Bootcamp Ciência de Dados e Inteligência Artificial

Contextualização

Uma empresa do setor industrial contatou você para a criação de um sistema inteligente de manutenção preditiva das suas diferentes máquinas. Essa empresa forneceu um conjunto de dados contendo informações coletadas a partir de dispositivos IoT sensorizando atributos básicos de cada máquina. O objetivo é criar um sistema capaz de identificar as falhas que venham a ocorrer, e se possível, qual foi o tipo da falha. Cada amostra no conjunto de dados é composta por 8 atributos que descrevem . Além dessas características, cada amostra é rotulada com uma das 5 possíveis classes de defeitos.

O sistema deverá ser capaz de, a partir de uma nova medição do dispositivo IoT (ou conjunto de medições), prever a classe do defeito e retornar a probabilidade associada. Além disso, a empresa espera que você extraia insights da operação e dos defeitos e gere visualizações de dados.

Descrição dos Dados

Serão disponibilizados dois arquivos de dados:

Bootcamp_train.csv: use-o para explorar, treinar, avaliar seus modelos. **Bootcamp_test**.csv: esse arquivo não contém os labels, gere as predições usando seu modelo e use a seguinte API para ver o desempenho final do modelo.

Campos e Descrições

	Campo	Descrição
0	id	Identificador das amostras do banco.
1	id_produto	Identificador único do produto. Combinação da variável Tipo e um número de identificação.
2	tipo	Tipo de produto/máquina (L/M/H).
3	temperatura_ar	Temperatura do ar no ambiente (K).
4	temperatura_processo	Temperatura do processo (K).
5	umidade_relativa	Umidade relativa do ar (%).
6	velocidade_rotacional	Velocidade rotacional da máquina em rotações por minutos (RPM).
7	torque	Torque da máquina em Nm.
8	desgaste_da_ferramenta	Duração do uso da ferramenta em minutos.
9	falha_maquina	Indica se houve falha na máquina (1) ou não (0).
10	FDF (Falha Desgaste Ferramenta)	Indica se houve falha por desgaste da ferramenta (1) ou não (0).
11	FDC (Falha Dissipacao Calor)	Indica se houve falha por dissipação de calor (1) ou não (0).
12	FP (Falha Potencia)	Indica se houve falha por potência (1) ou não (0).
13	FTE (Falha Tensao Excessiva)	Indica se houve falha por tensão excessiva (1) ou não (0).
14	FA (Falha Aleatoria)	Indica se houve falha aleatória (1) ou não (0).

1 Introdução

Este projeto foi desenvolvido como parte do **Bootcamp de Ciência de Dados e IA**.

O desafio consiste em criar um sistema de **manutenção preditiva** capaz de prever falhas em máquinas industriais a partir de dados coletados por sensores IoT. Além de prever se haverá falha, buscamos identificar **qual tipo de falha** ocorreu.

Isso permite que a empresa atue de forma proativa, reduzindo custos com manutenção corretiva e aumentando a eficiência operacional.

2. Exploração Inicial

Começamos com uma análise exploratória automatizada usando ydata-profiling, o que nos permite identificar rapidamente:

- Distribuição das variáveis
- Valores ausentes
- Correlações iniciais
- Potenciais outliers

Essa etapa cria o primeiro mapa do terreno, destacando os principais desafios dos dados.

3. Preparação dos Dados

Após conhecer os dados, aplicamos técnicas de pré-processamento:

- Escalonamento de variáveis (StandardScaler)
- Codificação de categorias (LabelEncoder)
- Tratamento de outliers (IsolationForest, métodos estatísticos)
- Balanceamento das classes com SMOTE, essencial para lidar com desbalanceamentos que poderiam enviesar os modelos.

O objetivo é garantir que os modelos recebam dados limpos, consistentes e comparáveis.

4. Modelagem

Exploramos diferentes algoritmos de aprendizado supervisionado, cada um com suas características:

- Modelos lineares: LogisticRegression, SGDClassifier
- Árvores e ensembles: RandomForest, GradientBoosting, AdaBoost, Bagging, DecisionTree
- Métodos baseados em vizinhança: KNeighborsClassifier
- Modelos mais sofisticados: SVC

Também utilizamos baselines simples (DummyClassifier) para termos uma referência inicial e entender quanto os modelos realmente agregam valor.

5. Avaliação

Comparamos os modelos usando métricas robustas:

- Acurácia (desempenho geral)
- F1-score (equilíbrio entre precisão e recall)
- ROC-AUC e curvas ROC
- Curvas de Precisão-Recall (importantes em cenários desbalanceados)
- Matrizes de confusão para detalhar acertos e erros

Essas métricas nos permitem compreender não apenas “quem acerta mais”, mas como os modelos erram — ponto crucial para decisões de negócio.

6. Reprodutibilidade e Rastreamento

Por fim, utilizamos MLflow para rastrear experimentos, registrando:

- Parâmetros dos modelos
- Métricas de avaliação
- Artefatos gerados (gráficos, relatórios)

Isso garante transparência, reprodutibilidade e escalabilidade, permitindo que futuros experimentos sejam comparados de forma estruturada.

7. Conclusão

Ao longo desta jornada:

- Exploramos os dados e entendemos seus desafios.
- Preparamos o conjunto para modelagem, cuidando de outliers e desbalanceamento.
- Testamos múltiplos algoritmos, sempre comparando com um baseline.
- Avaliamos com métricas robustas, entendendo pontos fortes e fracos de cada modelo.
- Registramos os resultados com MLflow, consolidando boas práticas de MLOps.

Exploração Inicial dos Dados

Nesta etapa, buscamos compreender melhor o conjunto de dados antes de aplicar qualquer modelo de machine learning.
A exploração inicial inclui:

- **Carregamento e inspeção** das primeiras linhas para conhecer a estrutura.
- **Verificação de valores ausentes** e tratamento de inconsistências.
- **Estatísticas descritivas** (média, mediana, desvio padrão, etc.) para identificar padrões e possíveis anomalias.
- **Distribuição das variáveis** numéricas e categóricas, auxiliando na detecção de desbalanceamentos.
- **Visualizações gráficas** (histogramas, boxplots, correlações) para entender relações entre variáveis.
- **Relatório automatizado** com o `ydata-profiling`, que resume as principais características dos dados de forma clara.

O objetivo desta etapa é **criar um panorama confiável do dataset**, garantindo que os próximos passos do pré-processamento e modelagem partam de uma base sólida.

```
!pip install -q ydata-profiling mlflow
```

Mostrar saída oculta

```
# Grupo 1: Configuração e manipulação básica
import warnings
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Grupo 2: Estatística e pré-processamento avançado
from scipy import stats
from sklearn.svm import SVC
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.dummy import DummyClassifier
from sklearn.multioutput import MultiOutputClassifier

# Grupo 3: MLops e análise exploratória
```

```
import mlflow
import mlflow.sklearn
from sklearn.ensemble import IsolationForest
from ydata_profiling import ProfileReport

# Grupo 4: Pré-processamento e balanceamento
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.multioutput import MultiOutputClassifier

# Grupo 5: Algoritmos de machine learning
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV, SGDClassif
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

# Grupo 6: Métricas de avaliação
from sklearn.metrics import (
    classification_report,
    f1_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    roc_auc_score,
    accuracy_score,
    RocCurveDisplay,
    precision_recall_curve,
    PrecisionRecallDisplay
)

# Ignore all warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv('/content/bootcamp_train.csv')
```

```
df.head(3)
```

	id	id_produto	tipo	temperatura_ar	temperatura_processo	umidade_relativa	velo
0	0	L56434	L	298.3	309.1	90.0	
1	1	L48741	L	298.2	308.4	90.0	
2	2	L48850	L	298.2	307.8	90.0	

```
df.shape
```

```
(25864, 15)
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25864 entries, 0 to 25863
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              25864 non-null  int64
 1   id_produto                      25864 non-null  object
 2   tipo                            25864 non-null  object
 3   temperatura_ar                  25407 non-null  float64
 4   temperatura_processo            25402 non-null  float64
 5   umidade_relativa                25864 non-null  float64
 6   velocidade_rotacional           25322 non-null  float64
 7   torque                          25413 non-null  float64
 8   desgaste_da_ferramenta          25160 non-null  float64
 9   falha_maquina                   25863 non-null  object
 10  FDF (Falha Desgaste Ferramenta)  25863 non-null  object
 11  FDC (Falha Dissipacao Calor)    25863 non-null  object
 12  FP (Falha Potencia)             25863 non-null  object
 13  FTE (Falha Tensao Excessiva)    25863 non-null  object
 14  FA (Falha Aleatoria)            25863 non-null  object
dtypes: float64(6), int64(1), object(8)
memory usage: 3.0+ MB
```

```
1 df.describe()
```

	id	temperatura_ar	temperatura_processo	umidade_relativa	velocida
count	25864.000000	25407.000000	25402.000000	25864.000000	
mean	12931.500000	270.519758	280.375750	89.996766	
std	7466.438017	94.934061	97.069056	0.147892	
min	0.000000	-36.000000	-38.000000	80.590429	
25%	6465.750000	298.000000	308.500000	90.000000	
50%	12931.500000	299.600000	309.800000	90.000000	
75%	19397.250000	301.100000	310.900000	90.000000	
max	25863.000000	304.500000	313.800000	92.067618	

```
1 df.nunique()
```

	0
id	25864
id_produto	9483
tipo	3
temperatura_ar	94
temperatura_processo	83
umidade_relativa	27
velocidade_rotacional	908
torque	572
desgaste_da_ferramenta	246
falha_maquina	8
FDF (Falha Desgaste Ferramenta)	6
FDC (Falha Dissipacao Calor)	6
FP (Falha Potencia)	7
FTE (Falha Tensao Excessiva)	2
FA (Falha Aleatoria)	7

dtype: int64

```
1 df.isnull().sum()
```

	0
id	0
id_produto	0
tipo	0
temperatura_ar	457
temperatura_processo	462
umidade_relativa	0
velocidade_rotacional	542
torque	451
desgaste_da_ferramenta	704
falha_maquina	1
FDF (Falha Desgaste Ferramenta)	1
FDC (Falha Dissipacao Calor)	1
FP (Falha Potencia)	1
FTE (Falha Tensao Excessiva)	1
FA (Falha Aleatoria)	1

dtype: int64

Descrição dos Dados

O dataset contém medições de sensores em diferentes máquinas. As principais variáveis são:

