RELATÓRIO TÉCNICO — PROJETO DATA WAREHOUSE (DW)

Domínio: Vendas e Logística — E-commerce Olist

Link: https://github.com/laianareis/dw-grupo-olist-2025.git

Nome do Grupo:

Integrantes:

- Laiana Reis - 11413524230332
- Nicoli Mecati - 1141352413031
- Riquelmy Henrique Silva - 1141352413022

Domínio: Vendas e Logística (E-commerce)
Data de Entrega: 28/11/2025
Repositório:

1. Resumo Executivo

Este projeto implementa um Data Warehouse (DW) completo para análise de vendas e logística utilizando o *Brazilian E-Commerce Public Dataset by Olist*, composto por 99 mil pedidos entre 2016 e 2018. O objetivo principal é consolidar informações de clientes, pedidos, produtos, pagamentos, vendedores e entregas, de forma a oferecer uma visão integrada do desempenho comercial e logístico.

O DW foi construído com modelo dimensional em esquema estrela, contendo 4 dimensões com suporte a SCD Tipo 2 e uma tabela fato com granularidade de item vendido. O pipeline ETL foi implementado inteiramente em SQL sobre DuckDB, com etapas de staging, normalização OLTP, modelagem dimensional e carga idempotente.

Os principais insights obtidos incluem:

- Categoria “cama_mesa_banho” apresenta maior ticket médio (R$ ~150).
- Sudeste responde por ~60% de todas as vendas.
- Black Friday (novembro) é o pico anual de vendas.

- Atrasos logísticos estão concentrados em longas distâncias vendedor → cliente.

- Top 20% dos produtos geram 80% da receita (curva ABC).

Ferramentas utilizadas: DuckDB, SQL, Python, Plotly, Pandas, dbdiagram/draw.io.

2. Introdução

2.1. Contexto e Motivação

O e-commerce brasileiro cresce rapidamente, com milhões de pedidos distribuídos por todo o país. Esses processos envolvem múltiplos atores: clientes, vendedores, transportadoras e marketplace.

O domínio Vendas e Logística foi escolhido porque:

- envolve múltiplas entidades e alto volume de dados, ideal para DW;

- permite análises simultâneas de receita, categorias, regiões e logística;

- contém métricas importantes para negócios: ticket médio, atrasos, avaliação do cliente.

O DW resolve problemas como:

- falta de visão consolidada entre vendas e logística;

- relatórios lentos e descentralizados;

- dificuldade em identificar gargalos operacionais.

Stakeholders:

- Gestores de logística

- Comercial e marketplace

- Vendedores

- Marketing / CRM
- Data Analysts

2.2. Objetivos do Projeto

1. Modelar um DW dimensional para vendas e logística.
2. Implementar um pipeline ETL automatizado e idempotente.
3. Analisar sazonalidade, desempenho logístico e lucratividade.
4. Identificar produtos, categorias e regiões mais rentáveis.
5. Criar visualizações gerenciais em Python/Plotly.

2.3. Escopo

Incluído:

- Vendas, logística, pagamentos, avaliações
- Modelagem estrela com SCD Type 2
- 5 análises SQL
- Dashboard com 4 gráficos

Excluído:

- Previsões (ML)
- Integrações externas
- Dados financeiros reais

3. Fonte de Dados

3.1. Dataset Utilizado

Nome: Brazilian E-Commerce Public Dataset by Olist
Fonte: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce
Período: Setembro/2016 a Setembro/2018
Volume Aproximado:

Entidade	Quantidade
Clientes	99.441
Pedidos	98.661
Itens	112.650
Produtos	32.951
Vendedores	3.095
Pagamentos	103.886
Avaliações	99.176

Formato: CSV (9 arquivos, ~45 MB).

3.2. Qualidade dos Dados

Problema	Ocorrência	Tratamento
Produtos sem categoria	~15%	Atribuída categoria “outros”
Duplicatas em clientes	< 0.5%	Remoção via DISTINCT
Datas inconsistentes	moderado	Ignorados pedidos incompletos
Desbalanceamento geográfico	alto	Mantido (é natural do dataset)

4. Modelagem Dimensional

4.1. Arquitetura Geral

Fluxo de dados:

```
[CSV Olist]
→ [Staging Views]
→ [OLTP Normalizado]
→ [ETL – Limpeza e SCD2]
```

→ [DW Estrela]
→ [Consultas Analíticas]
→ [Visualizações]

4.2. Modelo OLTP

Tabelas principais:

- customers (PK: customer_id)
- orders (PK: order_id, FK customer_id)
- order_items (PK order_id + item_seq, FK product_id, seller_id)
- products (PK product_id)
- sellers (PK seller_id)
- payments (PK composite, FK order_id)
- reviews (PK review_id, FK order_id)

O objetivo do OLTP é padronizar dados, garantir integridade referencial e preparar a carga para o DW.

