Infraestrutura de Dados com Apache Cassandra

Projeto de Disciplina — Arquitetura de dados para marketplace de varejo com foco em análise em larga escala.

Aluno: Caio Barroso **Disciplina:** Banco de Dados Não Relacionais — Apache Cassandra **Ferramentas:** Cassandra · Docker · Python · Pandas

CASSANDRA NOSQL ETL DOCKER PYTHON DATA ANALYTICS

Este documento descreve, de forma integrada, os aspectos conceituais (SQL vs NoSQL e Cassandra), o planejamento de infraestrutura, a configuração do ambiente, o processo de carga de dados, a preparação analítica em Python e as visualizações geradas para responder a perguntas de negócio de um marketplace de varejo.

1. Bancos de Dados SQL vs NoSQL

A disciplina parte da comparação entre modelos relacionais (SQL) e modelos não relacionais (NoSQL), pois a escolha de Cassandra só faz sentido quando se entende o tipo de problema que ele resolve.

1.1. SQL (Relacional)

Bancos de dados relacionais (como PostgreSQL, MySQL, SQL Server) utilizam um esquema rígido, tabelas normalizadas e linguagem SQL padronizada. São ideais para:

- Transações ACID fortes (sistemas financeiros, ERPs, CRMs).
- Relacionamentos complexos entre entidades (JOINs frequentes).
- Registros com estrutura estável e bem definida.

> **Exemplo típico de uso (SQL)**
>
> Sistema bancário com contas, clientes, transferências e saldos, exigindo consistência forte e integridade referencial.

1.2. NoSQL

Bancos NoSQL foram projetados para cenários de alta escala, volumes massivos de dados, distribuição geográfica e esquemas mais flexíveis. Existem vários subtipos:

- **Chave-valor:** Redis, DynamoDB.
- **Documento:** MongoDB, CouchDB.
- **Colunar distribuído:** Apache Cassandra, HBase.
- **Grafos:** Neo4j, JanusGraph.

Aspecto	SQL	NoSQL
Esquema	Rígido, definido antes da carga	Mais flexível, orientado a uso
Escalabilidade	Vertical (scale-up)	Horizontal (scale-out)
Consistência	ACID forte	Configurável (eventual, tunável)
Consultas	JOINs, ad-hoc SQL	Consultas otimizadas por chave e partição

> **Exemplo típico de uso (NoSQL / Cassandra)**
>
> Plataforma de marketplace com milhões de pedidos por dia, replicados em vários datacenters, onde a prioridade é alta disponibilidade e baixa latência para leitura de relatórios agregados de vendas, mesmo em caso de falha de nós isolados.

2. Introdução ao Apache Cassandra

Apache Cassandra é um banco de dados NoSQL distribuído, orientado a colunas, projetado para alta disponibilidade, escalabilidade horizontal e tolerância a falhas. Ele é amplamente utilizado em sistemas que exigem gravações intensas e leitura em baixa latência, em múltiplas regiões.

2.1. Principais características

- **Arquitetura peer-to-peer:** não há um nó mestre; todos os nós são equivalentes.
- **Replicação configurável:** fator de replicação por keyspace, suportando falhas de nós/datacenters.
- **Escalabilidade linear:** ao adicionar nós, aumenta-se a capacidade de leitura/escrita.
- **Modelo de dados baseado em colunas amplas:** tabelas otimizadas para queries específicas.
- **Consistência tunável:** leitura/escrita com níveis de consistência configuráveis (ONE, QUORUM, ALL, etc.).

2.2. Funcionamento em alto nível

Os dados em Cassandra são distribuídos no cluster por meio de *partition keys*, que determinam em quais nós as linhas serão fisicamente armazenadas. Cada tabela deve ser modelada a partir das consultas que serão executadas — ou seja, modelagem **query-driven**, e não normalizada como em bancos

relacionais.

> **Resumo:** Cassandra é adequado quando precisamos de grandes volumes de dados, replicação, disponibilidade alta e consultas bem definidas por chave ou partição (por exemplo, vendas por estado, categoria e período).

