Relatório - Modelo de Predição

**API - Machine Learning**

# **Estrutura de Arquivos**

sauter-university-waterworks/

├── api/

│ ├── main.py # funções de coleta

│ ├── upload\_gcp.py # Classe de upload GCP

│ └── chave/ # Pasta para chave GCP (não versionada)

│ └── sauter-university-waterworks-aa1bc765f223.json

│

├── dados\_ons/ # Pasta criada automaticamente para dados

│ ├── ENA\_2025\_original.csv # CSVs baixados

│ └── ENA\_2025.parquet # Parquets processados

│

├── logs/ # Pasta criada automaticamente para logs

│ ├── ons\_collector.log # Logs do processamento

│ └── api.log # Logs da API

│

├── requirements.txt # Dependências Python (completado)

├── requisicao.py # API com logging básico e paginação

├── Dockerfile # Container da aplicação

├── docker-compose.yml # Orquestração Docker

└── README.md # Documentação do projeto

# **Principais Componentes**

## **1. main.py**

### **main**

Contém toda a lógica de ETL (Extract, Transform, Load):

* baixar\_por\_resource\_id(resource\_id, nome\_arquivo)  
  Baixa um arquivo CSV do ONS usando o resource\_id do dataset.
* limpar\_nome\_colunas(nome\_coluna)  
  Normaliza os nomes de colunas (substitui caracteres inválidos por \_).
* df\_para\_parquet\_em\_string(df, arquivo\_parquet)  
  Converte qualquer DataFrame em arquivo Parquet com todas as colunas como STRING.
* processar\_arquivos(data\_inicio, data\_fim)  
  Função principal que:
  + Acessa a API do ONS (package\_show).
  + Identifica os arquivos referentes aos anos desejados.
  + Baixa os CSVs e carrega em Pandas.
  + Filtra pelo intervalo de datas (ena\_data).
  + Converte para Parquet (otimizado).
  + Gera relatório com tamanhos e taxa de compressão.

## **2. upload\_gcp.py**

### **upload\_gcp**

Classe responsável por enviar os arquivos para o Google Cloud Storage:

* UploadGCP.\_\_init\_\_(id\_projeto, nome\_bucket, caminho\_chave)  
  Inicializa cliente autenticado do GCP usando chave de serviço.
* upload\_para\_storage(caminho\_local, nome\_blob)  
  Faz upload do arquivo local para o bucket no caminho definido.

## **3. requisicao.py**

### **requisicao**

Implementa a API FastAPI que orquestra o fluxo:

#### **Configurações**

Define chave GCP (CAMINHO\_CHAVE), ID do projeto e bucket.

#### **Modelo IntervaloDatas**

Define payload esperado no POST:

{

"inicio": "2025/01/01",

"fim": "2025/09/20"

}

#### **Endpoint /coletar (POST)**

* Cria pasta local dados\_ons/.
* Executa main.processar\_arquivos para gerar os Parquets.
* Faz upload de cada arquivo para o GCS via UploadGCP.
* Retorna relatório JSON com caminhos e status de upload.

**Fluxo do Sistema**

1. Usuário chama a API via POST /coletar enviando datas de início e fim.
2. O sistema acessa a API do ONS e identifica quais arquivos CSV contêm os dados do período.
3. Faz download dos CSVs correspondentes e os salva em dados\_ons/.
4. Filtra as linhas pelo campo ena\_data dentro do intervalo solicitado.
5. Converte os resultados em arquivos Parquet (colunas normalizadas e tipadas como STRING).
6. Faz upload para o bucket do GCP, organizando com partição de data de ingestão:  
   gs://<NOME\_BUCKET>/raw/reservatorios\_ena/ingestion\_date=YYYY-MM-DD/ENA\_<ANO>.parquet
7. Retorna um JSON com status de cada arquivo.

**Exemplo de Uso**

#### **1. Rodar API localmente**

uvicorn api.requisicao:app --reload

#### **2. Chamada via curl**

curl -X POST "http://127.0.0.1:8000/coletar" \

-H "Content-Type: application/json" \

-d '{"inicio": "2025/01/01", "fim": "2025/09/20"}'

#### **3. Resposta esperada**

{

"mensagem": "Processamento e uploads concluídos",

"intervalo": {

"inicio": "2025/01/01",

"fim": "2025/09/20"

},

"resultados": [

{

"arquivo": "ENA\_2024.parquet",

"upload\_gcs": true,

"blob\_path": "gs://datalake-ons-ena/raw/reservatorios\_ena/ingestion\_date=2025-09-25/ENA\_2024.parquet"

},

{

"arquivo": "ENA\_2025.parquet",

"upload\_gcs": true,

"blob\_path": "gs://datalake-ons-ena/raw/reservatorios\_ena/ingestion\_date=2025-09-25/ENA\_2025.parquet"

}

]

}

**Observações**

* A pasta api/chave/ deve conter a chave JSON do serviço GCP com permissão de Storage Admin.
* Essa chave não deve ser versionada no Git.
* O bucket no GCP (datalake-ons-ena) precisa existir previamente.

