**Relatório de Análise para Previsão de Tempo de Vida Útil de Sensores**

**Alunos:**
Lucas Fidalgo Bitencourt - 2221061
José Guilherme Vasconcelos - 2221010
Pedro Henrique Pontes Fontana - 2111097

## Introdução

Este trabalho foi desenvolvido para analisar dados de sensores de unidades operacionais e prever o tempo de vida útil restante (RUL) das unidades, visando a implementação de manutenção preditiva.
O objetivo é explorar, limpar e analisar os dados, bem como aplicar técnicas de aprendizado de máquina para prever a vida útil restante das unidades.

## Importação de Bibliotecas

Inicialmente, importamos as bibliotecas necessárias para processar a limpeza dos dados, bem como para realizar a análise e visualização

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

## Carregamento do Dataset

O dataset é carregado a partir de um arquivo de texto e armazenado em um DataFrame:

```
PATH = '../data/raw/FD001.txt'
data = pd.read_csv(PATH, sep='\s+', header=None)
```

O banco de dados consiste em conjuntos de múltiplas séries temporais multivariadas, divididas em subconjuntos de treinamento e teste. Cada série temporal provém de um motor diferente, representando uma frota de

motores do mesmo tipo.
Os motores começam com diferentes níveis de desgaste inicial e variação de fabricação, que são desconhecidos para o usuário. Esse desgaste e variação são considerados normais e não são condições de falha.
Existem três configurações operacionais que afetam significativamente o desempenho do motor e estão incluídas nos dados, os quais também não são contaminados com ruído de sensores. No início de cada série temporal, o motor opera normalmente e desenvolve uma falha em algum ponto da série.
No conjunto de treinamento, a falha cresce até resultar na falha do sistema. Já no conjunto de teste, a série temporal termina algum tempo antes da falha do sistema. O objetivo é prever o número de ciclos operacionais restantes antes da falha nos dados de testes, ou seja, quantos ciclos operacionais o motor ainda pode operar após o último ciclo registrado.

**Visualização Inicial dos Dados**

Para entender a estrutura dos dados, as primeiras e últimas linhas do dataset, assim como sua forma, foram exibidas:

```
data.head()
data.tail()
print("Dataset shape ")
print(f"rows: {data.shape[0]}")
print(f"columns: {data.shape[1]}")
```

```
Dataset shape
rows: 20631
columns: 26
```

Os nomes das colunas foram alterados para refletir melhor as medições dos sensores e as configurações operacionais

```
columns = ['unit_number', 'time_in_cycles',
```

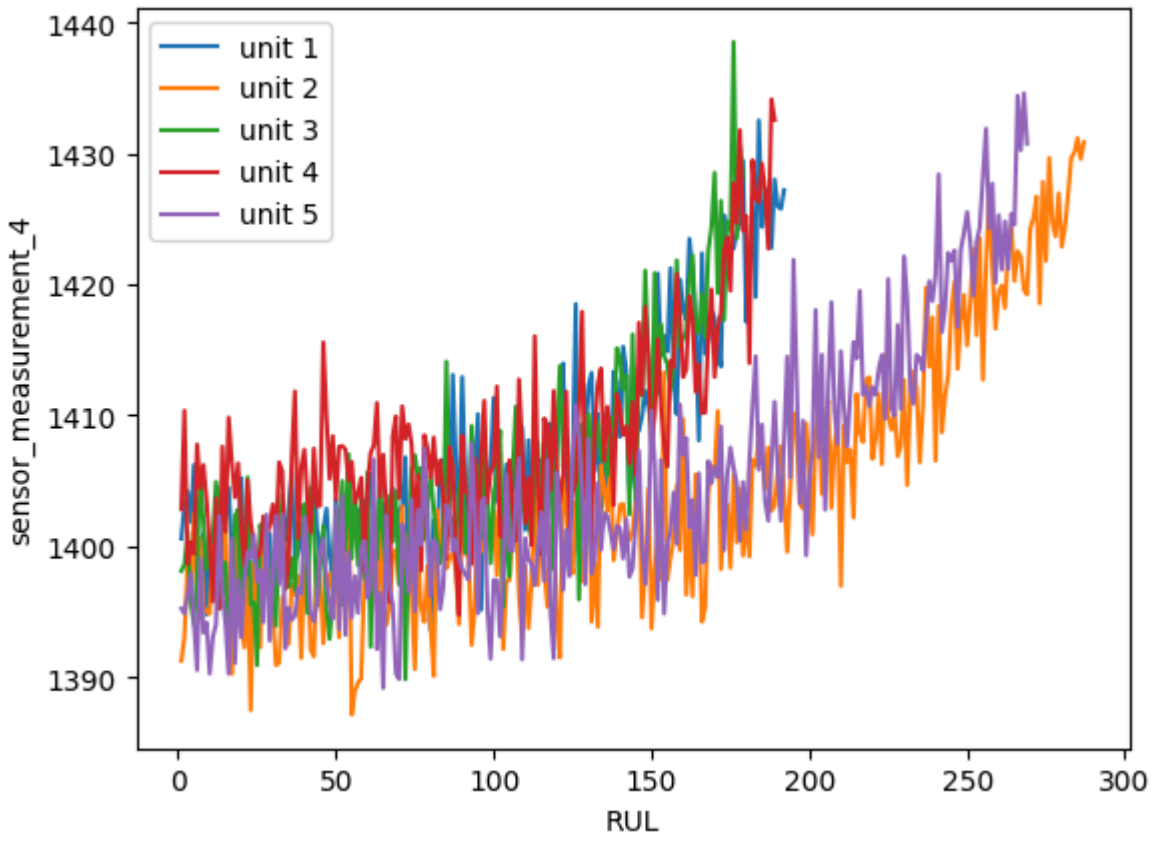
```
'operational_setting_1', 'operational_setting_2',
'operational_setting_3']
sensor_columns = ['sensor_measurement_{}'.format(i)
for i in range(1, 22)]
columns.extend(sensor_columns)
data.columns = columns data.head()
```

