', 'country', 'region', 'market']
df_dim_localidade = df_superstore[cols_geo].drop_duplicates().reset_index(drop=True)

# Geração da CHAVE SUBSTITUTA (PK)
df_dim_localidade.insert(0, 'chave_localidade', df_dim_localidade.index + 1)

print(f"Dim_Localidade criada com sucesso. {len(df_dim_localidade)} localidades únicas.")
df_dim_localidade.head(2)

# 4. CRIAÇÃO DA DIMENSÃO PRODUTO (DIM_PRODUCT)
cols_prod = ['product_id', 'product_name', 'category', 'sub_category']
df_dim_produto = df_superstore[cols_prod].drop_duplicates(subset=['product_id']).reset_index(drop=True)

# Geração da CHAVE SUBSTITUTA (PK)
df_dim_produto.insert(0, 'chave_produto', df_dim_produto.index + 1)

print(f"Dim_Produto criada com sucesso. {len(df_dim_produto)} produtos únicos.")
df_dim_produto.head(2)
```

2.2 Dimensão Cliente (Simulação Inicial SCD Tipo 2)

```


# 5. CRIAÇÃO DA DIMENSÃO CLIENTE (DIM_CUSTOMER) - SCD TIPO 2
# Apenas a combinação customer_ID + segment é considerada única para o histórico SCD 2 inicial
```

```
cols_cust = ['customer_ID', 'customer_name', 'segment']
df_dim_cliente = df_superstore[cols_cust].drop_duplicates().reset_index(drop=True)

# Geração da CHAVE SUBSTITUTA (PK)
df_dim_cliente.insert(0, 'chave_cliente', df_dim_cliente.index + 1)

# Implementação inicial de colunas SCD Tipo 2
df_dim_cliente['data_inicio_vigencia'] = df_superstore['order_date'].min().strftime('%Y-%m-%d')
df_dim_cliente['data_fim_vigencia'] = None # NULL, pois são registros atuais
df_dim_cliente['versao_atual'] = True

print(f"Dim_Cliente (SCD T2) criada com sucesso. {len(df_dim_cliente)} registros de cliente/segmento.")
df_dim_cliente.head(2)
```

Passo 3: Transformação (T) - Tabela Fato

```
# 6. CRIAÇÃO DA TABELA FATO (FACT_SALES)

# Cria uma cópia da tabela original para evitar modificar a fonte
df_fato = df_superstore.copy()

# --- MERGE com DIM_TEMPO ---
# Adiciona as chaves de tempo (AAAAMMDD) à Fato
df_fato['chave_tempo_pedido'] = df_fato['order_date'].dt.strftime('%Y%m%d').astype(int)
df_fato['chave_tempo_envio'] = df_fato['ship_date'].dt.strftime('%Y%m%d').astype(int)

# --- MERGE com DIM_LOCALIDADE ---
df_fato = pd.merge(
    df_fato,
    df_dim_localidade,
    on=['city', 'state', 'country', 'region', 'market'],
    how='left'
)

# --- MERGE com DIM_PRODUTO ---
df_fato = pd.merge(
    df_fato,
    df_dim_produto[['chave_produto', 'product_id']],
    on='product_id',
    how='left'
)

# --- MERGE com DIM_CLIENTE ---
# Usamos customer_ID E segment para ligar corretamente, devido ao SCD T2
df_fato = pd.merge(
    df_fato,
    df_dim_cliente[['chave_cliente', 'customer_ID', 'segment']],
    on=['customer_ID', 'segment'],
    how='left'
)