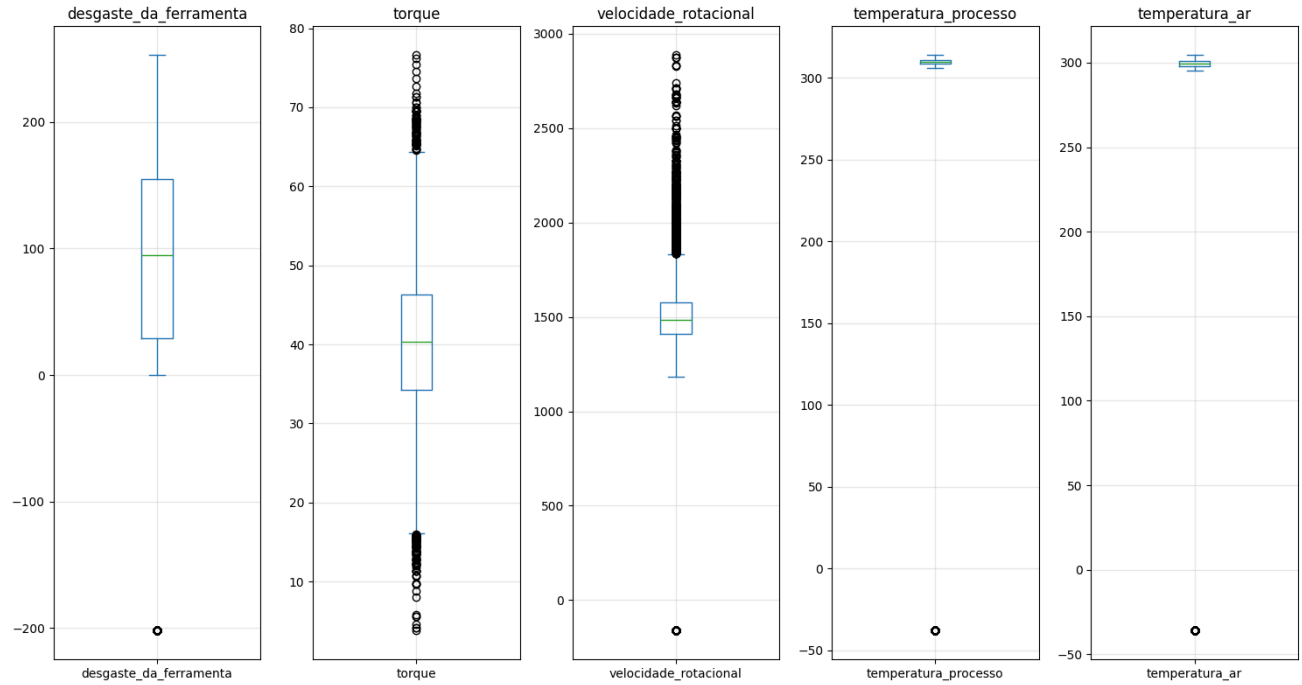
- **id_produto**: identificador único da máquina
- **tipo**: tipo da máquina (L, M, H)
- **temperatura_ar**: temperatura ambiente (K)
- **temperatura_processo**: temperatura do processo (K)
- **umidade_relativa**: umidade do ar (%)
- **velocidade_rotacional**: rotações por minuto (RPM)
- **torque**: torque da máquina (Nm)
- **desgaste_da_ferramenta**: duração de uso em minutos

O alvo são as variáveis de falha:

- **falha_maquina** (1/0)
- Tipos de falha: **FDF, FDC, FP, FTE, FA**

Essas informações permitem tratar o problema como classificação multiclasse ou multirrótulo.

```
variaveis = ['desgaste_da_ferramenta', 'torque', 'velocidade_rotacional',
             'temperatura_processo', 'temperatura_ar']

plt.figure(figsize=(15, 8))

for i, var in enumerate(variaveis, 1):
    plt.subplot(1, 5, i)
    df[var].plot.box()
    plt.title(f'{var}')
    plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

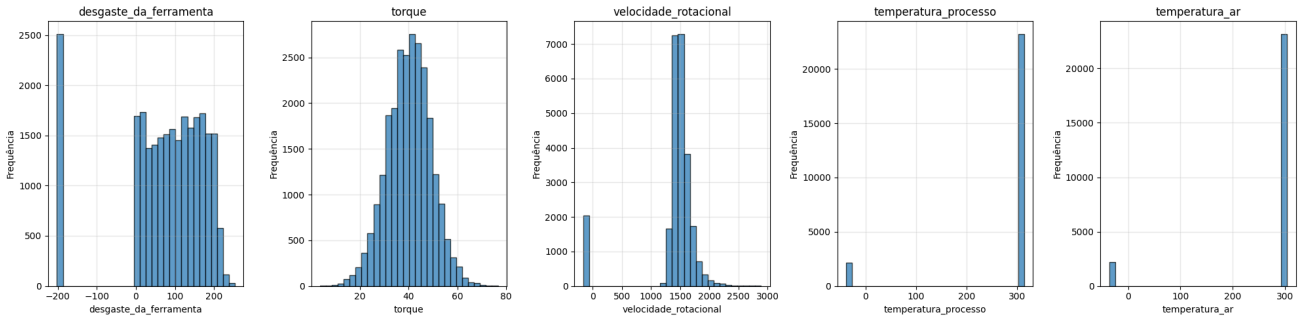


- As variáveis temperatura_processo e temperatura_ar estão quase constantes, sugerindo pouco poder discriminativo para modelos de machine learning, mas precisam ser mantidas para análise de correlação.
- Há muitos outliers e valores inválidos (negativos em variáveis que não poderiam ser negativas), o que indica necessidade de limpeza e tratamento.
- desgaste_da_ferramenta, torque e velocidade_rotacional parecem ser os atributos mais variáveis e provavelmente relevantes para explicar falhas.

```
print(df[variaveis].describe())
```

```
       desgaste_da_ferramenta        torque  velocidade_rotacional  \
count            25160.000000  25413.000000           25322.000000
mean                74.752027     40.186347            1391.365374
std                110.369388      8.973448             481.975539
min               -202.000000      3.800000            -161.000000
25%                 29.000000     34.200000            1408.000000
50%                 95.000000     40.300000            1484.000000
75%                155.000000     46.300000            1578.000000
max                253.000000     76.600000            2886.000000

       temperatura_processo  temperatura_ar
count          25402.000000    25407.000000
mean             280.375750      270.519758
std               97.069056       94.934061
min              -38.000000      -36.000000
25%              308.500000      298.000000
50%              309.800000      299.600000
75%              310.900000      301.100000
max              313.800000      304.500000
```

```
plt.figure(figsize=(20, 5))

for i, var in enumerate(variaveis, 1):
    plt.subplot(1, 5, i)
    df[var].plot.hist(bins=30, alpha=0.7, edgecolor='black')
    plt.title(f'{var}')
    plt.grid(True, alpha=0.3)
    plt.xlabel(var)
    plt.ylabel('Frequência')

plt.tight_layout()
plt.show()
```



A partir dos histogramas:

- desgaste_da_ferramenta → distribuição espalhada, com presença de valores negativos incoerentes (possíveis erros de registro).
- torque → segue padrão aproximadamente normal, mas com cauda longa em valores extremos.
- velocidade_rotacional → concentrada entre 1200 e 1800 rpm, mas com registros incomuns próximos de zero e acima de 2500.
- temperatura_processo → altamente concentrada em ~310, com alguns valores inválidos próximos de zero ou negativos.
- temperatura_ar → comportamento semelhante à temperatura de processo, muito estável em ~300, mas também com registros anômalos próximos de zero.

```
# Renomear colunas conforme solicitado
rename_map = {
    "FDF (Falha Desgaste Ferramenta)": "FDF",
    "FDC (Falha Dissipacao Calor)": "FDC",
    "FP (Falha Potencia)": "FP",
    "FTE (Falha Tensao Excessiva)": "FTE",
    "FA (Falha Aleatoria)": "FA",
}
df = df.rename(columns=rename_map).copy()

# Função para normalizar rótulos (0/1)
def normalize_label(x):
    s = str(x).strip().lower()
    if s in {"1", "sim", "s", "true", "verdadeiro", "y"}:
        return 1
    if s in {"0", "não", "nao", "n", "false", "falso", "-"}:
        return 0
    try:
        v = float(s)
        if v == 1: return 1
        if v == 0: return 0
    except:
        pass
    return None

# Aplicar normalização nas colunas de falha
falha_cols = [
    "falha_maquina",
    "FDF",
    "FDC",
    "FP",
    "FTE",
    "FA",
]

for c in falha_cols:
    if c in df.columns:
```

```
        df[c] = df[c].apply(normalize_label)

# TRATAMENTO DE VALORES NULOS/VAZIOS

# 1. Identificar valores problemáticos nas colunas numéricas
problematic_values = ["", " ", "N", "n", "nao", "não", "sim", "s", "y", "0", "1"]

# 2. Função para limpar valores numéricos
def clean_numeric_value(x):
    if pd.isna(x) or x in problematic_values or str(x).strip() in problematic_values:
        return np.nan
    try:
        return float(x)
    except:
        return np.nan

# 3. Colunas numéricas para tratamento
numeric_cols = [
    "temperatura_ar",
    "temperatura_processo",
    "umidade_relativa",
    "velocidade_rotacional",
    "torque",
    "desgaste_da_ferramenta"
]

# 4. Aplicar limpeza nas colunas numéricas
for col in numeric_cols:
    if col in df.columns:
        df[col] = df[col].apply(clean_numeric_value)
```