3. Contexto de Negócio e Conjunto de Dados

O projeto representa um marketplace de varejo com grande volume de transações, múltiplos estados e categorias de produto. O objetivo é suportar análises de:

- Vendas por período (mês/ano).
- Receita por estado e categoria.
- Relação entre preço e rating dos produtos.

O conjunto de dados utilizado é um dataset sintético (simulado) que representa o comportamento de transações em um marketplace, com arquivos em `.csv` e `.parquet` armazenados em `data/raw/`.

Estrutura de diretórios relevante

```
assets/
  img/
    Infnet-Logo.png
  evidencias/
    dataset_sintetico.png
    etl_analysis.png

data/
  raw/
    marketplace_sample_30.csv
    marketplace_bigdata_1M.parquet
  processed/
    vendas_por_mes.csv
    receita_estado_categoria.csv
    preco_rating_por_produto.csv

docker/
  docker-compose.yml
  marketplace_schema.cql

src/
  dataset_sintetico.py
  etl_analysis.py
  etl_cassandra.py
  plots_marketplace.py

img/
  grafico_linha_vendas_tempo.png
  grafico_barras_categoria_estado.png
  grafico_dispersao_preco_rating.png
```

Evidências de execução estão em `assets/evidencias/` e gráficos finais para o relatório em `img/`.

4. Perguntas de Negócio

As principais perguntas de negócio que orientam a modelagem e o processo analítico são:

1. **Receita e volume:** Como se comportam as vendas mensais (volume e receita) ao longo do tempo?
2. **Mix por estado/categoria:** Quais estados e categorias geram maior receita, e qual o ticket médio por combinação estado–categoria?
3. **Preço x qualidade:** Existe relação entre preço dos produtos e rating médio (avaliações) recebidos?

A modelagem em Cassandra, o ETL em Python e as visualizações finais foram desenhados para responder a essas perguntas.

5. Arquitetura e Planejamento de Infraestrutura

A infraestrutura planejada para hospedar os dados e suportar o processamento analítico foi baseada em:

- Cluster Cassandra com 2 nós (para alta disponibilidade e simulação de ambiente distribuído).
- Orquestração via Docker Compose.
- Volumes persistentes mapeados para dados e commit logs de cada nó.
- Container adicional para execução dos scripts Python (opcional, aqui rodado localmente na máquina host).

Planejamento básico (recursos)

- **Máquina host:** Windows com WSL/Docker Desktop.
- **Recursos mínimos:** 4 vCPUs, 8 GB RAM para ambiente de desenvolvimento.
- **Storage:** Volumes Docker para `/var/lib/cassandra` de cada nó (~10 GB).

- **Rede:** Rede bridge com portas expostas 9042 (CQL) para conexão dos clientes.

6. Configuração do Ambiente Computacional

A preparação do ambiente seguiu os passos abaixo, executados na máquina host:

6.1. Instalação de dependências

```
# Instalar Docker Desktop no Windows (via instalador oficial)
# Após instalação, garantir que o serviço Docker esteja em execução.

# Verificar versão do Docker no PowerShell
docker --version
```

Uma vez com Docker funcional, o cluster Cassandra é descrito em `docker/docker-compose.yml`.

6.2. Arquivo docker-compose.yml (trecho principal)

```
# Arquivo: docker/docker-compose.yml

version: '3.8'

services:
  cassandra-node1:
    image: cassandra:latest
    container_name: cassandra-node1
    environment:
      - CASSANDRA_CLUSTER_NAME=marketplace-cluster
      - CASSANDRA_SEEDS=cassandra-node1
      - CASSANDRA_START_RPC=true
    ports:
      - "9042:9042"
    volumes:
      - ./data/node1:/var/lib/cassandra
    networks:
      - cassandra-net

  cassandra-node2:
    image: cassandra:latest
    container_name: cassandra-node2
    environment:
      - CASSANDRA_CLUSTER_NAME=marketplace-cluster
      - CASSANDRA_SEEDS=cassandra-node1
    volumes:
      - ./data/node2:/var/lib/cassandra
    networks:
      - cassandra-net
    depends_on:
      - cassandra-node1

networks:
  cassandra-net:
    driver: bridge
```

O arquivo completo está disponível em `docker/docker-compose.yml` no repositório do projeto.