# --- SELEÇÃO FINAL DA TABELA FATO ---
cols_fato = [
    'row_id',
    'order_id',
    'chave_cliente',
    'chave_produto',
    'chave_localidade',
```

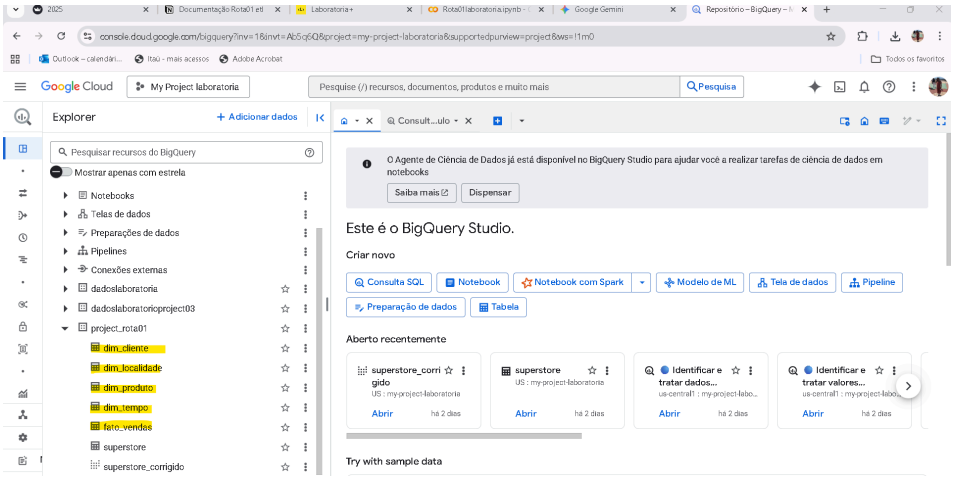
```
    'chave_tempo_pedido',
    'chave_tempo_envio',
    'sales',
    'quantity',
    'profit',
    'discount',
    'shipping_cost',
    'order_priority',
    'ship_mode'
]

df_fato_vendas = df_fato[cols_fato]
df_fato_vendas = df_fato_vendas.rename(columns={'order_id': 'chave_pedido'})

print(f"Fato_Vendas criada com sucesso. {len(df_fato_vendas)} transações.")
df_fato_vendas.head(2)
```

Passo 4: Carga (L) no Google BigQuery

```
# Mapeamento dos DataFrames para os nomes das tabelas no BigQuery
tables_to_load = {
    "dim_tempo": df_dim_tempo,
    "dim_localidade": df_dim_localidade,
    "dim_produto": df_dim_produto,
    "dim_cliente": df_dim_cliente,
    "fato_vendas": df_fato_vendas
}

# Configuração da carga (Load)
job_config = bigquery.LoadJobConfig(
    write_disposition="WRITE_TRUNCATE", # Sobrescreve a tabela se ela existir
)

print(f"Iniciando a carga de dados no BigQuery (Dataset: {DATASET_ID})...")

for table_name, df in tables_to_load.items():
    table_id = f"{PROJECT_ID}.{DATASET_ID}.{table_name}"

    try:
        # Envia o DataFrame para o BigQuery
        job = client.load_table_from_dataframe(
            df, table_id, job_config=job_config
        )
        job.result() # Espera o job terminar

        print(f"  [SUCESSO] Tabela '{table_name}' carregada. {job.output_rows} linhas.")

    except Exception as e:
        print(f"  [ERRO] Falha ao carregar a tabela '{table_name}': {e}")

