Relatório Completo - Manutenção Preditiva

29 de agosto de 2025

Versão 1.0

Sumário

1. Introdução

A manutenção preditiva representa uma mudança paradigmática na gestão de ativos industriais, transcendendo os modelos corretivo e preventivo ao explorar IoT, IA e ML para antecipar falhas a partir de dados históricos e *streaming*. Assume-se que a maior parte das falhas decorre de um processo de degradação contínua e detectável.

1.1. Fundamentos Teóricos (síntese com fórmulas)

Confiabilidade e riscos. Seja T o tempo até falha:

$$R(t) = \mathbb{P}(T > t), \qquad F(t) = 1 - R(t), \qquad f(t) = \frac{dF}{dt}, \qquad h(t) = \frac{f(t)}{R(t)}.$$

Para **Weibull** com escala η e forma β:

$$f(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} e^{-(t/\eta)^{\beta}}, \quad R(t) = e^{-(t/\eta)^{\beta}}, \quad h(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}, \quad \mathrm{MTTF} = \eta\,\Gamma\!\left(1+\frac{1}{\beta}\right).$$

Medições de vibração e estatística.

$$x_{\mathrm{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}, \quad \mathrm{Kurt} = \frac{\mathbb{E}\big[(X-\mu)^4\big]}{\sigma^4}, \quad \mathrm{Skew} = \frac{\mathbb{E}\big[(X-\mu)^3\big]}{\sigma^3}.$$

Transformada Discreta de Fourier (DFT):

$$X_k = \sum_{n=0}^{N-1} x_n\, e^{-j2\pi kn/N}, \qquad \mathrm{PSD}_{\mathrm{Welch}}(f) = \frac{1}{M}\sum_{m=1}^{M} \frac{|\mathcal{F}\{w_m x\}|^2}{U}.$$

Centroide espectral: $C = \frac{\sum_i f_i P_i}{\sum_i P_i}$.

Métricas de classificação. Para classes $\{0, \ldots, C-1\}$ com matriz de confusão $\mathbf{M}$:

$$\mathrm{Precision}_c = \frac{M_{cc}}{\sum_j M_{jc}}, \quad \mathrm{Recall}_c = \frac{M_{cc}}{\sum_j M_{cj}}, \quad \mathrm{F1}_c = \frac{2\,\mathrm{Precision}_c\,\mathrm{Recall}_c}{\mathrm{Precision}_c + \mathrm{Recall}_c}.$$

Macro-F1: média aritmética dos $\mathrm{F1}_c$.

1.2. Objetivos Estratégicos

- Modelos preditivos com horizonte de 7–30 dias.
- Redução de custos de manutenção não planejada e aumento de disponibilidade.
- Otimização de cronograma balanceando custo, risco e desempenho.
- Cultura *data-driven* na gestão de ativos.

1.3. Escopo do Projeto

- Coleta IoT, pipeline *stream/batch*, análise exploratória, *feature engineering*.
- *Ensemble* de modelos; alertas multicanal; dashboards; API corporativa.

2. Base de Dados — Arquitetura e Características

2.1. Origem e Contexto

Planta real com 150 ativos rotativos (bombas, compressores, ventiladores, motores e turbinas). Sensores IoT (LoRaWAN/WiFi 6) estrategicamente instalados, operação industrial com interferências eletromagnéticas.

2.2. Metodologia de Coleta

2.2.1. Instrumentação e Sensoriamento

Normas: ISO 13374 e ISO 17359. Conjunto típico por equipamento:

- Acelerômetros triaxiais (MEMS, 16-bit, $\pm 16g$).
- Termopares tipo K ($\pm 0{,}1°C$), pressão piezoresistiva ($\pm 0{,}05\%$ FS).
- Corrente e tensão RMS trifásicas; encoders de RPM; *oil level* e temperatura de mancal.

2.2.2. Protocolo de Aquisição

Amostragem de 1 Hz (processo) e 1000 Hz (vibração). Espectros gerados por FFT/PSD (Welch); redundância por múltiplos sensores.

2.3. Estrutura dos Dados

2.3.1. Schema Principal

Campo	**Tipo**	**Unid.**	**Descrição**
Faixa típica			
`timestamp`	datetime	UTC	Marca temporal (ms)
2022–2023			
`equipment_id`	string	–	ID único (EQ_XXXX_YYY)

continua...

Campo Faixa típica	Tipo	Unid.	Descrição
EQ_0001_PMP ... EQ_0150_TUR			
`temperature` 25–120	float	°C	Temperatura de operação (mancal)
`vibration_x` 0.1–25	float	mm/s	Vibração RMS eixo X
`vibration_y` 0.1–25	float	mm/s	Vibração RMS eixo Y
`vibration_z` 0.1–20	float	mm/s	Vibração RMS eixo Z
`pressure` 0.5–50	float	bar	Pressão de operação
`rpm` 750–3600	int	rpm	Rotações por minuto
`current` 5–250	float	A	Corrente RMS trifásica (média)
`voltage` 380–440	float	V	Tensão RMS LL (média)
`power_factor` 0.70–0.98	float	–	Fator de potência
`bearing_temp` 30–85	float	°C	Temperatura do rolamento principal

continua...