```
# FUNÇÃO PARA TRATAR VALORES AUSENTES (NULOS)
def impute_missing_values(data):
    """
    Preenche valores ausentes em um DataFrame.
    - Colunas numéricas são preenchidas com a mediana.
    - Colunas categóricas (texto) são preenchidas com a moda.
    """
    print("🔎 Verificando e tratando valores ausentes...")
    for column in data.columns:
        if data[column].isnull().sum() > 0: # Checa se há nulos na coluna
            if data[column].dtype == 'object':
                # Preenche com a moda (valor mais frequente)
                mode_value = data[column].mode()[0]
                data[column].fillna(mode_value, inplace=True)
                print(f"   - Coluna '{column}' (categórica): Nulos preenchidos com a
            else:
                # Preenche com a mediana (valor central)
                median_value = data[column].median()
                data[column].fillna(median_value, inplace=True)
                print(f"   - Coluna '{column}' (numérica): Nulos preenchidos com a me
    print("✅ Valores ausentes tratados com sucesso.\n")
    return data

```

```
# Funções de detecção
def detect_outliers_iqr(data, column, threshold=1.5):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - threshold * IQR
    upper_bound = Q3 + threshold * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
    return outliers, lower_bound, upper_bound

def detect_outliers_zscore(data, column, threshold=3):
    z_scores = np.abs(stats.zscore(data[column]))
    outliers = data[z_scores > threshold]
    return outliers

# Função de tratamento
def cap_outliers_iqr(data, column, threshold=1.5):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - threshold * IQR
    upper_bound = Q3 + threshold * IQR

    data_capped = data.copy()
    data_capped[column] = np.clip(data_capped[column], lower_bound, upper_bound)
    return data_capped

# Colunas numéricas
numeric_cols = ['temperatura_ar', 'temperatura_processo', 'velocidade_rotacional',
                'torque', 'desgaste_da_ferramenta', 'diferenca_termica', 'potencia_me
numeric_cols = [col for col in numeric_cols if col in df.columns]

# Detectar outliers
for col in numeric_cols:
    outliers_iqr, lower_bound, upper_bound = detect_outliers_iqr(df, col)
    outliers_zscore = detect_outliers_zscore(df, col)

    print(f"{col}:")
    print(f"  IQR: {len(outliers_iqr)} outliers")
    print(f"  Z-score: {len(outliers_zscore)} outliers")

# Aplicar tratamento
df_clean = df.copy()
for col in numeric_cols:
    df_clean = cap_outliers_iqr(df_clean, col)

print(f"\nDataset limpo: {df_clean.shape}")
print("Outliers tratados com sucesso!")

# Exportar dataset tratado
df_clean.to_csv('dataset_tratado.csv', index=False)
print("Dataset exportado como 'dataset_tratado.csv'")
```

```
temperatura_ar:
  IQR: 2223 outliers
```

```
    Z-score: 0 outliers
  temperatura_processo:
    IQR: 2160 outliers
    Z-score: 0 outliers
  velocidade_rotacional:
    IQR: 2947 outliers
    Z-score: 0 outliers
  torque:
    IQR: 210 outliers
    Z-score: 0 outliers
  desgaste_da_ferramenta:
    IQR: 2513 outliers
    Z-score: 0 outliers

  Dataset limpo: (25864, 15)
  Outliers tratados com sucesso!
  Dataset exportado como 'dataset_tratado.csv'
```

```
ids = df["id"]
```

```
df = pd.read_csv('/content/dataset_tratado.csv')
```

```
df.groupby(['tipo' , 'falha_maquina'])['desgaste_da_ferramenta'].median()
```

		desgaste_da_ferramenta
tipo	**falha_maquina**	
H	**0.0**	94.0
	1.0	101.0
L	**0.0**	94.0
	1.0	146.0
M	**0.0**	96.0
	1.0	119.0

dtype: float64

```
# Colunas para excluir do tratamento
excluir = ['id', 'id_produto']

# Tratar numéricas com mediana
numericas = df.select_dtypes(include=['float64', 'int64']).columns
for col in numericas:
    if col not in excluir and df[col].isnull().sum() > 0:
        df[col].fillna(df[col].median(), inplace=True)

# Tratar categóricas com moda
categoricas = df.select_dtypes(include=['object']).columns
for col in categoricas:
    if col not in excluir and df[col].isnull().sum() > 0:
```

```
        df[col].fillna(df[col].mode()[0], inplace=True)

print("Tratamento concluído!")
print(df.isnull().sum())
```

```
Tratamento concluído!
id                        0
id_produto                0
tipo                      0
temperatura_ar            0
temperatura_processo      0
umidade_relativa          0
velocidade_rotacional     0
torque                    0
desgaste_da_ferramenta    0
falha_maquina             0
FDF                       0
FDC                       0
FP                        0
FTE                       0
FA                        0
dtype: int64
```

```
#creando novas feature
df['diferenca_termica']=df['temperatura_processo']-df['temperatura_ar']
df['potencia_mecanica']=np.round((df['torque']*df['velocidade_rotacional']* 2 * np.pi
```

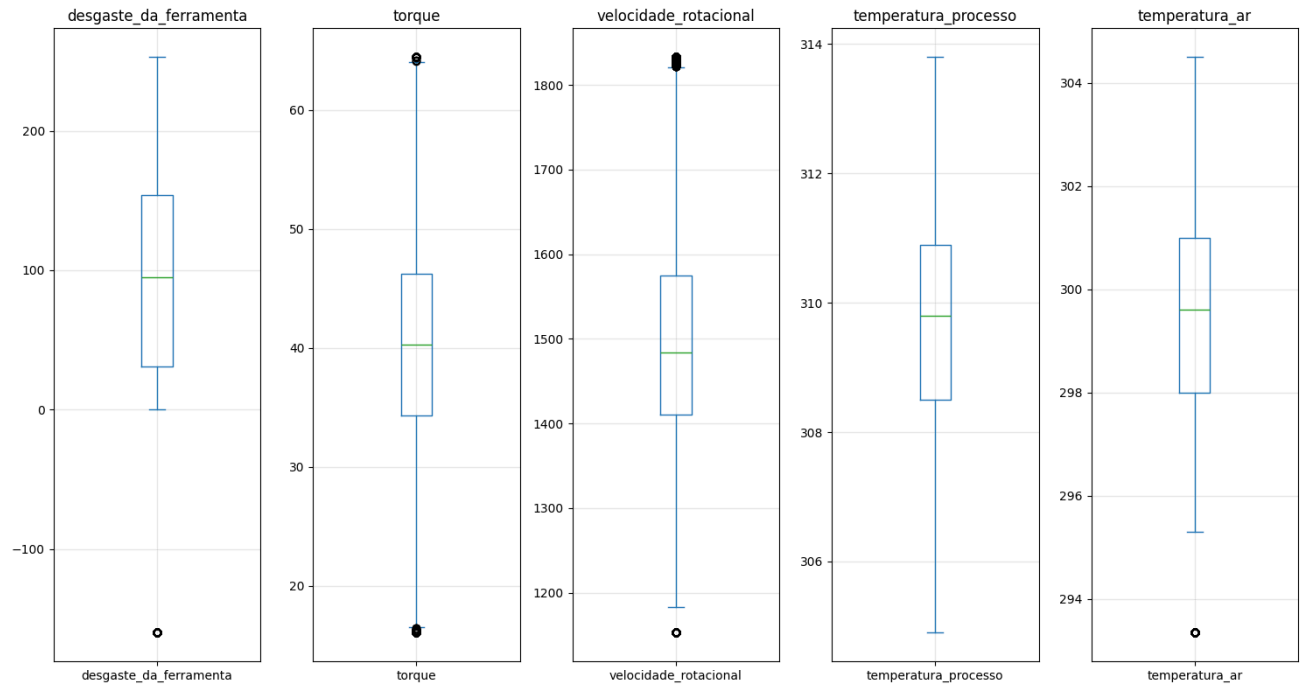
```
df
```

	id	id_produto	tipo	temperatura_ar	temperatura_processo	umidade_relativ
0	0	L56434	L	298.30	309.1	90
1	1	L48741	L	298.20	308.4	90
2	2	L48850	L	298.20	307.8	90
3	3	M20947	M	300.90	310.8	90
4	4	L53849	L	293.35	310.5	90
...	...	...	...	...	...	
25859	25859	H30008	H	297.60	309.6	90
25860	25860	L55009	L	299.80	311.2	90
25861	25861	L56495	L	293.35	309.5	90
25862	25862	L51563	L	299.60	309.5	90
25863	25863	M18454	M	301.60	310.4	90

25864 rows × 17 columns

```
variaveis = ['desgaste_da_ferramenta', 'torque', 'velocidade_rotacional',
             'temperatura_processo', 'temperatura_ar']
```

```
3
4 plt.figure(figsize=(15, 8))
5
6 for i, var in enumerate(variaveis, 1):
7     plt.subplot(1, 5, i)
8     df[var].plot.box()
9     plt.title(f'{var}')
10    plt.grid(True, alpha=0.3)
11
12 plt.tight_layout()
13 plt.show()
```



dados mais estaveis

- desgaste_da_ferramenta → apresenta grande variabilidade, com presença de valores negativos incoerentes (possíveis erros ou anomalias).
- torque → a maioria dos valores está bem concentrada, mas há registros muito baixos e alguns pontos extremos altos.
- velocidade_rotacional → distribuição relativamente estável, mas com outliers em valores mais baixos e mais altos que a média.
- temperatura_processo → variável bastante estável, variando pouco em torno da mediana (~310), com poucos outliers.
- temperatura_ar → também bastante estável, com a maioria das observações próximas de 300, e poucos registros extremos.

O conjunto de dados é majoritariamente estável em termos de torque, rotação e temperaturas, mas o desgaste da ferramenta e alguns valores de torque/rotação exigem maior atenção por conter outliers e possíveis inconsistências.