print("\nProcesso ETL (Carga) concluído!")
```

```
Iniciando a carga de dados no BigQuery (Dataset: project_rota01)...
[SUCESSO] Tabela 'dim_tempo' carregada. 1468 linhas.
[SUCESSO] Tabela 'dim_localidade' carregada. 3819 linhas.
[SUCESSO] Tabela 'dim_produto' carregada. 10292 linhas.
[SUCESSO] Tabela 'dim_cliente' carregada. 4873 linhas.
[SUCESSO] Tabela 'fato_vendas' carregada. 51290 linhas.
```

Processo ETL (Carga) concluído!



2.1.8 🔵 Projetar um pipeline em relação à ordem de atualização das tabelas.

Projeto de Pipeline: Ordem de Atualização de Tabelas

O pipeline de atualização garante que os dados sejam carregados no BigQuery de forma correta e eficiente, respeitando as dependências do Modelo Estrela.

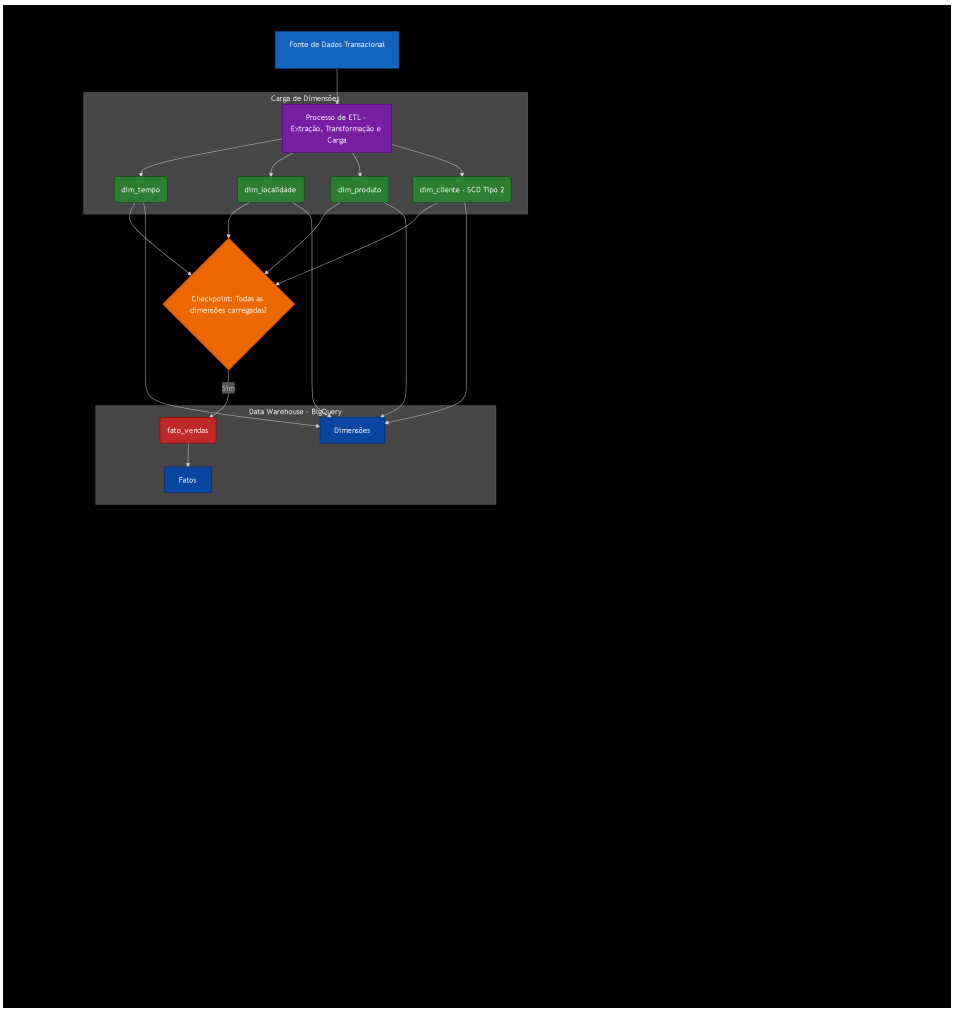
Ordem Proposta

O processo de carga (L) deve seguir esta ordem de prioridade:

Ordem	Tabela	Tipo	Chaves Primárias (PKs) Geradas	Dependências
1ª	**Dim_Tempo**	Independente	`chave_tempo` (AAAAMMDD)	Nenhuma
2ª	**Dim_Localidade**	Independente	`chave_localidade`	Nenhuma
3ª	**Dim_Produto**	Independente	`chave_produto`	Nenhuma
4ª	**Dim_Cliente**	Independente (SCD T2)	`chave_cliente`	Nenhuma
5ª	**Fato_Vendas**	Dependente	N/A	Dim_Tempo, Dim_Localidade, Dim_Produto, Dim_Cliente

