Projeto Prático - Pipeline Python para Dados via APIs

Objetivo do projeto

Construir um pipeline de **extração → tratamento → disponibilização de dados** usando Python (ou PySpark/Databricks, conforme preferência). Cada participante deve extrair dados de pelo menos duas APIs públicas distintas, realizar transformações e entregá-los em formato consumível via GIT **(CSV / Parquet / tabela em Snowflake/S3 / REST endpoint, Dashboard e entre outros de sua preferência)** disponibilizando os dados da melhor maneira possível, aplicando práticas de Engenharia de Dados (orquestração, particionamento, testes de qualidade, documentação).

Entregáveis obrigatórios (checklist)

- ☐ Link do repositório Git (GitHub/GitLab) de entrega do seu projeto;
- ☐ README.md com instruções (Ex.: setup, variáveis de ambiente, como executar);
- ☐ Código fonte (scripts ou notebook) e requirements (requirements.txt);
- ☐ Dados finais exportados (/dashboard/output/JSON/parquet/...) pelo menos 1 arquivo particionado com os dados apresentáveis em suas versões finais;
- ☐ Relatório de Data Quality uma possivel apresentação ou (JSON/CSV);
- ☐ Documentação dos endpoints API's usados (URL + query params);
- ☐ Arquivo run.sh ou instruções para rodar localmente caso executável;
- ☐ Diferencial Video/demo de 5–10 minutos (ou apresentação ao vivo no prazo);

Enunciado

Extrair dados de no mínimo 2 APIs públicas diferentes, aplicar tratamento e modelagem, e entregar um dataset limpo e documentado, explicando decisões técnicas e criando um pequeno script/DAG/notebook que possibilite reexecução sem dependências.

1 - Fonte dos Dados

Escolha no mínimo 2 APIs de diferentes fontes (não ambas do mesmo domínio).
Indique na README.md os endpoints usados e documentação.
Informe o motivo da escolha dessas fontes de dados e qual o objetivo inicial com o uso desses dados no seu projeto.

2 – Extração dos Dados

Criar scripts ou requisições em Python para consumir as APIs (ex.: requests/httpx ou PySpark + HTTP libs). Gerar de forma bem documentada os passos de requisição possíveis Headers e comentar possíveis bibliotecas e funções utilizadas.
- Lidar com paginação e rate-limits.
- Autenticação (se necessário) usando variáveis de ambiente (não comitar chaves).

3 - Tratamentos / Transformações
(mínimo esperado, mas incentivado a ir além)

Nesta etapa, é esperado que todos realizem tratamentos e transformações nos dados brutos obtidos pelas APIs e suas fontes de dados, através do uso de ferramentas cotidianas de sua preferência (Jupyter Notebook, Databricks, PyCharm, VS Code, entre outras...), simulando o trabalho que um Engenheiro de Dados realizaria em pipelines reais. Isso inclui ações como:

- Limpeza de dados inconsistentes ou nulos;
- Conversão e padronização de formatos (datas, números, textos);
- Normalização e categorização de dados;
- Filtragens e extrações de colunas relevantes;
- Enriquecimento com dados complementares entre fontes;
- Complementação de dados entre as diversas fontes de dados gerando retrabalhos e novas informações para a nova forma de dados com a junção das fontes por meio de consultas e joins;

⚠️ Mínimo obrigatório: Cada projeto deve conter ao menos essas 5 transformações bem definidas, com tratamento adequado para os dados recebidos de cada API.

🚀 Mas queremos mais!

Vamos avaliar o grau de sofisticação e criatividade nas transformações aplicadas. Mesmo que opcional, o uso de técnicas mais avançadas (como pivotamentos dos campos e dados, cruzamento de dados entre fontes, análise de consistência entre registros, preenchimento inteligente de campos nulos, entre outros) será altamente valorizado e influenciará na percepção do nível técnico do projeto.

📌 Nossa expectativa é que cada membro nos surpreenda com soluções criativas e eficazes, simulando situações do dia a dia de um profissional de engenharia de dados. Usem o conhecimento de vocês com Python e demonstrem o potencial!

4 - Qualidade de Dados

Implementar e nos demonstrar algumas checagens básicas nas suas fontes de dados (counts, null_ratio por coluna, range checks) e salvar um relatório de qualidade (JSON/CSV) agregar essas informações em campos do relatório e fornecer o código ou consulta realizado para a geração dessas validações nos seus dados.

Ex.:

FONTE_DADOS	QTD_ROWS	NULL_RATIO_MED	RANGE_CHECKS
REST Countries	128179	11%	88%
OpenAQ	127874256	13%	92%
OpenWeatherMap	929387928	7%	96%
SpaceX API	893279846	26%	99%

5 – Persistência dos Dados

Após o processo de tratamento, é essencial que os dados estejam prontos para consumo eficiente, seja por ferramentas analíticas, sistemas externos ou outros pipelines de

dados. Nesta etapa, vocês devem aplicar práticas reais de engenharia de dados, considerando performance, organização e reuso futuro dos dados.

✅ O que deve ser entregue:
Persistência final dos dados tratados:
- Utilizar um formato de armazenamento para disponibilização dos dados, preferencialmente com compressão (ex: Snappy/Delta table/.Parquet);
- Aplicar particionamento por uma coluna temporal relevante (ex.: ano/mês, data ou similar), para refletir boas práticas de otimização no armazenamento;
- Estruturar os dados como um Data Lake organizado, considerando a escalabilidade e o consumo futuro.

