**Digital Twin com Previsão de Falhas (Manutenção Preditiva):**

**Data iniciada: 22/06/25**

**Tempo trabalhado: 9H 10min**

Objetivo: Criar um modelo preditivo que simule o comportamento de uma máquina industrial e alerte sobre prováveis falhas futuras.

Descrição:

* Simule sensores (temperatura, vibração, rotação, corrente) com geração de dados em tempo real (ex: com ruído + eventos de falha).
* Modele um gêmeo digital da máquina usando dados históricos e de simulação.
* Treine algoritmos como Random Forest, LSTM ou XGBoost para prever tempo até falha (RUL).
* Forneça interface de visualização de status e previsões com alertas.

Stack:

Python, Pandas, Scikit-Learn, Keras, Grafana, InfluxDB, MQTT, Docker, Kubernetes, MLflow.

Destaques:

* Projeto completo de MLOps com ingestão, modelagem e monitoramento.
* Demonstra familiaridade com digital twins e predição industrial.

**Projeto Completo: Digital Twin com Previsão de Falhas (Manutenção Preditiva)**

**1. Objetivo  
Desenvolver um sistema completo baseado em Digital Twin para previsão de falhas de máquinas industriais. O sistema deve simular sensores em tempo real, prever o Remaining Useful Life (RUL) da máquina, emitir alertas pré-falha e disponibilizar visualização via dashboard.**

**2. Stack Tecnológica**

* **Linguagem principal: Python**
* **Bibliotecas de dados e modelagem:**
  + **Pandas, NumPy**
  + **Scikit-Learn**
  + **TensorFlow/Keras**
  + **XGBoost**
* **Simulação e ingestão de dados:**
  + **MQTT (Mosquitto Broker)**
  + **Python (geradores com ruído estocástico)**
* **Armazenamento e tempo real:**
  + **InfluxDB (time-series)**
* **Visualização:**
  + **Grafana**
* **Deploy e orquestração:**
  + **Docker**
  + **Kubernetes**
* **MLOps:**
  + **MLflow (rastreamento de experimentos e versionamento de modelos)**

**3. Componentes do Projeto**

**3.1 Simulação de Dados de Sensores**

* **Sensores simulados:**
  + **Temperatura**
  + **Vibração**
  + **Corrente elétrica**
  + **Rotação (RPM)**
* **Lógica de geração:**
  + **Baseado em distribuições Gaussianas com ruído**
  + **Introduzir "eventos de falha" (picos ou anomalias crescentes)**
  + **Enviar dados via MQTT para InfluxDB**

**3.2 Banco de Dados - InfluxDB**

* **Time-series de sensores**
* **Consultas via Grafana e scripts Python (InfluxDB-Python Client)**

**3.3 Modelagem do Gê̂meo Digital**

* **Treinamento com dados simulados e/ou históricos**
* **Algoritmos:**
  + **Random Forest (baseline)**
  + **XGBoost (alta performance)**
  + **LSTM (séries temporais)**
* **Métrica alvo:**
  + **RUL (Remaining Useful Life)**

**3.4 Deploy com Docker/Kubernetes**

* **Containerizar os serviços:**
  + **Simulador de sensores**
  + **API de previsão**
  + **InfluxDB + Grafana**
  + **Serviço MQTT**
* **Gerenciar com Kubernetes: escalabilidade e atualização**

**3.5 Interface de Visualização (Grafana)**

* **Dashboards:**
  + **Estado atual dos sensores**
  + **RUL previsto por algoritmo**
  + **Alertas de falha (via thresholding)**

**3.6 MLOps com MLflow**

* **Registro de modelos treinados**
* **Comparativo entre experimentos**
* **Deploy automático de melhor modelo**

**4. Conhecimentos Necessários**

* **Programacão em Python**
* **Machine Learning supervisionado e séries temporais**
* **Engenharia de dados e time-series**
* **Conceitos de Digital Twin**
* **Banco de dados InfluxDB**
* **Monitoramento com Grafana**
* **MLOps: experiment tracking, model registry (MLflow)**
* **Containeração com Docker e Kubernetes**
* **MQTT para IoT**