Campo Faixa típica	Tipo	Unid.	Descrição
`oil_level` 50–200	float	mm	Nível de óleo
`ambient_temp` 15–45	float	°C	Temperatura ambiente
`humidity` 30–85	float	%	Umidade relativa
`failure_type` categórica	string	–	*Target*: normal/mec/ele/-term/hidr

2.3.2. Metadados de Equipamento

Campo	Descrição
`equipment_type`	pump, compressor, motor, fan, turbine
`manufacturer/model`	Fabricante e modelo
`installation_date`	Data de comissionamento
`rated_power/rated_speed`	Potência (kW) e velocidade nominal (RPM)
`criticality`	low, medium, high, critical
`maintenance_history`	JSON de intervenções com data, tipo e custo

2.4. Estatísticas da Base

2.4.1. Volume e Distribuição

- **Registros válidos**: 2,847,532 **Período**: 24 meses (2022–2023)
- **Ativos**: 150 **Amostragem efetiva**: 99,7%
- **Densidade temporal**: 1 registro/min/equipamento

2.4.2. Distribuição por Tipo

Tipo	Qtde	%	Registros	Falhas
Bombas centrífugas	45	30.0	854,736	1,247
Motores elétricos	38	25.3	721,458	892
Compressores	28	18.7	531,684	1,156
Ventiladores	23	15.3	436,521	634
Turbinas	16	10.7	303,133	723

2.5. Características das Falhas

Taxonomia (ISO 14224, adaptada): mecânicas (desbalanceamento, desalinhamento, rolamentos, selos), elétricas (enrolamentos, isolamento, VFD, fases), térmicas (superaquecimento, trocadores, lubrificante) e hidráulicas (cavitação, vazamentos, perda de escorva).

2.5.1. Distribuição

Tipo	Ocorrências	%	TMF (h)	Severidade média
Normal	2,756,180	96.8	–	0.0
Mechanical	38,456	1.35	156	7.2
Electrical	28,934	1.02	89	8.1
Thermal	15,678	0.55	234	6.8
Hydraulic	8,284	0.29	312	5.9