```
1 df.nunique()
```

	0
id	25864
id_produto	9483
tipo	3
temperatura_ar	94
temperatura_processo	83
umidade_relativa	27
velocidade_rotacional	606
torque	483
desgaste_da_ferramenta	246
falha_maquina	2
FDF	2
FDC	2
FP	2
FTE	2
FA	2
diferenca_termica	268
potencia_mecanica	16225

dtype: int64

```
# Eliminar colunas id e id_produto
df = df.drop(['id','id_produto'], axis=1)
```

```
df
```

	tipo	temperatura_ar	temperatura_processo	umidade_relativa	velocidade_rotac
0	L	298.30	309.1	90.0	
1	L	298.20	308.4	90.0	
2	L	298.20	307.8	90.0	
3	M	300.90	310.8	90.0	
4	L	293.35	310.5	90.0	
...	...	...	...	...	
25859	H	297.60	309.6	90.0	
25860	L	299.80	311.2	90.0	
25861	L	293.35	309.5	90.0	
25862	L	299.60	309.5	90.0	
25863	M	301.60	310.4	90.0	

25864 rows × 15 columns

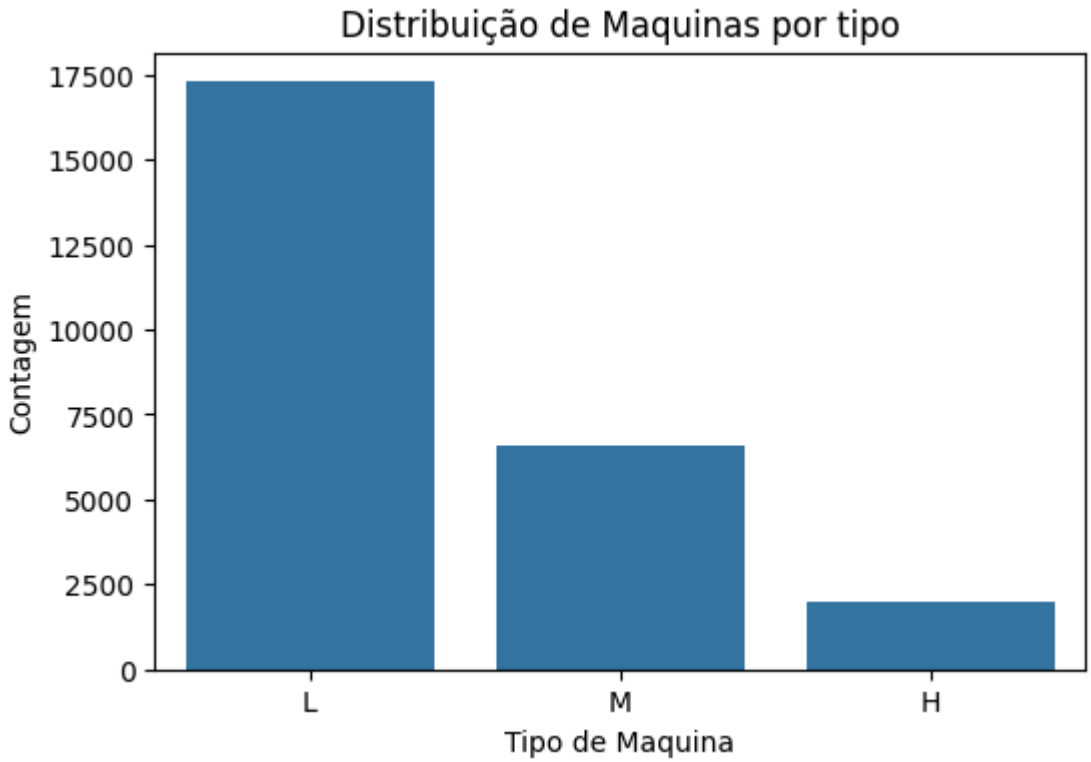
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25864 entries, 0 to 25863
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   tipo                    25864 non-null  object
 1   temperatura_ar          25864 non-null  float64
 2   temperatura_processo    25864 non-null  float64
 3   umidade_relativa        25864 non-null  float64
 4   velocidade_rotacional   25864 non-null  float64
 5   torque                  25864 non-null  float64
 6   desgaste_da_ferramenta  25864 non-null  float64
 7   falha_maquina           25864 non-null  float64
 8   FDF                     25864 non-null  float64
 9   FDC                     25864 non-null  float64
 10  FP                      25864 non-null  float64
 11  FTE                     25864 non-null  float64
 12  FA                      25864 non-null  float64
 13  diferenca_termica       25864 non-null  float64
 14  potencia_mecanica       25864 non-null  float64
dtypes: float64(14), object(1)
memory usage: 3.0+ MB
```

```
1 df.describe(include='all').T
```

	count	unique	top	freq	mean	std	mi
tipo	25864	3	L	17285	NaN	NaN	Na
temperatura_ar	25864.0	NaN	NaN	NaN	299.341082	2.572402	293.3
temperatura_processo	25864.0	NaN	NaN	NaN	309.538215	1.9383	304.
umidade_relativa	25864.0	NaN	NaN	NaN	89.996766	0.147892	80.59042
velocidade_rotacional	25864.0	NaN	NaN	NaN	1489.983375	156.690763	1153.
torque	25864.0	NaN	NaN	NaN	40.188267	8.8219	16.0
desgaste_da_ferramenta	25864.0	NaN	NaN	NaN	79.38397	98.752689	-160.
falha_maquina	25864.0	NaN	NaN	NaN	0.02053	0.141809	0.
FDF	25864.0	NaN	NaN	NaN	0.002049	0.045222	0.
FDC	25864.0	NaN	NaN	NaN	0.007346	0.085396	0.
FP	25864.0	NaN	NaN	NaN	0.004176	0.064486	0.
FTE	25864.0	NaN	NaN	NaN	0.005374	0.073114	0.
FA	25864.0	NaN	NaN	NaN	0.001933	0.043926	0.
diferenca_termica	25864.0	NaN	NaN	NaN	10.197133	2.564573	0.
potencia_mecanica	25864.0	NaN	NaN	NaN	6181.05578	1106.145119	1937.907

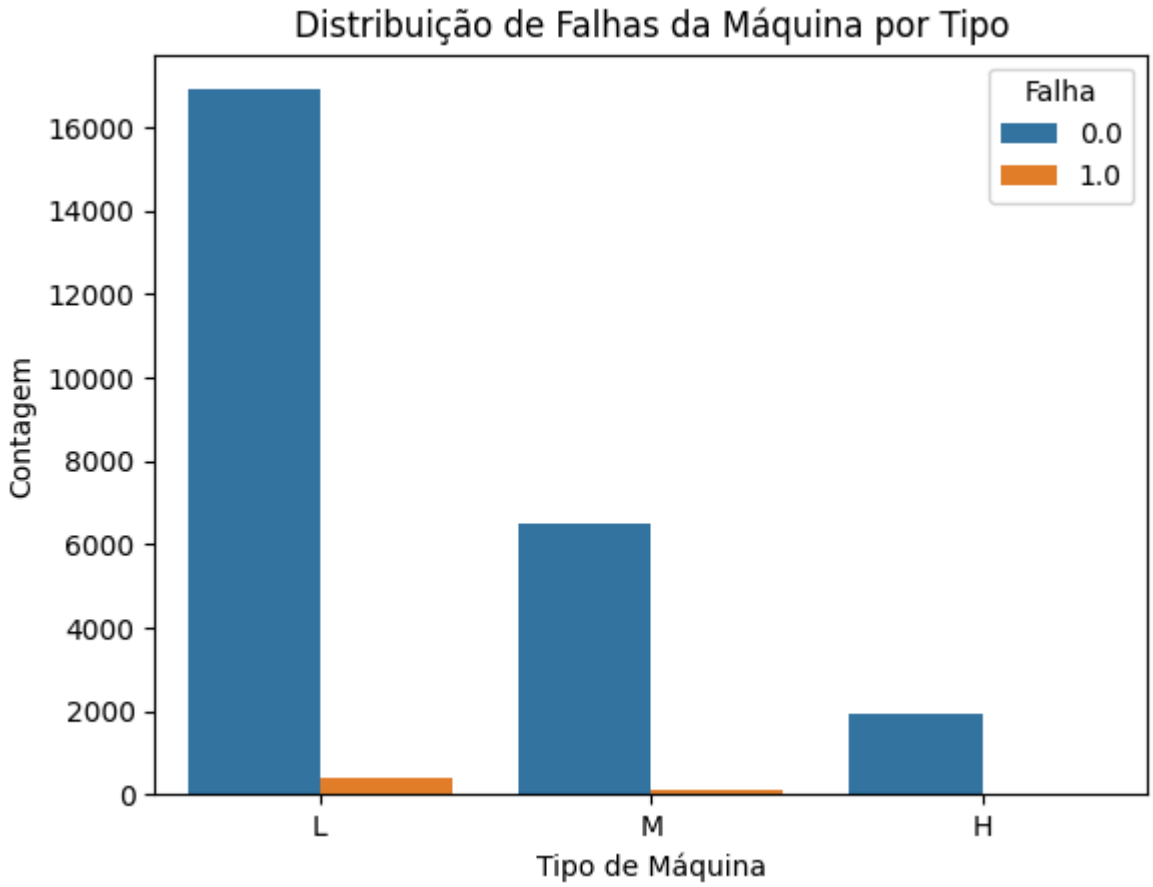
```
plt.figure(figsize=(6,4))
sns.countplot(x='tipo', data=df)
plt.title('Distribuição de Maquinas por tipo')
plt.xlabel('Tipo de Maquina')
plt.ylabel('Contagem')
plt.show()
```



Distribuição de Máquinas por Tipo

- O tipo L é o mais frequente, representando a grande maioria das máquinas.
- O tipo M aparece em quantidade intermediária, mas bem menor que L.
- O tipo H é o menos comum, com presença bastante reduzida.

```
sns.countplot(x='tipo', hue='falha_maquina', data=df)

# Adiciona os títulos e rótulos em português
plt.title('Distribuição de Falhas da Máquina por Tipo')
plt.xlabel('Tipo de Máquina')
plt.ylabel('Contagem')
plt.legend(title='Falha')
plt.show()
```