**Cálculo da Vida Útil remanescente (RUL) e Média Móvel**

Criamos uma função para calcular a vida útil remanescente (RUL) de cada unidade, e essa função foi aplicada aos dados:

```
def remaining_useful_life(data, column):
rul_data = data.copy()
time_variable_max = rul_data[column].max()
rul_data['RUL'] = time_variable_max -
rul_data[column] return rul_data['RUL'] units_ =
data['unit_number'].unique() subsets = [] for unit in
units_: subset = data[data['unit_number'] == unit]
subset['RUL'] = remaining_useful_life(subset,
'time_in_cycles') subsets.append(subset)
data_processed = pd.concat(subsets)
data_processed.dropna(inplace=True)
data_processed.reset_index(drop=True, inplace=True)
data_processed.head(10)
```

Além disso, geramos um gráfico para mostrar as medições dos sensores:



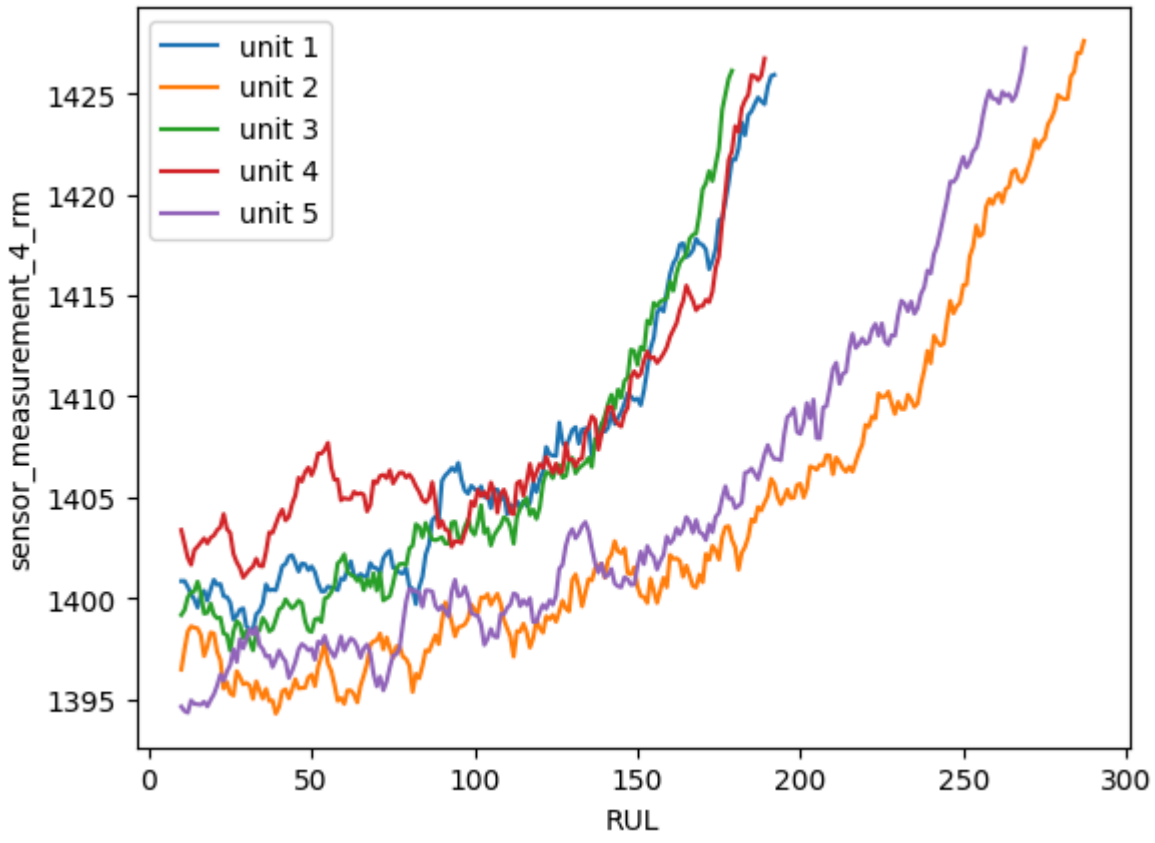

Para reduzir o ruído nos dados, foi calculada a média móvel das medições do sensores:

```
def rolling_mean(data, window_size: int = 5):
    rolling_mean_data = data.copy()
    for col in data.columns:
        rolling_mean_data[col] =
data[col].rolling(window=window_size).mean()
    rolling_mean_data.columns = [col + '_rm' for col
in data.columns]
    return rolling_mean_data

units_ = data['unit_number'].unique()
subsets = []
for unit in units_:
    subset = data[data['unit_number'] == unit]
    subset_units = subset['unit_number']
```

```
    subset_time = subset['time_in_cycles']
    subset_rul = subset['RUL']
    subset_rm =
rolling_mean(subset[subset.columns[2:-1]],
window_size=10)
    subset = pd.concat([subset_units, subset_time,
subset_rul, subset_rm], axis=1)