```
graph TD
    A[Fonte de Dados Transacional] --> B(Processo de ETL - Extração, Transformação e Carga)

    subgraph C[Carga de Dimensões]
        direction LR
        B --> C1(dim_tempo)
        B --> C2(dim_localidade)
        B --> C3(dim_produto)
        B --> C4(dim_cliente - SCD Tipo 2)
    end

    C1 --> D{Checkpoint: Todas as dimensões carregadas?}
    C2 --> D
    C3 --> D
    C4 --> D

    D -- Sim --> E(fato_vendas)

    subgraph F[Data Warehouse - BigQuery]
```

```
    direction TB
    C1 & C2 & C3 & C4 --> F1[Dimensões]
    E --> F2[Fatos]
end

style A fill:#1565c0,stroke:#003c8f,color:#ffffff
style B fill:#7b1fa2,stroke:#4a0072,color:#ffffff
style C1 fill:#2e7d32,stroke:#005005,color:#ffffff
style C2 fill:#2e7d32,stroke:#005005,color:#ffffff
style C3 fill:#2e7d32,stroke:#005005,color:#ffffff
style C4 fill:#2e7d32,stroke:#005005,color:#ffffff
style D fill:#ef6c00,stroke:#b53d00,color:#ffffff
style E fill:#c62828,stroke:#8e0000,color:#ffffff
style F1 fill:#0d47a1,stroke:#002171,color:#ffffff
style F2 fill:#0d47a1,stroke:#002171,color:#ffffff
```



Justificativa da Ordem de Atualização

Etapas 1, 2, 3 e 4: Prioridade das Dimensões

