📂 Estrutura de Arquivos Final:

```
projeto_btc_dados/
|
├── .github/
|   └── workflows/
|       └── pipeline.yml       <-- O Robô (Automação)
|
├── transformacao_btc/         <-- Pasta do dbt
|   ├── models/                <-- Seus SQLs (Silver/Gold)
|   ├── dbt_project.yml
|   └── profiles.yml           <-- Config de conexão (Local)
|
├── .gitignore                 <-- O Guardião (Segurança)
├── ingestao_bronze.py         <-- O Motor (Python + SQL)
└── requirements.txt (Opcional, mas recomendado)
```

1. O Motor de Ingestão (ingestao_bronze.py)

Tecnologia: Python + Psycopg2 (SQL Puro via Pooler 6543)

```
import os
import requests
import psycopg2
from dotenv import load_dotenv
from datetime import datetime, timezone

# 1. Configuração e Conexão
load_dotenv()

def get_db_connection():
    """Cria a conexão com o banco usando as variáveis do .env"""
    try:
        conn = psycopg2.connect(
```

```
            host=os.getenv("DB_HOST"),
            database=os.getenv("DB_NAME"),
            user=os.getenv("DB_USER"),
            password=os.getenv("DB_PASS"),
            port=os.getenv("DB_PORT", "6543")
        )
        return conn
    except Exception as e:
        print(f"❌ Erro ao conectar no banco: {e}")
        raise e

def obter_ultima_data_banco(cursor):
    """Consulta via SQL qual a data mais recente salva."""
    try:
        query = "SELECT updated_at FROM bronze_bitcoin ORDER BY updated_at DESC LIMIT 1;"
        cursor.execute(query)
        result = cursor.fetchone()

        if result:
            dt = result[0]
            if dt.tzinfo is None:
                dt = dt.replace(tzinfo=timezone.utc)
            return dt
        return None
    except Exception as e:
        print(f"⚠️ Erro ao verificar data no banco: {e}")
        return None

def ingestao_bronze():
    print("🧠 Iniciando Ingestão (Via SQL/Pooler)...")
```

```
    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        # 1. Verifica estado atual
        ultima_data = obter_ultima_data_banco(cursor)

        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("🗓️ Banco vazio ou sem dados. Buscaremos os últimos 365 dias.")

        # 2. Chamada na API (CoinGecko)
        api_url = f"https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&days={dias_para_buscar}&interval=daily"
        headers = {"User-Agent": "Mozilla/5.0"}

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

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

        dados_json = response.json()
        precos = dados_json.get("prices", [])
        print(f"📦 Registros retornados pela API: {len(precos)}")

        # 3. Processamento
```

```
    novos_dados = []
    for registro in precos:
        timestamp_ms = registro[0]
        valor_usd = registro[1]
        data_registro = datetime.fromtimestamp(timestamp_ms / 1000, tz=timezone.utc)

        if ultima_data is None or data_registro > ultima_data:
            novos_dados.append((
                "bitcoin",
                valor_usd,
                data_registro,
                datetime.now(timezone.utc)
            ))

    # 4. Inserção em Lote (Batch Insert)
    total_novos = len(novos_dados)
    if total_novos > 0:
        print(f"🚀 Inserindo {total_novos} novos registros...")
        query_insert = """
        INSERT INTO bronze_bitcoin (coin_id, price_usd, updated_at, ingestion_at)
        VALUES (%s, %s, %s, %s)
        """
        cursor.executemany(query_insert, novos_dados)
        conn.commit()
        print("✅ SUCESSO! Carga concluída.")
    else:
        print("✅ Banco já atualizado. Nenhum dado novo para inserir.")

except Exception as e:
    conn.rollback()
    print(f"❌ Erro Crítico: {e}")
```

```
    finally:
        cursor.close()
        conn.close()
        print("🔌 Conexão encerrada.")

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

Com certeza. Como seu Consultor Sênior, digo que essa é uma prática excelente: **Documentação de Legado**.

Aqui está o **"Snapshot Final" (Golden State)** do seu projeto projeto_btc_dados. Esse é o código exato que funcionou, com todas as correções de segurança (Secrets), performance (SQL/Psycopg2) e automação (GitHub Actions).

Você pode copiar isso e salvar no seu Notion, OneNote ou em um arquivo de texto para usar como "template" nos projetos da IBRA.

📂 Estrutura de Arquivos Final

Plaintext

```
projeto_btc_dados/
│
├── .github/
│   └── workflows/
│       └── pipeline.yml      <-- O Robô (Automação)
│
├── transformacao_btc/        <-- Pasta do dbt
│   ├── models/               <-- Seus SQLs (Silver/Gold)
│   ├── dbt_project.yml
│   └── profiles.yml          <-- Config de conexão (Local)
│
├── .gitignore                <-- O Guardião (Segurança)
├── ingestao_bronze.py        <-- O Motor (Python + SQL)
└── requirements.txt (Opcional, mas recomendado)
```

---

1. O Motor de Ingestão (ingestao_bronze.py)

Tecnologia: Python + Psycopg2 (SQL Puro via Pooler 6543)

Python

```
import os
import requests
import psycopg2
from dotenv import load_dotenv
from datetime import datetime, timezone

# 1. Configuração e Conexão
load_dotenv()

def get_db_connection():
    """Cria a conexão com o banco usando as variáveis do .env"""
    try:
        conn = psycopg2.connect(
            host=os.getenv("DB_HOST"),
            database=os.getenv("DB_NAME"),
            user=os.getenv("DB_USER"),
            password=os.getenv("DB_PASS"),
            port=os.getenv("DB_PORT", "6543")
        )
        return conn
    except Exception as e:
        print(f"❌ Erro ao conectar no banco: {e}")
        raise e

def obter_ultima_data_banco(cursor):
    """Consulta via SQL qual a data mais recente salva."""
    try:
```

```
        query = "SELECT updated_at FROM bronze_bitcoin ORDER BY updated_at DESC LIMIT 1;"
        cursor.execute(query)
        result = cursor.fetchone()

        if result:
            dt = result[0]
            if dt.tzinfo is None:
                dt = dt.replace(tzinfo=timezone.utc)
            return dt
        return None
    except Exception as e:
        print(f" ⚠️ Erro ao verificar data no banco: {e}")
        return None

def ingestao_bronze():
    print(" 🧠 Iniciando Ingestão (Via SQL/Pooler)...")

    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        # 1. Verifica estado atual
        ultima_data = obter_ultima_data_banco(cursor)

        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(" 🗓️ Banco vazio ou sem dados. Buscaremos os últimos 365 dias.")

        # 2. Chamada na API (CoinGecko)
```

```
    api_url = f"https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&days={dias_para_buscar}&interval=daily"
    headers = {"User-Agent": "Mozilla/5.0"}

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

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

    dados_json = response.json()
    precos = dados_json.get("prices", [])
    print(f" 📦 Registros retornados pela API: {len(precos)}")

    # 3. Processamento
    novos_dados = []
    for registro in precos:
        timestamp_ms = registro[0]
        valor_usd = registro[1]
        data_registro = datetime.fromtimestamp(timestamp_ms / 1000, tz=timezone.utc)

        if ultima_data is None or data_registro > ultima_data:
            novos_dados.append((
                "bitcoin",
                valor_usd,
                data_registro,
                datetime.now(timezone.utc)
            ))

    # 4. Inserção em Lote (Batch Insert)
```

```
        total_novos = len(novos_dados)
        if total_novos > 0:
            print(f" 🚀 Inserindo {total_novos} novos registros...")
            query_insert = """
            INSERT INTO bronze_bitcoin (coin_id, price_usd, updated_at, ingestion_at)
            VALUES (%s, %s, %s, %s)
            """
            cursor.executemany(query_insert, novos_dados)
            conn.commit()
            print(" ✅ SUCESSO! Carga concluída.")
        else:
            print(" ✅ Banco já atualizado. Nenhum dado novo para inserir.")

    except Exception as e:
        conn.rollback()
        print(f"❌ Erro Crítico: {e}")
    finally:
        cursor.close()
        conn.close()
        print(" 🔌 Conexão encerrada.")

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

---

2. O Robô de Automação (.github/workflows/pipeline.yml)

Localização: .github/workflows/pipeline.yml Obs: Configurado para rodar todo dia às 09:00 UTC (06:00 BRT) ou Manualmente.

```
name: Pipeline Bitcoin BTC

on:
  schedule:
```

```
    - cron: '0 9 * * *'
  workflow_dispatch:

env:
  DB_HOST: ${{ secrets.DB_HOST }}
  DB_USER: ${{ secrets.DB_USER }}
  DB_PASS: ${{ secrets.DB_PASS }}
  DB_NAME: ${{ secrets.DB_NAME }}
  DB_PORT: 6543

jobs:
  etl_pipeline:
    runs-on: ubuntu-latest

    steps:
      - name: 📥 Baixar Repositório
        uses: actions/checkout@v3

      - name: 🐍 Configurar Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: 📦 Instalar Dependências
        run: |
          pip install requests psycopg2-binary python-dotenv dbt-postgres

      - name: 🚀 Rodar Ingestão (Bronze)
        run: |
          python ingestao_bronze.py
```

```
    - name: ⚙️ Configurar dbt (Profile Seguro)
      run: |
        mkdir -p ~/.dbt
        echo "
        transformacao_btc:
          target: prod
          outputs:
            prod:
              type: postgres
              host: $DB_HOST
              user: $DB_USER
              password: $DB_PASS
              dbname: $DB_NAME
              port: 6543
              schema: silver
              threads: 1
        " > ~/.dbt/profiles.yml

    - name: 📐 Rodar Transformação (Silver/Gold)
      run: |
        cd transformacao_btc
        dbt deps
        dbt run --profiles-dir ~/.dbt
```

3. A Segurança (.gitignore)

Arquivo que impede o envio de senhas.

```
.env
venv/
__pycache__/
target/
dbt_packages/
```

```
logs/
*.log
```

4. Configuração Local do dbt (transformacao_btc/profiles.yml)

Para rodar na sua máquina via VS Code.

```
transformacao_btc:
  target: dev
  outputs:
    dev:
      type: postgres
      host: "{{ env_var('DB_HOST') }}"
      user: "{{ env_var('DB_USER') }}"
      password: "{{ env_var('DB_PASS') }}"
      dbname: "{{ env_var('DB_NAME') }}"
      port: 6543
      schema: silver
      threads: 1
```