6.3. Subida do cluster

```
# No PowerShell, a partir da raiz do projeto
cd "E:\desenvolvimento\Infraestrutura Cassandra [25E4_2]\docker"

# Subir os containers em segundo plano
docker-compose up -d

# Verificar se os containers estão rodando
docker ps
```

7. Modelagem de Dados no Cassandra

A modelagem foi conduzida de forma orientada às consultas, conforme boas práticas de Cassandra. As principais tabelas residem no keyspace `marketplace_ks`.

7.1. Criação do keyspace

```
-- Arquivo: docker/marketplace_schema.cql (trecho)

CREATE KEYSPACE IF NOT EXISTS marketplace_ks
WITH replication = {
```

```
  'class': 'SimpleStrategy',
  'replication_factor': 2
};
```

7.2. Tabela principal de transações

```
CREATE TABLE IF NOT EXISTS marketplace_ks.sales_transactions (
    order_id       uuid,
    order_date     date,
    state          text,
    category       text,
    product_id     text,
    product_name   text,
    quantity       int,
    unit_price     decimal,
    total_price    decimal,
    rating         decimal,
    PRIMARY KEY ((state, category), order_date, order_id)
) WITH CLUSTERING ORDER BY (order_date DESC, order_id ASC);
```

A partição composta (`state, category`) permite agrupar fisicamente os dados por estado e categoria, o que otimiza relatórios por combinação estado–categoria e período (pergunta de negócio 2).

7.3. Tabela de agregados mensais

```
CREATE TABLE IF NOT EXISTS marketplace_ks.sales_by_month (
    year_month     text,     -- ex: "2024-01"
    state          text,
    category       text,
    total_orders   bigint,
    total_quantity bigint,
    total_revenue  decimal,
    avg_ticket     decimal,
    PRIMARY KEY ((year_month), state, category)
) WITH CLUSTERING ORDER BY (state ASC, category ASC);
```

Esta tabela suporta consultas rápidas de vendas mensais (pergunta 1) e ticket médio por estado/categoria.

8. Carga de Dados no Cassandra (ETL) e Comprovação

A carga foi realizada via script Python `src/etl_cassandra.py`, que lê os arquivos de entrada, prepara os dados e insere nas tabelas do keyspace `marketplace_ks`.

8.1. Exemplo de script de carga em Python

```
# Arquivo: src/etl_cassandra.py (trecho)

from cassandra.cluster import Cluster
import pandas as pd

# Conexão com o cluster Cassandra
cluster = Cluster(['localhost'])  # cassandra-node1 exposto na porta 9042
session = cluster.connect('marketplace_ks')

# Leitura do dataset bruto
df = pd.read_parquet('data/raw/marketplace_bigdata_1M.parquet')

# Cálculo do total_price
df['total_price'] = df['quantity'] * df['unit_price']

# Inserção na tabela sales_transactions
insert_stmt = session.prepare("""
    INSERT INTO sales_transactions (
        state, category, order_date, order_id,
        product_id, product_name, quantity,
        unit_price, total_price, rating
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
""")

for _, row in df.iterrows():
    session.execute(
        insert_stmt,
        (
            row['state'],
            row['category'],
            row['order_date'].date(),
            row['order_id'],
            row['product_id'],
            row['product_name'],
            int(row['quantity']),
            float(row['unit_price']),
```

```
            float(row['total_price']),
            float(row['rating']),
        )
    )

cluster.shutdown()
```

8.2. Comprovação de carga bem sucedida

Após a execução do ETL, uma consulta de contagem foi executada via `cqlsh`:

```
-- Exemplo de validação no cqlsh

USE marketplace_ks;

SELECT COUNT(*) FROM sales_transactions;
```

Prints da execução do ETL e da estrutura dos dados estão salvos em `assets/evidencias/dataset_sintetico.png` e `assets/evidencias/etl_analysis.png`.

