part2

November 5, 2023

1 Desafio de Data Science da Tractian (Parte 2)

Nome: [Mauricio Barrios Castellanos]

E-mail: [mauricio.bc.89@gmail.com]

Data: [11/05/2023]

Descrição do Desafio:

Considerando os dados contidos nesses arquivos você deve completar as seguintes etapas:

1. Apresentar visualmente os dados contidos em cada arquivo, juntamente com as informações do ativo a que pertencem.
2. Desenvolver um modelo/função capaz de calcular o tempo de downtime e uptime para um ativo qualquer.
3. Desenvolver um modelo/função capaz de identificar mudanças nos padrões de vibração para um ativo qualquer.
4. Identificar possíveis falhas nos ativos utilizando o modelo desenvolvido no item 3 ou um novo modelo (a identificação deve ser autônoma e não uma análise visual).

1.1 Importação de Bibliotecas

- `numpy` (np): Matemática eficiente.
- `matplotlib.pyplot` (plt): Visualização de gráficos.
- `pandas` (pd): Manipulação de dados.
- `os` e `glob`: Lidam com arquivos e caminhos.
- `datetime` e `time`: Manipulação de datas e tempo.
- `seaborn`: Biblioteca para vizualição de dados

[1]:
```
# import libraries
# ================================================================
import numpy as np                      # mathematics
import matplotlib.pyplot as plt         # plot
import pandas as pd                     # pandas dataframe
import os                               # files sort
import glob                             # glob
import datetime                         # process timestamp
import time                             # time
import seaborn as sns
```

1.2 Importação de Dados

Nesta seção, se importaram dados para a análise . Os dados foram carregados em DataFrames do pandas `collects` e `assets`:

Além disso, realizou-se uma modificação no índice do DataFrame 'assets'. Isso envolveu a extração de informações relevantes da coluna 'sensors' em cada linha do DataFrame, nomeando-lo com o nome do sensor

[2]:
```
# import data
# ================================================================
collects_path = os.sep.join(['data', 'part2', 'collects.csv'])
assets_path = os.sep.join(['data', 'part2', 'assets.csv'])
collects = pd.read_csv(collects_path)
assets = pd.read_csv(assets_path)
assets.index = [assets['sensors'][i][2:-2] for i in assets.index]
```

1.3 Pré-Processamento de Dados

Nesta seção do notebook, é realizado o pré-processamento dos dados. Abaixo, são descritas as ações executadas:

- Remoção de Valores Nulos: Inicialmente, são removidas as linhas com valores NaN no DataFrame "collects" para garantir que os dados estejam completos e prontos para análise.
- Padronização das Direções dos Equipamentos: Alguns equipamentos possuem direções diferentes das habituais, como horizontal, vertical e axial. Para fins de análise, as direções são padronizadas.
- Identificação dos Modelos de Equipamentos: Os diferentes modelos de equipamentos disponíveis nos dados são identificados, e essa informação é armazenada na variável `model_names`.

[3]:
```
# data preprocessing
# ================================================================
# removing nan value ---------------------------------------------
collects = collects[~collects['params.accelRMS.x'].isna()]
collects.index = range(len(collects))
sensors = np.unique(collects['sensorId'])
# ----------------------------------------------------------------
'''some equipments the conventional horizontal, vertical axial
directions not apply. However convenience for analysis will
standardized.
 '''
sel_col = ['specifications.axisX', 'specifications.axisY',
  'specifications.axisZ']
assets.loc[:,sel_col] = assets.loc[:,sel_col].fillna('other')
assets = assets.fillna(0)
assets.loc[['MUR8453', 'MZU6388'],  'specifications.axisZ'] = 'axial'
assets.loc['MYS2071', 'specifications.axisY'] = 'vertical'
```

```
assets.loc['NAH4736', sel_col] = ['horizontal', 'vertical', 'axial']
assets.loc['NEW4797', sel_col] = ['horizontal', 'vertical', 'axial']
model_names =  np.unique(assets['modelType'])
assets.loc[:,sel_col]
```

[3]:

	specifications.axisX	specifications.axisY	specifications.axisZ
IAJ9206	vertical	horizontal	axial
LZY4270	axial	vertical	horizontal
MUR8453	horizontal	vertical	axial
MXK6435	horizontal	axial	vertical
MYD8706	horizontal	axial	vertical
MYS2071	axial	vertical	horizontal
MZU6388	horizontal	vertical	axial
NAH4736	horizontal	vertical	axial
NAI1549	axial	horizontal	vertical
NEW4797	horizontal	vertical	axial

1.4 Criação de um DataFrame para Análises

Nesta seção do notebook, um DataFrame denominado "df" é criado para posterior análise dos dados. Abaixo, são descritas as ações realizadas:

- As colunas "duration," "sampling_rate," "time_start," "sensorId," e "temp" são extraídas do DataFrame "collects" e adicionadas ao novo DataFrame "df.", sendo essas:
  - "duration": Esta coluna representa a duração das medições registradas e é extraída do DataFrame "collects."
  - "sampling_rate": A coluna "sampling_rate" indica a taxa de amostragem das medições, ou seja, com que frequência os dados são coletados pelo sensor.
  - "time_start": Esta coluna representa o momento em que as medições começam a ser registradas.
  - "sensorId": Cada sensor utilizado nas medições é identificado por um valor específico nesta coluna.
  - "temp": A coluna "temp" refere-se à temperatura registrada durante as medições.
- Para cada sensor na lista de sensores, informações adicionais são obtidas a partir do DataFrame "assets" e adicionadas ao DataFrame "df." Isso inclui as colunas:
  - "temp_max": Representa a temperatura máxima associada ao ativo.
  - "Downtime_max": Indica o tempo máximo de inatividade aceitável.
  - "power": Refere-se à potência relacionada ao ativo.
  - "rpm": Representa as rotações por minuto (RPM) associadas ao ativo.
  - "model_type": Identifica o tipo de maquina, o que é útil para clasificar elas pelo comportamento.
- O tipo de modelo é identificado e atribuído à coluna "model_type" com base na correspondência com os tipos de modelo previamente identificados.
- Os dados referentes às vibrações em velocidade e aceleração nas direções X, Y e Z de cada sensor são coletados e incorporados ao DataFrame "df". Para uma mlehor análise, as vibrações

são classificadas e categorizadas nas seguintes direções:

  - Horizontal (“accel_RMS_h” e “vel_RMS_h”).
  - Vertical (“accel_RMS_v” e “vel_RMS_v”).
  - Axial (“accel_RMS_a” e “vel_RMS_a”).

- As colunas “vel_RMS” e “accel_RMS” são calculadas com base nas medias das três direções e adicionadas ao DataFrame “df.”
- Por fim, o DataFrame “df” é ordenado com base nas colunas “sensorId” e “time_start,” e os índices são redefinidos para garantir uma organização adequada.

[4]:
```
# Create a dataframe for analyses
# ================================================================
df = pd.DataFrame()
df['duration'] = collects['params.duration']
df['sampling_rate'] = collects['params.sampRate']
df['time_start'] = collects['params.timeStart']
df['sensorId'] = collects['sensorId']
df['temp'] = collects['temp']
for sensor in sensors:
  dfi =  collects[collects['sensorId']==sensor]
  i = dfi.index
  df.loc[i, 'temp_max'] = assets.loc[sensor, 'specifications.maxTemp' ]
  df.loc[i, 'Downtime_max'] = assets.loc[sensor, 'specifications.maxDowntime' ]
  df.loc[i, 'power'] = assets.loc[sensor, 'specifications.power' ]
  df.loc[i, 'rpm'] = assets.loc[sensor, 'specifications.rpm' ]

  model_name = assets.loc[sensor, 'modelType' ]
  df.loc[i, 'model_type'] = np.where(model_name==model_names)[0][0]
  specs = ['X', 'Y', 'Z']
  directions = [assets.loc[sensor, ('specifications.axis' + j)][0]
    for j in specs]
  for d, s in zip(directions, specs):
    df.loc[i, ('accel_RMS_' + d)] = dfi['params.accelRMS.' + s.lower()]
    df.loc[i, ('vel_RMS_' + d)] = dfi['params.velRMS.' + s.lower()]
df['vel_RMS'] = df.loc[:,
  ['vel_RMS_' + d for d in directions]].sum(axis=1)/3
df['accel_RMS'] = df.loc[:,
  ['accel_RMS_' + d for d in directions]].sum(axis=1)/3
df = df.sort_values(by=['sensorId', 'time_start'])
df.index = range(len(df))
```

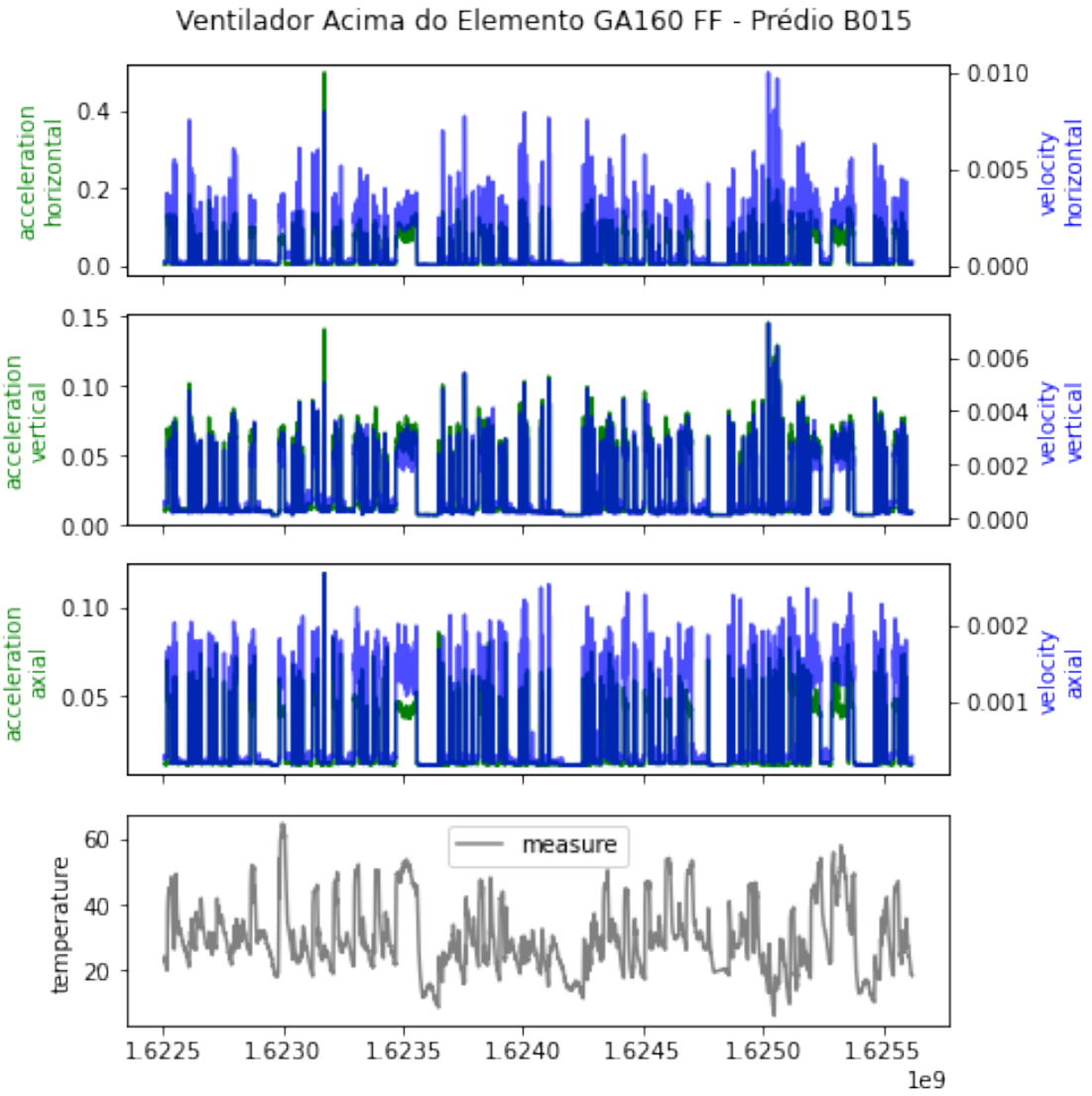
1.5 Presentação dos dados

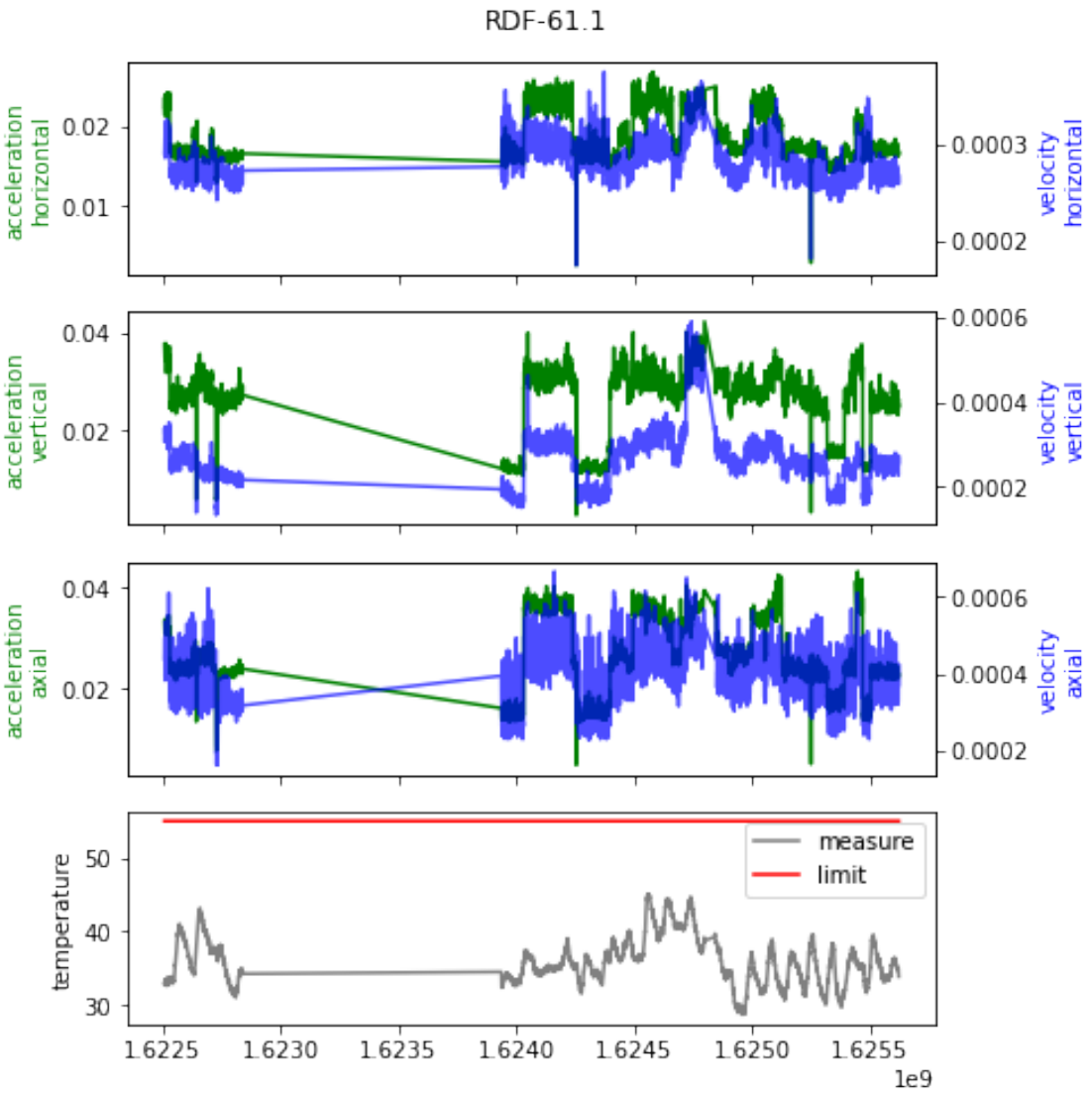
Nesta seção do notebook, os dados são apresentados visualmente para análise. Abaixo estão as ações realizadas:

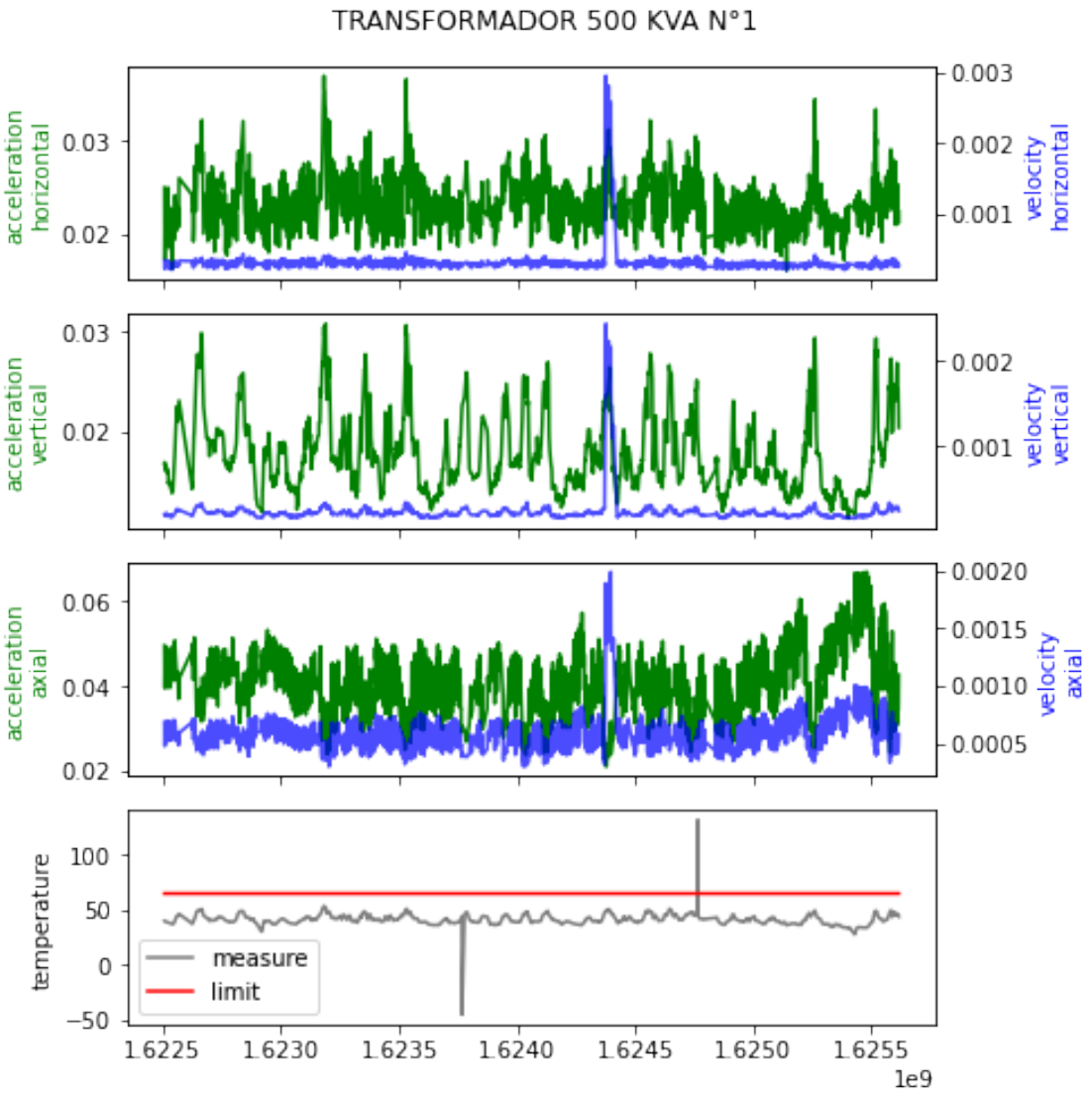
- Para cada sensor na lista de sensores, um gráfico é criado para representar as informações de vibração e temperatura ao longo do tempo.