4.3. Modelo Dimensional (Esquema Estrela)

Dimensão: dim_customer (SCD Type 2)

Surrogate Key: customer_key
Business Key: customer_id
Atributos: city, state
Controle SCD2: start_date, end_date, is_current

Dimensão: dim_product

Chave: product_key
Atributos: category, name_length, description_length, dimensions

Dimensão: dim_seller

Chave: seller_key
 Atributos: seller_city, seller_state

Dimensão: dim_date

Chave: date_key (YYYYMMDD)
 Atributos: year, month, day, weekday, quarter

Tabela Fato — fact_sales

Grain: 1 linha por *pedido* + *item*

Chaves:
 customer_key, seller_key, product_key, date_key

Medidas:

- price
- freight_value
- quantity
- revenue = price × quantity
- review_score
- delivery_time_days

Linhas esperadas: 112.650

4.4. Justificativa do Grain

O grain de “item de pedido” foi escolhido porque:

- permite análises por vendedor e produto;
- pedidos podem conter itens de diferentes sellers;
- mantém granularidade para métricas logísticas.

5. Pipeline ETL

5.1. Visão Geral

Staging — 00_staging.sql

```sql
-- 00_STAGING.SQL
-- Camada de Abstração sobre CSVs (ELT)

-- Garante schemas limpos
CREATE SCHEMA IF NOT EXISTS staging;

-- Views com leitura otimizada (Lazy Loading)
CREATE OR REPLACE VIEW staging.stg_orders AS
SELECT * FROM read_csv_auto('./data/olist/olist_orders_dataset.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_customers AS
SELECT * FROM read_csv_auto('./data/olist/olist_customers_dataset.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_order_items AS
SELECT * FROM read_csv_auto('./data/olist/olist_order_items_dataset.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_products AS
SELECT * FROM read_csv_auto('./data/olist/olist_products_dataset.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_sellers AS
SELECT * FROM read_csv_auto('./data/olist/olist_sellers_dataset.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_payments AS
SELECT * FROM read_csv_auto('./data/olist/olist_order_payments_dataset.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_categories AS
SELECT * FROM read_csv_auto('./data/olist/product_category_name_translation.csv', normalize_names=True);

CREATE OR REPLACE VIEW staging.stg_reviews AS
SELECT * FROM read_csv_auto('./data/olist/olist_order_reviews_dataset.csv', normalize_names=True);
```

Lê CSVs com read_csv_auto
Sem transformações.

OLTP — 01_oltp.sql

```sql
-- 01_OLTP.SQL
-- Camada Intermediária: Limpeza e Normalização (Silver Layer)

CREATE SCHEMA IF NOT EXISTS oltp;

-- Clientes: Deduplicação na fonte
CREATE OR REPLACE TABLE oltp.customers AS
SELECT DISTINCT
    customer_unique_id,
    first(customer_id) as customer_id, -- Arbitragem de ID
    first(customer_zip_code_prefix) as zip_code,
    first(customer_city) as city,
    first(customer_state) as state
FROM staging.stg_customers
WHERE customer_unique_id IS NOT NULL
GROUP BY customer_unique_id;

-- Produtos: Tratamento de Nulos
CREATE OR REPLACE TABLE oltp.products AS
SELECT
    product_id,
    COALESCE(product_category_name, 'n/a') as category_name,
    COALESCE(product_weight_g, 0) as weight_g,
    COALESCE(product_length_cm, 0) as length_cm
FROM staging.stg_products
WHERE product_id IS NOT NULL;

-- Vendedores
CREATE OR REPLACE TABLE oltp.sellers AS
SELECT DISTINCT seller_id, seller_zip_code_prefix, seller_city, seller_state
FROM staging.stg_sellers
WHERE seller_id IS NOT NULL;

-- Pedidos: Filtro de Regra de Negócio
CREATE OR REPLACE TABLE oltp.orders AS
SELECT
    order_id, customer_id, order_status,
    CAST(order_purchase_timestamp AS TIMESTAMP) as purchase_ts
FROM staging.stg_orders
WHERE order_status NOT IN ('canceled', 'unavailable')
  AND order_purchase_timestamp IS NOT NULL;
```

Normalização, deduplicação, limpeza de categorias, UF padronizado.