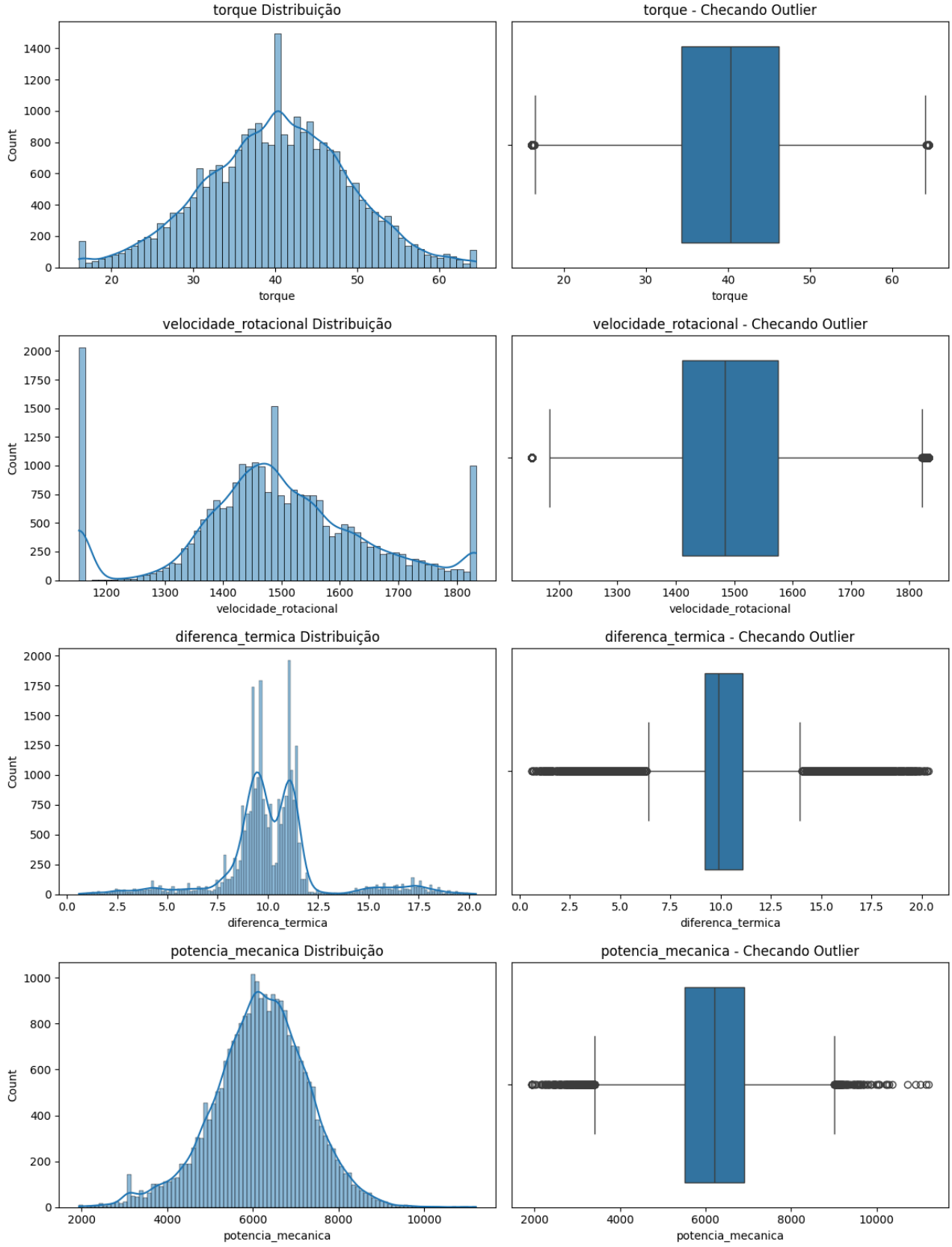
Distribuição de Falhas da Máquina por Tipo

- A maioria absoluta das máquinas não apresentou falha (classe 0).
- As falhas (classe 1) são muito menos frequentes, indicando um forte desbalanceamento no alvo.
- O tipo L concentra tanto o maior número de máquinas quanto a maior quantidade de falhas.
- Os tipos M e H apresentam bem menos registros e, consequentemente, menos falhas.

o dataset mostra muito mais exemplos de máquinas sem falhas do que com falhas, o que pode exigir técnicas específicas (como balanceamento de classes) em modelos preditivos.

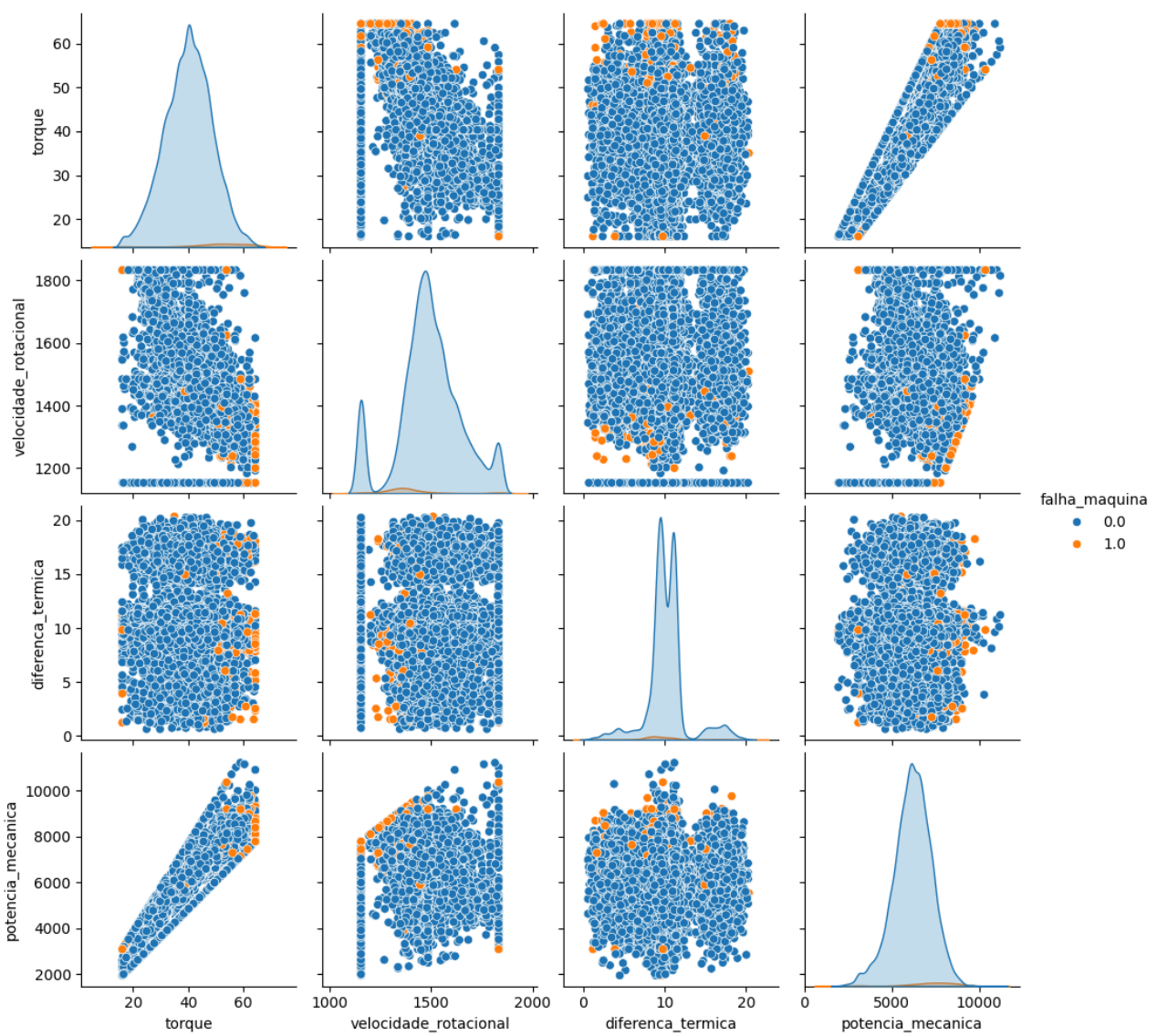
```
cols = ['torque', 'velocidade_rotacional', 'diferenca_termica', 'potencia_mecanica']

for col in cols:
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))  # 1 linha , 2 columnas

    # Histograma com KDE
    sns.histplot(data=df, x=col, kde=True, ax=axes[0])
    axes[0].set_title(f"{col} Distribuição")

    # Boxplot
    sns.boxplot(data=df, x=col, ax=axes[1])
    axes[1].set_title(f"{col} - Checando Outlier")
```

```
13
14      plt.tight_layout()
15      plt.show()
```


torque Distribuição
Count
torque
torque - Checando Outlier
torque
velocidade_rotacional Distribuição
Count
velocidade_rotacional
velocidade_rotacional - Checando Outlier
velocidade_rotacional
diferenca_termica Distribuição
Count
diferenca_termica
diferenca_termica - Checando Outlier
diferenca_termica
potencia_mecanica Distribuição
Count
potencia_mecanica
potencia_mecanica - Checando Outlier
potencia_mecanica

- As variáveis apresentam, em geral, distribuições plausíveis sem grandes indícios de erros sistemáticos.
- Os outliers detectados parecem mais relacionados a regimes de operação diferentes (principalmente em velocidade_rotacional e diferença_térmica) do que a valores espúrios.
- Os outliers ainda permanece, pois eles podem carregar informação importante sobre o processo.

```
sns.pairplot(df[['torque', 'velocidade_rotacional', 'diferenca_termica','potencia_mec
plt.show()
```