9. Extração, Manipulação e Exportação da Base Tratada

Para apoiar a resposta das perguntas de negócio, foi desenvolvido o script `src/etl_analysis.py`, responsável por:

- Ler os dados brutos.
- Executar operações de limpeza e manipulação.
- Criar variáveis derivadas (como `total_price` e `year_month`).
- Gerar bases agregadas e exportá-las em CSV.

9.1. Extração e criação de variáveis

```
# Arquivo: src/etl_analysis.py (trecho)

import pandas as pd

# Leitura dos dados brutos
df_raw = pd.read_parquet('data/raw/marketplace_bigdata_1M.parquet')

# Conversão de datas
df_raw['order_date'] = pd.to_datetime(df_raw['order_date'])

# Criação de nova variável: total_price
df_raw['total_price'] = df_raw['quantity'] * df_raw['unit_price']

# Criação de variável year_month (AAAA-MM)
df_raw['year_month'] = df_raw['order_date'].dt.to_period('M').astype(str)
```

9.2. Operações de manipulação de dados

```
# Agregado de vendas por mês
vendas_por_mes = (
    df_raw
    .groupby('year_month')
    .agg(
        total_orders=('order_id', 'nunique'),
        total_quantity=('quantity', 'sum'),
        total_revenue=('total_price', 'sum')
    )
    .reset_index()
)

# Agregado de receita por estado e categoria
receita_estado_categoria = (
    df_raw
    .groupby(['state', 'category'])
    .agg(
        total_revenue=('total_price', 'sum'),
        total_orders=('order_id', 'nunique'),
        avg_ticket=('total_price', 'mean')
    )
    .reset_index()
)

# Base para gráfico de dispersão (preço x rating por produto)
preco_rating_por_produto = (
    df_raw
    .groupby(['product_id', 'product_name'])
    .agg(
        avg_price=('unit_price', 'mean'),
        avg_rating=('rating', 'mean'),
        total_orders=('order_id', 'nunique')
    )
```

```
    .reset_index()
)
```

9.3. Exportação da base tratada para CSV

```
# Exporta as bases tratadas para data/processed/

vendas_por_mes.to_csv('data/processed/vendas_por_mes.csv', index=False)
receita_estado_categoria.to_csv('data/processed/receita_estado_categoria.csv', index=False)
preco_rating_por_produto.to_csv('data/processed/preco_rating_por_produto.csv', index=False)
```

Essas exportações atendem ao requisito de geração de base tratada em CSV para posterior análise e consumo por outras ferramentas.

10. Consultas de Leitura na Infraestrutura Cassandra

Com a base carregada, foram executadas consultas de leitura diretamente em Cassandra, tanto via `cqlsh` quanto via driver Python.

10.1. Exemplo de consulta de leitura em CQL

```
-- Vendas por estado e categoria em um mês específico

SELECT state, category, total_orders, total_revenue, avg_ticket
FROM marketplace_ks.sales_by_month
WHERE year_month = '2024-01';
```

10.2. Exemplo de consulta de leitura em Python

```
# Leitura de agregados mensais diretamente do Cassandra

from cassandra.cluster import Cluster

cluster = Cluster(['localhost'])
session = cluster.connect('marketplace_ks')

rows = session.execute("""
    SELECT year_month, state, category, total_orders, total_revenue, avg_ticket
    FROM sales_by_month
    WHERE year_month = '2024-01'
""")

for row in rows:
    print(row.year_month, row.state, row.category, row.total_orders, row.total_revenue)

cluster.shutdown()
```

11. Visualizações e Respostas às Perguntas de Negócio

As visualizações foram geradas em Python (script `src/plots_marketplace.py`) a partir dos arquivos tratados em `data/processed/`, com as figuras salvas na pasta `img/`.