**5. Fontes de Dados Reais / Simulados para Treinamento**

**5.1 Bases de Dados Públicas para RUL/Previsão de Falhas**

* **CMAPSS Dataset (NASA):**
  + [**https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository**](https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository)
* **PHM Society Data Challenge:**
  + [**https://www.phmsociety.org/events/data-challenges/**](https://www.phmsociety.org/events/data-challenges/)
* **Kaggle - Predictive Maintenance:**
  + [**https://www.kaggle.com/datasets/shubhendra7/air-pressure-system-aps-failure-at-scania-trucks**](https://www.kaggle.com/datasets/shubhendra7/air-pressure-system-aps-failure-at-scania-trucks)
* **SECOM dataset (semicondutores):**
  + [**https://archive.ics.uci.edu/ml/datasets/secom**](https://archive.ics.uci.edu/ml/datasets/secom)

**5.2 Gerador de Dados de Sensores Customizados**

* **Criar com NumPy e Scikit-Learn**
* **Alternativa: usar Simulink, AnyLogic ou MATLAB para dados mais realistas**

**6. Recursos e Bibliotecas úteis**

* **InfluxDB Python Client:**
  + [**https://github.com/influxdata/influxdb-client-python**](https://github.com/influxdata/influxdb-client-python)
* **Grafana + InfluxDB Tutorial:**
  + [**https://grafana.com/docs/grafana/latest/datasources/influxdb/**](https://grafana.com/docs/grafana/latest/datasources/influxdb/)
* **MLflow:**
  + [**https://mlflow.org/docs/latest/index.html**](https://mlflow.org/docs/latest/index.html)
* **Docker + Kubernetes:**
  + **Curso: Kubernetes da Linux Foundation (LFCS)**
* **Digital Twin conceitual:**
  + **Siemens, GE e IBM white papers**

**7. Estrutura de Diretórios Recomendada**

**project/**

**├── data/**

**│ ├── raw/**

**│ ├── processed/**

**├── notebooks/**

**├── models/**

**├── src/**

**│ ├── simulation/**

**│ ├── preprocessing/**

**│ ├── training/**

**│ ├── deployment/**

**├── docker/**

**├── kubernetes/**

**├── grafana\_dashboards/**

**├── mlruns/ (MLflow)**

**├── README.md**

**Explicação dos diretórios:**

**📁 data/**

**Repositório dos dados brutos, processados e possivelmente intermediários.**

**▸ raw/**

**Dados crus originais, sem nenhuma transformação.**

* **Exemplo: arquivos como train\_FD001.txt, test\_FD001.txt, RUL\_FD001.txt.**
* **Não se deve editar nada aqui manualmente.**

**▸ processed/**

**Dados após pré-processamento, normalização, geração de labels (RUL), e extração de features.**

* **Exemplo: arquivos .csv ou .npy com colunas como RUL, sensor\_2\_scaled, delta\_sensor\_11, etc.**
* **Pode conter janelas temporais, dados empilhados por timestep ou séries com padding para modelos LSTM.**

**📁 docker/**

**Ambiente de containerização com imagens auxiliares.**

* **Pode conter imagens para bancos de dados simulados, MLFlow Server, Prometheus, etc.**
* **Útil para manter o ambiente reproduzível e portátil.**

**📄 Dockerfile**

**Define a imagem base do seu projeto.**

* **Exemplo: imagem Python com scikit-learn, TensorFlow, pandas, matplotlib, mlflow, etc.**
* **Garante que o treinamento e o deploy usem exatamente o mesmo ambiente.**

**📁 grafana\_dashboards/**

**Dashboards para monitoramento do modelo em produção.**

* **Exemplo: painel de falhas preditas vs reais por motor.**
* **Pode se conectar ao MLflow, Prometheus, InfluxDB, ou logs do Kubernetes.**

**📁 kubernetes/**

**Arquivos de orquestração com Kubernetes YAML.**

* **Exemplo: deployment.yaml, service.yaml para seu modelo de IA ser executado como pod.**
* **Pode conter configuração de autoescalonamento, logging e monitoramento.**

**📁 mlruns/ (MLflow)**

**Diretório de experimentos de IA registrados via MLflow.**

* **Aqui ficam os logs automáticos de cada experimento, hiperparâmetros, métricas, artefatos.**
* **Permite reverter modelos, comparar versões e até servir modelos via REST API.**

**📁 models/**

**Contém os modelos treinados, checkpoints e metainformações.**

* **Exemplo: arquivos .h5, .pt, .joblib, ou salvos pelo MLflow.**
* **Pode haver subpastas por versão: v1/, v2/, final/.**

**📁 notebooks/**

**Prototipagem, visualizações e análise exploratória (EDA).**

* **Jupyter notebooks para:**
  + **Ver distribuição dos sensores.**
  + **Ver evolução do RUL.**
  + **Testar arquiteturas iniciais.**