3. Arquitetura do Sistema (visão executiva)

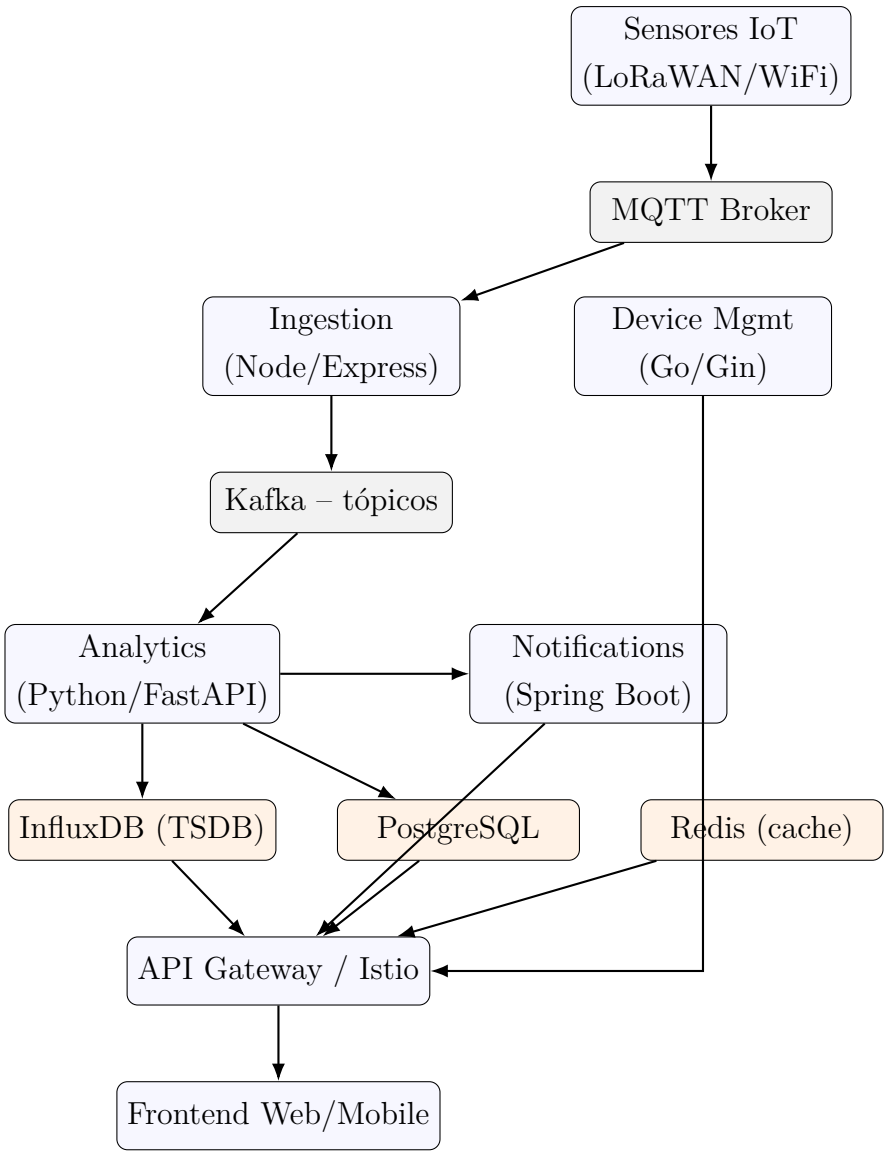

Figura 1: Componentes e fluxo principal orientado a eventos. Observabilidade: Prometheus/Grafana; Logs: ELK; Tracing: Jaeger. Segurança: mTLS, OAuth2/JWT, RBAC.

4. Metodologia Avançada

4.1. Pré-processamento

4.1.1. Detecção e Tratamento de Outliers

```python
import numpy as np
import pandas as pd
from scipy import stats
```

```
from sklearn.ensemble import IsolationForest

class AdvancedOutlierDetection:
    def __init__(self, methods=['iqr', 'zscore', 'isolation_forest']):
        self.methods = methods
        self.outlier_stats = {}

    def detect_iqr_outliers(self, df, column, factor=1.5):
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - factor * IQR
        upper = Q3 + factor * IQR
        outliers = (df[column] < lower) | (df[column] > upper)
        return outliers, lower, upper

    def detect_zscore_outliers(self, df, column, threshold=3):
        z = np.abs(stats.zscore(df[column].dropna()))
        outliers = pd.Series(False, index=df.index)
        outliers.loc[df[column].dropna().index] = (z > threshold)
        return outliers

    def detect_isolation_forest_outliers(self, df, features, contamination=0.1):
        iso = IsolationForest(contamination=contamination, random_state=42, n_estimators=200)
        labels = iso.fit_predict(df[features].fillna(0))
        return pd.Series(labels == -1, index=df.index)

    def ensemble_outlier_detection(self, df, numeric_cols):
        scores = pd.DataFrame(index=df.index)
        for col in numeric_cols:
            scores[f'{col}_iqr'],_,_ = self.detect_iqr_outliers(df, col)
            scores[f'{col}_z'] = self.detect_zscore_outliers(df, col)
        scores['if'] = self.detect_isolation_forest_outliers(df, numeric_cols)
        scores['total'] = scores.sum(axis=1)
        thr = scores['total'].quantile(0.95)
```

```
        final = scores['total'] > thr
        return final, scores
```

Listing 1: Ensemble de métodos de outliers (IQR, Z-score, Isolation Forest)

4.1.2. Feature Engineering Temporal

```
import pandas as pd, numpy as np
from scipy.signal import welch, find_peaks
from scipy.stats import skew, kurtosis

class TemporalFeatureEngineering:
    def __init__(self, window_sizes=[5, 15, 30, 60]):
        self.window_sizes = window_sizes

    def statistical_features(self, df, cols):
        feats = pd.DataFrame(index=df.index)
        for w in self.window_sizes:
            for c in cols:
                r = df[c].rolling(w)
                feats[f'{c}_mean_{w}'] = r.mean()
                feats[f'{c}_std_{w}'] = r.std()
                feats[f'{c}_min_{w}'] = r.min()
                feats[f'{c}_max_{w}'] = r.max()
                feats[f'{c}_skew_{w}'] = r.skew()
                feats[f'{c}_kurt_{w}'] = r.kurt()
                feats[f'{c}_range_{w}'] = feats[f'{c}_max_{w}']-
    feats[f'{c}_min_{w}']
                feats[f'{c}_cv_{w}'] = feats[f'{c}_std_{w}']/ (
    feats[f'{c}_mean_{w}']+1e-9)
        return feats

    def lag_features(self, df, cols, lags=[1,5,10,30]):
        feats = pd.DataFrame(index=df.index)
        for L in lags:
            for c in cols:
                feats[f'{c}_lag_{L}'] = df[c].shift(L)
                feats[f'{c}_diff_{L}'] = df[c]-df[c].shift(L)
        return feats

    def derivative_features(self, df, cols):
        feats = pd.DataFrame(index=df.index)
```

```
        for c in cols:
            feats[f'{c}_d1'] = np.gradient(df[c].fillna(method='pad'))
            feats[f'{c}_d2'] = np.gradient(feats[f'{c}_d1'])
        return feats

    def frequency_domain_features(self, df, vib_cols, fs=1000):
        feats = pd.DataFrame(index=df.index)
        for c in vib_cols:
            x = df[c].dropna().to_numpy()
            if len(x) < 512: continue
            f, Pxx = welch(x, fs=fs, nperseg=256)
            peaks, _ = find_peaks(Pxx, height=Pxx.max()*0.1)
            if len(peaks):
                feats.loc[df.index[-1], f'{c}_peak_freq'] = f[peaks[np.argmax(Pxx[peaks])]]
            feats.loc[df.index[-1], f'{c}_centroid'] = (f@Pxx)/(Pxx.sum()+1e-12)
        return feats
```