11.1. Gráfico de linhas — Vendas mensais (pergunta 1)

```
# Arquivo: src/plots_marketplace.py (trecho)

import pandas as pd
import matplotlib.pyplot as plt

# Vendas por mês
vendas_por_mes = pd.read_csv('data/processed/vendas_por_mes.csv')

plt.figure(figsize=(10, 5))
plt.plot(vendas_por_mes['year_month'], vendas_por_mes['total_revenue'], marker='o')
plt.xticks(rotation=45)
plt.xlabel('Ano-Mês')
plt.ylabel('Receita total')
plt.title('Receita mensal do marketplace')
plt.tight_layout()
plt.savefig('img/grafico_linha_vendas_tempo.png')
plt.close()
```

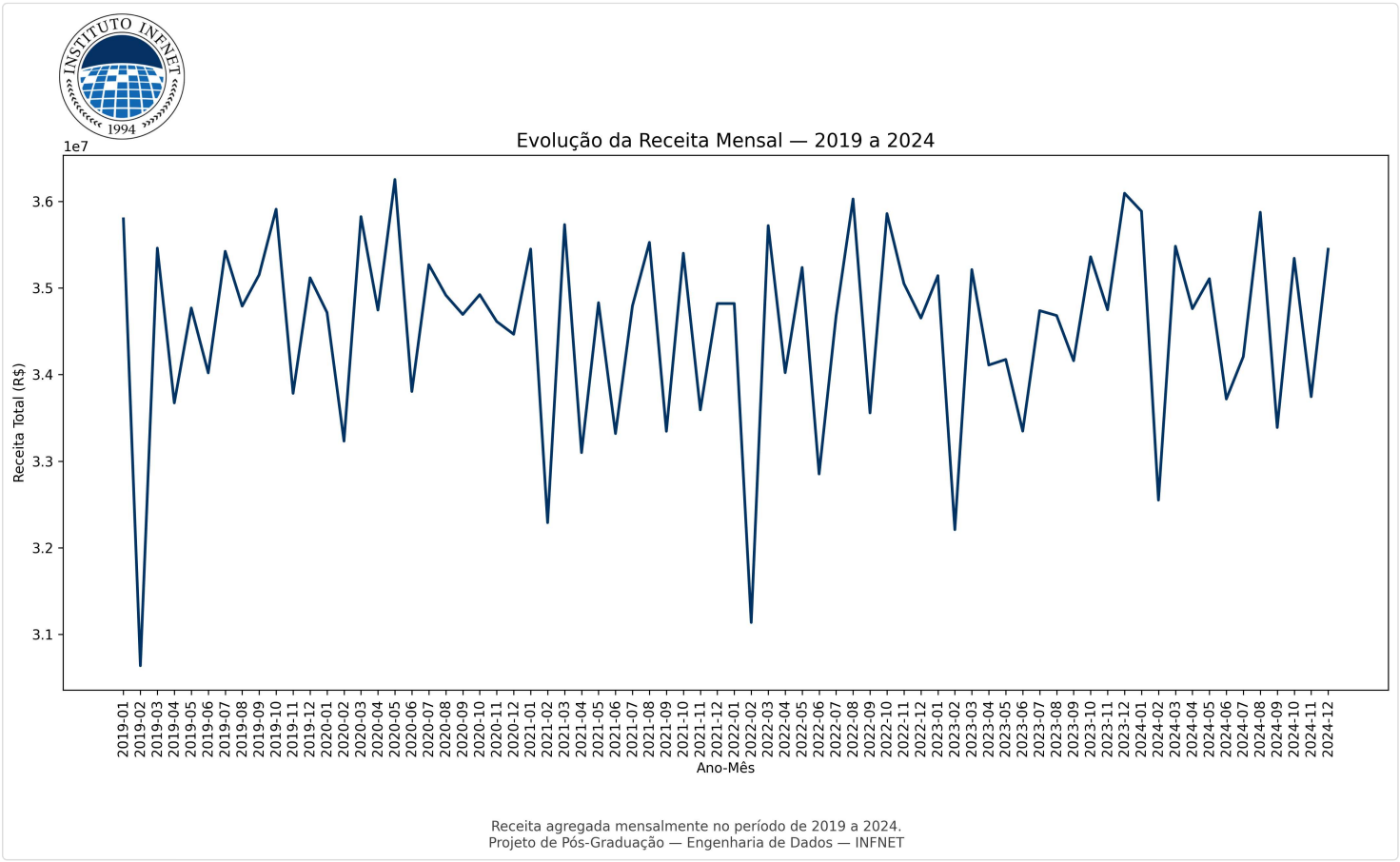

Figura 1 — Evolução da receita mensal do marketplace (arquivo: `img/grafico_linha_vendas_tempo.png`).