Explicação técnica da persistência:
Incluir um documento ou célula no notebook explicando:
- Por que escolheram o formato em questão Ex.:Parquet;
- Por que usaram determinada estratégia de particionamento;
- Como essa organização dos dados contribui para o desempenho e escalabilidade;
- Quais os possíveis cenários de consumo futuro dos dados.

Visualização dos dados tratados:
Criar uma visualização mínima para mostrar que os dados estão prontos para uso analítico;
Exemplos aceitáveis:
- Notebook com gráficos em Python (usando matplotlib, seaborn, plotly, etc.);
- Dashboard simples em ferramentas como Power BI, Tableau, Google Data Studio ou até Dash/Streamlit;
- Consumo dos dados através de uso do R em RStudio ou similar;
- Qualquer outro formato que apresente insights visuais claros sobre os dados.

🎯 Objetivo desta etapa:
Queremos avaliar o nível de entendimento das boas práticas que um(a) Engenheiro(a) de Dados deve ter ao trabalhar com grandes volumes de dados e preparar esses dados para múltiplas formas de consumo (dashboards, análises ad-hoc, sistemas downstream, etc).

Esperamos que vocês mostrem uma visão crítica sobre performance, escalabilidade, organização e legibilidade dos dados, como se estivessem entregando isso em um projeto real.

Checklist técnico / estrutura de repo (exemplo de entrega)

Exemplo de possíveis entregas realizadas via Git, conforme uma estrutura esperada.

```
project-data-api/
├─ README.md
├─ requirements.txt
├─ run.sh
├─ src/
|  ├─ extract_api1.py
|  ├─ extract_api2.py
|  ├─ transform.py
|  └─ load.py
├─ tests/
|  └─ test_transform.py
├─ notebooks/
|  └─ exploration.ipynb
├─ output/
|  └─ final_dataset/
|     └─ year=2025/month=08/part-000.parquet
└─ dq_report/
   └─ dq_2025-08-11.json
```

Boas práticas a incentivar:
- Uso de variáveis de ambiente para keys (os.getenv("OPENWEATHER_KEY"));
- Tratamento de retries com backoff (tenacity ou requests + lógica);
- Versionamento no Git (commits claros);
- Não commitar credenciais chaves de login em APIs utilizadas;

Sugestões de APIs públicas (escolha no mínimo 2)

Abaixo estão pares recomendados como exemplos de fontes de dados, cada link leva à documentação oficial. Eu marquei se exige chave (API key) ou não.

Opções úteis e variadas:

- **BancoCentralBrasil** — Expectativas de mercado para os indicadores macroeconômicos da Pesquisa Focus.
  Portal de Dados Abertos do Banco Central do Brasil
- **OpenWeatherMap** — dados meteorológicos (requere API key gratuita).
  openweathermap.orgdocs.openweather.co.uk
- **OpenAQ** — qualidade do ar (não exige key; ótimo para dados ambientais).
  OpenAQ Docsapi.openaq.org
- **GitHub REST API** — repositórios, commits, issues (exige token para chamadas autenticadas; muita variedade). GitHub Docs+1
- **REST Countries** — dados estáticos sobre países (sem key).
  restcountries.comPublicAPI
- **Exchange rates / exchangerate.host** — cotações (sem key ou com tier free).
  exchangerate.hostexchangerate-api.com
- **SpaceX API** — dados de lançamentos e foguetes (sem key). Ótimo para trabalhar com eventos históricos. r/SpaceX API DocsGitHub

- **disease.sh (COVID / disease API)** — dados de saúde pública (sem key, público). disease.sh+1
- **PokeAPI** — dados de "Pokémon" (sem key) — útil para exercícios lúdicos (bom para joins e normalização). pokeapi.co+1

Exemplos de endpoints (rápido)

OpenWeatherMap (current weather) — GET https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}. openweathermap.org

OpenAQ (latest measurements) — GET https://api.openaq.org/v2/latest?city={city}. api.openaq.org

PokeAPI (pokemon) — GET https://pokeapi.co/api/v2/pokemon/{id or name}. pokeapi.co

disease.sh (COVID world) — GET https://disease.sh/v3/covid-19/all or .../countries/{country}.
disease.sh

(Para cada API escolhida, inclua no README os endpoints exatos usados.)

Sugestões de pairing de APIs (com casos de uso rápidos)

- **OpenWeatherMap + OpenAQ** — correlação clima × qualidade do ar (ex.: juntar por cidade/latlon e criar indicadores por dia). openweathermap.orgapi.openaq.org

- **REST Countries + exchangerate.host** — agregar metadados de país com taxas de câmbio para análise econômica (join por código de moeda). restcountries.com exchangerate.host

- **GitHub API + StackExchange API** — cruzar atividade de repositórios (commits) com perguntas relacionadas (ex.: projetos open source e sua comunidade). GitHub Docs

- **disease.sh + OpenAQ** — investigar correlações entre doenças e qualidade do ar (ex.: casos por região vs índices de poluição). disease.sh api.openaq.org

- **SpaceX + REST Countries** — mapeamento de lançamentos por país de operação (ex.: localizar launchpads e agregar por país). r/SpaceX API Docsrestcountries.com

- **PokeAPI + any metadata API** — exercício lúdico para trabalhar joins, normalização e hierarquia (não precisa key). pokeapi.co

Critérios de avaliação

- ☐ - **Extração & Confiabilidade**
- ☐ - **Limpeza & Transformações**
- ☐ - **Modelagem & Persistência**
- ☐ - **Qualidade / Testes**
- ☐ - **Produção / Orquestração**
- ☐ - **Documentação & Reprodutibilidade**
- ☐ - **Apresentação & Demonstração**