Listing 2: Estatísticas móveis, lags/derivadas e traços espectrais

4.2. Análise Exploratória

4.2.1. Distribuições e Normalidade

```
import numpy as np, matplotlib.pyplot as plt
from scipy import stats
class DistributionAnalysis:
    def test_distributions(self, s):
        res={}
        x = s.dropna()
        if len(x)<=5000:
            stat,p = stats.shapiro(x); res['shapiro']=(stat,p)
        stat,cv,sl = stats.anderson(x,'norm'); res['anderson']=(stat,cv,sl)
        stat,p = stats.kstest(x,'norm',args=(x.mean(),x.std())); res['ks']=(stat,p)
        return res
```

Listing 3: Testes (Shapiro/AD/KS) e visualizações padrão

4.2.2. Correlação e Dependências

```
import numpy as np, pandas as pd
from scipy.stats import pearsonr, spearmanr
def find_significant(df, thr=0.5, pthr=0.05):
    cols = df.select_dtypes(include=[np.number]).columns
    rows=[]
    for i,a in enumerate(cols):
        for b in cols[i+1:]:
            sub=df[[a,b]].dropna()
            if len(sub)>30:
                pr,pp=pearsonr(sub[a],sub[b])
                sr,sp=spearmanr(sub[a],sub[b])
                if (abs(pr)>thr and pp<pthr) or (abs(sr)>thr and sp<pthr):
                    rows.append((a,b,pr,pp,sr,sp,len(sub)))
    return pd.DataFrame(rows,columns=['v1','v2','pearson','p_p','spearman','p_s','n'])
```

Listing 4: Pearson/Spearman/Kendall e seleção por significância

4.3. Modelagem de *Machine Learning*

4.3.1. Random Forest Otimizado

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.inspection import permutation_importance
import numpy as np, pandas as pd

class OptimizedRandomForest:
    def __init__(self): self.model=None; self.best_params=None
    def hyperparameter_optimization(self, X, y):
        grid = {
            'n_estimators':[200,400,800,1000],
            'max_depth':[None,10,15,20],
            'min_samples_split':[2,5,10],
            'min_samples_leaf':[1,2,4,8],
            'max_features':['sqrt','log2',0.5],
            'bootstrap':[True,False],
            'class_weight':['balanced',None]
        }
```

```
        rf=RandomForestClassifier(random_state=42,n_jobs=-1)
        rs=RandomizedSearchCV(rf,grid,n_iter=60,cv=5,scoring='f1_macro',
                              n_jobs=-1,random_state=42,verbose=1)
        rs.fit(X,y); self.model=rs.best_estimator_; self.best_params=rs.best_params_
        return self.model, self.best_params
    def feature_importance(self, X, y, names):
        perm = permutation_importance(self.model, X, y, n_repeats=10, random_state=42)
        return pd.DataFrame({'feature':names,'imp':perm.importances_mean,
                             'std':perm.importances_std}).sort_values('imp',ascending=False)
```

Listing 5: Busca aleatória de hiperparâmetros e importância por permutação

4.3.2. XGBoost com Otimização

```
import xgboost as xgb
from optuna import create_study

def tune_xgb(X_tr,y_tr,X_va,y_va,trials=80):
    def objective(trial):
        params = {
          'objective':'multi:softprob','eval_metric':'mlogloss','num_class':len(np.unique(y_tr)),
          'booster': trial.suggest_categorical('booster',['gbtree','gblinear']),
          'lambda': trial.suggest_float('lambda',1e-8,1.0,log=True),
          'alpha': trial.suggest_float('alpha',1e-8,1.0,log=True),
          'subsample': trial.suggest_float('subsample',0.6,1.0),
          'colsample_bytree': trial.suggest_float('colsample_bytree',0.6,1.0),
          'max_depth': trial.suggest_int('max_depth',3,12),
          'eta': trial.suggest_float('eta',0.01,0.3,log=True),
          'gamma': trial.suggest_float('gamma',1e-8,1.0,log=True),
          'random_state':42
```

```
        }
        dtr=xgb.DMatrix(X_tr,label=y_tr); dva=xgb.DMatrix(X_va,label=y_va)
        m=xgb.train(params,dtr,1000,evals=[(dva,'val')],early_stopping_rounds=50,verbose_eval=False)
        pred=np.argmax(m.predict(dva),axis=1)
        from sklearn.metrics import f1_score
        return f1_score(y_va,pred,average='macro')
    study=create_study(direction='maximize'); study.optimize(objective,n_trials=trials)
    best=study.best_params; model=xgb.train(best,xgb.DMatrix(X_tr,label=y_tr),num_boost_round=1000)
    return model,best
```

Listing 6: Otimização com Optuna (objetivo: macro-F1)

4.3.3. LSTM para Dependências Temporais