As tabelas **Dimensão (Tempo, Localidade, Produto e Cliente)** são independentes. Isso significa que a criação da `chave_localidade` não depende da `chave_produto`, e vice-versa.

- **Vantagem da Ordem Paralela/Indiferente:** No código que criamos (usando Pandas no Colab), essas dimensões são criadas no **ambiente de transformação (RAM)**, e o carregamento para o BigQuery ocorre em sequência. Na prática de engenharia de dados, a carga destas 4 tabelas pode ser feita em **paralelo** para economizar tempo.

- **A ordem 1 a 4 é flexível:** O importante é que todas elas estejam **completamente carregadas** antes do Passo 5.

Etapa 5: Última Prioridade (Tabela Fato)

A **Fato_Vendas** é a tabela central e é totalmente **dependente** de todas as chaves substitutas (PKs) das dimensões.

1. A Tabela Fato usa a `chave_tempo` para ligar as datas de Pedido e Envio.
2. Usa a `chave_localidade` para identificar onde a transação ocorreu.
3. Usa a `chave_produto` para identificar o item vendido.
4. Usa a `chave_cliente` para identificar quem comprou (e o segmento da época, no caso do SCD Tipo 2).

Veto do Checkpoint: Se a Tabela Fato for carregada antes das Dimensões, ela não encontrará as Chaves Estrangeiras (`chave_cliente`, `chave_produto`, etc.), e a integridade referencial do seu Data Warehouse será comprometida (o BigQuery aceita FKs, mas a lógica de negócio estaria quebrada).

