Classificação das estações meteorológicas de Pernambuco segundo dados do Inmet

André L. C. M. Silva, Caio L. Bezerra, Felipe C. Caminha, José B. O. Neto, Lucas S. F. Wanderley, Miguel C. Becker, Rodrigo T. M. Rodrigues

Centro de Estudos e Sistemas Avançados do Recife - C.E.S.A.R, Pernambuco - Brasil

alcms@cesar.school, clb@cesar.school, fcc3@cesar.school, jbon@cesar.school, lfsw@cesar.school, mcb4@cesar.school, rtmr@cesar.school

Abstract. *This article describes the development of a data pipeline designed to classify meteorological stations in the state of Pernambuco using data from the public Inmet API. The project follows the medallion architecture and uses pandas, SQLAlchemy, requests, and ML flow to orchestrate the ETL process, store the data, and train the machine learning model.*

Resumo. *Este artigo descreve a construção de um pipeline de dados para classificar estações meteorológicas de Pernambuco usando dados da API pública do Inmet. O projeto segue a arquitetura de medalhões e utiliza pandas, SQLAlchemy, requests e ML flow para orquestrar o ETL, armazenar os dados e treinar o modelo de machine learning.*

1. Entendimento Geral da arquitetura

A solução adota a arquitetura de medalhão, estruturada nas camadas Bronze, Prata e Ouro. A camada Bronze é responsável pela extração dos dados brutos a partir da API pública do Inmet, armazenando-os localmente sem qualquer tipo de pré-processamento. Na camada Prata, esses dados são tratados, padronizados e inseridos em um banco de dados PostgreSQL executado em container Docker, ficando disponíveis para consumo por ferramentas analíticas, como o Power BI. A camada Ouro realiza a normalização dos dados tratados com Standard Scaler, garantindo que todos os dados tenham média 0 e desvio padrão 1, gerando a versão final utilizada pelo processo de machine learning integrado ao ML Flow.

A automação do fluxo é feita por meio do Task Scheduler, que coordena a execução dos scripts responsáveis pela extração, transformação e normalização, bem como o acionamento dos containers Docker. Toda a comunicação com o banco de dados é feita via SQLAlchemy, enquanto as requisições à API são realizadas com a biblioteca requests.

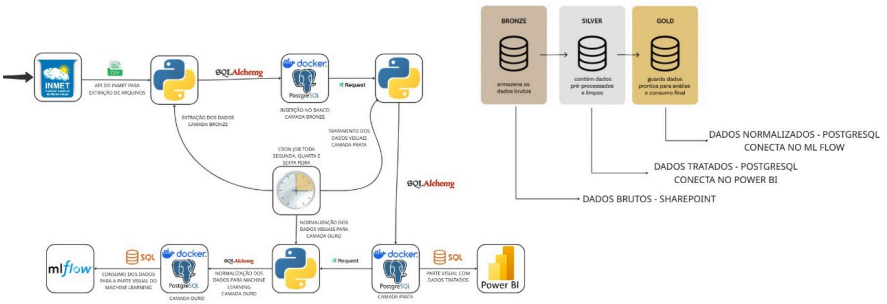

Figura 1. Prototipação da arquitetura do projeto

2. Tecnologias Utilizadas

No projeto citado foram utilizadas bibliotecas Python, banco de dados Postgresql, Docker, Power BI, API do Inmet e Task Scheduler.

2.1. Python

Além da conhecida e difundida linguagem de programação, utilizamos bibliotecas que funcionam como extensões dessa linguagem, são elas: **pandas** (responsável pelo tratamento de dados), **SQLAlchemy** (responsável pelo mapeamento e interações do Python com o banco de dados), **psycopg2** (responsável pela conexão com o banco de dados Postgresql), **Dotenv** (responsável pela confidencialidade de informações sensíveis do projeto), **Requests** (responsável por fazer requisições eficientes à API do Inmet) e **ML Flow** (Responsável pelo treinamento do modelo e aplicação do Machine Learning). Arquivos python são os responsáveis por retirar os dados da API, gerar arquivos csv, fazer seu tratamento e guardá-los no banco de dados, além de fazer a aplicação e o treinamento do modelo de Machine Learning.

2.2. Postgresql

Para guardar os dados e fazer com que a arquitetura medalhão projetada pudesse ser eficiente foi escolhido o banco de dados Postgresql, no qual denominamos de **bd_medalhao_inmet**, dentro do banco citado foram criadas três tabelas, **Silver_Inmet** (responsável por guardar os dados tratados e prontos para serem consumidos pela parte visual), **Gold_Inmet** (responsável por armazenar os dados normalizados, utilizados no treinamento do modelo) e a tabela **Cidades_Cluster** (responsável por armazenar os resultados obtidos através do ML Flow).

2.3. Docker

Para fazer a conexão e o acesso ao mesmo banco de dados em diferentes computadores, além de padronizar as versões das bibliotecas foi utilizado Docker, o qual persiste os dados mesmo que a imagem seja excluída e aberta novamente. Fazendo além de mais uma camada de segurança uma tecnologia ainda mais eficaz para a escalabilidade do projeto.

2.4. Power BI

Fomentando o visual do nosso projeto foi escolhida a ferramenta do Power BI, a qual foi integrada ao nosso banco de dados e consome as informações necessárias para produzir os gráficos apresentados, assim como extrair insights.

2.5. API do Inmet

Com o objetivo de abastecer a base de dados com informações reais utilizamos a API do Inmet, a qual fazemos requisições para obter os arquivos csv mais atualizados contendo dados que iniciam em 01/01/2020 e são atualizados hora a hora, com a informação se tornando público todo décimo dia do mês sucessor.