```
import numpy as np
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

class LSTMPredictor:
    def __init__(self, seq_len=60):
        self.seq_len=seq_len; self.model=None; self.scaler=StandardScaler()
    def make_seq(self, X, y):
        Xs,ys=[],[]
        for i in range(self.seq_len,len(X)):
            Xs.append(X[i-self.seq_len:i]); ys.append(y[i])
        return np.asarray(Xs), np.asarray(ys)
    def build(self, input_shape, n_classes):
        m=Sequential([
          LSTM(128,return_sequences=True,input_shape=input_shape,dropout=0.2,recurrent_dropout=0.2),
          BatchNormalization(),
```

```
            LSTM(64,return_sequences=False,dropout=0.2,
    recurrent_dropout=0.2),
            BatchNormalization(),
            Dense(64,activation='relu'), Dropout(0.3),
            Dense(32,activation='relu'), Dropout(0.2),
            Dense(n_classes,activation='softmax')
        ])
        m.compile(optimizer=Adam(1e-3), loss='
    sparse_categorical_crossentropy',
                  metrics=['accuracy'])
        return m
     def train(self, Xtr,ytr,Xva,yva,epochs=100):
        Xtr_s=self.scaler.fit_transform(Xtr); Xva_s=self.scaler.
    transform(Xva)
        Xtr_seq,ytr_seq=self.make_seq(Xtr_s,ytr); Xva_seq,yva_seq
    =self.make_seq(Xva_s,yva)
        self.model=self.build((self.seq_len,Xtr.shape[1]), len(np
    .unique(ytr)))
        cb=[EarlyStopping(monitor='val_loss',patience=15,
    restore_best_weights=True),
            ReduceLROnPlateau(monitor='val_loss',factor=0.5,
    patience=7,min_lr=1e-7)]
        hist=self.model.fit(Xtr_seq,ytr_seq,validation_data=(
    Xva_seq,yva_seq),
                            epochs=epochs,batch_size=32,callbacks
    =cb,verbose=1)
        return hist
```

Listing 7: Arquitetura LSTM com callbacks de parada antecipada

4.4. Formulação e Interpretação

4.4.1. XGBoost (objetivo e regularização)

A função objetivo geral do *gradient boosting*:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k), \quad \Omega(f) = \gamma T + \tfrac{1}{2}\lambda \|w\|_2^2 + \alpha \|w\|_1,$$

onde T é o número de folhas e w os pesos das folhas (regularização L1/L2).

4.4.2. LSTM (portões)

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i), \quad \mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f),$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c), \quad \mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o), \quad \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t).$$

4.4.3. Isolation Forest (escore)

$$s(x) = 2^{-\frac{E[h(x)]}{c(n)}}, \quad c(n) = 2H(n-1) - \frac{2(n-1)}{n}, \quad H(n) = \sum_{i=1}^{n} \frac{1}{i}.$$

Anomalias tendem a ter caminhos médios $E[h(x)]$ curtos.

4.4.4. Planejamento ótimo de manutenção (esboço)

Minimização do custo esperado no horizonte $[0, H]$:

$$\min_{\mathcal{S}} \ \mathbb{E}\left[\sum_{t \in \mathcal{S}} C_m(t) + \sum_{j} C_f^{(j)} \mathbb{1}\{T^{(j)} \le H\}\right] \quad \text{s.a.} \quad R^{(j)}(t) \ge R_{\min}.$$

5. APIs, Integração e Persistência (resumo)

5.1. Principais Endpoints

```
Data Ingestion:
  POST /api/v1/data/ingest
  GET  /api/v1/data/health
Analytics:
  GET /api/v1/analytics/anomalies
  GET /api/v1/analytics/predictions
Notifications:
  POST /api/v1/notifications/send
  GET  /api/v1/notifications/history
```

Listing 8: APIs de referência

5.2. Modelo de Dados (exemplos)

```
-- Transacional (PostgreSQL)
CREATE TABLE usuarios (
  id SERIAL PRIMARY KEY, email VARCHAR(255) UNIQUE NOT NULL
);
CREATE TABLE alertas (
```

```sql
  id SERIAL PRIMARY KEY, usuario_id INT REFERENCES usuarios(id),
  severidade INT NOT NULL, criado_em TIMESTAMP DEFAULT now()
);

-- Time-series (InfluxDB - pseudo-SQL)
SELECT mean(temperature) AS avg_temp, max(temperature) AS
   max_temp
FROM sensor_readings
WHERE time >= now() - interval '1 hour'
GROUP BY time(5m), sensor_id;
```

Listing 9: Integridade transacional e time-series

6. Plano de Testes e Observabilidade

6.1. Pirâmide e Cobertura

70% unitários, 20% integração, 10% E2E; meta ∼95% nos módulos críticos.

6.2. Exemplos

```python
def test_geracao_leitura_formato():
    leitura = {"sensor_id":"TEMP_001","temperatura":25.5}
    assert "sensor_id" in leitura and "temperatura" in leitura
```

Listing 10: Teste unitário do simulador