**Objetivo da pipeline:**

O objetivo da pipeline completo para ingestão, tratamento e disponibilização dos dados de reservatórios da ENA.   
 A ideia era permitir análises confiáveis e eficientes, mantendo histórico completo, evitando duplicadas, nulos, dados inconsistentes e dados com o valor 0.0 que podem interferir diretamente na análise e previsão .

**Etapa 1: Organização do Bucket no GCS**

Pensamento inicial: Eu queria ter uma estrutura clara e escalável para armazenar os dados, que permitisse fácil ingestão incremental e consultas eficientes.   
 Decidi usar a seguinte estrutura:

datalake-ons-ena/   
 raw/   
 reservatorios\_ena/   
 ingestion\_date=YYYY-MM-DD/   
 arquivo\_XXXX.parquet

Motivos da escolha:

* Particionar pelo caminho ingestion\_date=YYYY-MM-DD permite que o BigQuery detecte automaticamente a data de ingestão e leia apenas o que for necessário.
* Colocar os dados brutos na pasta raw garante que possamos sempre ter os dados originais intactos.
* Arquivos em Parquet: com o chema obrigatoriamente de todas as colunas como string.
* Essa organização facilita futuras análises e integrações com pipelines automatizados e evita que ela quebre.

**Etapa 2: Criação da Tabela Externa Raw**

Precisava consultar os arquivos no GCS sem mover os dados para BigQuery.   
 Criei uma tabela externa particionada pela data de ingestão para permitir consultas incrementais e criação das próximas tabelas.

Query utilizada:

CREATE OR REPLACE EXTERNAL TABLE sauter-university-waterworks.raw.bacias\_ena\_raw (   
 nom\_bacia STRING,   
 ena\_data STRING,   
 ena\_bruta\_bacia\_mwmed STRING,   
 ena\_bruta\_bacia\_percentualmlt STRING,   
 ena\_armazenavel\_bacia\_mwmed STRING,   
 ena\_armazenavel\_bacia\_percentualmlt STRING   
 )   
 WITH PARTITION COLUMNS (   
 ingestion\_date DATE   
 )   
 OPTIONS (   
 format = 'PARQUET',   
 uris = ['gs://datalake-ons-ena/raw/reservatorios\_ena/\*'],   
 hive\_partition\_uri\_prefix = 'gs://datalake-ons-ena/raw/reservatorios\_ena/',   
 require\_hive\_partition\_filter = false   
 );

Raciocínio:

* Mantive todas as colunas como STRING inicialmente para evitar problemas de leitura, e esse tratamento vai ser feito futuramente, como aqui é a tabela raw não precisa de nenhum tratamento além da partição.
* Partição por ingestion\_date permite que consultas filtradas leiam apenas os arquivos da data desejada, economizando custo e tempo.
* hive\_partition\_uri\_prefix garante que o BigQuery entenda corretamente a estrutura Hive-style do diretório.

**Etapa 3: Criação da Tabela Trusted**

Queria uma tabela interna, limpa, tipada corretamente, pronta para análise e com histórico incremental.   
 Decidi criar a tabela particionada também por ingestion\_date, consistente com a raw.

Query utilizada:

CREATE TABLE sauter-university-waterworks.processed.bacias\_ena\_trusted (   
 nom\_bacia STRING,   
 ena\_data DATE,   
 ena\_bruta\_bacia\_mwmed FLOAT64,   
 ena\_bruta\_bacia\_percentualmlt FLOAT64,   
 ena\_armazenavel\_bacia\_mwmed FLOAT64,   
 ena\_armazenavel\_bacia\_percentualmlt FLOAT64,   
 ingestion\_date DATE   
 )   
 PARTITION BY ingestion\_date;

Raciocínio:

* Tipagem correta evita erros de conversão e garante consistência.
* Partição permite consultas eficientes por ingestão diária.
* Tabela interna garante que os dados tratados fiquem disponíveis mesmo que os arquivos brutos mudem.