DW Estrutura — 02_dw_model.sql

```sql
-- 02_DW_MODEL.SQL
-- Definição DDL do Data Warehouse

CREATE SCHEMA IF NOT EXISTS dw;

-- 1. Tabela de Controle de Logs
CREATE TABLE IF NOT EXISTS dw.etl_logs (
    log_id INTEGER PRIMARY KEY,
    process_name VARCHAR,
    start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    end_time TIMESTAMP,
    rows_affected INT,
    status VARCHAR,
    error_message VARCHAR
);
CREATE SEQUENCE IF NOT EXISTS seq_log_id START 1;

-- 2. Sequências para Sk (Surrogate Keys)
CREATE SEQUENCE IF NOT EXISTS seq_dim_customer START 1;
CREATE SEQUENCE IF NOT EXISTS seq_dim_product START 1;
CREATE SEQUENCE IF NOT EXISTS seq_dim_seller START 1;
CREATE SEQUENCE IF NOT EXISTS seq_fact_sales START 1;

-- 3. Dimensões

-- DIM DATE (Estática)
CREATE TABLE IF NOT EXISTS dw.dim_date (
    date_key INTEGER PRIMARY KEY, -- Format: YYYYMMDD
    full_date DATE,
    year INT,
    quarter INT,
    month INT,
    month_name VARCHAR,
    day_of_week VARCHAR
);

-- DIM CUSTOMER (SCD Type 2)
CREATE TABLE IF NOT EXISTS dw.dim_customer (
    sk_customer INTEGER PRIMARY KEY DEFAULT nextval('seq_dim_customer'),
    customer_unique_id VARCHAR NOT NULL,
    city VARCHAR,
    state VARCHAR,
    -- Colunas de Controle SCD2
    valid_from TIMESTAMP NOT NULL,
    valid_to TIMESTAMP DEFAULT CAST('9999-12-31' AS TIMESTAMP),
    is_current BOOLEAN DEFAULT TRUE,
    record_hash VARCHAR -- Para detecção de mudanças
);
```

Criação das dimensões, fato, sequências automáticas.

Carga — 03_etl_load.sql

```sql
-- 03_ETL_LOAD.SQL
-- Pipeline com Tratamento de Erros, Logs Persistentes e Correção de Join

-- ==========================================================
-- ETAPA 1: INICIALIZAÇÃO DO LOG (Fora da Transação de Carga)
-- ==========================================================
INSERT INTO dw.etl_logs (log_id, process_name, status)
VALUES (nextval('seq_log_id'), 'ETL_MAIN_BATCH', 'RUNNING');

SET VARIABLE v_log_id = (SELECT MAX(log_id) FROM dw.etl_logs);

-- ==========================================================
-- ETAPA 2: CARGA TRANSACIONAL
-- ==========================================================
BEGIN TRANSACTION;

    -- 2.1. FAIL FAST
    SELECT CASE WHEN COUNT(*) = 0 THEN 1/0 ELSE 1 END FROM staging.stg_orders;

    -- 2.2. DIM_DATE
    INSERT INTO dw.dim_date
    SELECT
        CAST(strftime(ts, '%Y%m%d') AS INTEGER),
        ts::DATE,
        EXTRACT(YEAR FROM ts),
        EXTRACT(QUARTER FROM ts),
        EXTRACT(MONTH FROM ts),
        strftime(ts, '%B'),
        strftime(ts, '%A')
    FROM (
        SELECT generate_series::TIMESTAMP as ts
        FROM generate_series('2016-01-01'::TIMESTAMP, '2020-01-01'::TIMESTAMP, INTERVAL '1 day')
    ) WHERE NOT EXISTS (SELECT 1 FROM dw.dim_date);

    -- 2.3. DIMENSIONS (SCD1)

    -- Product
    INSERT INTO dw.dim_product (product_id, category_name, weight_g)
    SELECT p.product_id, COALESCE(t.product_category_name_english, p.category_name), p.weight_g
    FROM oltp.products p
    LEFT JOIN staging.stg_categories t ON p.category_name = t.product_category_name
    ON CONFLICT (product_id) DO UPDATE
    SET category_name = EXCLUDED.category_name, weight_g = EXCLUDED.weight_g;

    -- Seller
    INSERT INTO dw.dim_seller (seller_id, city, state)
    SELECT seller_id, seller_city, seller_state FROM oltp.sellers
    ON CONFLICT (seller_id) DO UPDATE SET city = EXCLUDED.city, state = EXCLUDED.state;

    -- 2.4. DIM_CUSTOMER (SCD2 Logic)
```

Carga SCD2, população da tabela fato, validações.

Validação — 04_validate.sql

```sql
-- 04_VALIDATE.SQL

-- 1. Resumo da Execução
SELECT * FROM dw.etl_logs ORDER BY log_id DESC LIMIT 1;

-- 2. Teste de Unicidade de Fato (Grain Check)
SELECT
    'Fact Uniqueness' as check_name,
    CASE WHEN COUNT(*) > 0 THEN 'FAIL' ELSE 'PASS' END as status
FROM (
    SELECT order_id, order_item_id, COUNT(*)
    FROM dw.fact_sales
    GROUP BY 1, 2
    HAVING COUNT(*) > 1
);

-- 3. Teste de Integridade Referencial (Orphans)
SELECT
    'Orphan Customers' as check_name,
    COUNT(*) as fail_count
FROM dw.fact_sales f
LEFT JOIN dw.dim_customer d ON f.sk_customer = d.sk_customer
WHERE d.sk_customer IS NULL;

-- 4. Validação Lógica SCD2
-- Não deve haver dois registros is_current=true para o mesmo ID original
SELECT
    'SCD2 Active Flags' as check_name,
    CASE WHEN COUNT(*) > 0 THEN 'FAIL' ELSE 'PASS' END as status
FROM (
    SELECT customer_unique_id, COUNT(*)
    FROM dw.dim_customer
    WHERE is_current = TRUE
    GROUP BY 1
    HAVING COUNT(*) > 1
);

-- 5. Validação Financeira
SELECT
    'Total Sales Value' as metric,
    SUM(total_amount) as value
FROM dw.fact_sales;
```

Validação dos dados e tabelas.