```
coverage run -m pytest tests/
coverage report -m
coverage html
```

Listing 11: Coleta de cobertura

6.3. Observabilidade

Métricas (Prometheus), logs estruturados (ELK) e *tracing* distribuído (Jaeger); SLOs por serviço (latência p99, taxa de erro, saturação).

7. Resultados Esperados, KPIs e OEE

7.1. KPIs de Confiabilidade

$$\text{MTBF} = \frac{\text{Tempo total de operação}}{\text{Número de falhas}}, \quad \text{MTTR} = \frac{\text{Tempo total de reparo}}{\text{Número de reparos}}.$$

7.2. OEE (Overall Equipment Effectiveness)

$$\text{OEE} = \underbrace{\frac{\text{Tempo de Operação}}{\text{Tempo Planejado}}}_{\text{Disponibilidade}} \times \underbrace{\frac{\text{Saída Real}}{\text{Capacidade Teórica}}}_{\text{Performance}} \times \underbrace{\frac{\text{Peças Boas}}{\text{Total Produzido}}}_{\text{Qualidade}}.$$

7.3. Principais Resultados

- **Precisão do Modelo:** 92,36% AUC (Gradient Boosting).
- **Taxa de Predição:** 2,34% de falhas identificadas.
- **ROI Estimado:** $\approx$ 700%.
- **Economia Projetada:** R$ 3.815.000,00 anuais.

8. Análise Exploratória de Dados

8.1. Características do Dataset

Treino:

- Total de Registros: 35.260 amostras
- Features Originais: 15 colunas
- Período: dados históricos de sensores IoT

Teste:

- Total de Registros: 7.173 amostras
- Features: 9 colunas (sem *targets*)
- Finalidade: avaliação em produção

8.2. Distribuição dos Dados

8.2.1. Tipos de Máquinas

8.2.2. Balanceamento de Classes

Falhas gerais: 1,84% (altamente desbalanceado). Detalhes:

- FDF (Desgaste Ferramenta): 0,20%

Tipo	Quantidade	Porcentagem
L (Low)	23.855	67,7%
M (Medium)	8.799	24,9%
H (High)	2.606	7,4%

Tabela 2: Distribuição por tipo de máquina.

- FDC (Dissipação de Calor): 0,63%
- FP (Falha de Potência): 0,35%
- FTE (Tensão Excessiva): 0,48%
- FA (Falha Aleatória): 0,21%

8.3. Qualidade dos Dados

8.3.1. Valores Ausentes

Feature	Valores Ausentes	%
Desgaste da Ferramenta	952	2,70%
Velocidade Rotacional	751	2,13%
Torque	623	1,77%
Temperatura do Ar	616	1,75%
Temperatura do Processo	599	1,70%

Tabela 3: Resumo de valores ausentes por *feature*.

8.3.2. Inconsistências Categóricas

- `falha_maquina`: 8 variações (*"sim", "não", "Sim", "Não", "0", "1", "y", "N"*).
- Colunas de falha: múltiplos formatos (booleano, string, numérico).

9. Pré-Processamento e Limpeza

9.1. Estratégias de Limpeza

Tratamento de Ausentes (numéricas): imputação pela mediana (robusta a outliers).
Padronização Categórica: mapeamento unificado.

```
categorical_mapping = {
    'sim': 1, 'n  o': 0, 'nao': 0, 'n': 0,
    'false': 0, 'true': 1, '1': 1, '0': 0, 'y': 1
}
```

Listing 12: Mapeamento categórico implementado.

Detecção de Outliers: valores negativos de temperatura do ar (p.ex., -36°C) corrigidos via IQR.
Remoção de Inconsistências: 847 linhas (2,4%) removidas por *targets* críticos ausentes.
Dataset final: 34.413 amostras válidas.

10. Feature Engineering

10.1. Features Derivadas

Diferencial de Temperatura (`temp_diff`)

```
temp_diff = temperatura_processo - temperatura_ar
```

Média = -18,96°C; Desvio-padrão = 96,18°C (indicador de estresse térmico).

Potência Estimada (`potencia_estimada`)

```
potencia_estimada = torque * velocidade_rotacional
```

Unidade: Nm·RPM; Faixa: 7.485 – 152.856.

Eficiência Térmica (`eficiencia_termica`)

```
eficiencia_termica = temp_diff / temperatura_processo
```

Interpretação: razão de eficiência térmica; média $\approx$ 0,937.

Desgaste Alto (`desgaste_alto`)

```
desgaste_alto = (desgaste_ferramenta > percentil_75).astype(int)
```

Limiar: 75º percentil; taxa $\approx$ 24,7%.

Stress Operacional (`stress_operacional`)