    subsets.append(subset)
data = pd.concat(subsets)
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
data.head(10)
```

Dessa forma, com a média móvel, o ruído dos dados foi reduzido conforme se observa na imagem abaixo.




Após realizar análise exploratória dos dados para entender melhor a

estrutura dos dados, foram reveladas as seguintes estatísticas descritivas sobre a vida útil remanescente dos motores:

° **Média de Ciclo Restantes:** Em média, os motores possuem 107 ciclos de vida útil restante. Isso indica que, em geral, os motores no conjunto de dados têm cerca de 107 ciclos antes de falhar.

° **Máximo de Ciclos Restantes:** A vida útil restante máxima observada no conjunto de dados é 361 ciclos. Esse valor representa o maior número de ciclos restantes registrado para um motor antes de falha.

° **Percentil 75:** 75% dos motores têm 155 ciclos de vida útil restante ou menos. Esse percentil indica que a maioria dos motores no conjunto de dados está prevista para falhar dentro de 155 ciclos.

° **Percentil 25:** 25% dos motores têm 25 ciclos de vida útil restante ou menos. Esse valor reflete que um quarto dos motores está próximo da falha, com apenas 25 ciclos restantes ou menos.

° **Distribuição da Vida Útil Restante:** A distribuição da vida útil remanescente apresenta uma assimetria (skewness) de 0.5, sugerindo uma leve inclinação à direita.

° **RUL Inferior a 200 Ciclos:** A maioria dos pontos de dados tem menos de 200 ciclos de vida útil restante.

° **Outliers:** Pontos de dados com RUL superior a 300 ciclos podem ser considerados outliers, indicando valores atípicos na distribuição.