**📁 src/ (coração do código)**

**▸ deployment/**

**Código para servir o modelo treinado (API REST, gRPC, etc.).**

* **Pode usar FastAPI, Flask, TorchServe, TensorFlow Serving, etc.**
* **Exemplo: endpoint POST /predict que recebe dados do motor e retorna RUL.**

**▸ preprocessing/**

**Scripts para limpar, normalizar e transformar dados.**

* **normalize\_data.py, generate\_labels.py, sliding\_window.py, etc.**
* **Contém lógica para transformar raw → processed.**

**▸ simulation/**

**Códigos que simulam o sistema físico (Digital Twin) ou geram dados sintéticos.**

* **Exemplo: simulação de um motor com falhas programadas.**
* **Pode ser também um modelo físico/matemático (ODEs, termodinâmica, desgaste).**

**▸ training/**

**Código para treinar os modelos de IA.**

* **train.py, train\_lstm.py, train\_xgboost.py.**
* **Contém definição do modelo, função de perda, métrica, callbacks, salvamento de modelos, integração com MLflow.**

**8. Cronograma de Desenvolvimento (Sprints)**

* **Sprint 1 - Setup:**
  + **Instalação de Docker, InfluxDB, Grafana, MQTT**
  + **Criação do simulador de sensores**
* **Sprint 2 - Ingestão e Armazenamento:**
  + **Pipeline MQTT → InfluxDB**
  + **Dashboard inicial no Grafana**
* **Sprint 3 - Modelagem Preditiva:**
  + **Engenharia de features + treinamento RUL**
  + **Teste com Random Forest, XGBoost e LSTM**
* **Sprint 4 - MLOps:**
  + **Integração MLflow**
  + **Containerização do pipeline**
* **Sprint 5 - Deploy e Monitoramento:**
  + **Kubernetes + escalabilidade**
  + **Alertas em tempo real**

**9. Conclusão  
Este projeto cobre o ciclo completo de um sistema preditivo industrial com digital twin, combinando simulação de sensores, machine learning, visualização e MLOps. É uma excelente demonstração de competências em IA industrial, engenharia de dados e automação de sistemas críticos.**

**10. Anexos e Complementos Sugeridos**

* **Referências à bibliografia de manutenção preditiva**
* **PDF de white papers de Digital Twins**
* **Diagrama de arquitetura (em PlantUML ou draw.io)**
* **Scripts de deploy automatizado (Helm + K8s)**

**Análise do problema:**

**Estamos diante de um dos conjuntos de dados mais emblemáticos em manutenção preditiva baseada em aprendizado de máquina: o CMAPSS (Commercial Modular Aero-Propulsion System Simulation). Vamos destrinchar cuidadosamente.**

**🎯 Objetivo do Problema**

**Você deve prever o número de ciclos restantes (RUL — Remaining Useful Life) de uma turbofan engine com base em leituras de sensores ao longo do tempo.**

**📁 Sobre os Arquivos**

**Você tem:**

* **train\_FDxxx.txt: Dados de treino. Cada linha representa um ciclo de uma turbina específica até sua falha total.**
* **test\_FDxxx.txt: Dados de teste. Cada turbina foi monitorada até certo ponto antes da falha.**
* **RUL\_FDxxx.txt: Solução do teste. Mostra quantos ciclos restavam antes da falha no último ciclo registrado em test\_FDxxx.txt.**

**Tela de celular com fundo preto

O conteúdo gerado por IA pode estar incorreto.**

**Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.**

**🧠 Natureza do Problema**

**É um problema de regressão supervisionada temporal, altamente dependente de:**

* **Tendências ao longo do tempo.**
* **Condições de operação (colunas 3, 4, 5).**
* **Medidas dos sensores (colunas 6 a 26).**

**Você pode formular como:**

**python**

**CopiarEditar**

**Input: sequência de ciclos do motor i, incluindo sensores e condições operacionais**

**Output: RUL estimado no último ciclo**

**📊 Os sensores importam? Devo saber o que eles medem?**

**Resposta: Depende do seu objetivo**

* **Se você for usar modelos de Deep Learning ou regressão automática, você não precisa saber o que cada sensor mede — o modelo aprende padrões diretamente.**
* **Mas se quiser explicabilidade, engenharia de features ou modelos híbridos com física (Digital Twin físico+estatístico), entender os sensores é essencial.**