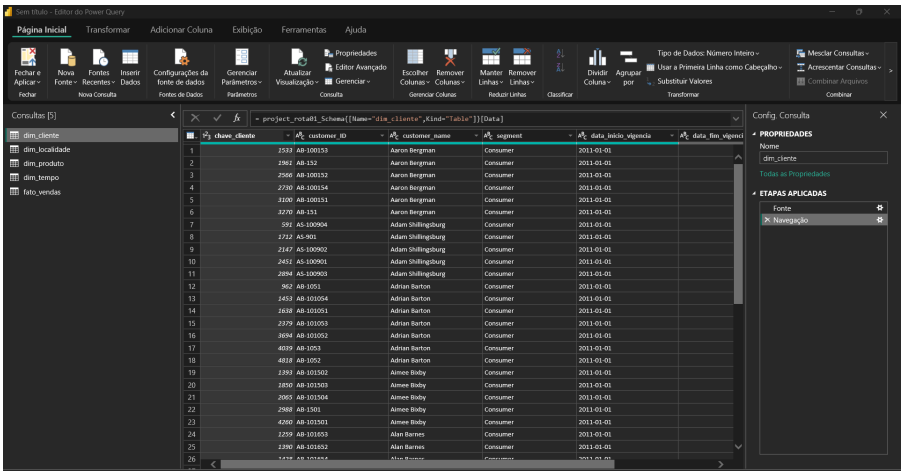
Estrutura do Pipeline de Código (Confirmação)

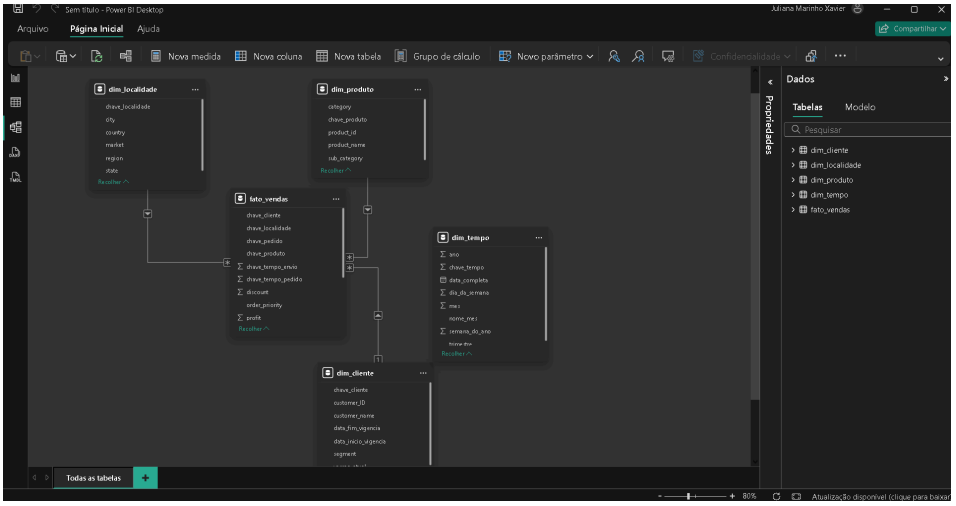
O código Python que implementamos nos passos anteriores já segue rigorosamente essa ordem:

1. **Transformação (T):** Cria os DataFrames: `df_dim_tempo`, `df_dim_localidade`, `df_dim_produto`, `df_dim_cliente`.
2. **Transformação (T):** Cria o DataFrame `df_fato_vendas` usando as chaves das dimensões.
3. **Carga (L):** O *loop* de carregamento para o BigQuery foi intencionalmente escrito na ordem correta:Python

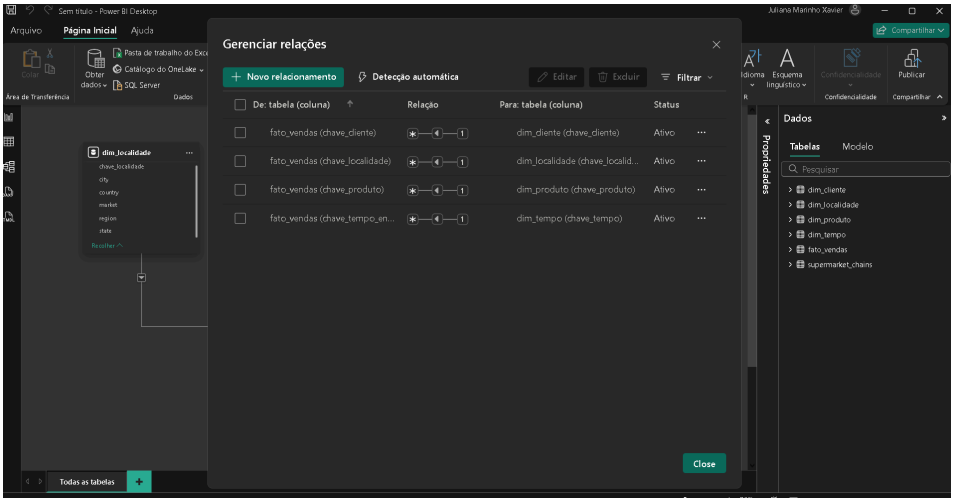
```
# Mapeamento para o carregamento no BigQuery
tables_to_load = {
    "dim_tempo": df_dim_tempo,
    "dim_localidade": df_dim_localidade,
    "dim_produto": df_dim_produto,
    "dim_cliente": df_dim_cliente,
    "fato_vendas": df_fato_vendas # Fato Vendas é sempre o último!
}
# O loop executa essa ordem.
```

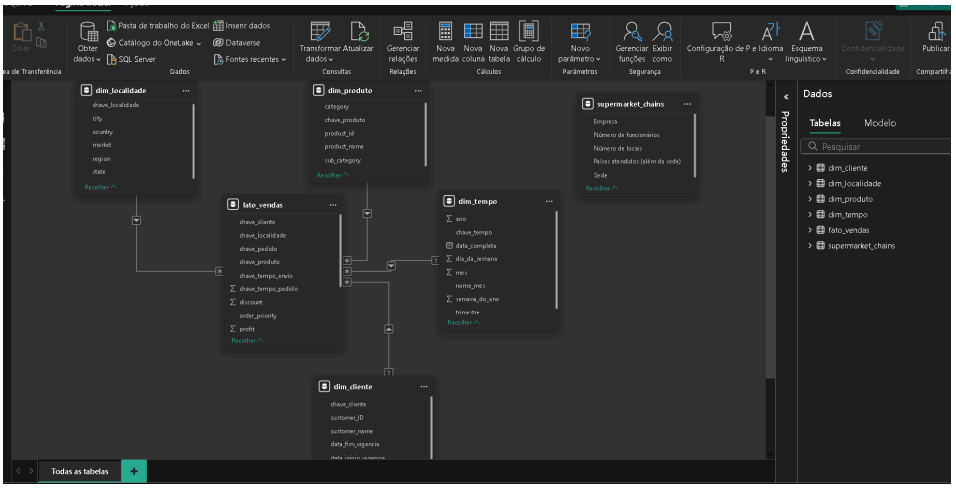
🟩 2.2 Apresentar resultados



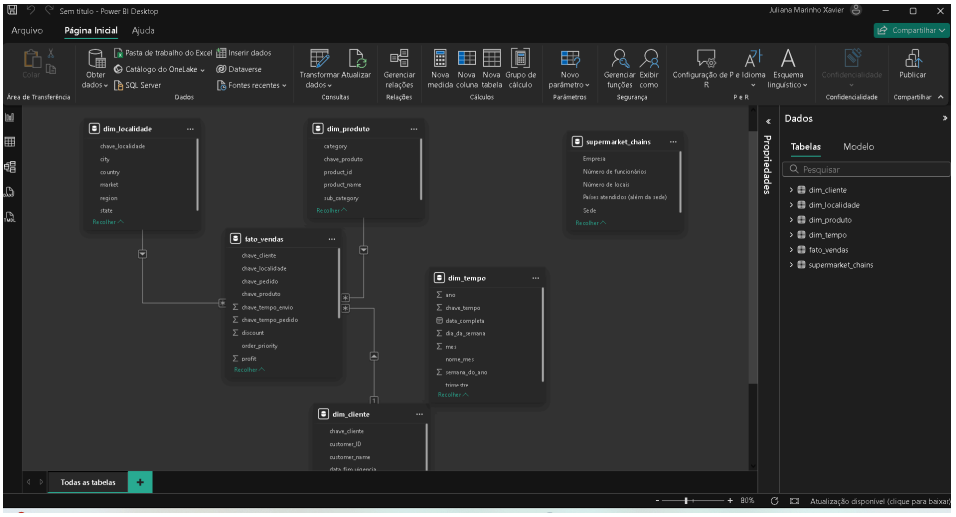


A dim_tempo não se conectou automaticamente, portanto fiz o ajuste manual:





Carga da tabela extraída em web scrapping