Além disso, algumas medições dos sensores são constantes ao longo do tempo, indicando que esses sensores não variam e podem ser excluídos da análise:

- Sensor 01
- Sensor 05
- Sensor 06
- Sensor 10
- Sensor 16
- Sensor 18
- Sensor 19

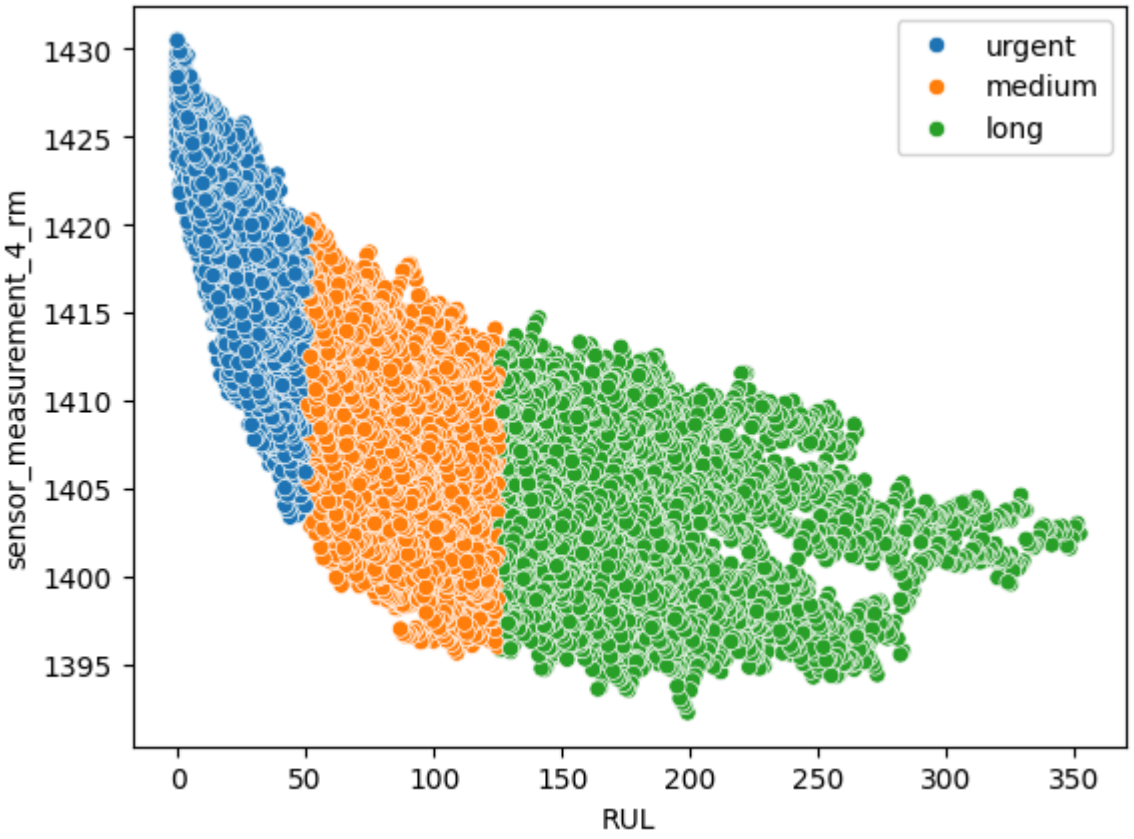
**Criação de Classificação Multiclasse**