**Por exemplo:**

* **Sensor 2 → fan speed**
* **Sensor 11 → HPC outlet pressure**
* **Sensor 15 → HPC temperature**

**(*Veja a referência do paper para detalhes dos sensores*)**

**🔧 Modelagem possível**

**1. Regressão direta**

* **Entradas: vetor dos sensores no último ciclo (ou média dos últimos N ciclos)**
* **Saída: RUL**
* **Modelos: XGBoost, Random Forest, Regressão Linear**

**2. Modelos sequenciais**

* **Entradas: sequência temporal dos últimos N ciclos de sensores**
* **Modelos: LSTM, GRU, Transformer, Temporal CNN**

**3. Digital Twin híbrido (visionário e robusto)**

* **Modelagem física baseada em conhecimento de motores (termodinâmica, pressão, RPM)**
* **Combinado com modelos estatísticos para estimativa de RUL residual**

**📌 Sugestão de fluxo de trabalho**

1. **Escolha o dataset FD001 para começar.**
2. **Pré-processamento:**
   * **Normalize os dados por motor**
   * **Calcule RUL para o train**
   * **Use uma janela deslizante (sliding window) para treinar em séries temporais**
3. **Modelagem:**
   * **Comece com regressão simples (e.g., XGBoost)**
   * **Evolua para modelos temporais (LSTM, Transformer)**
4. **Validação:**
   * **RMSE, MAE**
   * **Scoring por limites: erro pequeno quando faltam poucos ciclos pode ser crítico**
5. **Explicabilidade e Engenharia:**
   * **Avalie quais sensores mais impactam a previsão**
   * **Crie features derivadas (e.g., variação de pressão por ciclo)**

**📚 Referência técnica**

**Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). *Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation*. Proceedings of PHM08.**

**Eles descrevem as falhas físicas e a construção do simulador, que é essencial para um verdadeiro Digital Twin.**

**Diferença conceitual: train vs test**

**🔧 train\_FDxxx.txt (Treinamento)**

* **Cada linha representa um ciclo de operação de um motor até sua falha completa.**
* **Cada motor (ou "engine") tem seu ciclo de vida completo registrado, desde o início até a falha crítica (run-to-failure).**
* **Você pode calcular o RUL verdadeiro diretamente:**

**python**

**CopiarEditar**

**RUL = (ciclo final do motor) - (ciclo atual)**

* **Serve para o modelo aprender padrões de degradação.**

**✅ O modelo vê a evolução total da falha e aprende a prever a "vida restante" com base nas leituras dos sensores.**

**🔍 test\_FDxxx.txt (Teste)**

* **Cada motor também tem linhas correspondentes aos seus ciclos operacionais, como no treino.**
* **MAS: esses motores ainda não falharam no fim do arquivo.**
* **O arquivo termina em um ponto arbitrário antes da falha real.**

**✅ O objetivo é que seu modelo estime quantos ciclos ainda faltam para a falha a partir desse último ciclo disponível.**

**Mas como validar?**

**📄 RUL\_FDxxx.txt: o "gabarito" do teste**

* **Esse arquivo diz quantos ciclos faltavam até a falha real a partir do último ciclo disponível de cada motor no test.**

**Exemplo:  
Se o motor 001 no test tem 135 ciclos registrados, e a entrada correspondente em RUL\_FDxxx.txt for 53, isso quer dizer:**

**O motor falharia no ciclo 135 + 53 = 188.  
Você não vê esses 53 ciclos, só deve prever que eles existem.**

**📉 Comparação ilustrativa**

| **Conjunto** | **Motor** | **Início** | **Fim visível** | **Fim real** | **RUL disponível?** |
| --- | --- | --- | --- | --- | --- |
| **train** | **001** | **1** | **250** | **250** | **✅ (calculado)** |
| **test** | **001** | **1** | **135** | **188** | **❌ (usa RUL\_FDxxx)** |

**🧠 Analogia simples**

**Imagine que você monitora 100 pacientes cardíacos:**

* **No train, você monitora até a parada cardíaca e anota tudo: pode estudar a curva de deterioração completa.**
* **No test, o monitoramento foi interrompido enquanto o paciente ainda vivia, e você precisa prever quanto tempo resta com base nas últimas medições.**

**Para avançar na modelagem de dados:**

**Modelos que exploram essa dependência temporal (ex: LSTM, GRU, Transformer, Temporal Convolutional Networks) costumam entregar resultados mais precisos.**