Esboço Conceitual do Dashboard Superstore (Modelo Estrela)

O dashboard é dividido em uma estrutura de **3 linhas principais**, seguindo a hierarquia de importância (Visão Geral → Tendência → Detalhe).

1. Área Superior: Filtros e KPIs (Visão Geral)

Esta área deve ocupar a largura total do dashboard e é o primeiro lugar onde o usuário olha.

BLOCO ESQUERDO (Filtros)	BLOCO CENTRAL (KPIs)	BLOCO DIREITO (KPIs)
Filtros Globais (Slicers):	**Cartão 1:** Vendas Totais (`SUM(sales)`)	**Cartão 3:** Lucro Total (`SUM(profit)`)
Ano (`ano` da Dim_Tempo)	**Cartão 2:** Quantidade Total (`SUM(quantity)`)	**Cartão 4:** Margem de Lucro (`Profit` / `Sales`)
País (`country` da Dim_Localidade)	**Cartão 5:** Custo Médio de Envio (`AVG(shipping_cost)`)	**Cartão 6:** Modo de Envio mais Usado (`ship_mode`)

2. Área Central: Performance Temporal (Tendências)

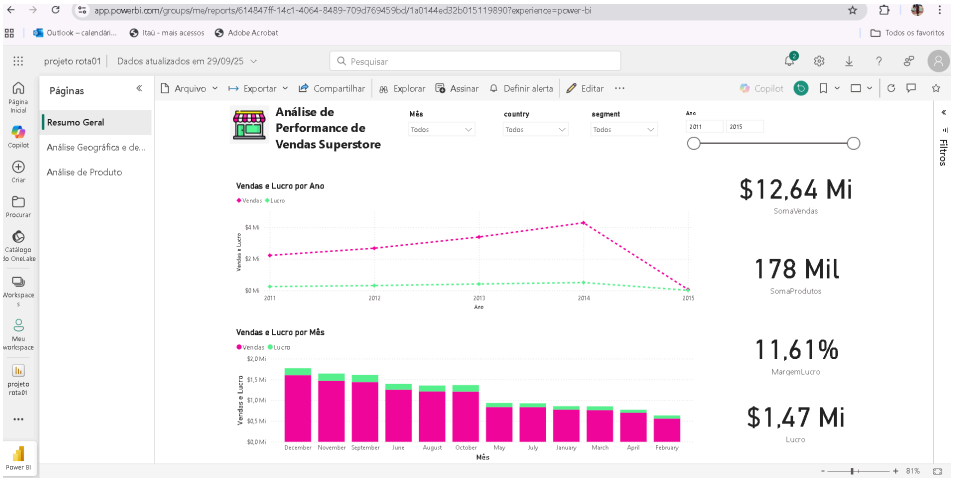
Esta é a área mais importante para a análise de tendências e deve ter os gráficos maiores.

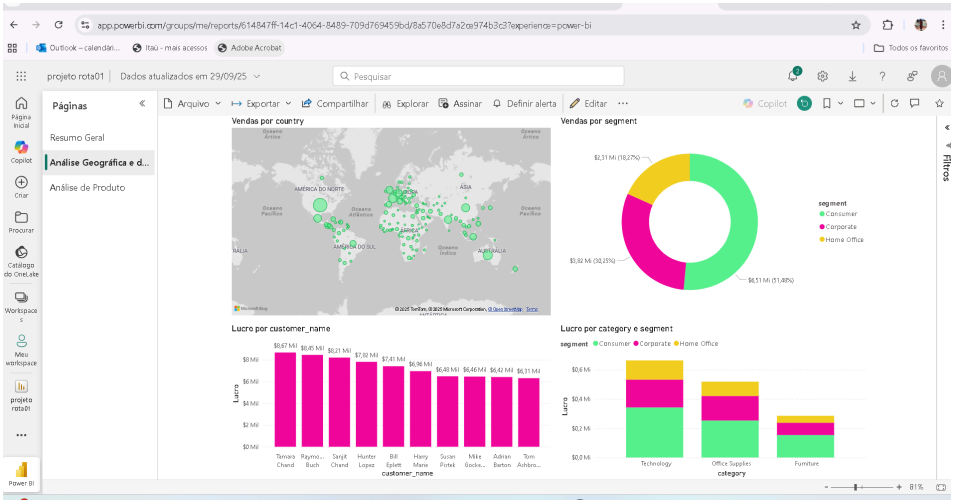
BLOCO ESQUERDO GRANDE	BLOCO DIREITO MÉDIO
Gráfico de Linha (Dual Axis): Vendas vs. Lucro ao Longo do Tempo	**Gráfico de Barras/Colunas:** Sazonalidade de Vendas por Mês