**Etapa 4: Procedure para popular a Trusted**

Eu precisava automatizar a limpeza, tipagem, remoção de duplicadas e inserção incremental dos dados da raw para a trusted.   
 O objetivo era garantir que:

* Nenhum valor nulo ou inválido fosse inserido.
* Linhas duplicadas fossem removidas.
* Apenas os dados corretos fossem mantidos.

Query utilizada:

CREATE OR REPLACE PROCEDURE sauter-university-waterworks.processed.popular\_trusted\_bacias()   
 BEGIN   
 MERGE sauter-university-waterworks.processed.bacias\_ena\_trusted T   
 USING (   
 WITH cleaned\_source\_data AS (   
 SELECT   
 COALESCE(nom\_bacia, '') AS nom\_bacia,   
 PARSE\_DATE('%Y-%m-%d', ena\_data) AS ena\_data,   
 SAFE\_CAST(ena\_bruta\_bacia\_mwmed AS FLOAT64) AS ena\_bruta\_bacia\_mwmed,   
 SAFE\_CAST(ena\_bruta\_bacia\_percentualmlt AS FLOAT64) AS ena\_bruta\_bacia\_percentualmlt,   
 SAFE\_CAST(ena\_armazenavel\_bacia\_mwmed AS FLOAT64) AS ena\_armazenavel\_bacia\_mwmed,   
 SAFE\_CAST(ena\_armazenavel\_bacia\_percentualmlt AS FLOAT64) AS ena\_armazenavel\_bacia\_percentualmlt,   
 ingestion\_date   
 FROM sauter-university-waterworks.raw.bacias\_ena\_raw   
 )   
 SELECT \*   
 FROM cleaned\_source\_data   
 WHERE ena\_bruta\_bacia\_mwmed > 0   
 AND ena\_bruta\_bacia\_percentualmlt > 0   
 AND ena\_armazenavel\_bacia\_mwmed > 0   
 AND ena\_armazenavel\_bacia\_percentualmlt > 0   
 QUALIFY ROW\_NUMBER() OVER(   
 PARTITION BY nom\_bacia, ena\_data   
 ORDER BY ingestion\_date DESC   
 ) = 1   
 ) R   
 ON T.nom\_bacia = R.nom\_bacia AND T.ena\_data = R.ena\_data   
 WHEN MATCHED THEN   
 UPDATE SET   
 ena\_bruta\_bacia\_mwmed = R.ena\_bruta\_bacia\_mwmed,   
 ena\_bruta\_bacia\_percentualmlt = R.ena\_bruta\_bacia\_percentualmlt,   
 ena\_armazenavel\_bacia\_mwmed = R.ena\_armazenavel\_bacia\_mwmed,   
 ena\_armazenavel\_bacia\_percentualmlt = R.ena\_armazenavel\_bacia\_percentualmlt,   
 ingestion\_date = R.ingestion\_date   
 WHEN NOT MATCHED THEN   
 INSERT (nom\_bacia, ena\_data, ena\_bruta\_bacia\_mwmed, ena\_bruta\_bacia\_percentualmlt,   
 ena\_armazenavel\_bacia\_mwmed, ena\_armazenavel\_bacia\_percentualmlt, ingestion\_date)   
 VALUES (R.nom\_bacia, R.ena\_data, R.ena\_bruta\_bacia\_mwmed, R.ena\_bruta\_bacia\_percentualmlt,   
 R.ena\_armazenavel\_bacia\_mwmed, R.ena\_armazenavel\_bacia\_percentualmlt, R.ingestion\_date);   
 END;

Raciocínio detalhado:

* Usei MERGE incremental para atualizar linhas existentes e inserir novas, garantindo consistência.
* A CTE cleaned\_source\_data transforma os tipos e aplica tratamento inicial.
* Filtro > 0 garante que linhas com valores inválidos (0 ou nulos) sejam ignoradas.
* QUALIFY ROW\_NUMBER() remove duplicadas mantendo a linha mais recente.
* A procedure não tem parâmetros, processa todas as ingestions disponíveis.

**Etapa 5: Chamada da Procedure**

Query utilizada:

CALL sauter-university-waterworks.processed.popular\_trusted\_bacias();

* Chamada simples e direta, pronta para ser agendada com Cloud Scheduler para execução diária.
* Evita necessidade de especificar datas manualmente.

**Conclusão:**

O projeto estruturou uma pipeline de dados confiável e escalável:

* Bucket GCS organizado por ingestão.
* Tabela externa raw para consulta direta de arquivos.
* Tabela trusted limpa, tipada e particionada.
* Procedure para automação e consistência dos dados.
* Boas práticas aplicadas: particionamento, tipagem, remoção de duplicadas, filtro de valores inválidos.