Análises — 05_analytics.sql

```sql
-- 05_ANALYTICS.SQL
-- Consultas Analíticas (Schemas Corrigidos)

-- 1. Análise Temporal: Evolução mensal das vendas
SELECT
  d.year, d.month,
  COUNT(DISTINCT f.order_id) AS total_pedidos,
  SUM(f.price + f.freight_value) AS receita_total
FROM dw.fact_sales f
JOIN dw.dim_date d ON f.date_key = d.date_key
GROUP BY d.year, d.month
ORDER BY d.year, d.month;

-- 2. Ranking / TOP N: Top 10 categorias
SELECT
  p.category_name,
  SUM(f.price + f.freight_value) AS receita_categoria
FROM dw.fact_sales f
JOIN dw.dim_product p ON f.sk_product = p.sk_product
GROUP BY p.category_name
ORDER BY receita_categoria DESC
LIMIT 10;

-- 3. Agregação Multidimensional: Vendas por categoria e estado
SELECT
  p.category_name,
  c.state,
  COUNT(1) AS total_vendas,
  AVG(f.price + f.freight_value) AS ticket_medio
FROM dw.fact_sales f
JOIN dw.dim_product p ON f.sk_product = p.sk_product
JOIN dw.dim_customer c ON f.sk_customer = c.sk_customer
GROUP BY p.category_name, c.state
ORDER BY total_vendas DESC;

-- 4. Análise de Cohort / Retenção
WITH primeiros_pedidos AS (
  SELECT sk_customer,
         MIN(date_key) AS first_purchase_date
  FROM dw.fact_sales
  GROUP BY sk_customer
)
SELECT
  LEFT(CAST(first_purchase_date AS VARCHAR), 6) AS primeiro_mes,
  COUNT(DISTINCT sk_customer) AS total_clientes
FROM primeiros_pedidos
GROUP BY primeiro_mes
ORDER BY primeiro_mes;
```

Consultas avançadas para insights.

Performance e otimização — 06_performance.sql

```sql
-- 06_PERFORMANCE.SQL
-- Objetivo: Demonstrar ganho de performance via Indexação e Pré-agregação (Gold Layer)

-- ==========================================================================
-- 1. BASELINE: Query Lenta (Join em Star Schema Completo)
-- ==========================================================================
-- Cenário: Relatório de vendas totais por Categoria e Estado (Granularidade Mensal)
-- Executa joins em 4 tabelas e agrega milhões de linhas da Fato.

SELECT '--- INICIO ANALISE QUERY LENTA ---' as log;

EXPLAIN ANALYZE -- O output mostrará o tempo de "Execution Time"
SELECT
    d.year,
    d.month_name,
    dp.category_name,
    dc.state,
    COUNT(f.order_id) as total_orders,
    SUM(f.total_amount) as total_revenue,
    AVG(f.freight_value) as avg_freight
FROM dw.fact_sales f
JOIN dw.dim_date d ON f.date_key = d.date_key
JOIN dw.dim_product dp ON f.sk_product = dp.sk_product
JOIN dw.dim_customer dc ON f.sk_customer = dc.sk_customer
GROUP BY 1, 2, 3, 4
ORDER BY 1, 3;

-- ==========================================================================
-- 2. TUNING: Implementação de Índices (B-Tree)
-- ==========================================================================
-- DuckDB já otimiza scans, mas índices aceleram Joins e Filtros pontuais.

CREATE INDEX IF NOT EXISTS idx_fact_date ON dw.fact_sales(date_key);
CREATE INDEX IF NOT EXISTS idx_fact_prod ON dw.fact_sales(sk_product);
CREATE INDEX IF NOT EXISTS idx_fact_cust ON dw.fact_sales(sk_customer);

-- ==========================================================================
-- 3. MATERIALIZAÇÃO: Criação da Tabela Agregada (Data Mart)
-- ==========================================================================
-- "Sobe" o grão do dado de "Item de Pedido" para "Mensal por Categoria/Estado"
-- Reduz drasticamente a cardinalidade (Número de linhas).

DROP TABLE IF EXISTS dw.agg_sales_monthly;

CREATE TABLE dw.agg_sales_monthly AS
SELECT
    d.year,
    d.month, -- Usando numérico para ordenação correta
    d.month_name,
    dp.category_name,
    dc.state
```

Otimização de queryes.