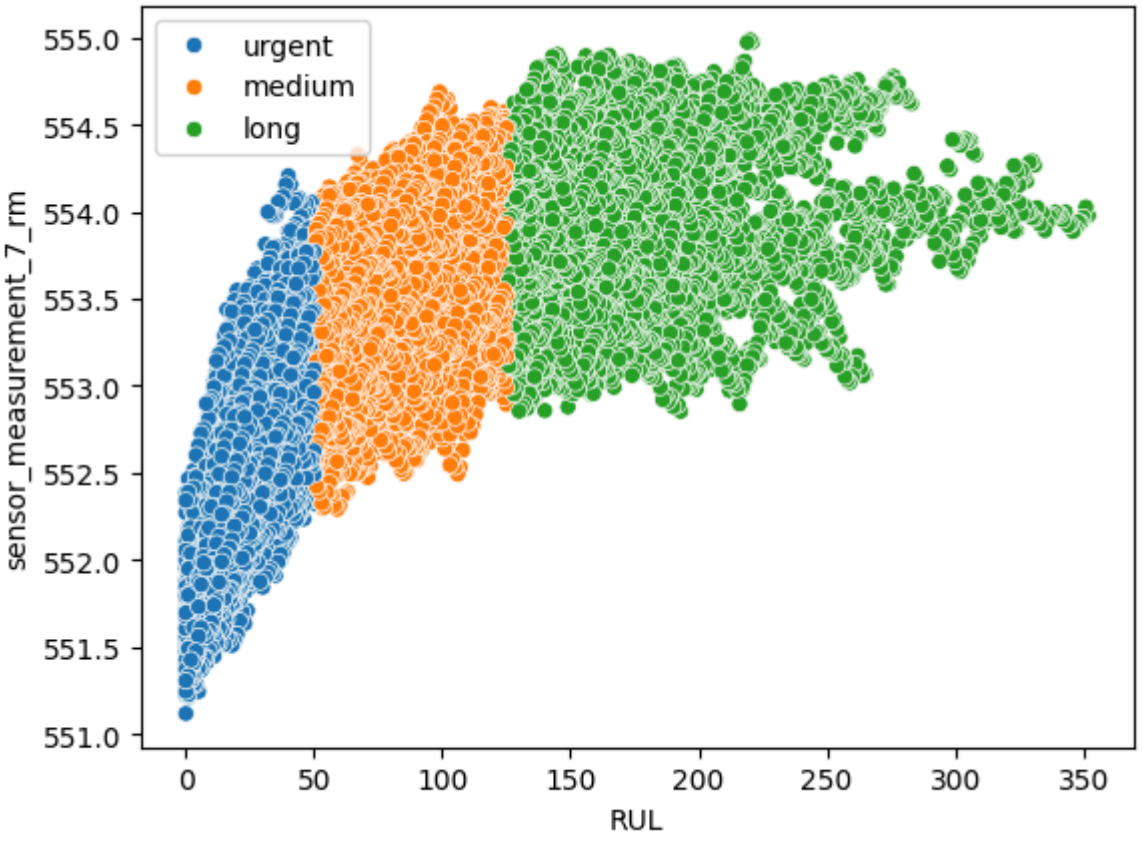
Os dados foram classificados em categorias de urgência de manutenção para facilitar a análise e a previsão:

```
def make_multiclass_classification(data, column,
thresholds, labels, categorical=False):
data_multiclass = data.copy() bins = [-float('inf')]
+ thresholds + [float('inf')] if categorical: labels
= list(labels.values()) else: labels =
list(labels.keys()) data_multiclass[column] =
pd.cut(data_multiclass[column], bins=bins,
labels=labels) return data_multiclass[column]
thresholds = [50, 125] labels = {0: 'urgent', 1:
'medium', 2: 'long'} data['maintanance_urgency'] =
make_multiclass_classification(data, 'RUL',
thresholds, labels, categorical=True)
data['maintanance_urgency'].value_counts(normalize=Tr
ue)
```

