Documentação Técnica: Pipeline de Ingestão Bitcoin (Camada Bronze)

Status: ✅ Concluído Data: 30/01/2026

1. Visão Geral

Este módulo é responsável pela ingestão de dados de mercado do Bitcoin. Ele implementa um pipeline incremental inteligente:

1. **Verifica qual a última data gravada no Data Warehouse.**
2. **Busca dados na API externa (CoinGecko).**
3. **Filtra apenas os registros novos (Delta) para evitar duplicidade.**
4. **Realiza a carga no Supabase via API segura.**

2. Stack Tecnológica

- **Linguagem: Python 3.12+**
- **Armazenamento (Data Warehouse): Supabase (PostgreSQL)**
- **Fonte de Dados: CoinGecko API (Free Tier)**
- **Segurança: Variáveis de ambiente (.env) e Políticas de Segurança de Linha (RLS).**

Passo 1: Preparação do Terreno (Ambiente)

Siga estes comandos no terminal do seu VS Code:

1. **Crie a pasta do projeto**
2. **Crie e ative o Ambiente Virtual (essencial para não bagunçar seu Python):**

   python -m venv venv

   Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

   .\venv\Scripts\activate

3. Configuração do Ambiente

3.1. Dependências

Bibliotecas necessárias para execução:

pip install pandas requests supabase python-dotenv

3.2. Credenciais (.env)

O arquivo .env na raiz do projeto deve conter:

 # URL do Projeto Supabase

SUPABASE_URL=https://seu-projeto.supabase.co

 # Chave 'service_role' (Secret) para permissão de escrita

SUPABASE_KEY=sb_secret_...

4. Estrutura do Banco de Dados

Como utilizamos a API do Supabase, a tabela foi criada via SQL Editor para garantir compatibilidade e segurança.

Script SQL:

```
CREATE TABLE IF NOT EXISTS bronze_bitcoin (
id SERIAL PRIMARY KEY,
coin_id TEXT,
price_usd NUMERIC,
updated_at TIMESTAMP WITH TIME ZONE, -- Importante: Com fuso horário
ingestion_at TIMESTAMP WITH TIME ZONE
 );

-- Habilitar segurança e permitir acesso via API
ALTER TABLE bronze_bitcoin ENABLE ROW LEVEL SECURITY;
CREATE POLICY "Permitir Ingestao Service Role" ON bronze_bitcoin FOR ALL USING
(true) WITH CHECK (true);
```

5. Código Fonte Final (ingestao_bronze.py)

Este script contém as correções para:

- **Erro de Fuso Horário:** Força comparação UTC vs UTC.
- **Limite da API:** Busca apenas os últimos 365 dias (limite do plano grátis).
- **Bloqueio de API:** Usa User-Agent para simular um navegador.

```
import os
import requests
import time
from supabase import create_client, Client
from dotenv import load_dotenv
from datetime import datetime, timezone

# --- CONFIGURAÇÃO INICIAL ---
load_dotenv()
url = os.getenv("SUPABASE_URL")
key = os.getenv("SUPABASE_KEY")

if not url or not key:
    raise ValueError("❌ Erro: Credenciais do Supabase (URL/KEY) faltando no .env")
```

```
supabase: Client = create_client(url, key)

def obter_ultima_data_banco():
    """
    Consulta o Supabase para saber qual a data mais recente salva.
    Retorna um objeto datetime com fuso horário (UTC) ou None.
    """
    try:
        response = supabase.table("bronze_bitcoin")\
            .select("updated_at")\
            .order("updated_at", desc=True)\
            .limit(1)\
            .execute()

        dados = response.data
        if dados and len(dados) > 0:
            data_str = dados[0]['updated_at']

            # 1. Converte string ISO para datetime
            # O replace corrige formatos que vêm como 'Z'
            dt = datetime.fromisoformat(data_str.replace('Z', '+00:00'))

            # 2. BLINDAGEM DE FUSO HORÁRIO
            # Se a data vier "ingênua" (sem fuso), forçamos UTC para evitar erro de
comparação
            if dt.tzinfo is None:
                dt = dt.replace(tzinfo=timezone.utc)

            return dt
        return None
    except Exception as e:
        print(f" ⚠️ Aviso: Não foi possível ler última data (Banco vazio ou erro): {e}")
        return None

def ingestao_bronze():
    print(" 🧠 Iniciando Ingestão Inteligente (Camada Bronze)...")

    # 1. Verifica estado atual do Banco (Watermark)
    ultima_data = obter_ultima_data_banco()

    # Limite da API Grátis da CoinGecko é 365 dias
    dias_para_buscar = "365"

    if ultima_data:
        print(f" 🗓️ Última data no banco: {ultima_data.strftime('%Y-%m-%d %H:%M:%S
%Z')}")
    else:
```

```
        print(f"🗓️ Banco vazio. Buscaremos os últimos {dias_para_buscar} dias.")

    # 2. Configura Chamada na API
    api_url =
f"https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&day
s={dias_para_buscar}&interval=daily"

    # Header essencial para evitar erro 403/429
    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"
    }

    try:
        print(f"📡 Baixando dados da CoinGecko...")
        response = requests.get(api_url, headers=headers, timeout=20)

        if response.status_code != 200:
            print(f"❌ Erro da API: {response.status_code}")
            print(f"   Mensagem: {response.text}")
            return

        dados_json = response.json()
        precos = dados_json.get("prices", [])

        print(f"📦 Registros encontrados na API: {len(precos)}")

        # 3. Processamento e Filtro (Delta)
        novos_registros = []

        for registro in precos:
            timestamp_ms = registro[0]
            valor_usd = registro[1]

            # Converte timestamp UNIX (ms) para Datetime UTC
            data_registro = datetime.fromtimestamp(timestamp_ms / 1000,
tz=timezone.utc)

            # LÓGICA INCREMENTAL:
            # Só insere se (Banco Vazio) OU (Data Nova > Data Banco)
            if ultima_data is None or data_registro > ultima_data:
                payload = {
                    "coin_id": "bitcoin",
                    "price_usd": valor_usd,
                    "updated_at": data_registro.isoformat(),
                    "ingestion_at": datetime.now(timezone.utc).isoformat()
                }
```

```
            novos_registros.append(payload)

        # 4. Inserção em Lote (Batch Insert)
        total_novos = len(novos_registros)

        if total_novos > 0:
            print(f" 🚀 Inserindo {total_novos} novos registros no Supabase...")

            # Quebra em lotes de 5000 para performance
            tamanho_lote = 5000
            for i in range(0, total_novos, tamanho_lote):
                lote = novos_registros[i:i + tamanho_lote]
                supabase.table("bronze_bitcoin").insert(lote).execute()
                print(f"   -> Lote enviado com sucesso.")
                time.sleep(1) # Pausa para respeitar limite da API

            print(" ✅ SUCESSO! Ingestão concluída.")
        else:
            print(" ✅ O banco já está atualizado. Nenhum registro novo encontrado.")

    except Exception as e:
        print(f" ❌ Erro Crítico durante a execução: {e}")

if __name__ == "__main__":
    ingestao_bronze()
```