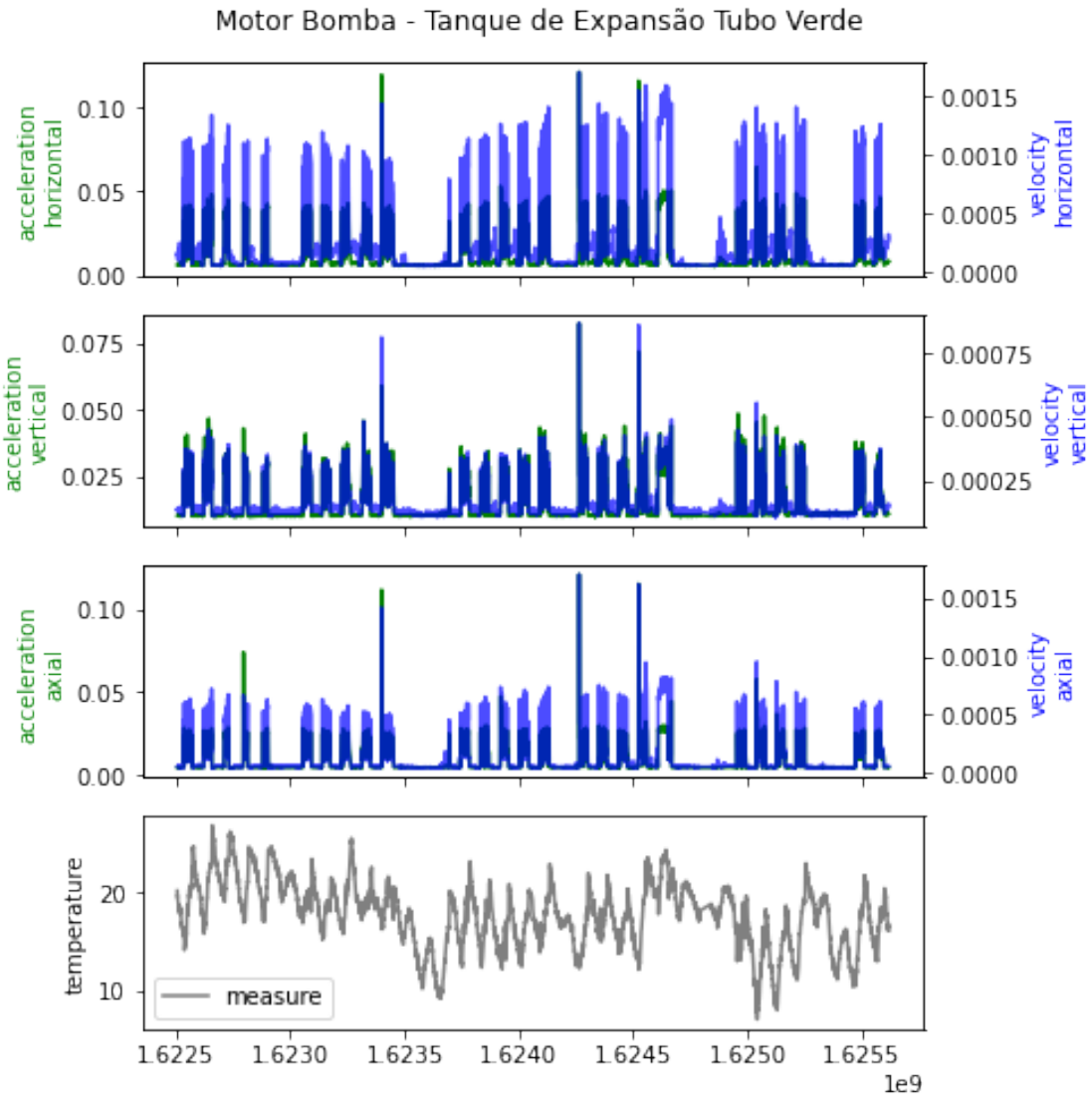
- Os gráficos são organizados em quatro subplots em um único conjunto de figuras, sendo que cada sensor possui seu próprio conjunto de subplots da seguinte maneira:
  - Subplot 0: Apresenta os dados de vibração de velocidade e aceleração na direção horizontal.
  - Subplot 1: Mostra os dados de vibração de velocidade e aceleração na direção vertical.
  - Subplot 2: Exibe os dados de vibração de velocidade e aceleração na direção axial.
  - Subplot 3: Representa os dados de temperatura, juntamente com seu limite, se aplicável.
- O gráfico mostra as medidas de aceleração (em verde) e velocidade (em azul) nas direções correspondentes ao sensor.
- A temperatura também é exibida sendo a medida real representada em cinza e, se aplicável, o limite máximo de temperatura (em vermelho).
- O título do gráfico é definido como o nome do ativo associado aos dados.
- O gráfico é compartilhado no eixo X para facilitar a comparação temporal.

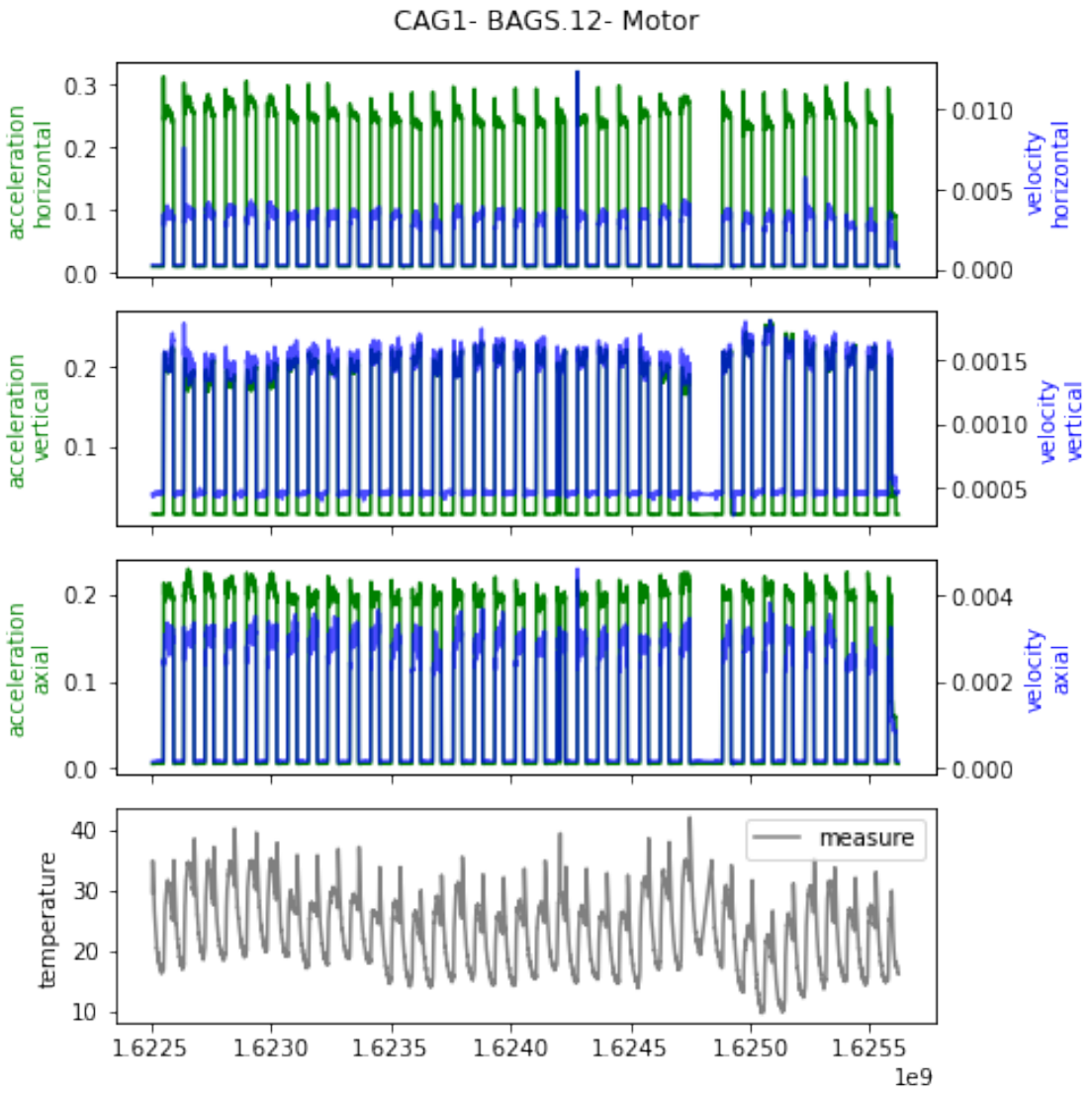
[5]:
```
# Present data
# ================================================================
for sensor in sensors:
  dfi = df[df['sensorId']==sensor]
  t =np.array(dfi['time_start'])
  lab1 = [ 'acceleration', 'velocity']
  lab2 = [ 'horizontal', 'vertical', 'axial']
  fig, axs = plt.subplots(4, 1, sharex=True, figsize=(7,7))
  ax2 = [0, 0, 0]
  fig.suptitle(assets['name'][sensor])
  for j, d in enumerate(directions):
    axs[j].plot(t, dfi['accel_RMS_' + d], color='g')
    ax2[j] = axs[j].twinx()
    ax2[j].plot(t, dfi['vel_RMS_'+ d], color='b', alpha=0.7)
    axs[j].set_ylabel(lab1[0] + '\n' + lab2[j], color='g')
    ax2[j].set_ylabel(lab1[1] + '\n' + lab2[j], color='b')
  axs[3].plot(t, dfi['temp'], color='gray', label='measure')
  if np.mean(dfi.temp_max)>0:
    axs[3].plot(t, dfi['temp_max'], label='limit', color='r')
  axs[3].set_ylabel('temperature')
  axs[3].legend()
  fig.tight_layout()
```

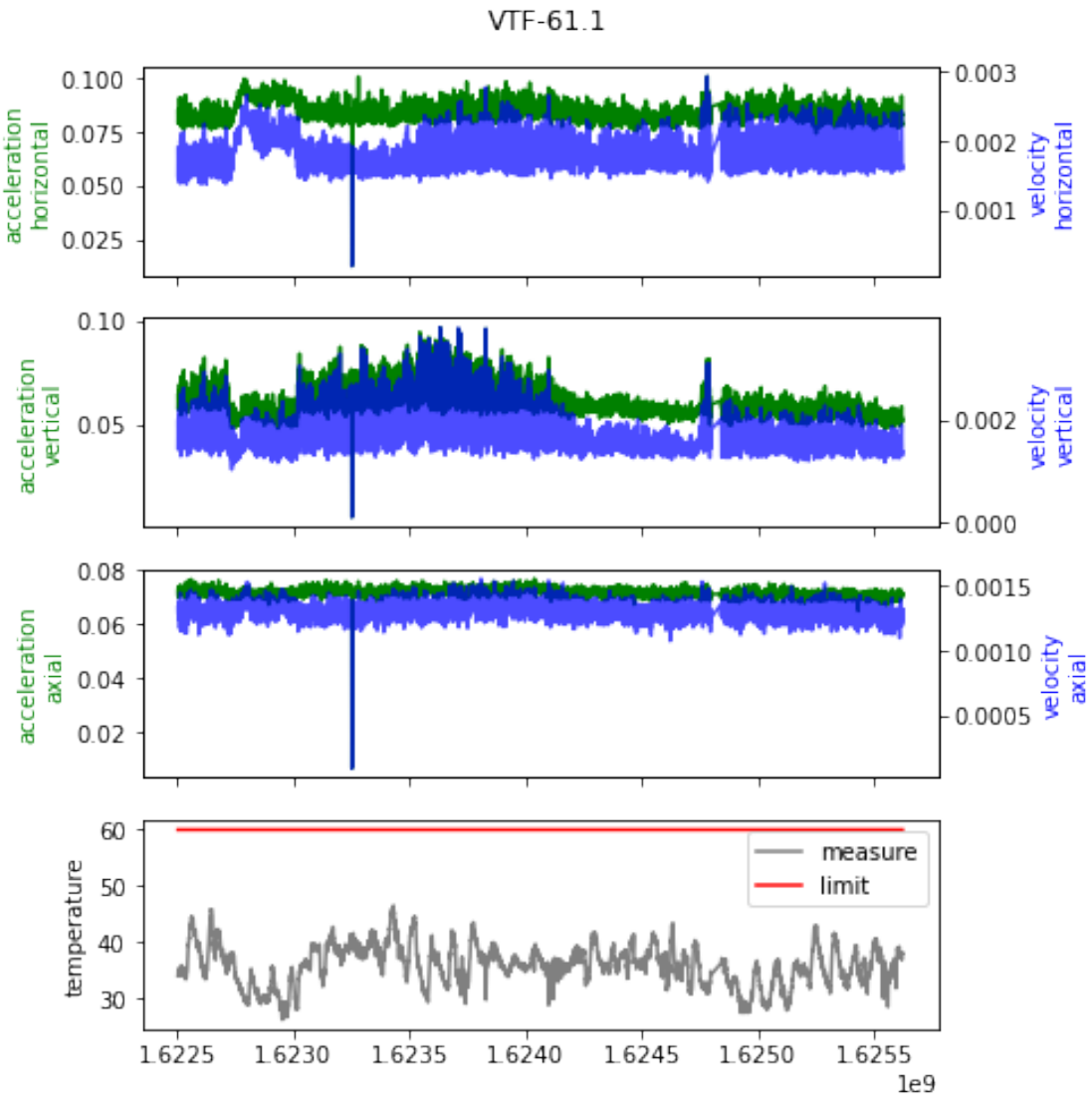

Ventilador Acima do Elemento GA160 FF - Prédio B015
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
1e9

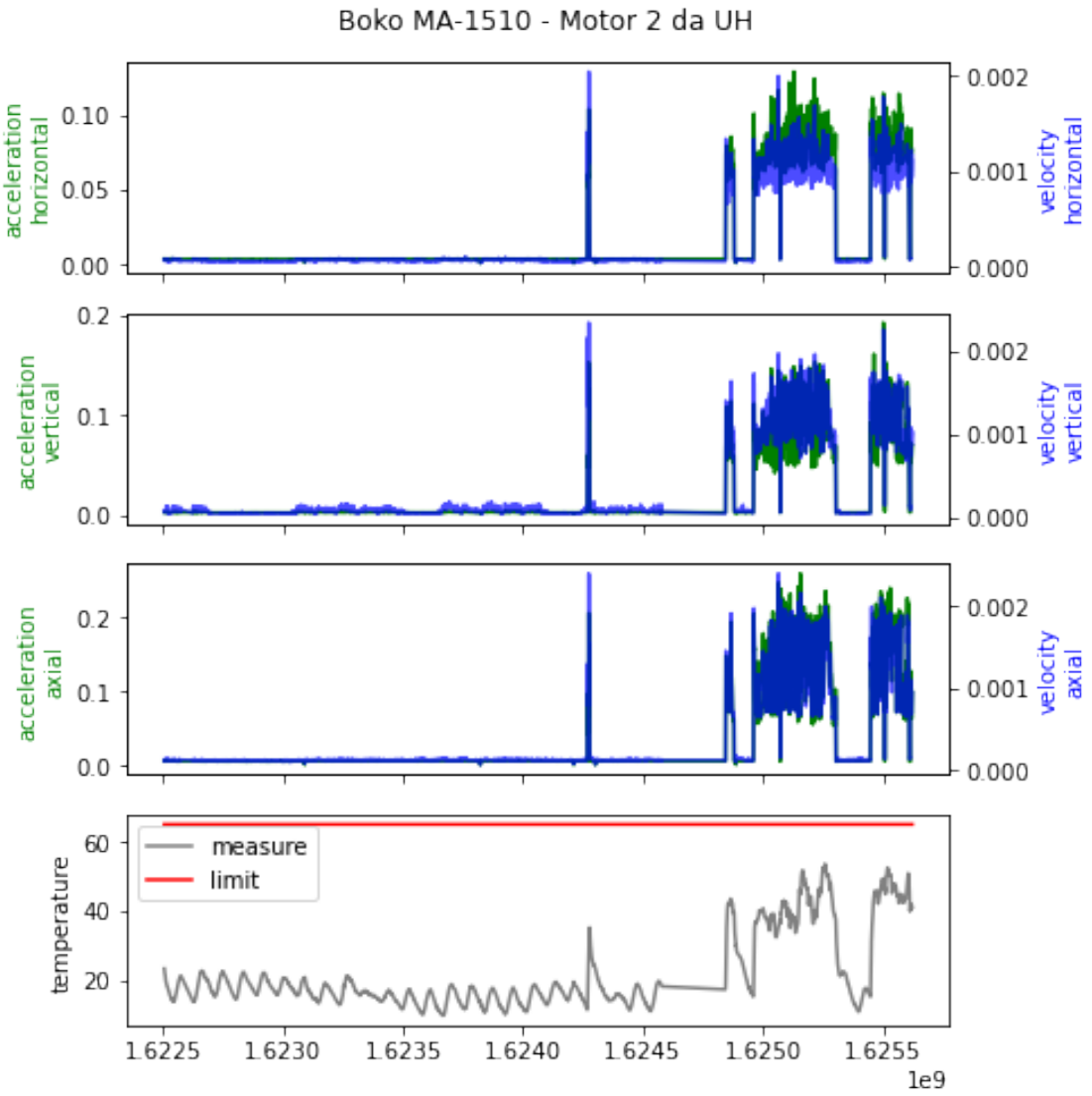

RDF-61.1
acceleration horizontal
0.02
0.01
velocity horizontal
0.0003
0.0002
acceleration vertical
0.04
0.02
velocity vertical
0.0006
0.0004
0.0002
acceleration axial
0.04
0.02
velocity axial
0.0006
0.0004
0.0002
temperature
50
40
30
measure
limit
1.6225
1.6230
1.6235
1.6240
1.6245
1.6250
1.6255
1e9