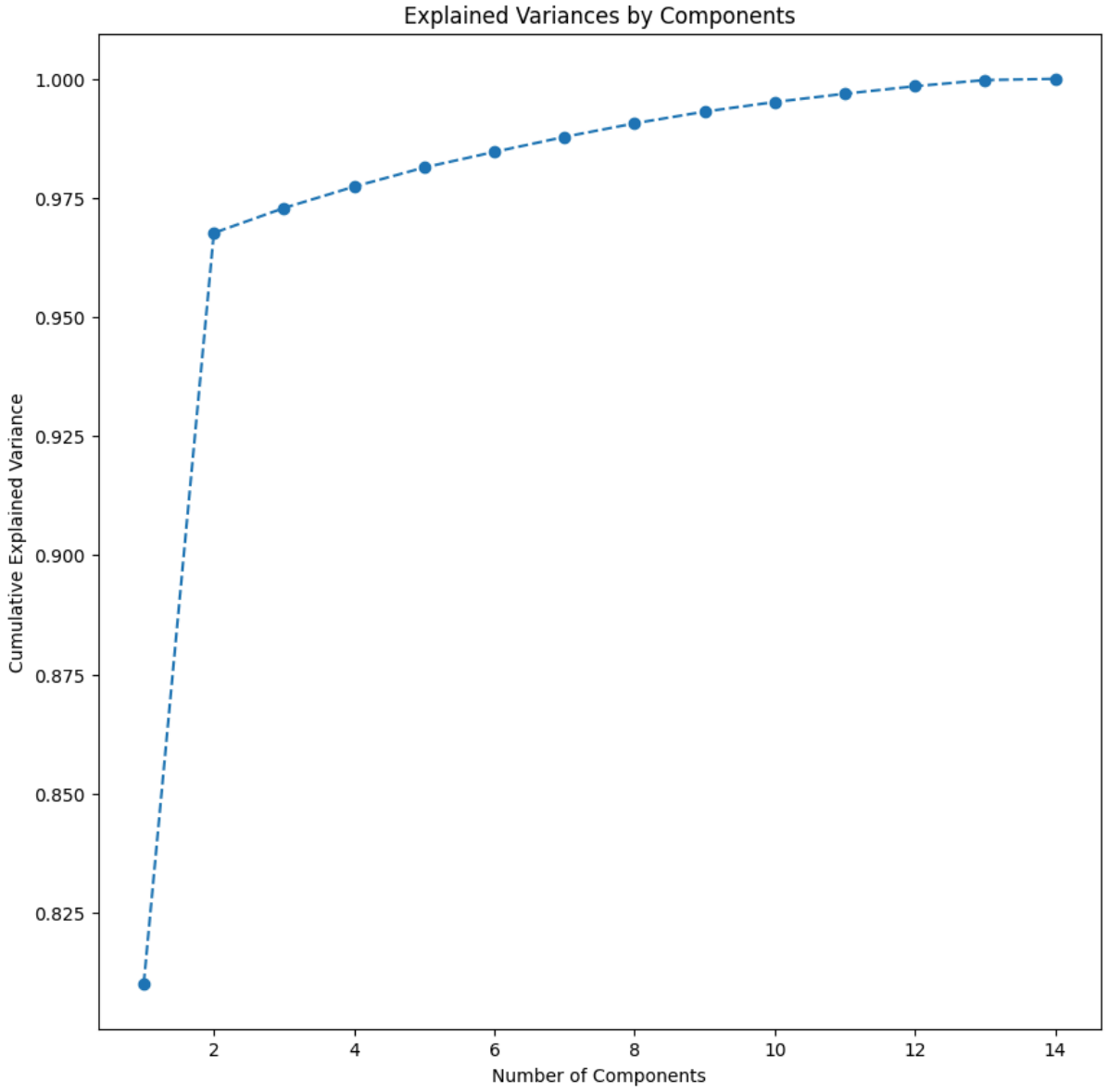
Assim, os rótulos foram definidos da seguinte forma:

**urgente (0)** - 50 ciclos ou menos de vida útil restante
**média (1)** - entre 50 e 125 ciclos de vida útil restante
**longa (2)** - 125 ciclos ou mais de vida útil restante