**Objetivo do Modelo Preditivo:**

O objetivo deste modelo preditivo é estimar a Evolução de Energia Natural Afluente (ENA**)** das bacias hidrológicas contempladas no banco de dados do ONS com a máxima similaridade possível em relação aos valores históricos registrados. A similaridade é medida preferencialmente por métricas padronizadas de erro e correlação (MAPE, MAE), de forma que as previsões sejam confiáveis.

**Ferramenta Utilizada: JupyterLab**

Optei peloJupyterLab por sua integração nativa com a Google Cloud Platform (GCP), em especial com o Vertex AI, o que possibilita treinamento, validação e deploy de modelos de forma escalável e rastreável. Além disso, sua interface interativa acelera o desenvolvimento exploratório e garante produtividade acadêmica, enquanto os recursos de MLOps do Vertex AI asseguram governança e confiabilidade em ambiente empresarial.

**Etapa 1: Preparação dos dados**

Foi desenvolvido um sistema automatizado em Python para coleta dos dados históricos de Energia Natural Afluente (ENA), disponibilizados pelo Operador Nacional do Sistema Elétrico (ONS), cobrindo o período de 2000 a 2025 (26 anos). Os dados são obtidos no formato CSV diretamente do repositório público do ONS em AWS S3 e armazenados de forma organizada em diretórios anuais.

A implementação foi estruturada em funções modulares, que permitem reutilização e manutenção simplificada:

* download\_ena\_data(year) - realiza o download dos arquivos anuais diretamente da fonte oficial, com verificação de status HTTP e tratamento de exceções.
* save\_to\_directory(file, path) - garante a criação automática de diretórios e armazenamento padronizado por ano.
* validate\_response() - confirma a integridade do conteúdo antes de salvar (checagem de resposta válida).
* main() - função orquestradora, responsável por percorrer o período configurado (2000–2025), contabilizar os downloads realizados e registrar o progresso via logs.

O sistema inclui estratégias de robustez como timeout de 30 segundos, intervalo de 0,5 segundos entre requisições para evitar sobrecarga no servidor, e logging detalhado para acompanhamento em tempo real. O design modular facilita ajustes, como alteração do período analisado ou destino de armazenamento.

**Etapa 2: Implementação do modelo ARIMA**

O sistema de modelagem preditiva baseado em ARIMA foi desenvolvido para previsão da Energia Natural Afluente (ENA), com foco em confiabilidade e robustez estatística. Sua arquitetura modular foi organizada em funções especializadas: ENADataProcessor, responsável pelo carregamento, limpeza e agregação dos dados; TemporalSplitter, que executa a divisão temporal em treino, validação e teste de forma cronológica; RigorousARIMAModel, que aplica testes de estacionariedade, seleção automática de ordens e diagnósticos residuais; e ENAARIMAModel, que integra o fluxo completo, executando validação walk-forward e consolidando métricas de performance.

Além das funções centrais de modelagem, o sistema incorpora rotinas de fallback, que testam diferentes configurações (ARIMA simplificado, AR(1), média móvel) para garantir previsões mesmo em condições críticas. Funções auxiliares, como adf\_test() e kpss\_test(), asseguram a robustez estatística, enquanto os módulos de visualização e logging permitem acompanhamento transparente dos resultados e geração de relatórios empresariais.

**Conclusão:**

O projeto apresentou uma solução completa e integrada para o ciclo de dados e modelagem preditiva da Energia Natural Afluente (ENA). A primeira etapa, de coleta automatizada, assegurou a construção de uma base histórica robusta (2000–2025), organizada e rastreável, eliminando a necessidade de processos manuais e garantindo qualidade, consistência e escalabilidade no acesso às informações oficiais disponibilizadas pelo ONS.

Na segunda etapa, foi desenvolvido o sistema de modelagem ARIMA, estruturado em arquitetura modular e orientada a funções. Essa camada de análise incorporou testes estatísticos rigorosos, validação temporal via walk-forward e mecanismos de fallback, garantindo previsões robustas, auditáveis e altamente similares aos dados reais. A combinação de métricas de desempenho (MAE, RMSE, MAPE) com diagnósticos de resíduos (ADF, KPSS, Ljung-Box, Jarque-Bera) assegurou tanto a precisão preditiva quanto a validade estatística do modelo.

**Objetivo do Dashboard:**

O objetivo deste dashboard é fornecer uma ferramenta de visualização de dados interativa e intuitiva para a análise dos dados de reservatórios da ENA. A plataforma permite que usuários explorem tendências históricas, comparem o desempenho entre diferentes bacias hidrográficas e identifiquem rapidamente status críticos, tudo a partir de uma fonte de dados confiável e centralizada.