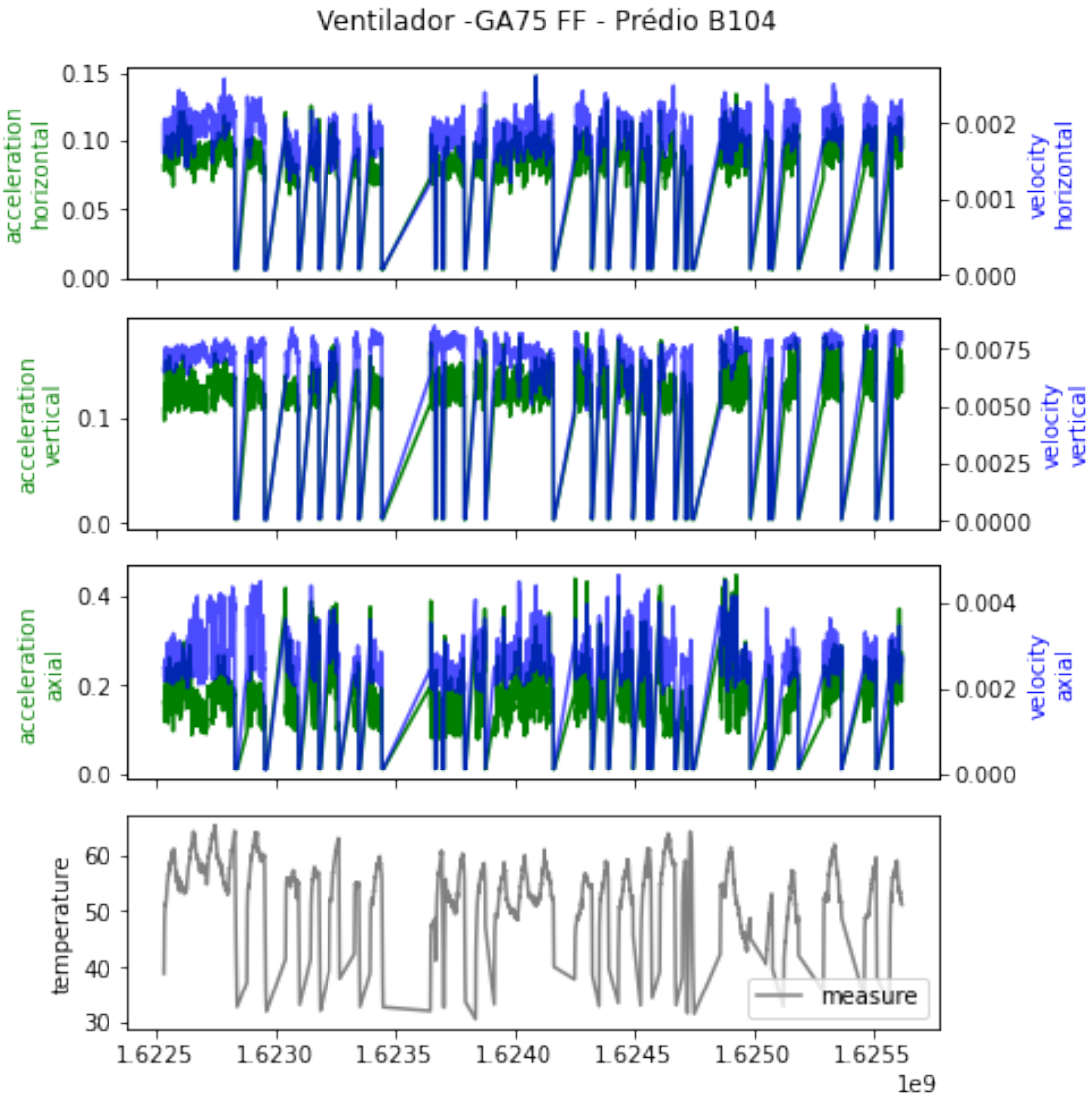

TRANSFORMADOR 500 KVA N°1
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
limit
1e9

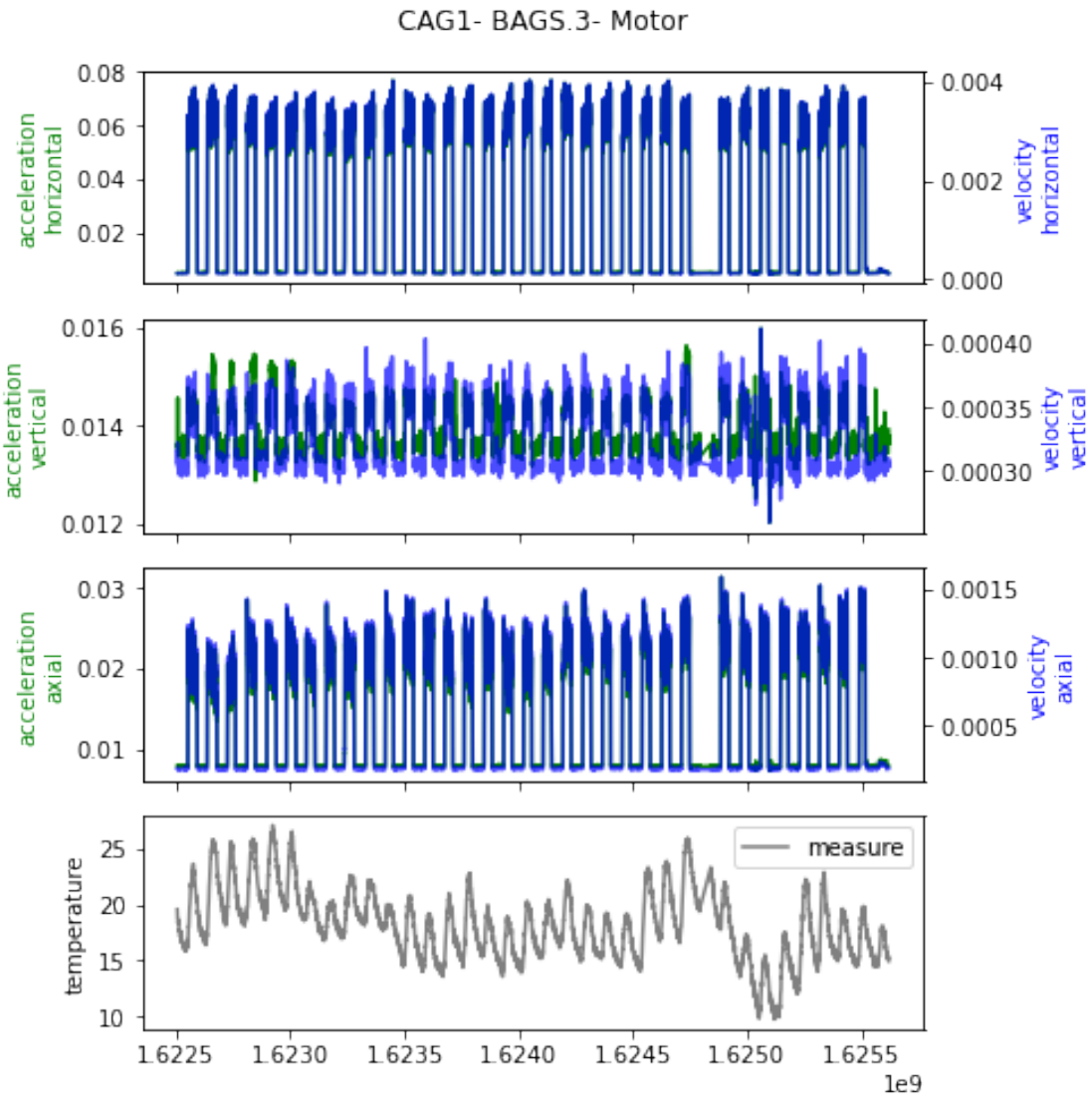

Motor Bomba - Tanque de Expansão Tubo Verde
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
1e9

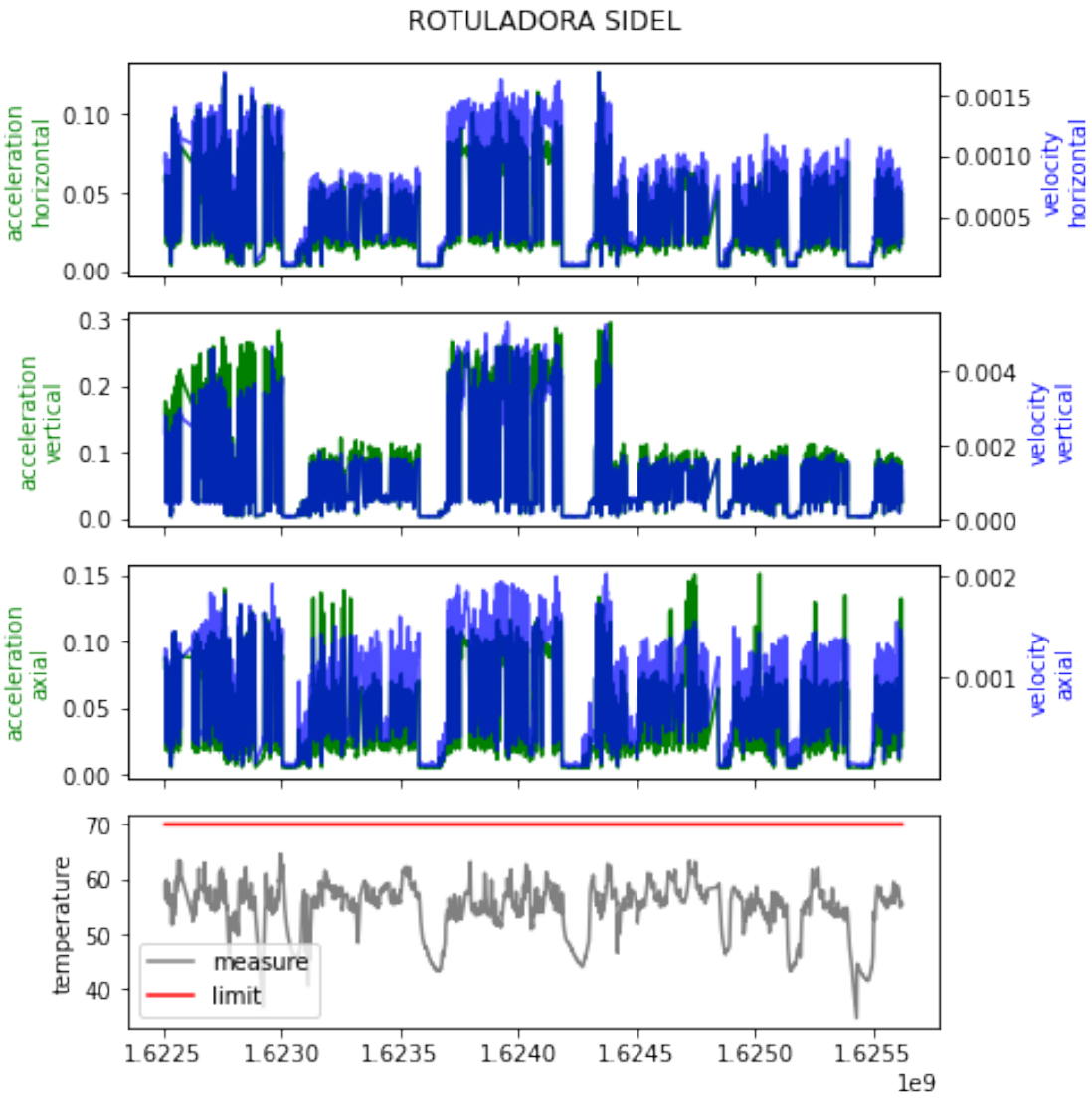

CAG1- BAGS.12- Motor
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
1e9


VTF-61.1
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
limit
1e9


Boko MA-1510 - Motor 2 da UH
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
limit
1e9


Ventilador -GA75 FF - Prédio B104
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
1e9


CAG1- BAGS.3- Motor
acceleration horizontal
velocity horizontal
acceleration vertical
velocity vertical
acceleration axial
velocity axial
temperature
measure
1e9



1.6 Modelo para calcular o tempo de downtime e uptime.

1.6.1 Rotulagem dos Dados

Os dados são rotulados, onde o valor '0' é atribuído ao Tempo de Inatividade e o valor '1' ao Tempo de Atividade. O processo de rotulagem é realizado para cada sensor (ativo) na lista de sensores, seguindo os seguintes passos:

1. Primeiramente, os dados específicos desse sensor são selecionados e copiados em um DataFrame separado denominado 'dfi.'
2. O nome do sensor e o tipo de modelo associado são obtidos para fins de identificação.
3. O tempo decorrido entre as medições é calculado e adicionado como uma nova coluna denominada 'dt' ao DataFrame 'dfi.'

4. Os dados são rotulados como ‘0’ (Tempo de Inatividade) ou ‘1’ (Tempo de Atividade), com base em um limite de vibração. Esse limite varia de acordo com o ativo.

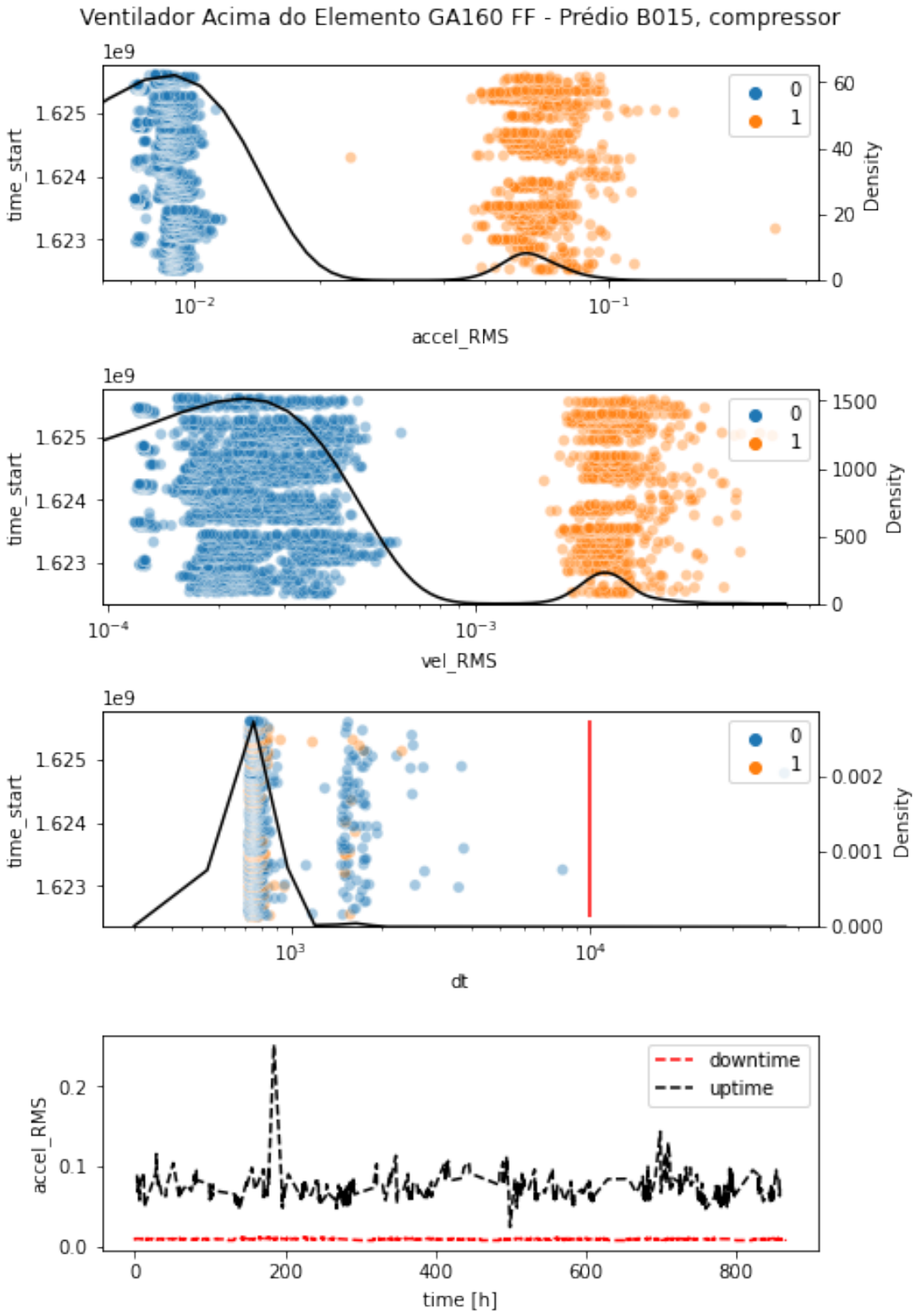
5. Os rótulos ‘0’ ou ‘1’ são atribuídos à coluna ‘class’ no DataFrame ‘dfi.’