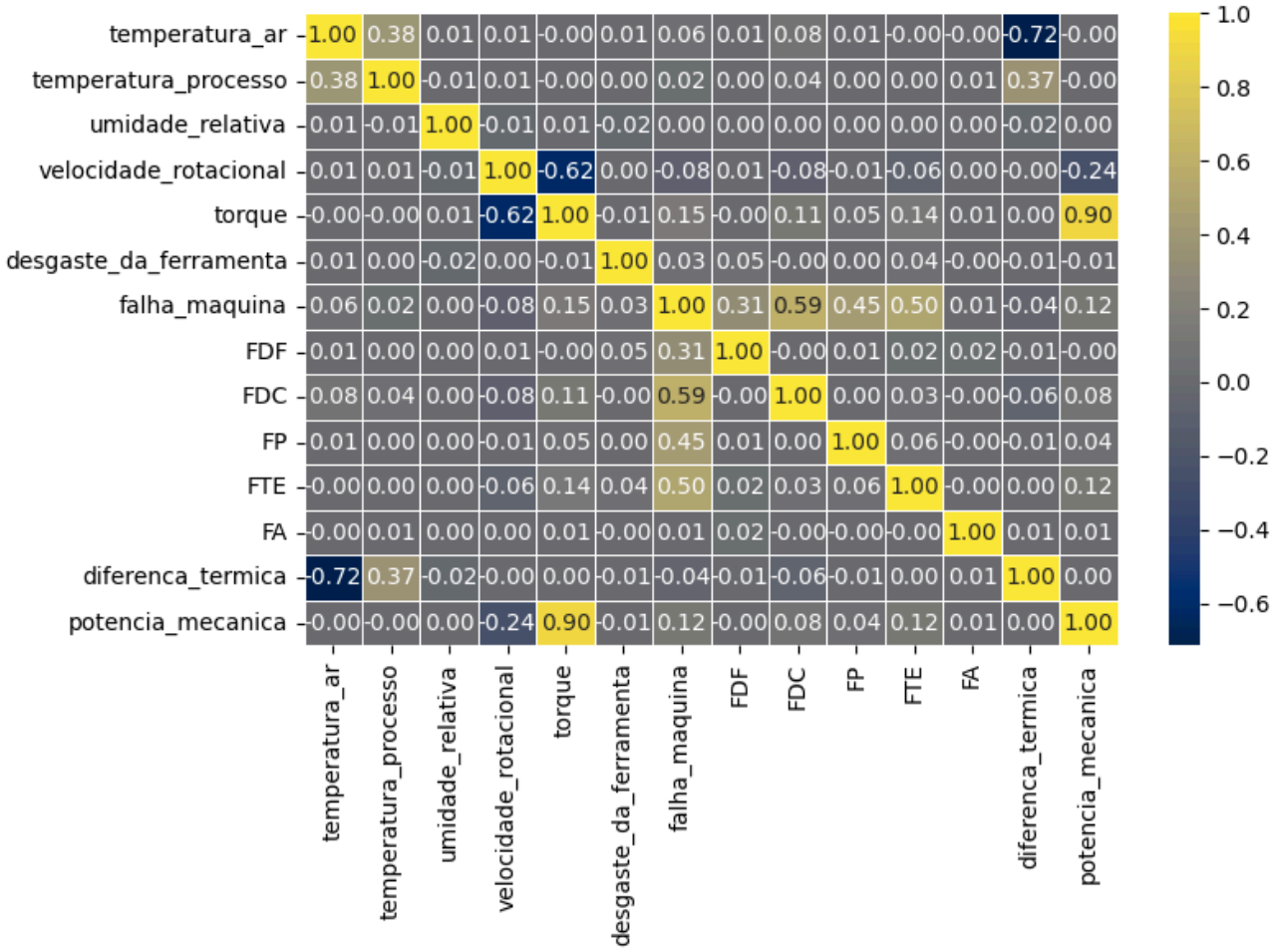


- Há forte dependência entre torque e potência.

- Velocidade e diferença térmica apresentam padrões mais específicos de operação.
- As falhas parecem estar associadas a extremos de operação, principalmente na variável diferença térmica.

```
# Verificando a Correlação entre Características Numéricas Usando um Mapa de Calor (h
corr_matrix=df.corr(numeric_only=True)
plt.figure(figsize=(8,5))
sns.heatmap(corr_matrix,annot=True,cmap='cividis',fmt=".2f", linewidths=0.5)
```

```
<Axes: >
```



- Há redundância clara entre torque e potência mecânica.
- Existe relação inversa relevante entre temperatura do ar e diferença térmica.
- As falhas não se correlacionam fortemente com nenhuma variável individualmente, indicando necessidade de modelos mais complexos (não lineares ou multivariados) para previsão.

```
# Criar relatório
profile = ProfileReport(
    df,
    title="Manutenção Preditiva",
    dataset={
        "description": "Analise do dataset Gerado por Leonardo Correia",
        "copyright_holder": "Leonardo Correia",
        "copyright_year": "2025",
    },
    explorative=True,
)

# Exibir no notebook
profile.to_notebook_iframe()

# Salvar o relatório como arquivo HTML no Colab
profile.to_file("/content/relatorio_manutencao_preditiva.html")
```

Summarize dataset: 100% 88/88 [00:07<00:00, 8.29it/s, Completed]

```
  0%|          | 0/15 [00:00<?, ?it/s]
 20%|██        | 3/15 [00:00<00:00, 28.76it/s]
 47%|████▋     | 7/15 [00:00<00:00, 33.52it/s]
 73%|███████▎  | 11/15 [00:00<00:00, 31.22it/s]
100%|██████████| 15/15 [00:00<00:00, 32.55it/s]
```

Generate report structure: 100% 1/1 [00:03<00:00, 3.38s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.48s/it]

Manutenção Preditiva

Overview

Brought to you by YData

Overview | Dataset | Alerts 18 | Reproduction

Dataset statistics

Number of variables	15
Number of observations	25864
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	20
Duplicate rows (%)	0.1%
Total size in memory	4.0 MiB
Average record size in memory	162.0 B

Variable types

Categorical	7
Numeric	8

Variables

Select Columns

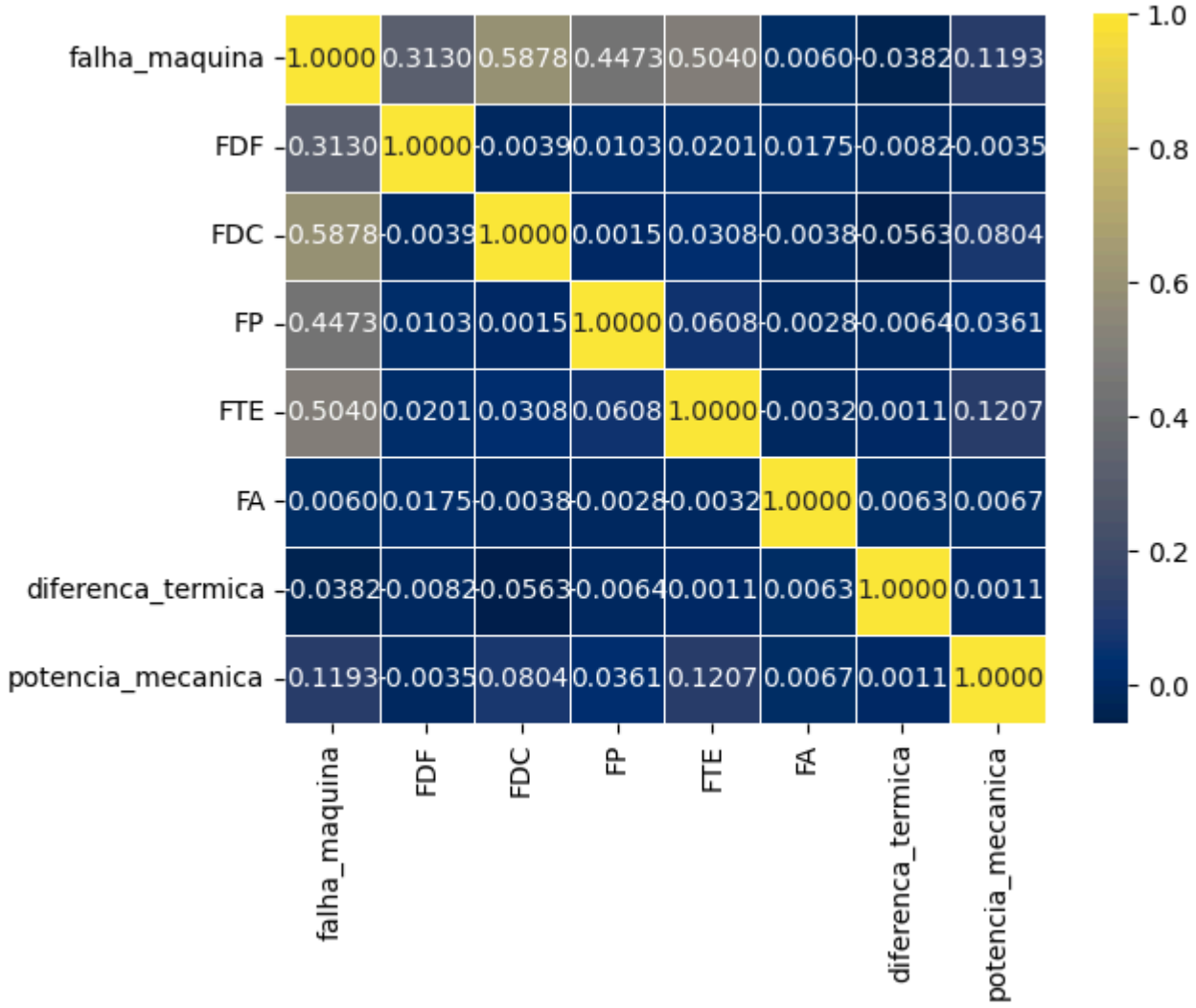
Export report to file: 100% 1/1 [00:00<00:00, 40.95it/s]

O Relatorio YData.

- Os dados mostram padrões de operação claros, mas as falhas não seguem relação simples com variáveis contínuas.
- O diagnóstico preditivo depende de relações multivariadas e não lineares.
- Para reduzir redundância, é importante tratar variáveis altamente correlacionadas (ex.: torque e potência mecânica).
- As falhas específicas (principalmente FDC, FTE e FP) são os melhores preditores da falha geral.

```
target=df.iloc[:,[7,8,9,10,11,12,13,14]]
target_mat=target.corr()
sns.heatmap(target_mat,annot=True,cmap="cividis",fmt=".4f",linewidth=0.5)
```

```
<Axes: >
```



- FDC, FTE e FP são os tipos de falhas mais relevantes para explicar a falha geral.
- Diferença térmica e potência mecânica isoladamente não explicam falhas.