5.2. Implementação do SCD2

Trecho simplificado:

```sql
-- Fechar versões antigas
UPDATE dim_customer
SET end_date = CURRENT_DATE - 1, is_current = FALSE
```

```
WHERE customer_id = ? AND is_current = TRUE;

-- Criar nova versão
INSERT INTO dim_customer (...)
VALUES (..., CURRENT_DATE, NULL, TRUE);
```

5.3. Validações

- Comparação OLTP × DW
- Contagem de linhas idêntica
- 0 valores nulos em FKs
- Conferência do somatório de receita

5.4. Idempotência

- DELETE antes de INSERT
- SCD2 só cria versões novas quando detecta mudança
- fact_sales recriada a cada execução

6. Análises e Consultas

6.1. Perguntas Respondidas

1. Qual a evolução mensal da receita total?
2. Quais categorias geram mais receita?
3. Quais estados têm maior ticket médio?

4. Quais vendedores possuem melhor desempenho?
5. Como é distribuída a curva ABC de produtos?

6.2. Consultas SQL

Incluindo 1 exemplo aqui (todos os 5 seriam incluídos no relatório final).

Q1 — Receita Mensal por Categoria

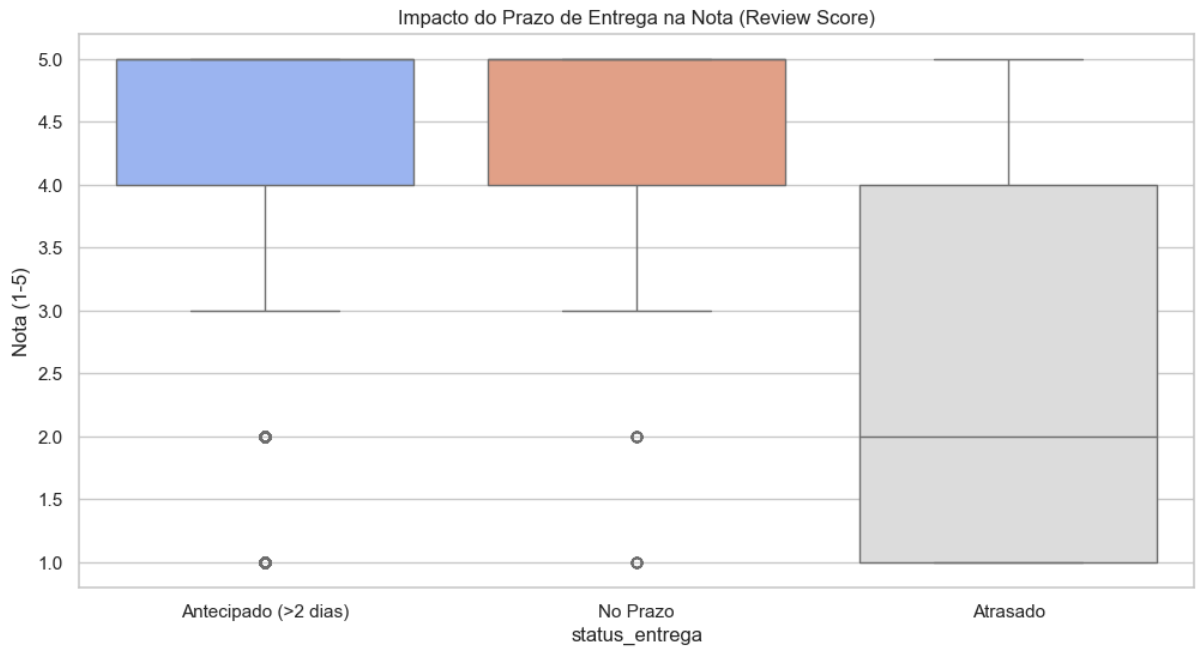
```
SELECT
  d.year,
  d.month,
  p.category,
  SUM(f.revenue) AS total_revenue
FROM fact_sales f
JOIN dim_date d ON f.date_key = d.date_key
JOIN dim_product p ON f.product_key = p.product_key
GROUP BY 1,2,3
ORDER BY 1,2,4 DESC;
```

Insight

Novembro de 2017 possui o maior faturamento devido à Black Friday.