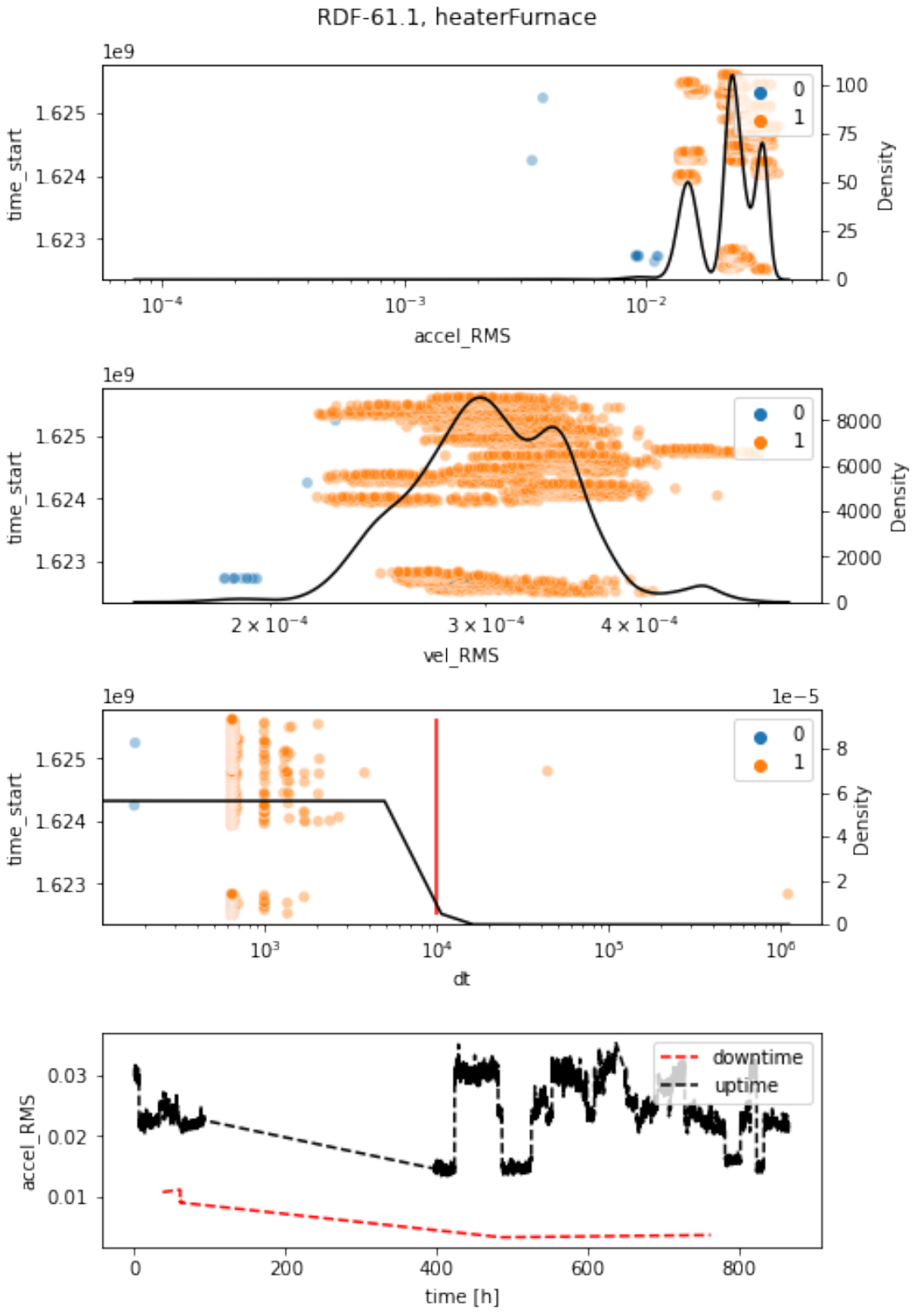
No gráfico, a representação dos dados é organizada em quatro subplots e segue o padrão a seguir:

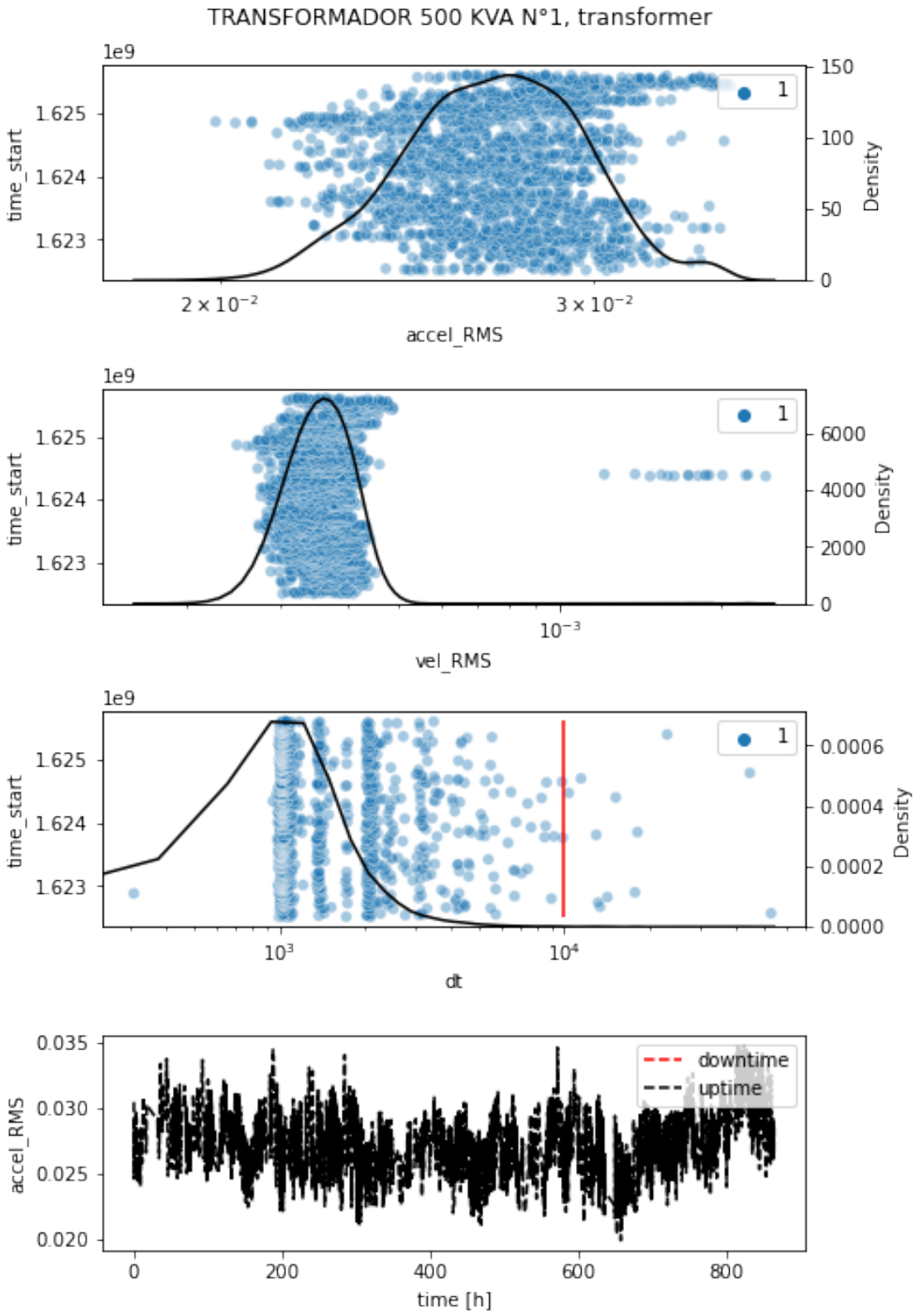
- O título do gráfico é definido com base no nome e tipo de modelo do sensor.
- Os dois primeiros subplots (0 a 1) apresentam informações relacionadas à aceleração, velocidade e tempo da medição.
  - Os pontos no gráfico representam os dados rotulados como ‘Tempo de Inatividade’ (azul) e ‘Tempo de Atividade’ (laranja) no eixo esquerdo.
  - As curvas KDE (Kernel Density Estimation) em preto representam a distribuição dos dados no eixo direito.
  - Os eixos horizontais têm escala logarítmica.
- O último subplot (3) exibe a vibração de aceleração em relação ao tempo para ambas as classificações. As linhas pontilhadas representam a aceleração durante o ‘Tempo de Inatividade’ (vermelho) e ‘Tempo de Atividade’ (preto) ao longo do tempo.”

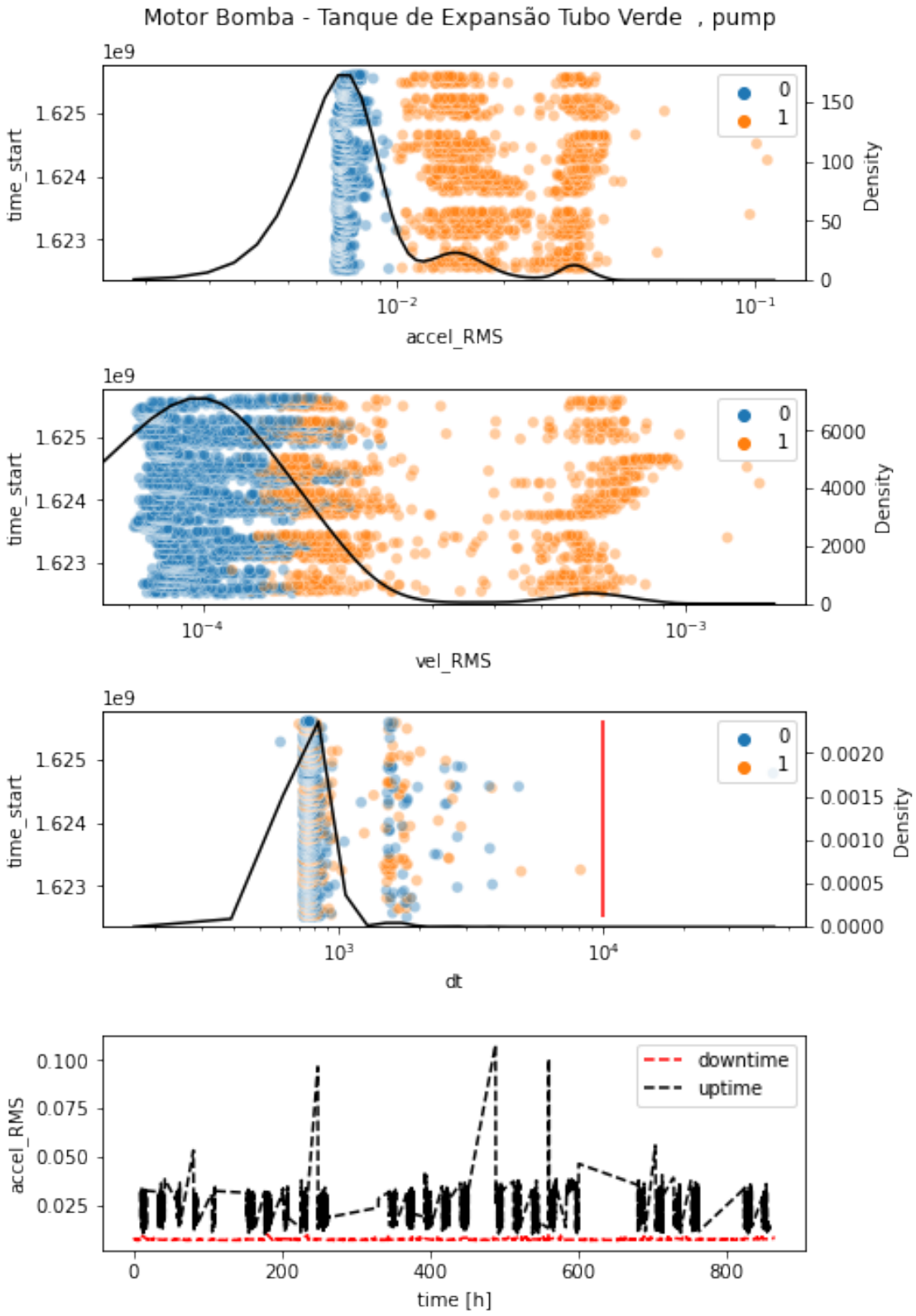
[6]:
```
# labeling data 0 for downtime and 1 for uptime
# ================================================================
for j, sensor in enumerate(sensors):
  # define data of each sensor ----------------------------------
  dfi = df[df['sensorId']==sensor].copy()
  index = dfi.index
  title = '%s, %s'%(
    assets['name'][sensor], assets['modelType'][sensor])
  dtime = np.array(dfi.time_start[1:]
    ) - np.array(dfi.time_start[:-1])
  dfi['dt'] = np.r_[dtime, dtime[-1]]

  # labeling the data of corresponding sensor threshold  ----------
  th_values = {1: 1.2e-2, 2: 1.9e-2, 3: 1e-2, 9: 3e-2}
  th = th_values.get(j, 2.1e-2) # threshold

  vib_acel = np.array(dfi['accel_RMS'])
  dfi['class']           = 0*(vib_acel<th) + 1*(vib_acel>th)
  df.loc[index, 'class'] = 0*(vib_acel<th) + 1*(vib_acel>th)

  # plot the view of classification ------------------------------
  fig, axs = plt.subplots(4, 1, figsize=(7, 10))
  ax_0 = list(axs)
  ax_1 = [0, 0, 0]
  fig.suptitle(title)

  # axis 0 a 2
  labels = ['accel_RMS', 'vel_RMS', 'dt']
  lab = ['downtime', 'uptime']
  for i in range(3):
```

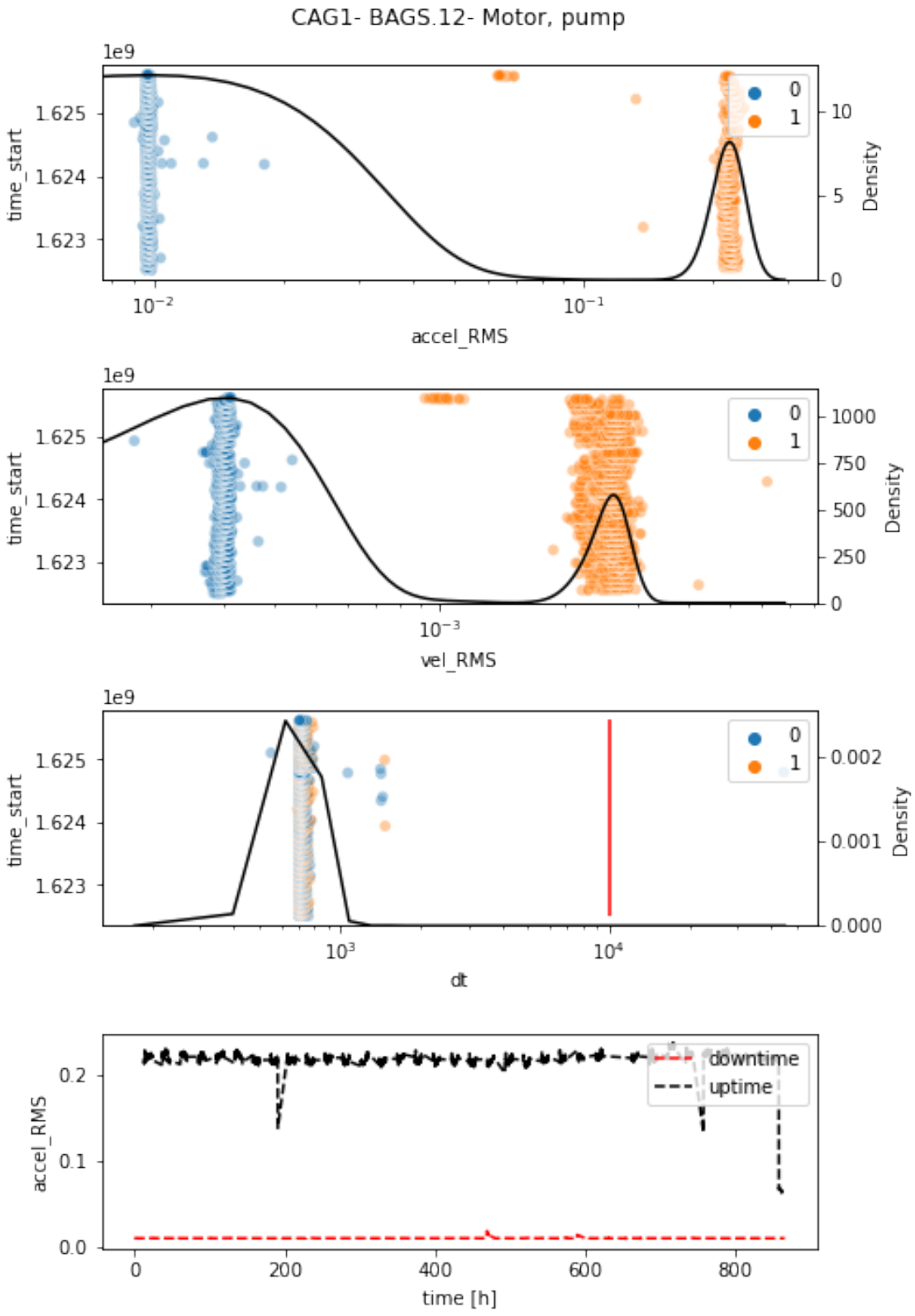
```
  ax_1[i] = ax_0[i].twinx()
  sns.scatterplot(dfi, alpha=0.4,
    x=labels[i], y='time_start', hue='class', ax =ax_0[i])
  sns.kdeplot(dfi[labels[i]], ax =ax_1[i], color='k')
  #if i<2:
  ax_1[i].set_xscale('log')
  ax_0[i].legend(loc='upper right')
ax_0[2].vlines(1e4,dfi['time_start'].min(),
  dfi['time_start'].max(), color='r')

# axis 3
for Class, color  in enumerate(['r', 'k']):
  t = dfi.time_start
  t = t[dfi['class']==Class] - t.min()
  ax_0[3].plot(t/3600, dfi.accel_RMS[dfi['class']==Class],'--',
    color=color,  label=lab[Class])
ax_0[3].set_xlabel('time [h]')
ax_0[3].set_ylabel('accel_RMS')
ax_0[3].legend(loc='upper right')
fig.tight_layout()
```