Este gráfico permite observar sazonalidade, crescimento ou queda na receita ao longo dos meses, respondendo diretamente à pergunta de negócio sobre a evolução das vendas ao longo do tempo.

11.2. Gráfico de barras — Receita por estado e categoria (pergunta 2)

```
# Gráfico de barras por estado e categoria

receita_estado_categoria = pd.read_csv('data/processed/receita_estado_categoria.csv')

# Exemplo: filtrar top N categorias por estado
top = (
    receita_estado_categoria
    .sort_values('total_revenue', ascending=False)
    .head(20)
)

plt.figure(figsize=(10, 6))
plt.barh(
    top['state'] + ' - ' + top['category'],
    top['total_revenue']
)
plt.xlabel('Receita total')
plt.ylabel('Estado - Categoria')
plt.title('Receita por estado e categoria')
plt.tight_layout()
plt.savefig('img/grafico_barras_categoria_estado.png')
plt.close()
```

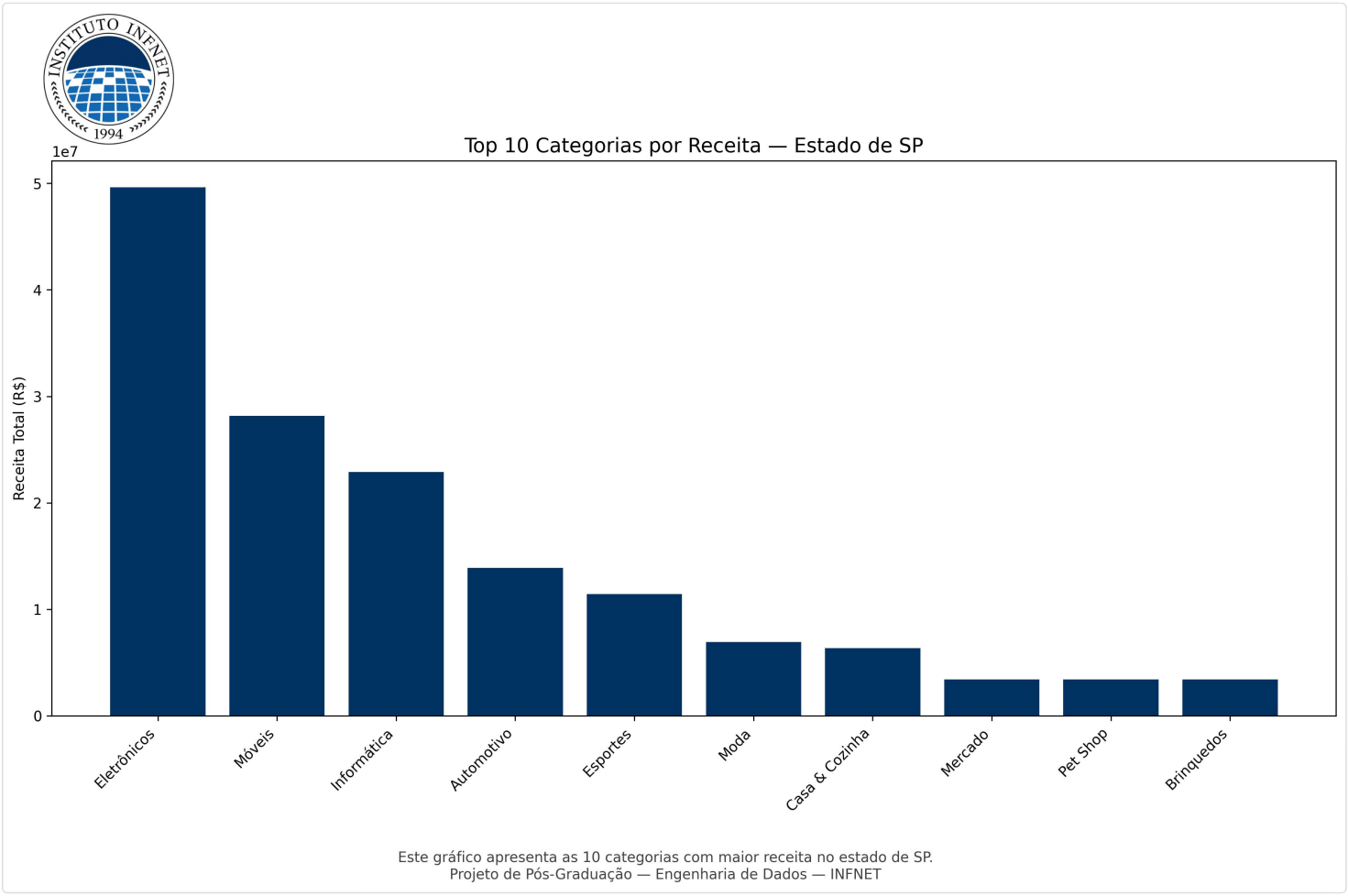

Figura 2 — Receita por estado e categoria (arquivo: `img/grafico_barras_categoria_estado.png`).

A visualização deixa evidente quais combinações estado–categoria concentram a maior parte da receita, além de permitir comparar o ticket médio (via tabela `sales_by_month` e agregados em Python).

11.3. Gráfico de dispersão — Preço vs rating (pergunta 3)

```
# Gráfico de dispersão preço x rating por produto

preco_rating = pd.read_csv('data/processed/preco_rating_por_produto.csv')

plt.figure(figsize=(8, 6))
plt.scatter(
    preco_rating['avg_price'],
    preco_rating['avg_rating'],
    alpha=0.6
)
plt.xlabel('Preço médio do produto')
plt.ylabel('Rating médio')
plt.title('Dispersão preço x rating dos produtos')
plt.tight_layout()
plt.savefig('img/grafico_dispersao_preco_rating.png')
plt.close()
```