**Ferramenta Utilizada: Looker Studio (Google)**

* **Motivos da Escolha:** Optamos pelo Looker Studio por sua integração nativa e de alta performance com o Google BigQuery, eliminando a necessidade de exportar dados. Sua interface *drag-and-drop* acelera o desenvolvimento, e suas funcionalidades de interatividade (filtros, controles de período) permitem a criação de uma experiência de análise rica e customizável para o usuário final.

**Etapa 1: Fonte de Dados (Data Source)**

* **Pensamento inicial:** Para garantir a precisão e a performance do dashboard, era fundamental conectá-lo diretamente à tabela mais limpa e confiável do nosso pipeline de dados.
* **Implementação:**
* **Conector:** Google BigQuery.
* **Projeto:** sauter-university-waterworks
* **Dataset:** processed
* **Tabela:** bacias\_ena\_trusted
* **Justificativa:** Utilizamos a tabela bacias\_ena\_trusted como única fonte da verdade. Esta tabela é o resultado final do nosso pipeline de ETL, contendo dados já limpos, tipados, desduplicados e validados. Conectar o dashboard a esta tabela garante que todas as visualizações reflitam dados consistentes e de alta qualidade, prontos para a análise.

**Etapa 2: Componentes Visuais e Justificativas**

O dashboard foi estruturado para contar uma história, guiando o usuário do geral para o específico.

**1. Indicadores Chave (Scorecards)**

* **Descrição:** Exibem métricas únicas e agregadas em destaque no topo do dashboard, como "Média de ENA Bruta (MWmed)" e "Total de Reservatórios em Nível Crítico".
* **Justificativa:** Scorecards oferecem uma visão macro e imediata dos indicadores de performance mais importantes (KPIs). Eles são o ponto de partida da análise, respondendo à pergunta: "Qual é a situação geral agora?".

**2. GRÁFICO DE SÉRIE TEMPORAL**

* **Descrição:** Um gráfico de linhas mostrando a evolução da ena\_bruta\_bacia\_mwmed ao longo do tempo (ena\_data).
* **Justificativa:** É a melhor visualização para analisar **tendências, ciclos e sazonalidade**. Permite ao usuário identificar padrões de cheia e seca, comparar o desempenho atual com períodos anteriores e detectar anomalias ou desvios significativos ao longo do tempo.

**3. Gráfico de Barras Horizontais**

* **Descrição:** Um ranking das bacias (nom\_bacia) ordenado pela ena\_armazenavel\_bacia\_percentualmlt.
* **Justificativa:** O gráfico de barras é ideal para **comparações e ranqueamento**. Ele responde de forma clara à pergunta: "Quais bacias estão com os maiores ou menores percentuais de energia armazenável?". A orientação horizontal facilita a leitura dos nomes das bacias.

**Objetivo do Levantamento de Infraestrutura:**

O objetivo para este projeto é estabelecer uma base de operações e segurança robusta e totalmente automatizada no Google Cloud Platform, provisionando os ambientes essenciais para o modelo preditivo de Machine Learning. O levantamento visa automatizar o ciclo de vida completo da infraestrutura e do código.

**Ferramentas utilizadas: Terraform (HashiCorp) e Github Actions (Github)**

O Terraform foi escolhido como a ferramenta fundamental para o levantamento da infraestrutura (IaC - Infrastructure as Code) neste projeto devido a uma combinação de funcionalidades que atendem diretamente aos requisitos de automação, governança e boas práticas de DevOps e Engenharia de Software. O Terraform é uma ferramenta de código aberto que codifica APIs em arquivos de configuração declarativos, com a infraestrutura descrita usando uma sintaxe de configuração de alto nível (HCL). A ferramenta garante que o ambiente seja replicável e idêntico em diferentes etapas do ciclo de vida do software, eliminando erros causados por processos manuais.

O GitHub Actions foi escolhido como a ferramenta de *pipeline* (CI/CD - Integração Contínua/Entrega Contínua) para este projeto devido à sua integração nativa com o código-fonte e sua capacidade de automatizar o ciclo de vida completo do software, desde a construção da infraestrutura até a implantação da aplicação, seguindo as diretrizes de DevOps e segurança. A plataforma GitHub é o repositório de código-fonte utilizado pelo projeto. Ao usar o GitHub Actions, a automação está diretamente integrada ao repositório Git, facilitando a adesão às boas práticas de versionamento (branch, PR, reviews) e a rastreabilidade das mudanças de infraestrutura e aplicação.