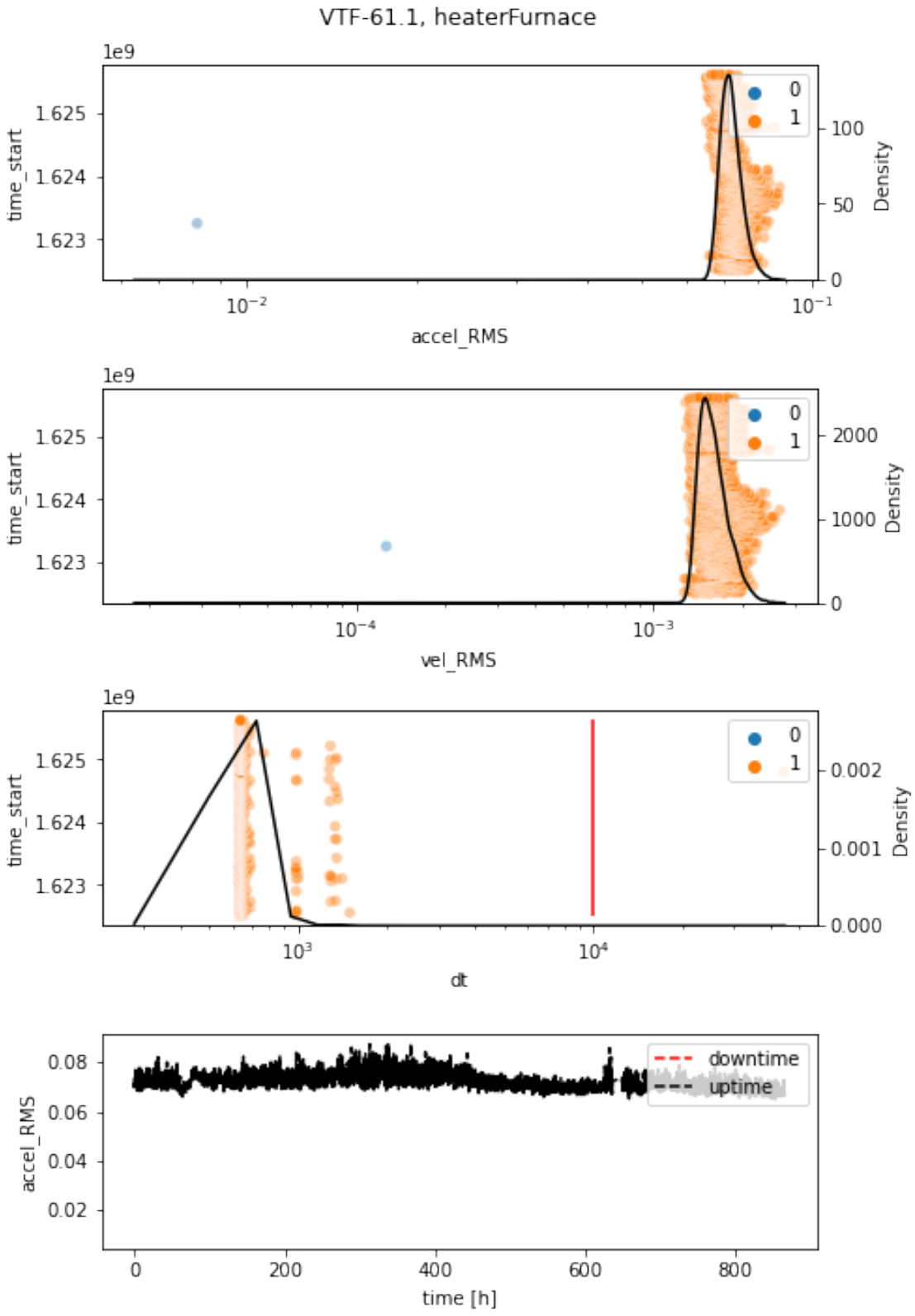

Ventilador Acima do Elemento GA160 FF - Prédio B015, compressor
1e9
time_start
1.625
1.624
1.623
Density
0
1
accel_RMS
vel_RMS
dt
downtime
uptime
accel_RMS
time [h]

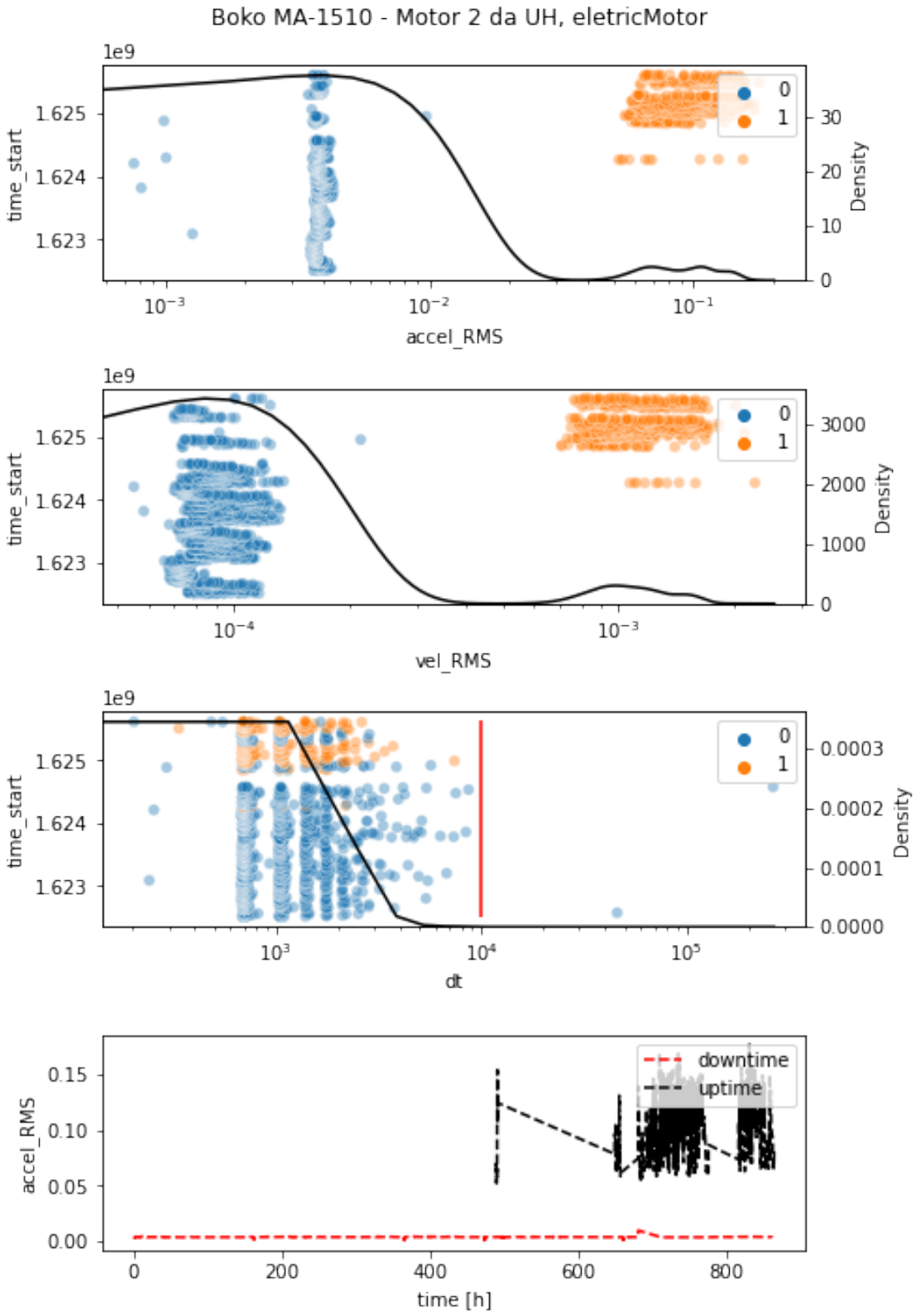

RDF-61.1, heaterFurnace
1e9
time_start
1.625
1.624
1.623
accel_RMS
Density
0
1
vel_RMS
dt
1e−5
downtime
uptime
accel_RMS
time [h]

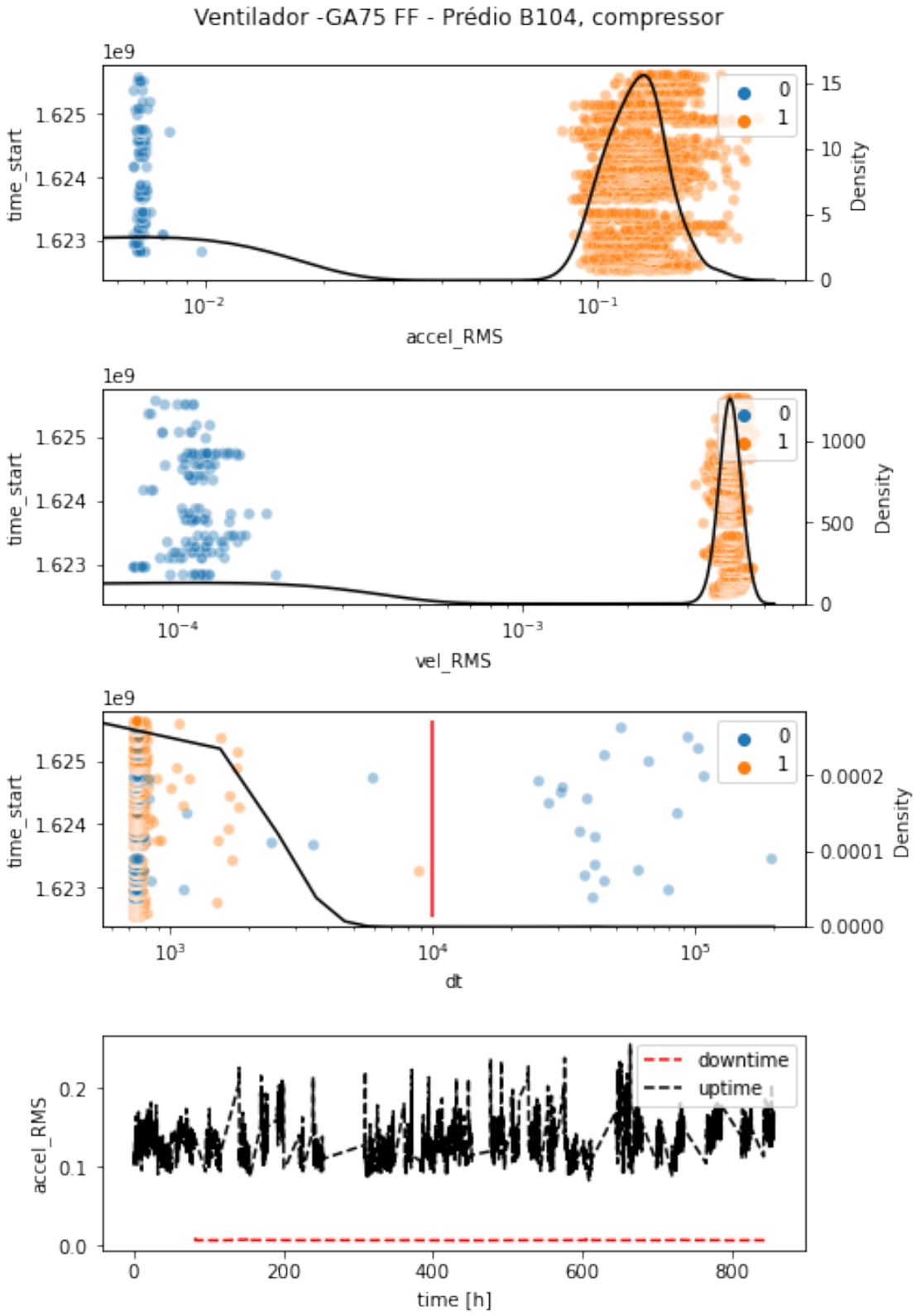

TRANSFORMADOR 500 KVA N°1, transformer
1e9
time_start
1.625
1.624
1.623
150
100
50
0
Density
1
2×10^{-2}
3×10^{-2}
accel_RMS
1e9
time_start
1.625
1.624
1.623
6000
4000
2000
0
Density
1
10^{-3}
vel_RMS
1e9
time_start
1.625
1.624
1.623
0.0006
0.0004
0.0002
0.0000
Density
1
10^{3}
10^{4}
dt
accel_RMS
0.035
0.030
0.025
0.020
downtime
uptime
0
200
400
600
800
time [h]

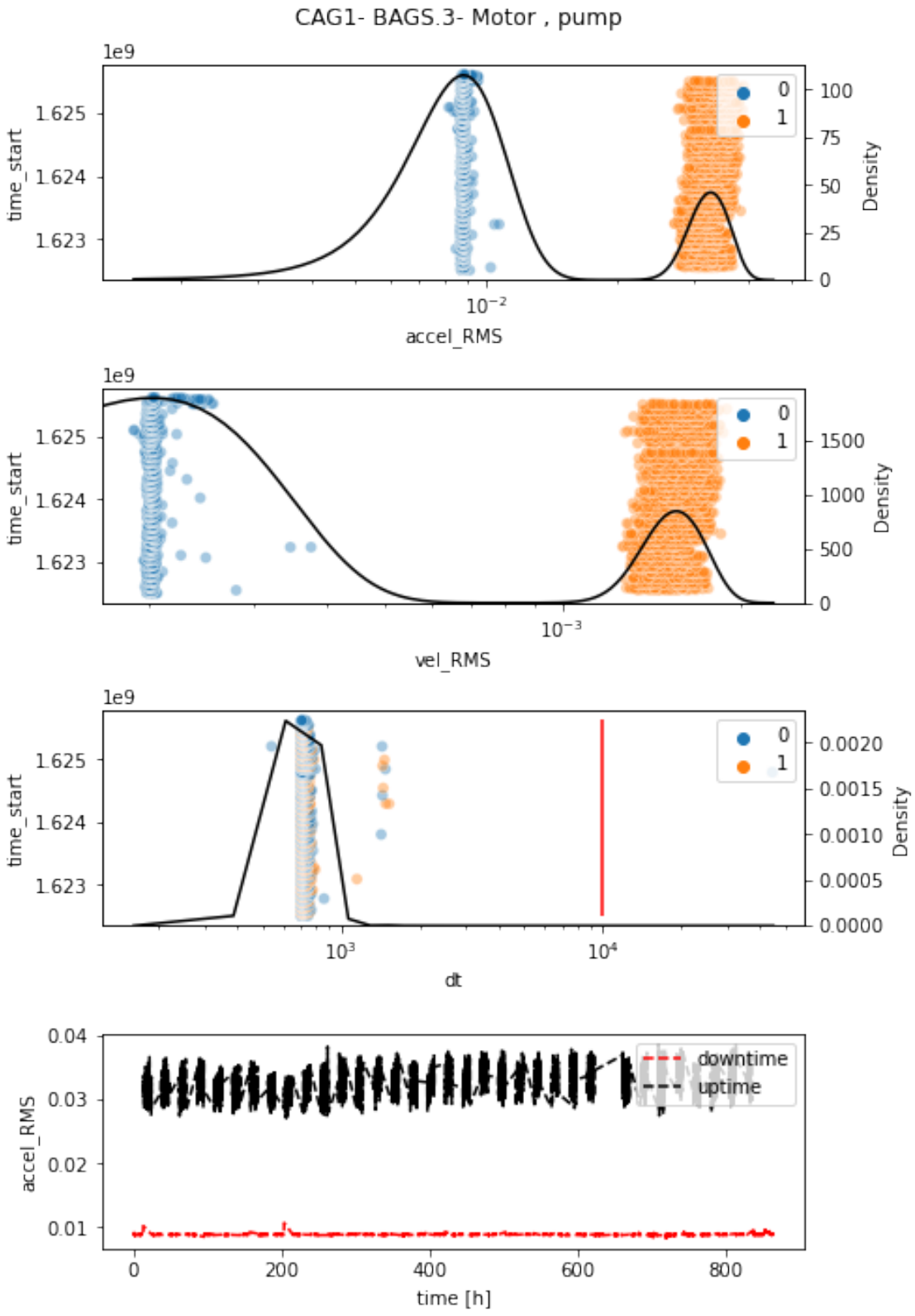

Motor Bomba - Tanque de Expansão Tubo Verde , pump
1e9
time_start
1.625
1.624
1.623
0
1
150
100
50
0
Density
10^{-2}
10^{-1}
accel_RMS
1e9
time_start
1.625
1.624
1.623
0
1
6000
4000
2000
0
Density
10^{-4}
10^{-3}
vel_RMS
1e9
time_start
1.625
1.624
1.623
0
1
0.0020
0.0015
0.0010
0.0005
0.0000
Density
10^{3}
10^{4}
dt
accel_RMS
0.100
0.075
0.050
0.025
downtime
uptime
0
200
400
600
800
time [h]