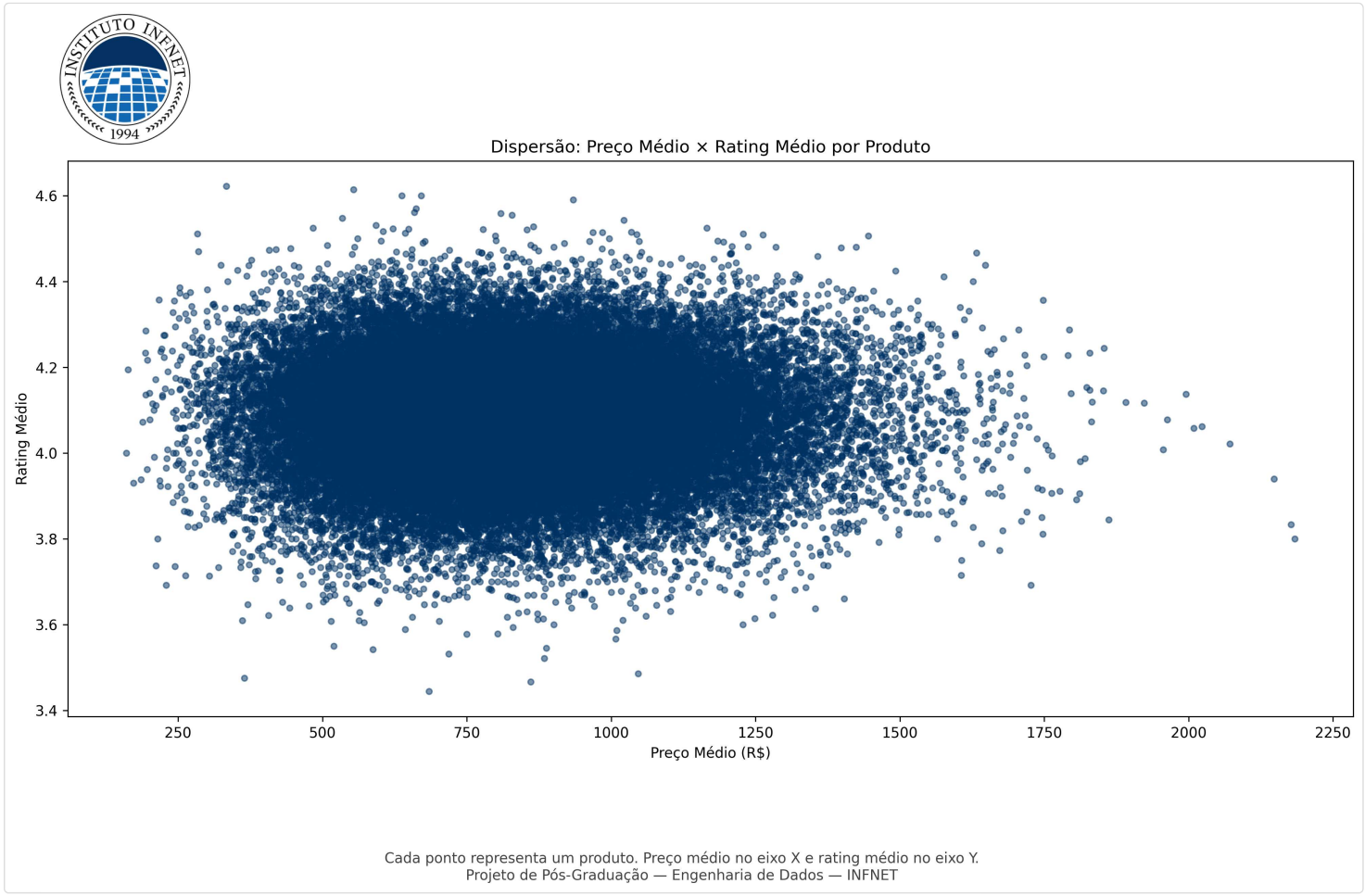

Figura 3 — Dispersão entre preço médio e rating dos produtos (arquivo: `img/grafico_dispersao_preco_rating.png`).

O gráfico permite identificar se produtos mais caros tendem a receber avaliações melhores ou piores, além de destacar outliers (produtos caros com rating baixo ou baratos com rating muito alto).

12. Evidências de Execução

As evidências visuais da execução dos scripts e da preparação dos dados foram salvas na pasta `assets/evidencias/`:

Arquivo	Descrição
`assets/evidencias/dataset_sintetico.png`	Print da execução do script `dataset_sintetico.py` e visualização da estrutura básica do dataset.
`assets/evidencias/etl_analysis.png`	Print da execução do script `etl_analysis.py`, incluindo geração de agregados e exportação para CSV.

13. Conclusões e Próximos Passos

13.1. Resultados alcançados

- Comparação conceitual entre SQL e NoSQL e contextualização do uso de Cassandra.
- Definição de perguntas de negócio alinhadas ao cenário de marketplace.
- Planejamento e configuração de uma infraestrutura Cassandra em Docker com múltiplos nós.