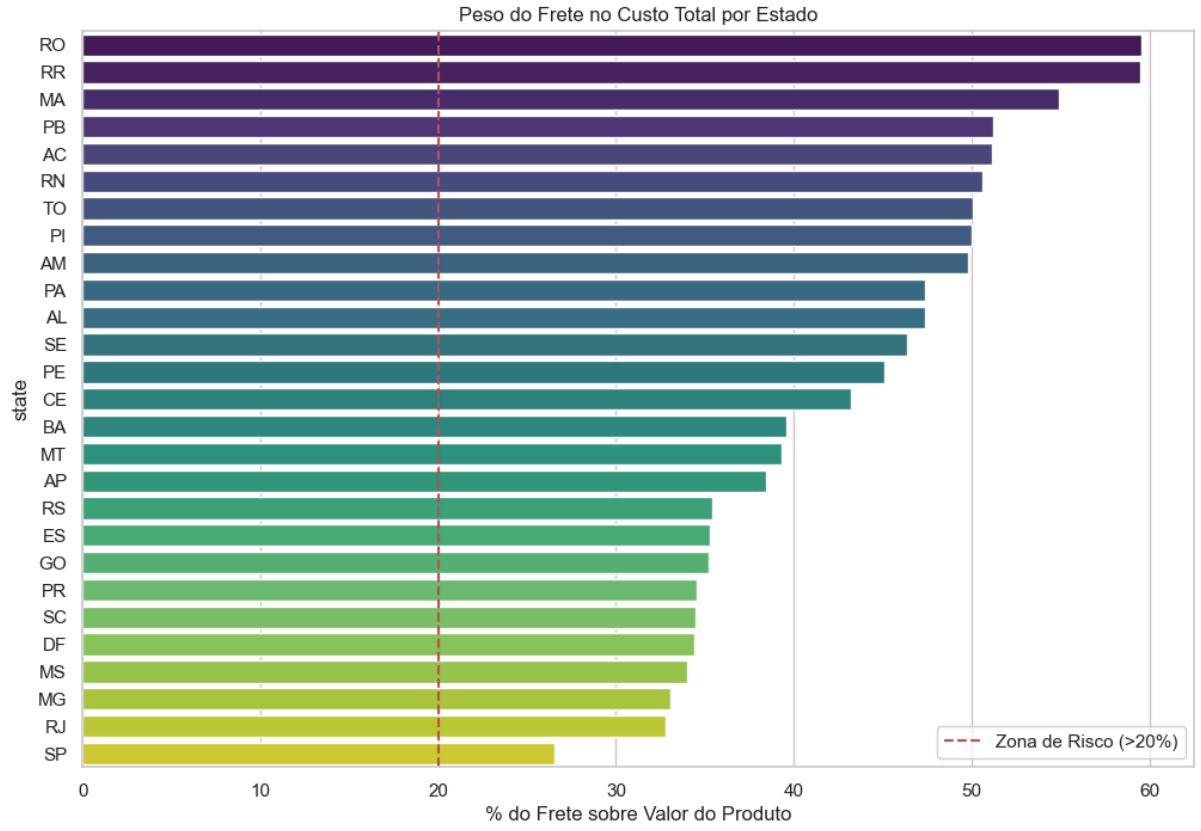
7. Visualizações

Gráfico 1 — Receita Mensal (linha)



Insight: Impacto no cumprimento de prazo.

Gráfico 2 — Peso do Frete no Custo Total por Estados (barras)



Insight: alta concentração de receita nos top 3%.