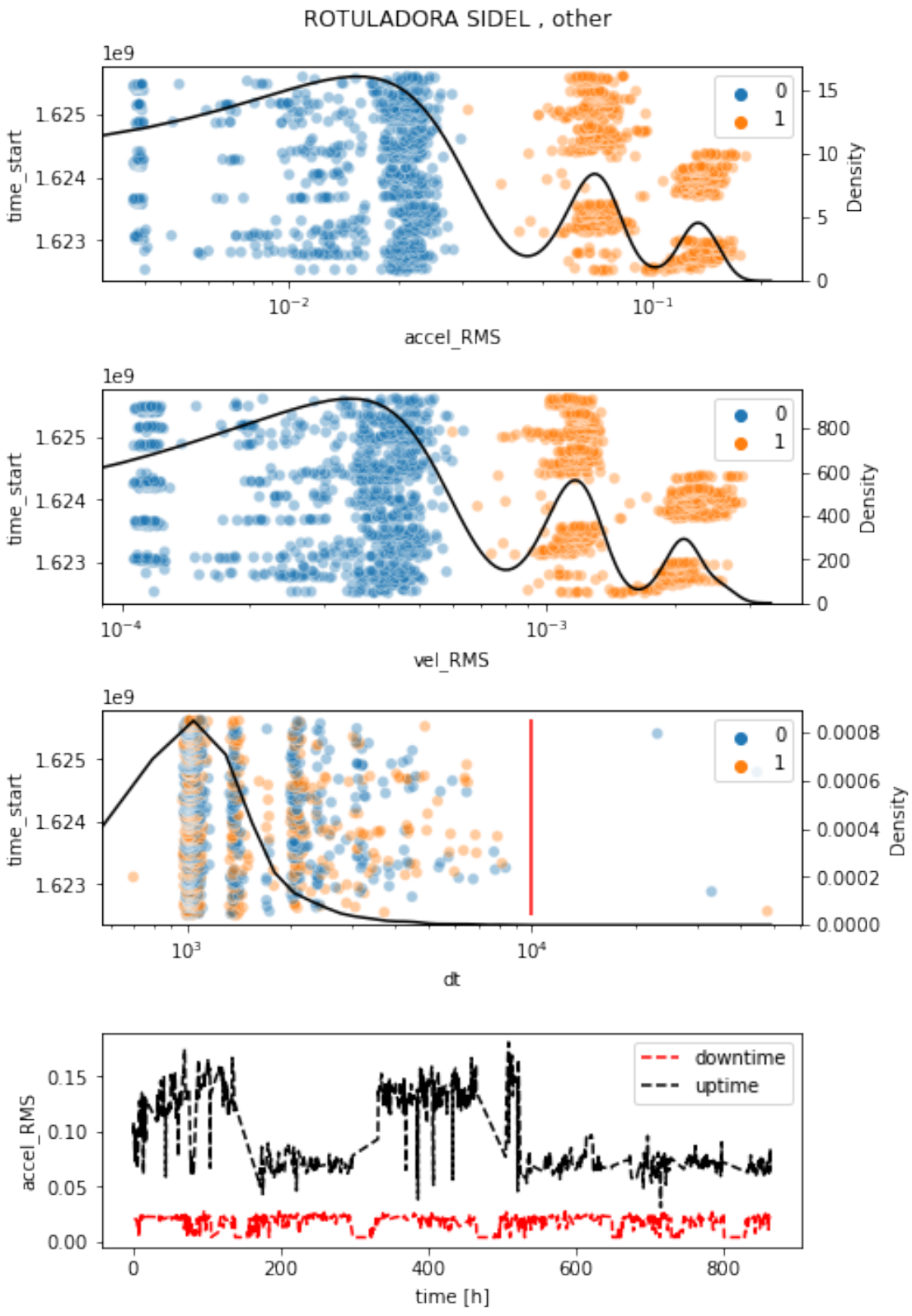

CAG1- BAGS.12- Motor, pump
1e9
time_start
1.625
1.624
1.623
10^{-2}
10^{-1}
accel_RMS
0
1
Density
10
5
0
1e9
time_start
1.625
1.624
1.623
10^{-3}
vel_RMS
0
1
Density
1000
750
500
250
0
1e9
time_start
1.625
1.624
1.623
10^{3}
10^{4}
dt
0
1
Density
0.002
0.001
0.000
accel_RMS
0.2
0.1
0.0
0
200
400
600
800
time [h]
downtime
uptime


VTF-61.1, heaterFurnace
1e9
time_start
1.625
1.624
1.623
0
1
100
50
0
Density
10^{-2}
10^{-1}
accel_RMS
1e9
time_start
1.625
1.624
1.623
0
1
2000
1000
0
Density
10^{-4}
10^{-3}
vel_RMS
1e9
time_start
1.625
1.624
1.623
0
1
0.002
0.001
0.000
Density
10^{3}
10^{4}
dt
accel_RMS
0.08
0.06
0.04
0.02
downtime
uptime
0
200
400
600
800
time [h]


Boko MA-1510 - Motor 2 da UH, eletricMotor
1e9
time_start
1.625
1.624
1.623
Density
0
1
accel_RMS
vel_RMS
dt
accel_RMS
downtime
uptime
time [h]


Ventilador -GA75 FF - Prédio B104, compressor
1e9
time_start
1.625
1.624
1.623
0
1
Density
accel_RMS
vel_RMS
dt
downtime
uptime
accel_RMS
time [h]


CAG1- BAGS.3- Motor , pump
1e9
time_start
1.625
1.624
1.623
10^{-2}
accel_RMS
Density
0
1
100
75
50
25
0
1e9
time_start
1.625
1.624
1.623
10^{-3}
vel_RMS
Density
0
1
1500
1000
500
0
1e9
time_start
1.625
1.624
1.623
10^{3}
10^{4}
dt
Density
0
1
0.0020
0.0015
0.0010
0.0005
0.0000
accel_RMS
0.04
0.03
0.02
0.01
0
200
400
600
800
time [h]
downtime
uptime


ROTULADORA SIDEL , other
1e9
time_start
1.625
1.624
1.623
0
1
Density
accel_RMS
vel_RMS
dt
downtime
uptime
accel_RMS
time [h]

1.6.2 Árvore de Decisão para Classificação de 'downtime' e 'uptime' de Ativos

Nesta seção do notebook, é criada uma árvore de decisão para classificar os ativos em duas categorias: "Tempo de Inatividade" e "Tempo de Atividade." Abaixo estão as ações realizadas:

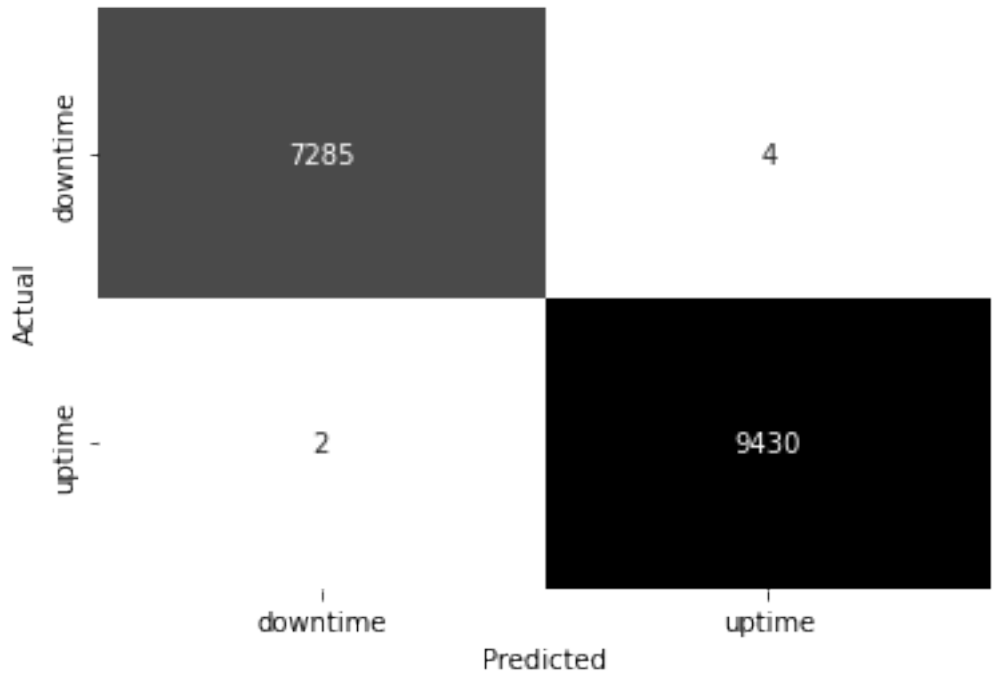
- Seleção de Carateristicas: Nesta etapa, são selecionadas as características mais relevantes para a classificação, o que inclui:
  - 'vel_RMS,': Esta característica representa a soma do valor dos dados de RMS (Root Mean Square) da velocidade de vibração nas três direções.
  - 'accel_RMS,': Representa o valor do RMS da aceleração nas três direções.
  - 'model_type,': Esta característica identifica o tipo de modelo associado ao ativo (bomba, motor, compressor, etc.).
  - 'rpm.': Indica as rotações por minuto (RPM) associadas ao ativo.
- Divisão dos Dados: Os dados são divididos em conjuntos de treinamento e teste, com 50% dos dados destinados ao conjunto de teste.
- Criação e Treinamento do Classificador de Árvore de Decisão:Um classificador de árvore de decisão é criado e treinado usando o conjunto de treinamento.
- Previsões nos Dados de Treinamento e Teste: O classificador faz previsões nos conjuntos de treinamento e teste.
- Avaliação da Precisão do Modelo: A precisão do modelo é avaliada usando a métrica 'accuracy_score' nos conjuntos de treinamento e teste, indicando a porcentagem de previsões corretas. mostrando uma accuracy perto de 1.
- Exportação da Árvore de Decisão: A árvore de decisão é exportada para uma representação textual usando 'export_text' para visualizar a estrutura da árvore e as regras de decisão. Tendo só 16 folhas.
- Criação de uma Matriz de Confusão: É gerada uma matriz de confusão para avaliar o desempenho do modelo, tendo menos de 0.1% de falsos positivos nos dois casos 'downtime' e 'uptime'.

[7]:
```
# classification decision tree for downtime and uptime asset
# ============================================================
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import export_text
from sklearn.metrics import confusion_matrix
# select features
columns = ['vel_RMS', 'accel_RMS', 'model_type','rpm']
X = np.array(df.loc[:, columns])
y = np.array(df.loc[:, 'class'])

# Split the data into training and testing sets -------------------
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.5, random_state=21)
```

```
# Create and train a Decision Tree Classifier ---------------------
clf = DecisionTreeClassifier(random_state=21)
clf.fit(X_train, y_train)

# Make predictions on the train and test data set -----------------
y_pred_tr = clf.predict(X_train)
y_pred = clf.predict(X_test)

# Evaluate the model's accuracy -----------------------------------
accuracy_tr = accuracy_score(y_train, y_pred_tr)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy train:", accuracy_tr)
print("Accuracy test:", accuracy)
print("Number of leaf nodes:", clf.get_n_leaves())

# Export the Decision Tree to a text-based representation
tree_text = export_text(clf, feature_names=columns)
print(tree_text)

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)
lab = ['downtime', 'uptime']
sns.heatmap(cm, annot=True, fmt='d', cmap='Greys',
  cbar=False, xticklabels=lab, yticklabels=lab)
plt.xlabel('Predicted');  plt.ylabel('Actual'); plt.show()
```

```
Accuracy train: 1.0
Accuracy test: 0.999641169786496
Number of leaf nodes: 16
|--- accel_RMS <= 0.01
|   |--- accel_RMS <= 0.01
|   |   |--- accel_RMS <= 0.01
|   |   |   |--- class: 0.0
|   |   |--- accel_RMS >  0.01
|   |   |   |--- rpm <= 2645.00
|   |   |   |   |--- class: 0.0
|   |   |   |--- rpm >  2645.00
|   |   |   |   |--- class: 1.0
|   |--- accel_RMS >  0.01
|   |   |--- rpm <= 2650.00
|   |   |   |--- class: 0.0
|   |   |--- rpm >  2650.00
|   |   |   |--- class: 1.0
|--- accel_RMS >  0.01
|   |--- accel_RMS <= 0.02
|   |   |--- rpm <= 1599.00
|   |   |   |--- rpm <= 729.00
```

```
|   |   |   |   |--- class: 1.0
|   |   |   |--- rpm >  729.00
|   |   |   |   |--- class: 0.0
|   |   |--- rpm >  1599.00
|   |   |   |--- vel_RMS <= 0.00
|   |   |   |   |--- class: 1.0
|   |   |   |--- vel_RMS >  0.00
|   |   |   |   |--- class: 0.0
|   |--- accel_RMS >  0.02
|   |   |--- accel_RMS <= 0.03
|   |   |   |--- vel_RMS <= 0.00
|   |   |   |   |--- class: 1.0
|   |   |   |--- vel_RMS >  0.00
|   |   |   |   |--- model_type <= 3.50
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- model_type >  3.50
|   |   |   |   |   |--- class: 1.0
|   |   |--- accel_RMS >  0.03
|   |   |   |--- accel_RMS <= 0.03
|   |   |   |   |--- vel_RMS <= 0.00
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- vel_RMS >  0.00
|   |   |   |   |   |--- model_type <= 3.50
|   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |--- model_type >  3.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |--- accel_RMS >  0.03
|   |   |   |   |--- class: 1.0
```



1.6.3 Função para Prever 'Downtime' e 'Uptime'

A função "calculate_dw_up" permite calcular o tempo de inatividade (downtime) e tempo de atividade (uptime) de um equipamento (ativo) medido por um sensor específico em um DataFrame de dados.

Parâmetros: - df: DataFrame contendo os dados do sensor. - sensor: Identificador do sensor para o qual o downtime e uptime devem ser calculados.

Retorno: - Downtime e uptime em segundos, juntamente com o intervalo total de tempo das medições.

Funcionamento da Função: 1. A função começa selecionando os dados do sensor específico no DataFrame "df" e garante que eles estejam ordenados cronologicamente.

2. Os intervalos de tempo entre as medições de vibração são calculados para determinar os momentos em que ocorrem as transições entre downtime e uptime. O último intervalo de tempo também é adicionado para abranger a última medição.

3. Em seguida, a função usa um classificador de aprendizado de máquina previamente treinado chamado "clf" para prever se cada medição corresponde a downtime (0) ou uptime (1).

4. A função calcula o tempo total abrangido pelas medições, o tempo de uptime (soma dos intervalos de tempo em que o equipamento estava ativo) e o tempo de downtime (soma dos intervalos de tempo em que o equipamento estava inativo).

5. Os valores de downtime, uptime e o intervalo de tempo total são retornados como resultados da função.

[8]:
```python
# function to predict downtime and uptime
# ================================================================
def calculate_dw_up(df, sensor):
  '''
    Calculate the downtime and uptime in DataFrame 'df' for \
      the equipment measured by the sensor.

    Parameters:
    - df: DataFrame containing sensor data.
    - sensor: Sensor identifier for which downtime and uptime \
      are calculated.

    Returns:
    - Downtime and uptime in seconds, and the total time span \
      of measurements.
  '''
  # get delta times between vibration measurement
  dfi = df[df['sensorId']==sensor].copy() # select sensor data

  # sorts the data by the 'time_start' column to ensure chronological order.
  dfi = dfi.sort_values(by='time_start').copy() # ensure order

  # computes the time intervals between consecutive vibration measurements.
  dtimes = np.array(dfi.time_start[1:]    # get delta times
    ) - np.array(dfi.time_start[:-1])
  last_dt = df['duration'].iloc[-1]       # add dtime last
  dtimes = np.r_[dtimes,  last_dt]        #     measurement

  # predict downtime or uptime in each measurement
  #   using a machine learning classifier 'clf'.
  columns = ['vel_RMS', 'accel_RMS', 'model_type','rpm']
  class_pred = clf.predict(np.array(dfi.loc[:, columns]))

  # Calculates the total time span of measurements,
  #   uptime in seconds, and downtime in seconds.
  total_time = dfi.time_start.max() - dfi.time_start.min() + last_dt
  uptime   = np.sum(dtimes*(class_pred==1))
  downtime = np.sum(dtimes*(class_pred==0))
  return downtime, uptime, total_time
```

1.6.4 Previsão de Tempos para Cada Máquina

Os tempos de operação total, downtime e uptime são previstos e exibidos para cada máquina, com base nos dados de sensores e no modelo de aprendizado de máquina. Abaixo estão as ações realizadas:

- Iteração por Sensores (maquinas ou ativos):
  - Para cada sensor na lista de sensores, as seguintes informações são obtidas:

        * Nome da máquina (“machine”).
        * Tipo de modelo da máquina (“mtype”).
        * Máximo tempo de downtime especificado para a máquina (“maxdw”).
- Cálculo de Downtime e Uptime:
  - A função “calculate_dw_up” é chamada para calcular o downtime, uptime e o tempo total de operação da máquina com base nos dados do sensor.
- Exibição dos Resultados:
  - Os resultados são exibidos na forma de uma saída formatada para cada máquina, com informações detalhadas:
    * Nome da máquina.
    * Tipo de modelo da máquina.
    * Máximo tempo de downtime especificado.
    * Tempos calculados de operação total, downtime e uptime para a máquina.