- Modelagem orientada a queries com tabelas de transações e agregados mensais.
- Carga de dados e validação via CQL e Python.
- Processo de extração, manipulação, criação de variáveis e exportação em CSV em Python.
- Visualizações (linha, barras, dispersão) respondendo às principais perguntas de negócio.

13.2. Próximos passos sugeridos

- Adicionar autenticação e controle de acesso ao cluster Cassandra.
- Explorar topologias multi-datacenter para cenários de alta disponibilidade geográfica.
- Criar novas tabelas derivadas ou materialized views para SLAs específicos de consulta.
- Orquestrar o pipeline ETL com ferramentas como Apache Airflow ou Dagster.
- Publicar dashboards em ferramentas de BI conectadas à base tratada.

14. Repositório Git do Projeto

Todo o código-fonte do projeto (scripts Python, arquivos Docker, dados de amostra e evidências gráficas) está disponível no repositório:

Repositório Git: https://github.com/CBarrosoBRRJ/PD-INFRAESTRUTURA-CASSANDRA-25E4_2

```
# Como clonar o repositório

git clone https://github.com/CBarrosoBRRJ/PD-INFRAESTRUTURA-CASSANDRA-25E4_2.git
cd PD-INFRAESTRUTURA-CASSANDRA-25E4_2
```