2.5. Task Scheduler

Finalizando e mantendo todo o pipeline orquestrado e ativo tem-se o Task Scheduler do Windows, o qual executa toda décimo dia do mês o script, a fim de atualizar os dados retornados pela API do Inmet e consequentemente o banco de dados e o Power BI. Segue configuração do cron job local:

3. Estrutura de Pastas do Projeto

- ❖ data/ (Pasta com dados iniciais, não tratados ou semi tratados)
- inmet_raw (Arquivos brutos retirados da API do Inmet)
- inmet_etl_bronze (Arquivos com tratamento mínimo para se tornarem minimamente usáveis)

- ❖ mlruns/ (Pasta com logs de cada um dos processos de treinamento do modelo que ocorreram)

- ❖ mlutils/ (Pasta com o script do banco de dados gerado pelo ML Flow)

- ❖ Arquivos na raiz do projeto
- .env (Informações de conexão com o banco de dados conectados ao Docker)
- .gitignore (Arquivos que foram gerados no código mas não subimos para o git)
- Docker-compose.yml (Orquestrador das imagens Docker)
- Dockerfile (Arquivo de configuração do Docker)
- Requirements.txt (Contém todas as bibliotecas que devemos baixar para executar o projeto)

- ❖ medalhao/ (Pasta com todos os códigos necessários para o pipeline e arquitetura do projeto funcionar da melhor maneira)
- connections.py (Responsável pela conexão com o banco de dados)
- Etl_bronze.py (Responsável pela limpeza para deixar os arquivos minimamente usáveis)
- Etl_gold.py (Responsável pela normalização dos dados)
- Etl_populate.py (Responsável por tratar os dados de teste)
- Etl_silver.py (Responsável pelo tratamento dos dados)
- extract.py (Responsável por fazer extração dos dados)
- main.py (Responsável por orquestrar o pipeline e fazer com ocorra tudo como previsto)
- Ml_train.py (Arquivo que faz o teste e implementa o modelo de Machine Learning)
- models.py (Responsável pela criação das tabelas e suas colunas no banco de dados)
- populate.py (Responsável por fazer acesso de teste a API do Inmet)

- ❖ trendz/ (Pasta com o arquivo do dashboard Power BI)

- ❖ reports/ (Pasta com o relatório do projeto em formato pdf)

4. Machine Learning

Fizemos um modelo que calcula pelo método da silhueta o k ideal, após isso faz a checagem pelo método do cotovelo, caso seja diferente ele tira a média entre os 2, prevalecendo em caso de números decimais o arredondamento mais próximo. Após achar a quantidade ideal de clusters rodamos o algoritmo do k-means para separar as cidades/estações meteorológicas em grupos, o k-means foi escolhido por ter dados somente numéricos, baixo número de variáveis, altamente eficiente e de fácil interpretação. Coloquei a regra que nenhuma cidade pode ficar isolada em um grupo, isso é de extrema importância para que possamos entender quais cidades têm características semelhantes quanto a atributos como temperatura, velocidade do vento, umidade e quantidade de chuva. Informação essa que é guardada no banco de dados e alimenta o Power BI, além de ter seus logs registrados na pasta mlruns. O processo para o treinamento correto do modelo se inicia com o agrupamento das médias diárias por cidade de cada uma das métricas observadas, após isso faz o agrupamento em clusters e salva na tabela do banco de dados Cidades_Cluster.

4.1. Cluster 0

Ficaram juntos no mesmo cluster as cidades que pertencem ao Agreste e a Zona da Mata por terem um clima mais úmido e ameno, tendo entre elas a temperatura mais baixa entre os grupos, maior umidade relativa do ar, taxa de precipitação mais alta e mais distribuída, além de velocidade do vento moderado. Resultando nas cidades de Caruaru, Garanhuns e Palmares.

4.2. Cluster 1

Ficaram juntos no mesmo cluster as cidades que pertencem ao Sertão semiárido por terem um clima mais quente e seco, tendo entre elas baixa umidade, pouca chuva, baixa precipitação e ventos mais fracos. Resultando nas cidades de Cabrobó, Floresta, Ouricuri, Petrolina e Salgueiro.

4.3. Cluster 2

Ficaram juntos no mesmo cluster as cidades que pertencem à transição entre Agreste e Sertão por terem um clima intermediário, tendo temperatura mediana, precipitação moderada, umidade variável durante o ano e velocidade do vento constante. Resultando nas cidades de Arcoverde, Ibimirim, Serra Talhada e Surubim.

5. Passo a Passo para Execução

1. Constrói as imagens
    a. docker-compose build
2. Sobe os serviços
    a. docker-compose up
3. Para todos os containers
    a. docker-compose down

4. Para iniciar o visual do ML Flow
    a. python -m mlflow ui --port 5000
5. Para acessar URL do ML Flow
    a. http://127.0.0.1:5000
6. Agora basta abrir o arquivo .pbix que está na pasta trendz/

6. Resultado Visual

Abaixo prints do nosso arquivo com o resultado visual do projeto desenvolvido.

Figura 3. Página inicial do projeto

Figura 4. Página com dados detalhados por estação meteorológica

7. Links Úteis

GitHub:

- ➔ https://github.com/andrecastrom06/Pipeline_Inmet.git

Especificações do Projeto:

- ➔ https://classroom.google.com/c/Nzc5NzcyODMwNjQx/m/NzkyMTc1ODgwNjE4/details