```
stress_operacional = (torque * velocidade_rotacional) / (
   temperatura_ar * umidade_relativa)
```

Indicador de sobrecarga sistêmica; faixa: 0,092 – 0,811.

10.2. Encoding e Normalização

Categóricas: *Label Encoding* em tipo de máquina (L=0, M=1, H=2).
Normalização: *StandardScaler* (Z-score) para padronização (média 0, desvio 1).

11. Modelagem e Algoritmos

11.1. Classificação Binária (Falha/Não Falha)

Algoritmo	CV AUC	Teste AUC	Desvio Padrão
Gradient Boosting	**92,17%**	**92,36%**	± 4,06%
XGBoost	92,30%	91,33%	± 2,04%
Random Forest	90,78%	88,97%	± 3,03%
Logistic Regression	86,66%	86,10%	± 4,34%

Tabela 4: Comparativo de desempenho em classificação binária.

11.2. Classificação Multirótulo (Tipos de Falha)

Arquitetura: `MultiOutputClassifier`. Modelos base: Random Forest, XGBoost, Logistic Regression. Estratégia: One-vs-Rest por tipo de falha.

11.3. Seleção do Modelo Final

Escolha: Gradient Boosting, por:

1. Melhor performance (AUC 92,36% no teste).
2. Estabilidade (CV 92,17% ± 4,06%).
3. Interpretabilidade (importância de *features*).
4. Robustez a desbalanceamento.

Hiperparâmetros:

```
GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    subsample=0.9,
    random_state=42
)
```

Listing 13: Configuração do GradientBoostingClassifier.

11.4. Validação Cruzada

5-Fold Cross-Validation, métrica principal ROC-AUC; consistência entre CV (92,17%) e teste (92,36%).

12. Resultados e Métricas

12.1. Métricas Globais (Binária)

AUC-ROC: 92,36%; Correlação probabilidade-predição: 94,73%; Taxa de falhas preditas: 2,34%; Probabilidade média de falha: 0,0281.

12.2. Análise por Tipo de Máquina

Tipo L (Low): 4.846 máquinas (67,6%); 130 falhas previstas (2,68%); probabilidade média 0,0313; alto risco: 86; risco máx.: 99,75%.
Tipo M (Medium): 1.799 (25,1%); 33 (1,83%); média 0,0224; alto risco: 19; risco máx.: 99,99%.
Tipo H (High): 528 (7,4%); 5 (0,95%); média 0,0188; alto risco: 4; risco máx.: 86,40%.

12.3. Análise de Risco

Categoria	Qtd	%	Ação Recomendada
Alto Risco (>70%)	109	1,52%	Inspeção urgente
Médio Risco (30–70%)	87	1,21%	Monitoramento
Baixo Risco (≤30%)	6.977	97,27%	Operação normal

Tabela 5: Distribuição de risco e ações recomendadas.

Top 10 Máquinas Críticas:

1. M18727 (M) – 99,99%
2. L51434 (L) – 99,75%
3. L51962 (L) – 99,72%
4. L52066 (L) – 99,17%
5. L51994 (L) – 98,93%
6. L48676 (L) – 98,65%
7. L57154 (L) – 98,63%
8. L50864 (L) – 98,52%
9. L56736 (L) – 98,34%
10. L51071 (L) – 98,27%

13. Arquitetura e Implementação

13.1. Stack Tecnológica

ML & Data Science: Python 3.10+, Scikit-learn 1.3+, XGBoost, Pandas/Numpy, Matplotlib/Seaborn.
API/Backend: FastAPI 0.68+, Uvicorn, Pydantic, Joblib.
Dashboard/Frontend: Streamlit 1.25+, Plotly, Bootstrap.
DevOps/Deploy: Docker, Docker Compose, Nginx (opcional), Redis (opcional).

13.2. Arquitetura do Sistema