[9]:
```
# predict times for each machine
# ================================================================
msgs = ['total operation time = ', 'downtime = ', 'uptime = ']
for sensor in sensors:
  # define data of each sensor
  machine = assets['name'][sensor],
  mtype = assets['modelType'][sensor]
  maxdw = assets['specifications.maxDowntime'][sensor]

  # calculate downtime and uptime
  downtime, uptime, total_time = calculate_dw_up(df, sensor)
  print((' %s '%(machine)).center(65, ' '))
  print('machine type : ', mtype)
  print('maximum downtime : ', maxdw)
  times = [total_time, downtime, uptime]
  for msg, time in zip(msgs, times):
    time_formated = str(datetime.timedelta(seconds=time))
    print(msg, time_formated)
```

```
    Ventilador Acima do Elemento GA160 FF - Prédio B015
machine type :  compressor
maximum downtime :  48.0
total operation time =  36 days, 1:03:27.711000
downtime =  28 days, 7:56:14.711000
uptime =  7 days, 17:07:13
                RDF-61.1
machine type :  heaterFurnace
maximum downtime :  0.25
total operation time =  36 days, 1:01:18.711000
downtime =  2:35:01
uptime =  35 days, 22:26:17.711000
           TRANSFORMADOR 500 KVA N°1
machine type :  transformer
maximum downtime :  0.0
```

```
total operation time =  35 days, 23:38:34.711000
downtime =  0:00:00
uptime =  35 days, 23:38:34.711000
      Motor Bomba - Tanque de Expansão Tubo Verde
machine type :  pump
maximum downtime :  48.0
total operation time =  36 days, 0:58:59.711000
downtime =  25 days, 14:07:21.711000
uptime =  10 days, 10:51:38
            CAG1- BAGS.12- Motor
machine type :  pump
maximum downtime :  0.0
total operation time =  36 days, 1:03:08.711000
downtime =  21 days, 7:48:01.711000
uptime =  14 days, 17:15:07
               VTF-61.1
machine type :  heaterFurnace
maximum downtime :  0.25
total operation time =  36 days, 1:01:48.711000
downtime =  0:10:41
uptime =  36 days, 0:51:07.711000
          Boko MA-1510 - Motor 2 da UH
machine type :  eletricMotor
maximum downtime :  0.0
total operation time =  36 days, 1:07:23.711000
downtime =  29 days, 15:17:03
uptime =  6 days, 9:50:20.711000
         Ventilador -GA75 FF - Prédio B104
machine type :  compressor
maximum downtime :  48.0
total operation time =  35 days, 16:52:28.711000
downtime =  16 days, 7:54:18
uptime =  19 days, 8:58:10.711000
            CAG1- BAGS.3- Motor
machine type :  pump
maximum downtime :  0.0
total operation time =  36 days, 1:03:17.711000
downtime =  21 days, 18:30:48.711000
uptime =  14 days, 6:32:29
             ROTULADORA SIDEL
machine type :  other
maximum downtime :  1.0
total operation time =  36 days, 1:03:26.711000
downtime =  19 days, 14:06:55.711000
uptime =  16 days, 10:56:31
```

1.7 Função para Identificar Mudanças nos Padrões de Vibração

A função "changes_patterns" é projetada para analisar dados de sensores e identificar mudanças de comportamento, tanto suaves quanto abruptas (soft changes e hard changes), bem como identificar outliers.

1.7.1 Mudanças Suaves (Soft Changes) e Abruptas (Hard Changes)

As mudanças de comportamento são identificadas com base em critérios específicos. 'S'oft changes' representam mudanças mais sutis que podem ser considerados como um aviso previo antes da falha acontecer, enquanto 'hard changes' são mudanças mais abruptas. As mudanças suaves são identificadas quando um valor excede a média do sinal mais duas vezes o desvio padrão. Já as mudanças abruptas são identificadas quando um valor excede a média mais três vezes o desvio padrão. Esses critérios foram escolhidos de forma a detectar desvios significativos em relação à média, indicando potenciais mudanças no comportamento da máquina.

1.7.2 Outliers em Transformadores, Motores e Bombas Centrífuga

Observou-se que transformadores, motores e bombas centrífugas apresentam uma relação linear entre os dados de aceleração e vibração. Portanto, desvios desse comportamento podem indicar problemas na operação do equipamento e resultar em falhas subsequentes. Esses outliers são identificados por meio da correlação de Pearson entre as últimas 20 medições de vibração e aceleração.

1.7.3 Função

O funcionamento da função "changes_patterns" é apresentado a seguir:

Parâmetros: - df: DataFrame contendo os dados do sensor. - sensor: Identificador do sensor para o qual a análise está sendo realizada.

Retorno: - Um DataFrame chamado 'changes' com informações sobre soft changes, hard changes e outliers.

Funcionamento da Função: 1. Os dados do sensor específico são extraídos do DataFrame "df" para análise.

2. Os dados são filtrados para incluir apenas as medições de uptime, usando um classificador de aprendizado de máquina previamente treinado ("clf").
3. Os sinais de interesse para a análise são definidos, incluindo os sinais de aceleração ('accel_RMS') e velocidade ('vel_RMS') nas direções horizontal ('h'), vertical ('v') e axial ('a').
4. Para cada sinal de interesse, são adicionadas médias móveis (moving averages) e desvios padrão (standard deviations) calculados com base em uma janela de 20 medições.
5. A correlação de Pearson entre a aceleração e a velocidade é calculada para cada eixo ('h', 'v', 'a').
6. Soft changes e hard changes no comportamento são identificados para os sinais selecionados com base em critérios específicos, como superar médias e desvios padrão.
7. Outliers são identificados para transformadores, motores e bombas centrífugas com base no tipo de modelo ('model_type') e na correlação de Pearson.

8. As informações sobre soft changes, hard changes e outliers são armazenadas no DataFrame 'changes' e retornadas como resultado da função.

[10]:
```python
# functions to identify changes in vibration patterns
# ====================================================================
def changes_patterns(df, sensor):
  """
  Analyzes sensor data to identify changes in behavior and outliers.

  Parameters:
  - df: DataFrame containing sensor data.
  - sensor: Sensor identifier for which the analysis is performed.

  Returns:
  - A DataFrame 'changes' with information about soft changes,\
    hard changes, and outliers.

  """
  # Get data for the specified sensor from the DataFrame
  dfi = df[df['sensorId'] == sensor].copy()
  index = dfi.index

  # Filter the data for uptime measurements using
  #   a machine learning classifier (clf)
  columns = ['vel_RMS', 'accel_RMS', 'model_type', 'rpm']
  class_pred = clf.predict(np.array(dfi.loc[:, columns]))
  dfi = dfi.iloc[class_pred == 1, :]
  index_pred = index[class_pred == 1]

  # Define signals of interest for analysis
  signals_a = ['accel_RMS_' + i for i in ['h', 'v', 'a']]
  signals_v = ['vel_RMS_' + i for i in ['h', 'v', 'a']]
  signals = signals_a + signals_v

  # Add moving average and standard deviation for selected signals
  for signal in signals:
    dfi.loc[:, signal + '_mean'] = dfi[signal].rolling(
      window=20, min_periods=0).mean()
    dfi.loc[:, signal + '_std'] = dfi[signal].rolling(
      window=20, min_periods=0).std()

  # Calculate Pearson correlation between acceleration and
  #   velocity for each axis (h, v, a)
  for i in ['h', 'v', 'a']:
    dfi.loc[:, 'cor_accel_vel_' + i] = np.abs(
      dfi['accel_RMS_' + i].rolling(
        window=20, min_periods=1).corr(dfi['vel_RMS_' + i]))
```

```
# Identify soft and hard changes in behavior for the selected signals
changes = pd.DataFrame(
  index=index, columns=['soft', 'hard', 'outliers'])
changes.loc[:, :] = 0
for signal in signals:
  mean = dfi.loc[:, signal + '_mean']
  std = dfi.loc[:, signal + '_std']
  x = dfi.loc[:, signal]
  changes.loc[index_pred, 'soft'] += 1 * (
    x > (mean + 2 * std))  # Identify soft changes in behavior
  changes.loc[index_pred, 'hard'] += 1 * (
    x > (mean + 3 * std))  # Identify hard changes in behavior

# Identify outliers in transformer, motors, and centrifugal
#   pumps based on model_type and correlation
if np.isin(dfi['model_type'].mean(), [1, 4, 5]):
  for i in ['h', 'v', 'a']:
    if dfi['power'].mean()>20 and dfi['model_type'].mean()==4:
      pass
    else:
      changes.loc[index_pred, 'outliers'] += 1 * (
        dfi.loc[:, 'cor_accel_vel_' + i] < 0.9)

return changes
```

1.8 Teste da Função para Previsão de Falhas

Após a implementação a função é testada, a avaliação das mudanças suaves é realizada para identificar possíveis indícios de danos no sistema. Automaticamente, os pontos que apresentam mais de dois incrementos suaves simultâneos nas análises das vibrações "vib_RMS_accel" e "vib_RMS_vel" em qualquer uma das três direções são selecionados como possiveis danos.

Da mesma forma, quaisquer mudanças abruptas no comportamento também são consideradas indicativas de possíveis falhas. Além disso, dados que apresentem mais de dois outliers em qualquer uma das duas direções são igualmente considerados possíveis falhas.

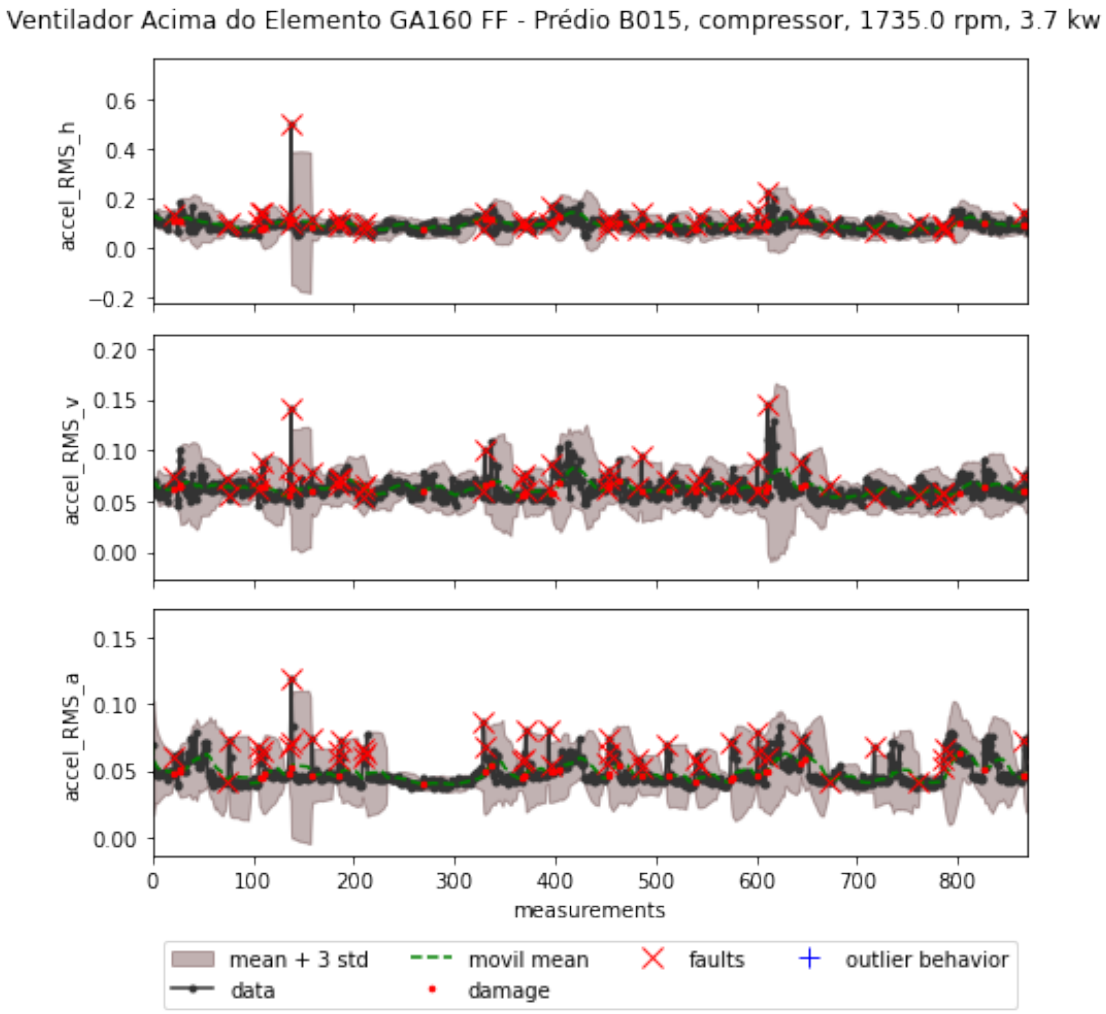
A função "changes_patterns" é testada para identificar mudanças de comportamento em equipamentos. O teste é realizado para cada sensor na lista de sensores disponíveis. Para cada sensor, as seguintes etapas são executadas:

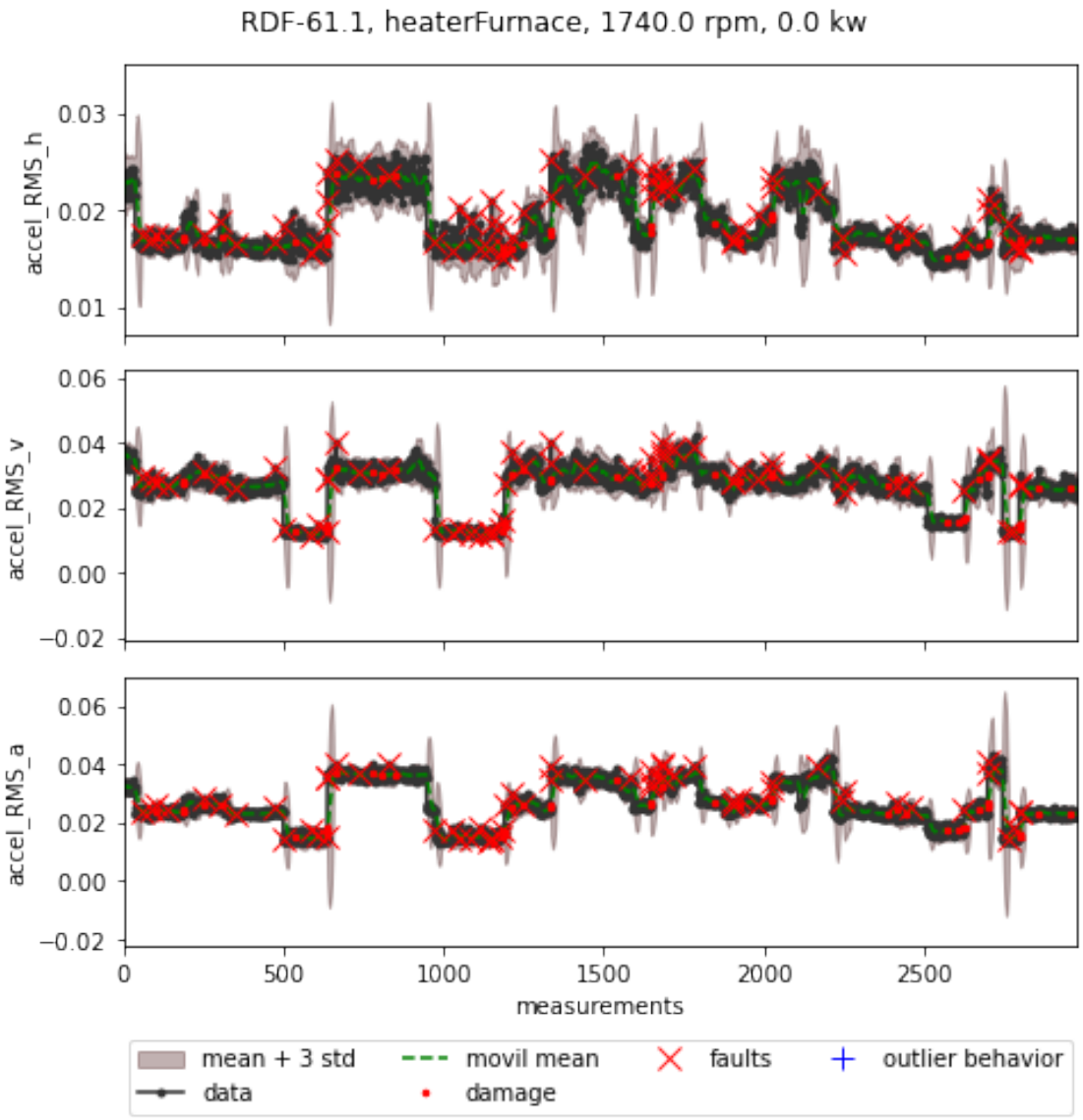
1. A função "changes_patterns" é chamada para identificar mudanças de comportamento no sensor em análise. O resultado é armazenado na variável "changes."
2. Os dados específicos desse sensor são selecionados e copiados em um DataFrame separado, chamado "dfi."
3. Os dados são filtrados para incluir apenas as medições de uptime, usando um classificador de aprendizado de máquina previamente treinado ("clf").