Eixo X: Data (`data_completa` - hierarquia de Ano/Mês)	**Eixo X:** Mês (`nome_mes` da Dim_Tempo)
Eixo Y1 (Vendas): `SUM(sales)`	**Eixo Y:** `SUM(sales)`
Eixo Y2 (Lucro): `SUM(profit)`	**Foco:** Identificar os meses de pico (sazonalidade) de forma rápida.
Foco: Mostrar a tendência geral e a correlação entre vendas e lucro.	

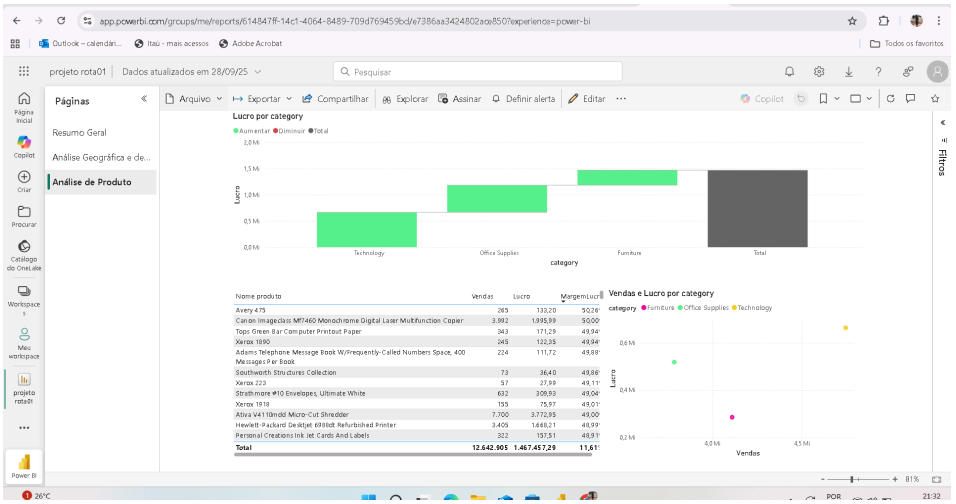
3. Área Inferior: Segmentação e Detalhe (Onde e O Quê)

Esta área detalha as dimensões geográficas, de cliente e de produto.

BLOCO ESQUERDO	BLOCO CENTRAL	BLOCO DIREITO
Mapa Coroplético (ou de Pontos): Vendas por Localidade	**Gráfico de Rosca/Pizza:** Vendas por Segmento de Cliente	**Gráfico de Barras (Hierárquico):** Lucro por Categoria de Produto
Localização: `state` ou `country` da Dim_Localidade	**Legenda:** `segment` da Dim_Cliente	**Eixo Y:** `category` e `sub_category` (Permite **Drill-Down**)
Valores: `SUM(sales)`	**Valores:** `SUM(sales)`	**Eixo X:** `SUM(profit)`
Foco: O mapa visualiza rapidamente as áreas de maior receita.	**Foco:** Distribuição da receita entre os segmentos (*Consumer*, *Corporate*, *Home Office*).	**Foco:** Identificar os produtos mais lucrativos e permitir que o usuário explore a fundo (clicando na Categoria).







projeto rota01.pbix