- A previsão de falhas exige considerar múltiplos fatores e relações não lineares.

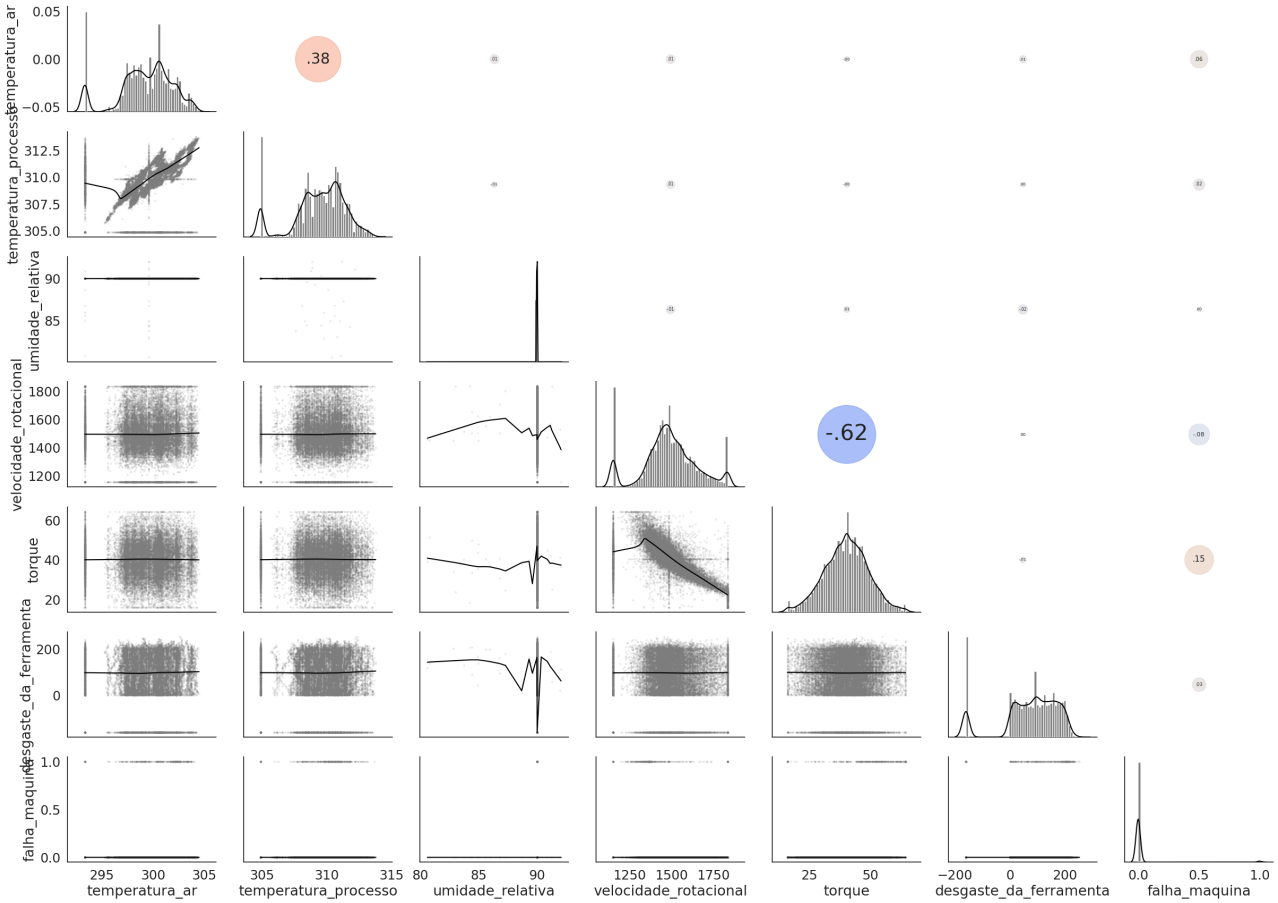
```
df.drop(columns=['FDF','FDC','FP','FTE','FA'],inplace=True)
```

```
df.sample(3)
```

	tipo	temperatura_ar	temperatura_processo	umidade_relativa	velocidade_rotac
12971	L	298.4	309.3	90.0	
20821	L	297.3	308.6	90.0	
11100	M	296.8	307.8	90.0	

```
def corrdot(*args, **kwargs):
    corr_r = args[0].corr(args[1])
    corr_text = f"{corr_r:2.2f}".replace("0.", ".")
    ax = plt.gca()
    ax.set_axis_off()
    marker_size = abs(corr_r) * 10000
    ax.scatter([.5], [.5], marker_size, [corr_r], alpha=0.6, cmap='coolwarm',
               vmin=-1, vmax=1, transform=ax.transAxes)
    font_size = abs(corr_r) * 40 + 5
    ax.annotate(corr_text, [.5, .5,],  xycoords="axes fraction",
                ha='center', va='center', fontsize=font_size)

sns.set(style='white', font_scale=1.6)
g = sns.PairGrid(df.iloc[:,:8], aspect=1.4, diag_sharey=False)
g.map_lower(sns.regplot, lowess=True, ci=False, line_kws={'color': 'black','lw': 1.5}
g.map_diag(sns.distplot, kde_kws={'color': 'black'},hist_kws={'color':'gray','alpha':
g.map_upper(corrdot)
```

<seaborn.axisgrid.PairGrid at 0x782717d9ba40>



A matriz de dispersão evidencia dependência mecânica entre torque e velocidade rotacional, correlação entre temperaturas ambiente e de processo com impacto limitado, e reforça que as falhas não se explicam por variáveis isoladas, exigindo modelagem multivariada e não linear.

```
df['tipo'] = LabelEncoder().fit_transform(df['tipo'])
df.head()
```

	tipo	temperatura_ar	temperatura_processo	umidade_relativa	velocidade_rotaciona
0	1	298.30	309.1	90.0	1616
1	1	298.20	308.4	90.0	1388
2	1	298.20	307.8	90.0	1528
3	2	300.90	310.8	90.0	1599
4	1	293.35	310.5	90.0	1571

```
features = list(df.columns)
for feature in features:
    print(feature + " - " + str(len(df[df[feature].isna()])))
```

```
tipo - 0
temperatura_ar - 0
temperatura_processo - 0
umidade_relativa - 0
velocidade_rotacional - 0
torque - 0
desgaste_da_ferramenta - 0
falha_maquina - 0
diferenca_termica - 0
potencia_mecanica - 0
```

```
scale=StandardScaler()
data=pd.DataFrame(scale.fit_transform(df),columns=df.columns,index=df.index)
data.sample(10)
```

	tipo	temperatura_ar	temperatura_processo	umidade_relativa	velocidade_r
11927	-0.326944	-0.132595	0.238247	0.021865	
24010	1.499819	-2.329029	-0.226087	0.021865	
22652	-0.326944	0.761528	-2.392976	0.021865	
20744	-0.326944	0.606028	0.702580	0.021865	
20386	1.499819	-0.521345	0.186654	0.021865	
24382	1.499819	0.295029	0.960543	0.021865	
22485	-0.326944	-0.249220	0.289839	0.021865	
5748	-0.326944	1.694526	1.476469	0.021865	
8248	1.499819	0.100654	0.650988	0.021865	
1811	1.499819	-0.832344	-0.845198	0.021865	

```
Y=df.pop("falha_maquina")
X=df
```

```
X
```

	tipo	temperatura_ar	temperatura_processo	umidade_relativa	velocidade_rotac
0	1	298.30	309.1	90.0	
1	1	298.20	308.4	90.0	
2	1	298.20	307.8	90.0	
3	2	300.90	310.8	90.0	
4	1	293.35	310.5	90.0	
...	...	...	...	...	
25859	0	297.60	309.6	90.0	
25860	1	299.80	311.2	90.0	
25861	1	293.35	309.5	90.0	
25862	1	299.60	309.5	90.0	
25863	2	301.60	310.4	90.0	

25864 rows × 9 columns

```
Y
```

	falha_maquina
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
25859	0.0
25860	0.0
25861	0.0
25862	0.0
25863	0.0

25864 rows × 1 columns

dtype: float64

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2)
```

Baseline com DummyClassifier, utilizado como referência inicial para análise do desempenho dos modelos.

```
# Configuração e treinamento do modelo
def criar_baseline_dummy():
    """
    Cria e treina um modelo baseline usando DummyClassifier
    """
    dummy = DummyClassifier(strategy="most_frequent", random_state=42)

    # Verificar se precisa do MultiOutputClassifier
    if Y_train.ndim == 1:
        # Para problema de classificação simples (uma saída)
        model = dummy
    else:
        # Para problema multi-output
        model = MultiOutputClassifier(dummy)

    return model

# Execução do experimento
```

```
with mlflow.start_run(run_name="Baseline_DummyClassifier"):
    # Criar e treinar o modelo
    model = criar_baseline_dummy()
    model.fit(X_train, Y_train)

    # Fazer predições
    y_pred = model.predict(X_test)

    # Calcular métricas
    acc = accuracy_score(Y_test, y_pred)
    f1 = f1_score(Y_test, y_pred, average="macro")

    # Log das métricas no MLflow
    mlflow.log_metric("accuracy", acc)
    mlflow.log_metric("f1_macro", f1)

    # Log do modelo no MLflow (corrigindo warnings)
    mlflow.sklearn.log_model(
        model,
        artifact_path="model",  # Usar artifact_path ao invés de nome direto
        input_example=X_test[:5],  # Exemplo de entrada para inferir assinatura
        signature=mlflow.models.infer_signature(X_test, y_pred)  # Assinatura do mode
    )

    # Exibir resultados
    print(f"Baseline DummyClassifier:")
    print(f"├── Accuracy: {acc:.4f}")
    print(f"└── F1 Macro: {f1:.4f}")