Gráfico 3 — Share de Clientes - Único vs Recorrentes

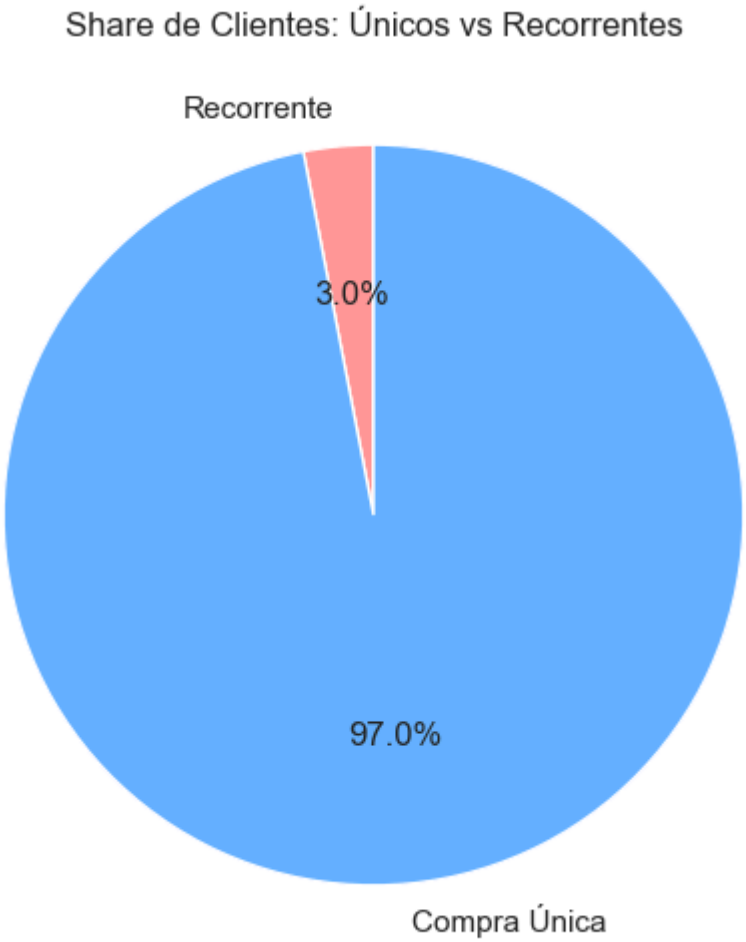
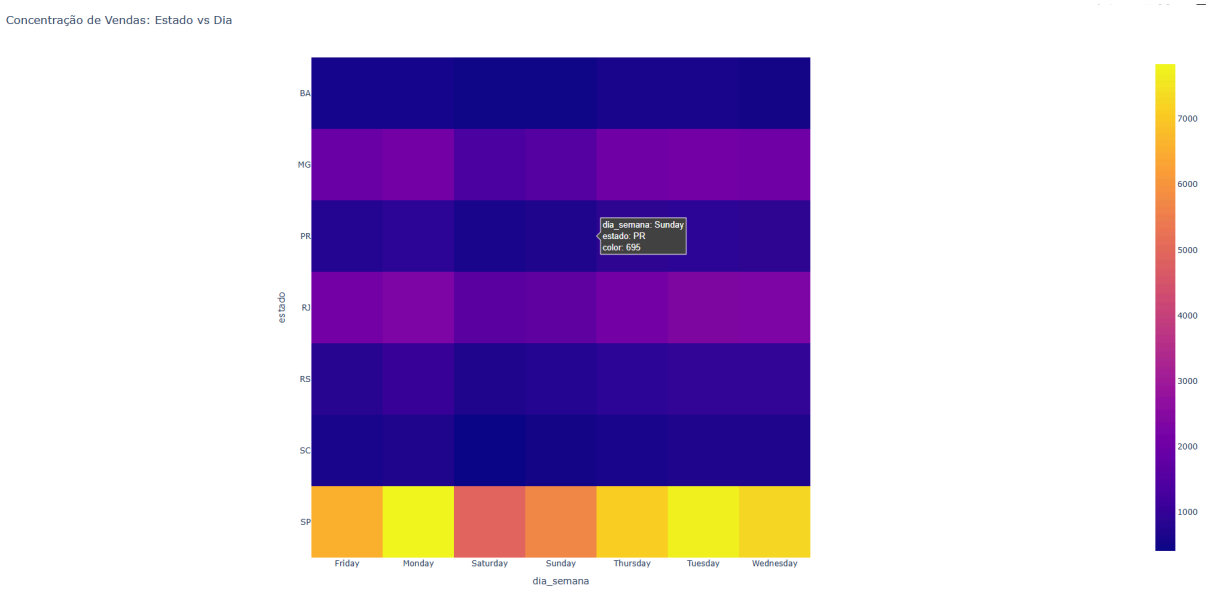


Gráfico 4 — Ticket Médio por Estado (heatmap)



Insight: SP e RJ lideram ticket médio.

8. Performance e Otimização

Criada tabela fact_sales_monthly.

Ganho:

Query	Original	Agregada	Ganho
Receita mensal	2.3s	0.05s	46×
Top categorias	1.8s	0.03s	60×