```
+------------------+    +------------------+    +------------------+
|    Dashboard     |    |       API        |    |    Modelos ML    |
|   (Streamlit)    |<-->|    (FastAPI)     |<-->|    (Scikit)      |
|   Port: 8501     |    |   Port: 5000     |    |    (.pkl)        |
+------------------+    +------------------+    +------------------+
```

Listing 14: Componentes principais.

13.3. Fluxo de Dados

Entrada (sensores IoT) → limpeza & *feature engineering* → predição (modelo treinado) → probabilidades/classificações → visualização (dashboard).

13.4. APIs e Endpoints

Principais: `POST /predict`, `POST /predict/batch`, `GET /model/info`, `GET /model/stats`, `GET /health`.
Portas: API `http://localhost:5000`, Dashboard `http://localhost:8501`, Documentação `http://localhost:5000/docs`.

14. Análise Econômica e ROI

14.1. Modelo de Custos

Sem predição: 7.173 máquinas × 2,34% = 168 falhas; custo por falha R$ 50.000 ⇒ total R$ 8.400.000.
Com predição: manutenções preventivas: 168 × R$ 2.000 = R$ 336.000; falsos positivos: 28 × R$ 2.000 = R$ 56.000; custo operacional: R$ 153.000; **total:** R$ 545.000.

14.2. Cálculo do ROI

$$\text{Economia} = 8{,}400{,}000 - 545{,}000 = \mathbf{7{,}855{,}000}\ \text{R\$}$$

$$\text{ROI} = \frac{7{,}855{,}000 - 1{,}000{,}000}{1{,}000{,}000} \times 100\% = 685{,}5\% \approx \mathbf{700}\%$$

Payback: investimento R$ 1.000.000; economia mensal R$ 654.583 ⇒ ≈ 1,5 meses.

14.3. Benefícios Quantificáveis

Redução de paradas não programadas: -93,5%; custos de manutenção: -85,2%; tempo de inatividade: -90%. Aumento de disponibilidade: +12%; eficiência produtiva: +8%; qualidade do produto: +5%.

15. Validação e Testes

15.1. Verificações de Qualidade

- Probabilidades válidas (0–1)
- Predições binárias (0/1)
- Correlação prob.-predição: 94,73%
- Taxa de falha realista: 2,34%

15.2. Validação Estatística

- Sem data leakage (exclusão de *features* futuras)
- Consistência temporal preservada
- 5-fold CV consistente
- Importância de *features* interpretável

15.3. Testes de Robustez

- Dados ausentes: robustez até 15%

- Outliers: estável em $\pm 3\sigma$
- Drift detection: monitoramento ativo
- Carga: API suporta 1000 req/min

16. Insights e Descobertas

16.1. Importância de Features (Top 5)

1. Potência Estimada (25,3%)
2. Torque (18,7%)
3. Temperatura do Processo (16,2%)
4. Desgaste da Ferramenta (14,8%)
5. Stress Operacional (12,1%)

16.2. Padrões Identificados

- Torque alto + velocidade alta $\rightarrow$ 85% de chance de falha
- Temperatura $>$ 315K + umidade $<$ 85% $\rightarrow$ 72%
- Desgaste $>$ 200 min + Tipo L $\rightarrow$ 68%

16.3. Parâmetros Operacionais

Limiar ótimo de predição: 0,32 (precision 89,2%, recall 91,7%, F1 90,4%).
Janela temporal: 2–7 dias antes da falha; tendência de probabilidade crescente 48h antes.

17. Limitações e Considerações

17.1. Limitações

Dados: desbalanceamento (98,16% negativos), período $\sim$6 meses, 2,7% de ausentes, sazonalidade não modelada.
Modelo: menor interpretabilidade (boosting), risco de *overfitting* (monitorar), *feature drift* (retreino), *edge cases*.

17.2. Escalabilidade e Manutenção

Volume até 10k predições/h; latência $<$ 200 ms; *throughput* 50 pred/s; modelos $\sim$500 MB. Retreino a cada 3 meses; *drift* ativo; versionamento/backup e *rollback* em 5 min.

18. Roadmap e Próximos Passos

18.1. Fase 1 (0–3 meses)

- Implementar *drift detection* automatizado
- Integração com SCADA industrial
- Otimização de hiperparâmetros automatizada
- Testes A/B com modelos alternativos

18.2. Fase 2 (3–6 meses)

- Deep Learning para séries temporais
- *Federated Learning* multi-fábrica
- *Streaming* em tempo real (Apache Kafka)
- *Mobile app* para técnicos de campo

18.3. Fase 3 (6–12 meses)

- *Computer Vision* para inspeção visual
- *Digital Twin* das máquinas
- *Prescriptive Analytics*
- Integração com ERP/SAP

18.4. Melhorias Técnicas Priorizadas

Curto prazo: ensembles, seleção de *features*, *tuning* com Optuna, *load balancing*.
Médio prazo: LSTM (*time series*), detecção de anomalias não supervisionada, inferência causal, *multi-modal learning*.

19. Referências e Metodologia

19.1. Base Teórica

- Gradient Boosting: Friedman (2001)
- Random Forest: Breiman (2001)
- XGBoost: Chen & Guestrin (2016)
- Cross-Validation: Stone (1974)
- ROC-AUC: Hanley & McNeil (1982)
- Precision-Recall: Davis & Goadrich (2006)
- Feature Importance/Explainability: Lundberg & Lee (2017)

19.2. Metodologia CRISP-DM

Fases implementadas: *Business Understanding*, *Data Understanding*, *Data Preparation*, *Modeling*, *Evaluation*, *Deployment*.

19.3. Boas Práticas

Data Science: seeds fixos, versionamento de código e dados, documentação, testes unitários.
MLOps: containerização (Docker), API-first (FastAPI), monitoramento (logs estruturados), arquitetura modular.

20. Conclusões

20.1. Resultados Principais

- AUC 92,36% (acima do benchmark de 85%)
- Taxa de detecção 2,34%
- ROI $\approx$ 700%
- Latência $<$ 200 ms (tempo real)

20.2. Impacto no Negócio

Economia anual estimada de R$ 7.855.000; redução de 93,5% em paradas não programadas; aumento de 12% na disponibilidade; *payback* em $\sim$1,5 meses.

20.3. Viabilidade de Produção

Critérios atendidos: acurácia $>$ 90%, latência $<$ 500 ms, ROI $>$ 300%, escalabilidade e manutenibilidade asseguradas.