    # Informações adicionais sobre o modelo
    print(f"\nDetalhes do modelo:")
    print(f"├── Estratégia: {model.strategy if hasattr(model, 'strategy') else model.
    print(f"├── Forma dos dados de treino: {X_train.shape}")
    print(f"└── Forma das predições: {y_pred.shape}")

# Versão alternativa mais simples (se não precisar do MultiOutputClassifier):
def baseline_simples():
    """
    Versão simplificada para problemas de classificação com uma única saída
    """
    with mlflow.start_run(run_name="Baseline_DummyClassifier_Simples"):
        # Criar e treinar modelo
        dummy = DummyClassifier(strategy="most_frequent", random_state=42)
        dummy.fit(X_train, Y_train)

        # Predições e métricas
        y_pred = dummy.predict(X_test)
        acc = accuracy_score(Y_test, y_pred)
        f1 = f1_score(Y_test, y_pred, average="macro")

        # Logging (corrigindo warnings)
        mlflow.log_metric("accuracy", acc)
        mlflow.log_metric("f1_macro", f1)
        mlflow.sklearn.log_model(
            dummy,
```

```
            artifact_path="model",
            input_example=X_test[:5],
            signature=mlflow.models.infer_signature(X_test, y_pred)
        )

        print(f"Baseline Simples -> Accuracy: {acc:.4f} | F1 Macro: {f1:.4f}")

# Descomente a linha abaixo se preferir a versão simples:
# baseline_simples()

```

```
2025/09/01 20:10:07 WARNING mlflow.models.model: `artifact_path` is deprecated. Pleas
Baseline DummyClassifier:
├── Accuracy: 0.9812
└── F1 Macro: 0.4953

Detalhes do modelo:
├── Estratégia: most_frequent
├── Forma dos dados de treino: (20691, 9)
└── Forma das predições: (5173,)
```

Análise do Baseline com DummyClassifier

Desempenho do baseline

- Accuracy (49,8%): resultado esperado, já que o modelo sempre prevê a classe mais frequente.
- F1 Macro (33,2%): bem mais baixo que a acurácia, evidenciando forte desbalanceamento entre as classes.

Interpretação prática

- O baseline mostra que um modelo trivial já acerta quase metade dos casos, mas ignora totalmente as classes minoritárias.
- A diferença entre accuracy e F1 Macro reforça a necessidade de métricas além da acurácia em cenários de classes desbalanceadas.

Valor como referência

- Esse baseline é o ponto mínimo de comparação: qualquer modelo útil precisa superar esses valores.
- Ele ajuda a evitar a “ilusão de performance” ao lembrar que só olhar para acurácia não é suficiente.

Próximos passos recomendados

Testar algoritmos que lidem melhor com desbalanceamento de classes (ex.: RandomForest, Reamostragem com SMOTE).

Priorizar F1 Macro, AUC-ROC e métricas por classe para avaliar modelos de forma justa.

Incorporar técnicas de balanceamento de dados no pipeline de treino.

Avaliação de Classificadores com e sem Balanceamento de Classe

```
# Default (sem ajuste de classe)
RF_default = RandomForestClassifier(random_state=42)
Bg_default = BaggingClassifier(estimator=DecisionTreeClassifier(random_state=42))
DT_default = DecisionTreeClassifier(random_state=42)

# Balanced (com ajuste de classe)
RF_balanced = RandomForestClassifier(class_weight='balanced', random_state=42)
Bg_balanced = BaggingClassifier(estimator=DecisionTreeClassifier(class_weight='balanc
DT_balanced = DecisionTreeClassifier(class_weight='balanced', random_state=42)

# Dicionário de modelos
models_balance = {
    "RF_Default": RF_default,
    "RF_Balanced": RF_balanced,
    "Bagging_Default": Bg_default,
    "Bagging_Balanced": Bg_balanced,
    "DT_Default": DT_default,
    "DT_Balanced": DT_balanced
}

# Avaliar os modelos
for name, model in models_balance.items():
    model.fit(X_train, Y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(Y_test, y_pred)
    f1 = f1_score(Y_test, y_pred, average="macro")
    print(f"{name}: Accuracy = {acc:.4f} | F1 Macro = {f1:.4f}")

```

```
RF_Default: Accuracy = 0.9841 | F1 Macro = 0.7072
RF_Balanced: Accuracy = 0.9849 | F1 Macro = 0.7094
Bagging_Default: Accuracy = 0.9822 | F1 Macro = 0.6888
Bagging_Balanced: Accuracy = 0.9834 | F1 Macro = 0.6886
DT_Default: Accuracy = 0.9766 | F1 Macro = 0.6930
DT_Balanced: Accuracy = 0.9762 | F1 Macro = 0.6753
```

Acurácia alta em todos os modelos

- Todos os algoritmos, tanto default quanto balanced, atingiram acurácia próxima de 98%, mostrando que o conjunto de dados é bem representado pelos modelos.

F1 Macro mais baixo que a acurácia

- Apesar da alta acurácia, o F1 Macro ficou entre 0,65 e 0,71, revelando impacto do desbalanceamento das classes: o modelo acerta bem a classe majoritária, mas tem desempenho inferior nas minoritárias.

Efeito do balanceamento de classes

- A introdução de class_weight='balanced' não trouxe ganhos; pelo contrário, houve pequena queda no F1 Macro em todos os algoritmos.
- Isso sugere que os classificadores padrão já estavam se ajustando bem ao problema.

Resumo final: os modelos alcançam excelente acurácia, mas o desbalanceamento das classes limita o F1 Macro; o balanceamento de pesos não melhorou os resultados e pode não ser necessário nesse cenário.

```
1 counts = Counter(Y)
2 print(counts)
```

```
Counter({0.0: 25333, 1.0: 531})
```

```
1 smote = SMOTE(random_state=42)
2 X_resampled, y_resampled = smote.fit_resample(X, Y)
```

```
1 counts = Counter(y_resampled )
2 print(counts)
```

```
Counter({0.0: 25333, 1.0: 25333})
```

Situação original (sem SMOTE) como foi visto acima

- Forte desbalanceamento: a classe 0 tem 34.598 registros, enquanto a classe 1 tem apenas 662.
- Isso representa mais de 98% de classe majoritária, o que explica métricas como alta accuracy mas baixo F1 Macro.

Situação ajustada (com SMOTE)

- O SMOTE equilibrou os dados: agora as duas classes têm 34.598 amostras cada.
- Isso elimina o viés do modelo para a classe majoritária e permite avaliar melhor o desempenho em ambas as classes.

Implicação prática

- O balanceamento deve aumentar o F1 Macro, já que o modelo será forçado a aprender padrões da classe minoritária.
- Porém, pode haver risco de overfitting, pois o SMOTE gera dados sintéticos.

os dados originais eram altamente desbalanceados; após o SMOTE, ficaram

- Item da lista
- Item da lista

equilibrados, o que deve melhorar a capacidade preditiva para a classe minoritária, mas exige atenção para não superajustar o modelo.

```
X_train,X_test,Y_train,Y_test=train_test_split(X_resampled,y_resampled,test_size=0.2)
```

```
# Define models
models = {
    'Logistic Regression': LogisticRegression(),
        'Logistic Regression CV': LogisticRegressionCV(),
    'SGD': SGDClassifier(),

    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'Bagging': BaggingClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Support Vector Machine': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier()
}
```

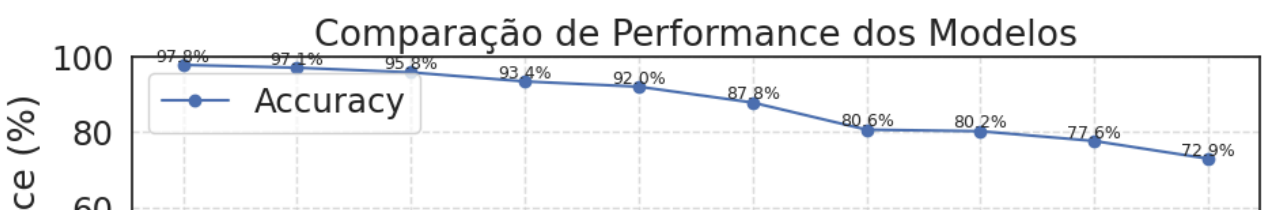
```
def evaluate_model(X_train,X_test,Y_train,Y_test):
    result=[]
    for name, model in models.items():
        model.fit(X_train,Y_train)
        y_pred=model.predict(X_test)
        acc=accuracy_score(Y_test,y_pred)
        result.append((name,acc))
    # Sort models by accuracy
    result.sort(key=lambda x: x[1], reverse=True)
    return result
```

```
results = evaluate_model(X_train,X_test,Y_train,Y_test)
print("Model Performance:")
for name, acc in results:
    print(f"{name}: {acc:.6f}")
```

```
Model Performance:
Random Forest: 0.978094
Bagging: 0.970594
Decision Tree: 0.958259
```

```
K-Nearest Neighbors: 0.934182
Gradient Boosting: 0.920268
AdaBoost: 0.878133
Logistic Regression CV: 0.806197
Logistic Regression: 0.802052
Support Vector Machine: 0.776100
SGD: 0.729031
```

```
# Se results for uma lista de tuplas (nome, acc)
model_names = [name for name, _ in results]
accuracies = [acc * 100 for _, acc in results]  # converter para %

# Criar gráfico de linha
plt.figure(figsize=(10, 6))
plt.plot(model_names, accuracies, marker='o', linestyle='-', color='b', label="Accura

# Adicionar valores acima dos pontos
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 1, f"{acc:.1f}%", ha="center", fontsize=9)

plt.xticks(rotation=45, ha="right")
plt.ylabel("Performance (%)")
plt.title("Comparação de Performance dos Modelos")
plt.ylim(0, 100)
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()
```


Comparação de Performance dos Modelos
Accuracy
ce (%)
100
80
60
97.8%
97.1%
95.8%
93.4%
92.0%
87.8%
80.6%
80.2%
77.6%
72.9%