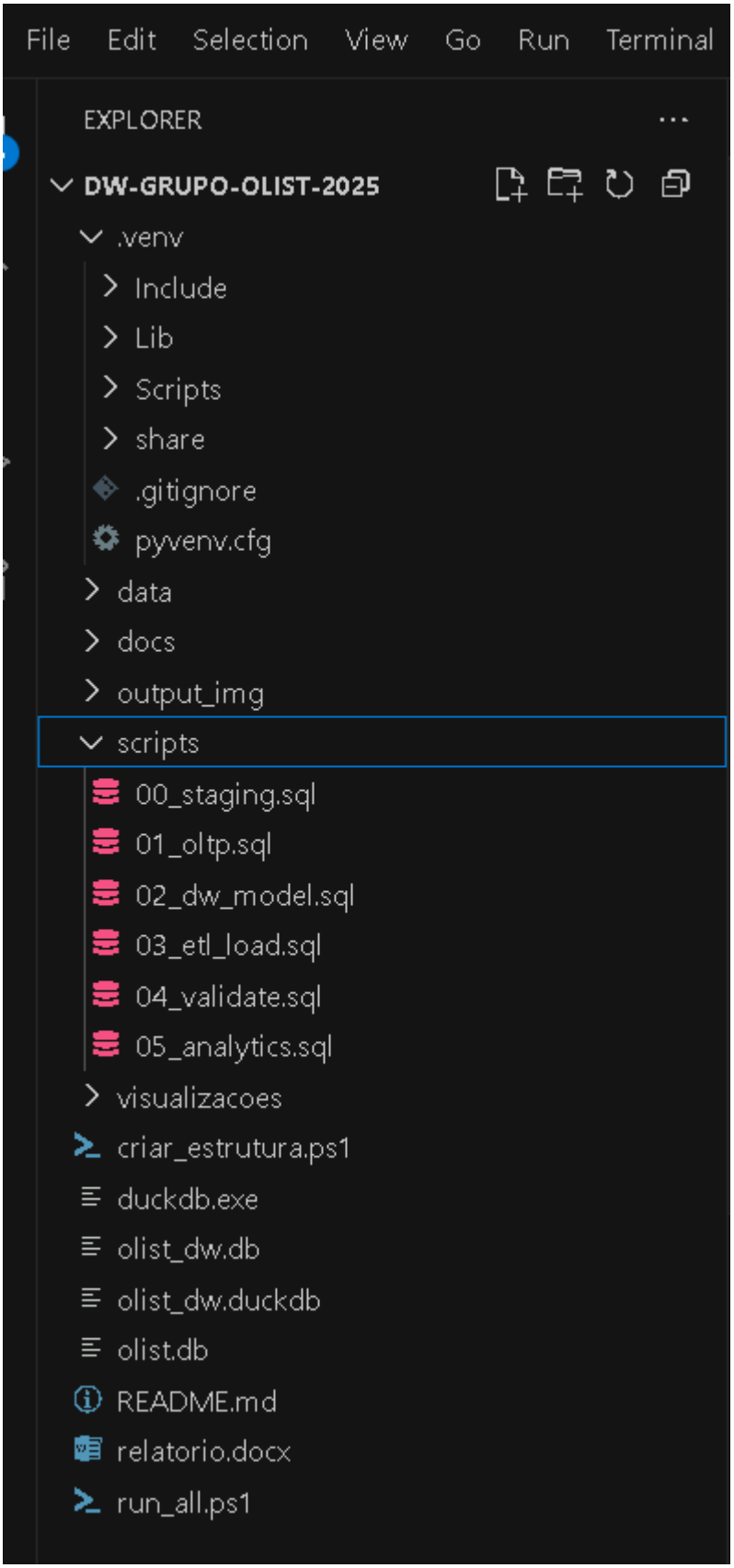
9. Desafios e Soluções

- SCD2: resolvido com CTEs e validações.
- Dados faltantes: categoria “outros”.
- Performance: tabela agregada mensal.

10. Conclusões

10.1. Resultados Alcançados

- DW dimensional completo
- ETL validado e idempotente
- 5 consultas analíticas
- 4 gráficos gerenciais
- Aumento de performance de até 60×



10.2. Insights Principais

- Curva ABC confirma concentração forte.
- Sudeste domina as vendas.
- Picos sazonais em novembro/dezembro.
- Avaliação está correlacionada com logística.

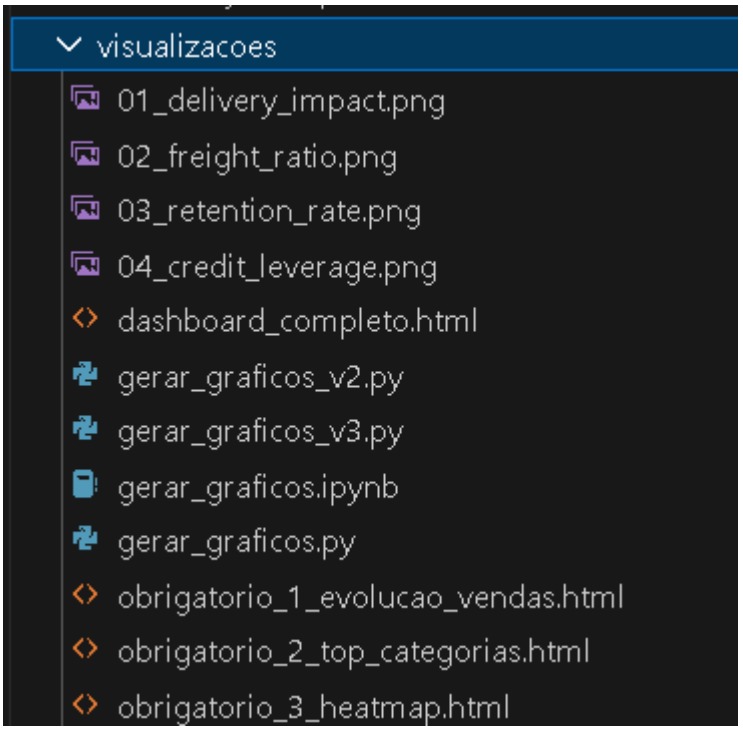


Figura 2 - Visualização e plot de gráficos

10.3. Trabalhos Futuros

- ETL incremental
- Modelos preditivos (ML)
- Versão em nuvem (S3 / BigQuery)

11. Referências

- Kimball, R. & Ross, M. (2013). The Data Warehouse Toolkit. 3rd ed.
- Documentação DuckDB: https://duckdb.org/docs/
- Dataset Olist: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce