Para organizar o seu projeto de forma lógica no GitHub e garantir que tanto as consultas SQL quanto o código Python sejam facilmente compreensíveis e reutilizáveis, você pode seguir uma estrutura de diretórios bem definida. Aqui está uma sugestão de como organizar seu repositório:

**Estrutura do Projeto**

**Detalhamento de cada parte:**



**1. README.md**

O arquivo README.md deve ser o primeiro arquivo que a pessoa lê. Aqui, você pode explicar o que o projeto faz, como configurá-lo e executá-lo. Inclua instruções para a instalação das dependências, uso do código, e detalhes sobre o formato do dataset e como ele é carregado para o banco de dados.

**2. requirements.txt**

Este arquivo deve listar as bibliotecas Python necessárias para o funcionamento do projeto, como psycopg2 (para conexão com o PostgreSQL), pandas, sqlalchemy, etc. Exemplo:

psycopg2

pandas

sqlalchemy

numpy

**3. data/**

Dentro desse diretório, você pode separar os dados em subdiretórios como raw/ (dados originais) e processed/ (dados que foram carregados no banco ou processados de alguma forma). É uma boa prática manter o diretório de dados no repositório para facilitar o compartilhamento de exemplos, mas lembre-se de não subir dados confidenciais ou grandes demais (use .gitignore para arquivos muito grandes).

**4. sql/**

O diretório sql/ pode armazenar todos os scripts SQL necessários para o projeto. Você pode criar diferentes arquivos de script para cada etapa:

* create\_tables.sql: Contém as queries para criar as tabelas.
* constraints.sql: Contém as queries para definir as constraints (chaves primárias, estrangeiras, etc.).
* data\_cleaning.sql: Caso você precise fazer alguma limpeza de dados diretamente no SQL (por exemplo, remover duplicatas, corrigir tipos de dados, etc.), este arquivo pode armazenar essas queries.

**5. src/**

Este diretório deve conter o código Python. Aqui você vai separar o código em módulos, dependendo das responsabilidades do script:

* **data\_loader.py**: Responsável por baixar, descompactar e carregar os dados para o banco de dados. Esse script também pode ser responsável por chamar os scripts SQL de criação de tabelas, se necessário.
* **data\_processor.py**: Scripts para fazer o pré-processamento e transformação dos dados antes de carregá-los para o banco, se necessário.
* **db\_connector.py**: Conexão com o banco de dados PostgreSQL, que pode usar psycopg2 ou SQLAlchemy.
* **utils.py**: Funções auxiliares que podem ser reutilizadas em vários scripts (como conversões de formato, validações, etc.).

**6. notebooks/**

Este diretório pode ser usado para Jupyter Notebooks, caso você queira explorar os dados antes de realizar o processamento e carregamento, ou se precisar realizar uma análise inicial. Isso é opcional, mas pode ser útil para documentação e análise exploratória.

**7. tests/**

Sempre que possível, é bom adicionar testes. Este diretório pode conter testes unitários para os scripts em src/. Isso ajuda a garantir que as funções e o carregamento de dados estão funcionando corretamente.

* test\_data\_loader.py: Testa o carregamento de dados no banco.
* test\_db\_connector.py: Testa a conexão com o banco de dados.
* test\_data\_processor.py: Testa as transformações realizadas nos dados.

**Dicas Adicionais:**

* **.gitignore**: Lembre-se de adicionar um arquivo .gitignore para não subir arquivos desnecessários, como dependências do Python (\_\_pycache\_\_), arquivos de dados grandes, ou arquivos específicos de IDEs (ex: .vscode, .idea).

Exemplo de .gitignore:

\_\_pycache\_\_/

\*.pyc

\*.pyo

\*.env

\*.db

\*.log

data/raw/\*.zip

* **Versionamento de dados**: Caso os dados sejam grandes, considere usar ferramentas de versionamento como o **DVC** (Data Version Control), que ajudam a gerenciar datasets de maneira mais eficiente em repositórios Git.

Com essa estrutura organizada, fica fácil separar as responsabilidades entre os códigos e as consultas SQL, além de tornar o repositório mais fácil de entender e usar para outras pessoas.