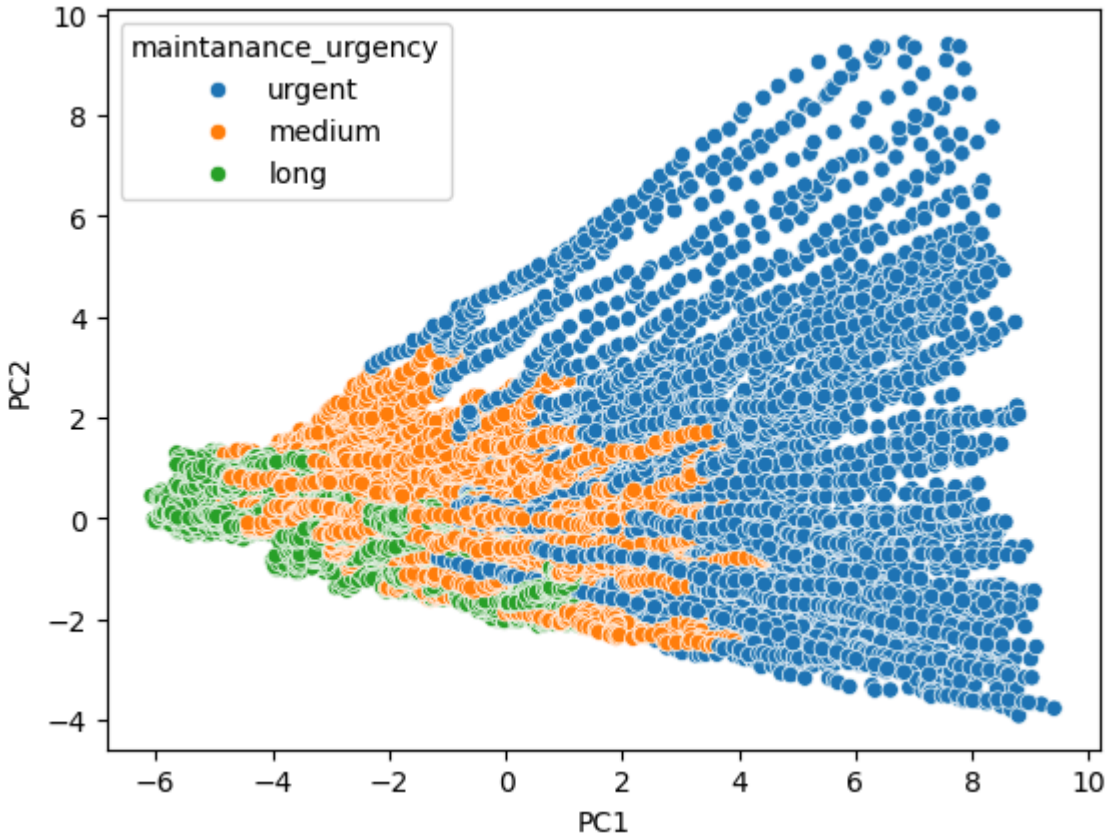

urgent
medium
long
sensor_measurement_4_rm
1430
1425
1420
1415
1410
1405
1400
1395
0
50
100
150
200
250
300
350
RUL


urgent
medium
long
sensor_measurement_7_rm
555.0
554.5
554.0
553.5
553.0
552.5
552.0
551.5
551.0
0
50
100
150
200
250
300
350
RUL

Além disso, fizemos análise do principal componente e conforme gráfico abaixo, apenas 2 componentes principais são capazes de explicar mais de 95% da variância do dataset.




Também observamos que os pontos de dados urgentes parecem mais dispersos, cobrindo uma parte maior do gráfico, embora componham a minoria dos dados, conforme imagem abaixo.




**Escolha do Modelo**

Para prever a urgência de manutenção, foram utilizados dois modelos de aprendizado de máquina: o K-Nearest Neighbors (KNN) e o Random Forest. A escolha desses modelos se deu pela sua capacidade de lidar com classificações multiclasse e pela facilidade de interpretação dos resultados.

Os dados foram divididos em conjuntos de treinamento e teste, e os dados de treinamento foram normalizados usando o StandardScaler:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Dividindo os dados em conjuntos de treinamento e
teste
```

```
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)

# Exibindo a forma dos conjuntos de treinamento e
teste
print(f"Shape of the training features:
{X_train.shape}")
print(f"Shape of the testing features:
{X_test.shape}")
print(f"Shape of the training target:
{y_train.shape}")
print(f"Shape of the testing target: {y_test.shape}")

# Normalizando os dados
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

O modelo KNN foi ajustado com os dados de treinamento escalonados e avaliado com os dados de teste:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Ajustando o modelo KNN
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_scaled, y_train)

# Prevendo as categorias
y_pred_knn = knn_model.predict(X_test_scaled)
```

```
# Relatório de classificação
print("KNN Classifier")
print(classification_report(y_test, y_pred_knn))
```

```
KNN Classifier
              precision    recall  f1-score   support

        long       0.96      0.97      0.96      1427
      medium       0.95      0.96      0.95      1522
      urgent       0.98      0.97      0.98       998

    accuracy                           0.96      3947
   macro avg       0.97      0.96      0.96      3947
weighted avg       0.96      0.96      0.96      3947
```

Já o modelo Random Forest foi ajustado com os dados de treinamento escalonados e avaliado com os dados de teste:

```
from sklearn.ensemble import RandomForestClassifier
```

```
# Ajustando o modelo Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_scaled, y_train)

# Prevendo as categorias
y_pred_rf = rf_model.predict(X_test_scaled)

# Relatório de classificação
print('Random Forest Classifier')
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier
              precision    recall  f1-score   support

        long       0.92      0.95      0.93      1427
      medium       0.89      0.90      0.90      1522
      urgent       0.97      0.90      0.93       998

    accuracy                           0.92      3947
   macro avg       0.92      0.92      0.92      3947
weighted avg       0.92      0.92      0.92      3947
```

**Conclusão**

A partir dessa análise decidimos utilizar o modelo de classificação KNN, tendo em vista que apresentou boas métricas de avaliação. Essas análises ajudam a prever a vida útil restante das unidades operacionais, permitindo a implementação eficaz de estratégias de manutenção preditiva.