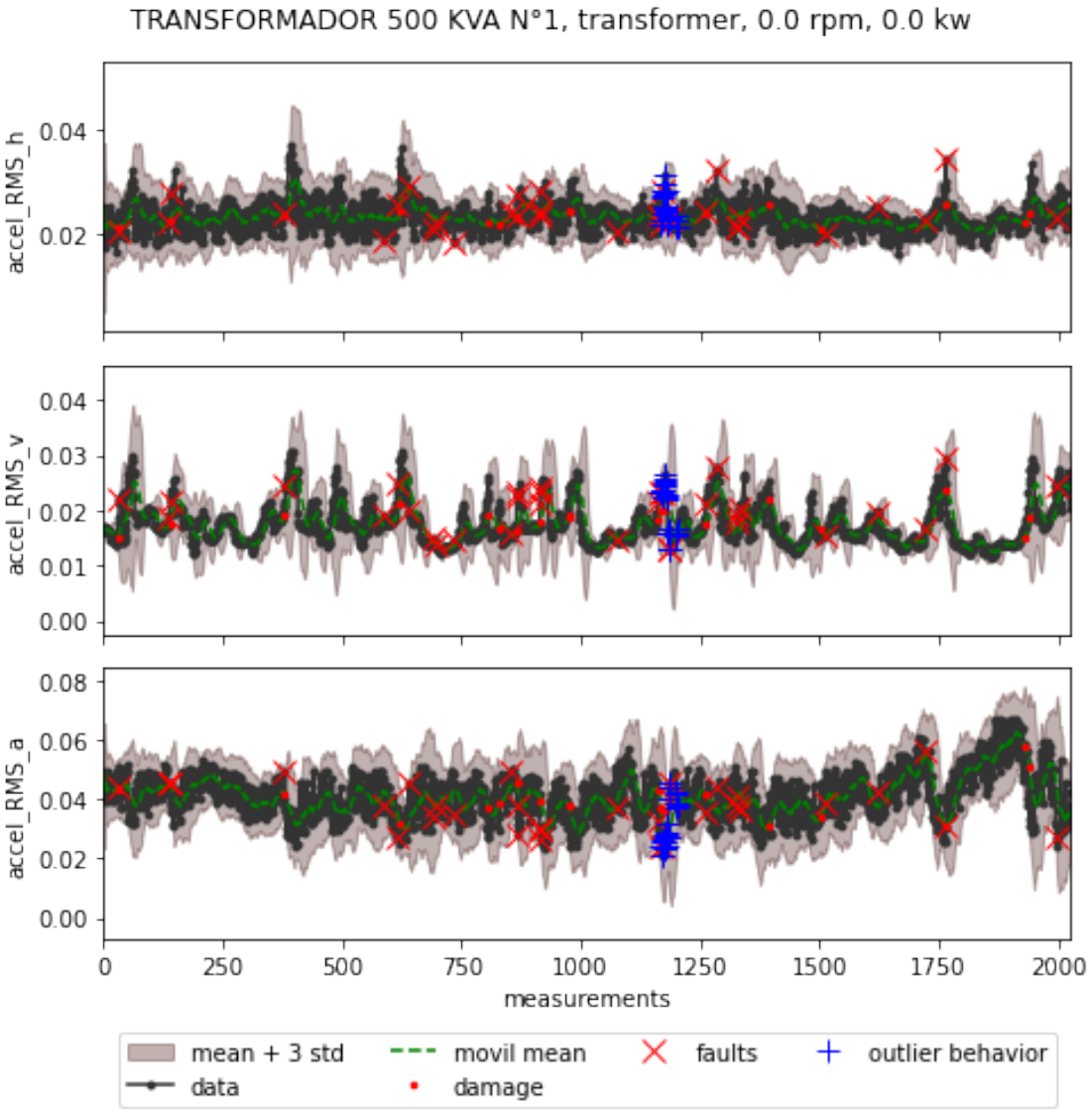
4. As possíveis falhas são identificadas como aqueles pontos em que ocorrem simultaneamente mais de duas mudanças suaves, mais de dois outliers e mudanças abruptas no comportamento.
5. Um gráfico é criado para visualizar as informações. O gráfico possui três subplots empilhados verticalmente e compartilha o mesmo eixo x.
6. Para cada um dos subplots, as seguintes informações são plotadas:
    - A faixa de três desvios padrão acima e abaixo da média é preenchida para destacar o comportamento esperado.
    - Os dados reais são representados por pontos.
    - A média móvel dos dados é exibida como uma linha tracejada.
    - Pontos vermelhos indicam mudanças suaves no comportamento.
    - Marcadores "x" vermelhos indicam falhas abruptas (hard changes).
    - Marcadores "+" azuis destacam comportamentos atípicos (outliers).

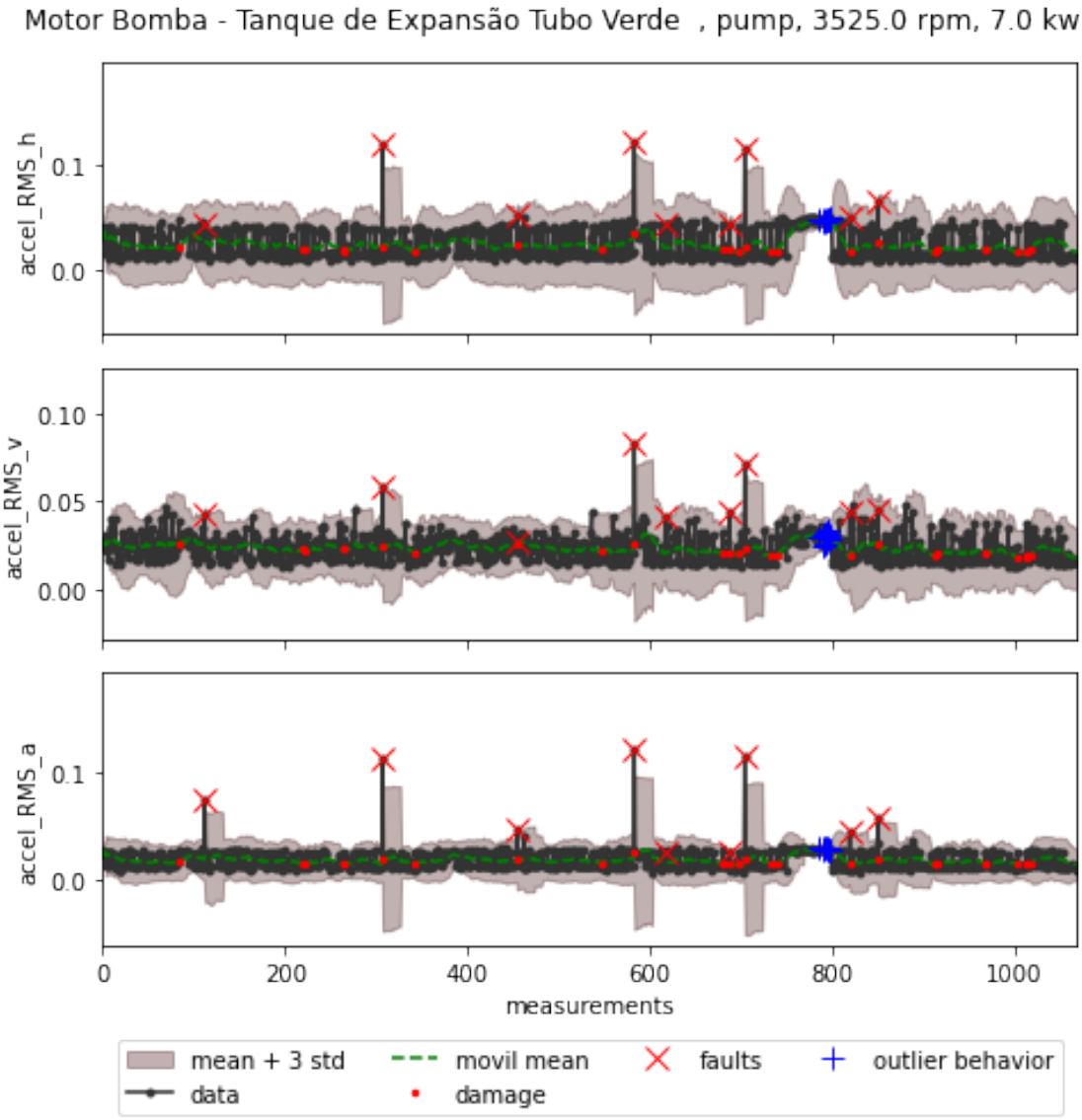
[11]:
```
# test function in fault prediction
# ================================================================
for j, sensor in enumerate(sensors):

  # define data of each sensor ------------------------------------
  title = '%s, %s, %s rpm, %s kw'%(
    assets['name'][sensor], assets['modelType'][sensor],
    assets['specifications.rpm'][sensor],
    assets['specifications.power'][sensor])

  # identify changes in sensor ------------------------------------
  changes = changes_patterns(df, sensor)

  # Get data for the specified sensor from the DataFrame ----------
  dfi = df[df['sensorId'] == sensor].copy()
  index = dfi.index

  # Filter the data for uptime measurements using
  #   a machine learning classifier (clf) ------------------------
  columns = ['vel_RMS', 'accel_RMS', 'model_type', 'rpm']
  class_pred = clf.predict(np.array(dfi.loc[:, columns]))
  dfi = dfi.iloc[class_pred == 1, :]
  index_pred = index[class_pred == 1]

  # fault identification ------------------------------------------
  t = dfi.loc[index_pred, 'time_start']
  t = np.arange(len(t))
  soft = changes.loc[index_pred, 'soft']>2
  hard = changes.loc[index_pred, 'hard']>0
  outliers = changes.loc[index_pred, 'outliers']>2

# plot ----------------------------------------------------------
  fig, ax = plt.subplots(3, 1, figsize=(7,7), sharex=True)
```

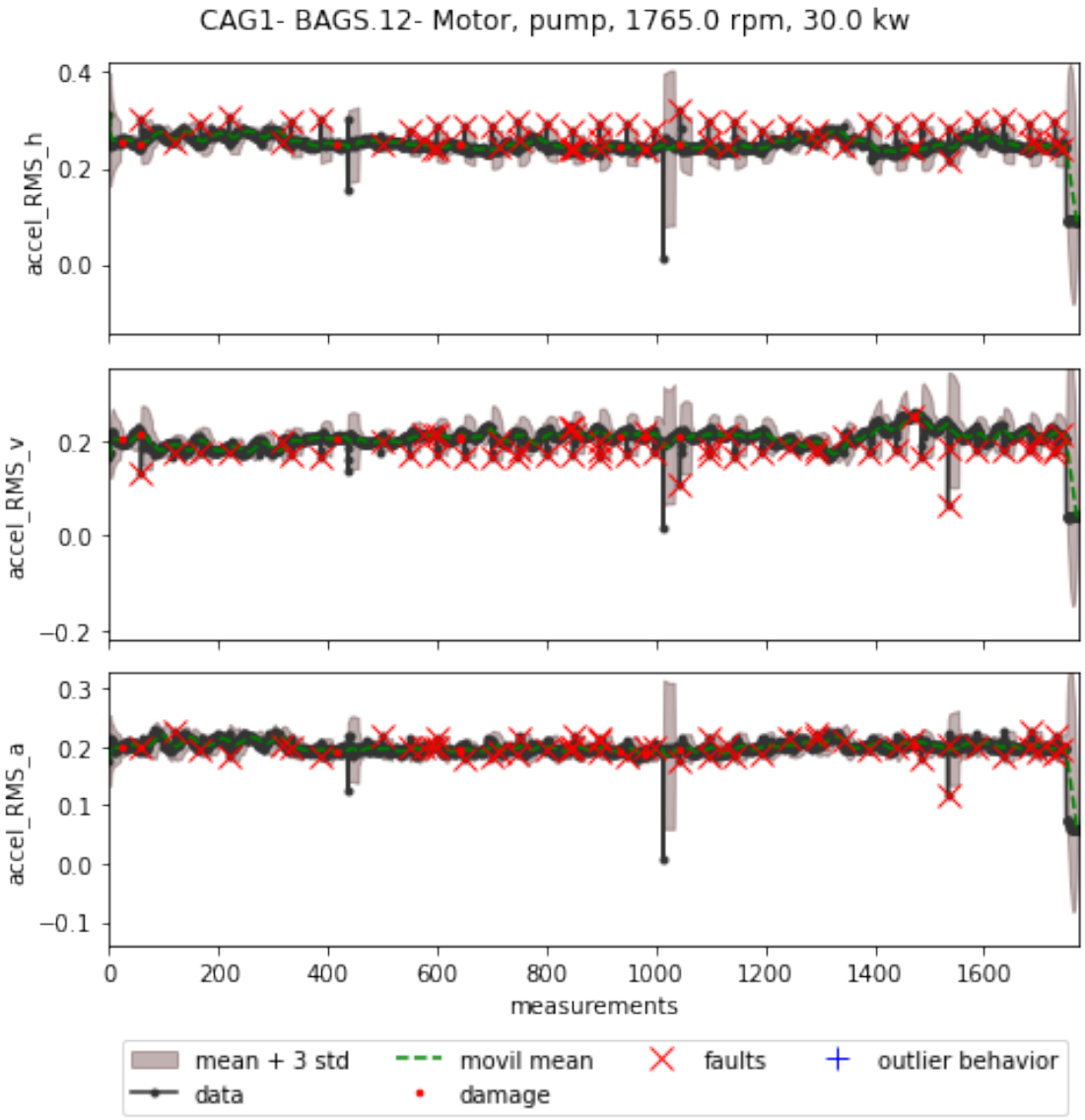
```
  fig.suptitle(title)
  for j, signal in enumerate(['accel_RMS_' + i for i in ['h', 'v', 'a']]):
    x = np.array(dfi.loc[index_pred, signal])
    x_mean = np.array(
      dfi[signal].rolling(window=20, min_periods=0).mean())
    x_std = np.array(
      dfi[signal].rolling(window=20, min_periods=0).std())
    ax[j].fill_between(t, x_mean - 3*x_std, x_mean + 3*x_std,
      color=(0.2, 0, 0), alpha=0.3, label='mean + 3 std')
    ax[j].plot(t, x, '.-', color=(0.2, 0.2, 0.2), label='data')
    ax[j].plot(t, x_mean, '--', color='g', label='movil mean')
    ax[j].plot(t[soft], x_mean[soft], '.', color='r', markersize=5,
      label='damage')
    ax[j].plot(t[hard], x[hard], 'x', color='r', markersize=10,
      label='faults')
    ax[j].plot(t[outliers], x[outliers], '+', color='b',
      markersize=10, label='outlier behavior')
    ax[j].set_ylabel(signal)
    ax[j].set_ylim(np.nanmin(x - 3*x_std),
                   np.nanmax(x + 3*x_std))
  ax[j].set_xlabel('measurements')
  ax[j].set_xlim(t.min(), t.max())
  ax[j].legend(loc='upper center', bbox_to_anchor=(0.5, -0.3), ncol=4)
  fig.tight_layout()
```

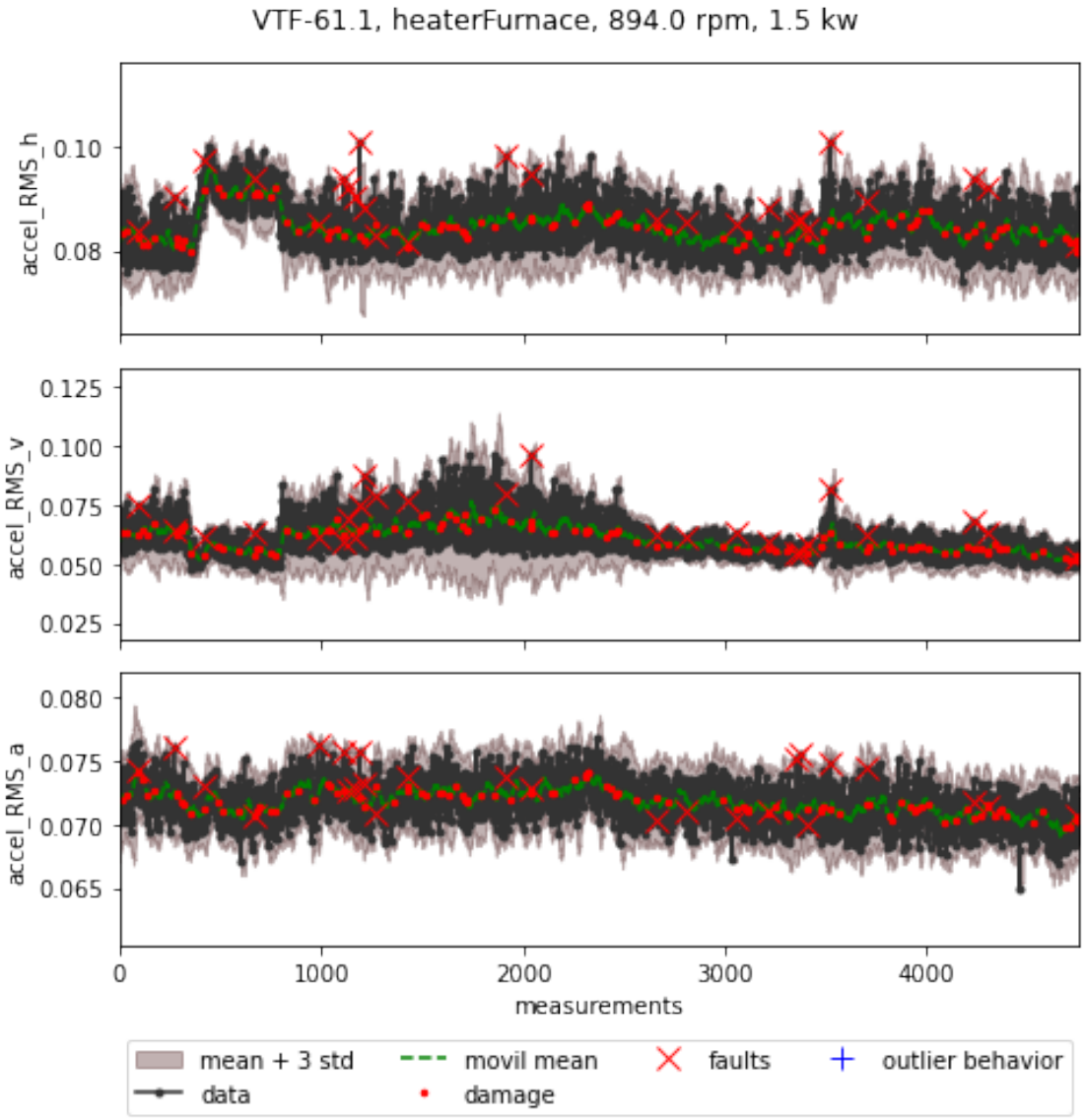

Ventilador Acima do Elemento GA160 FF - Prédio B015, compressor, 1735.0 rpm, 3.7 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
measurements
mean + 3 std
movil mean
faults
outlier behavior
data
damage

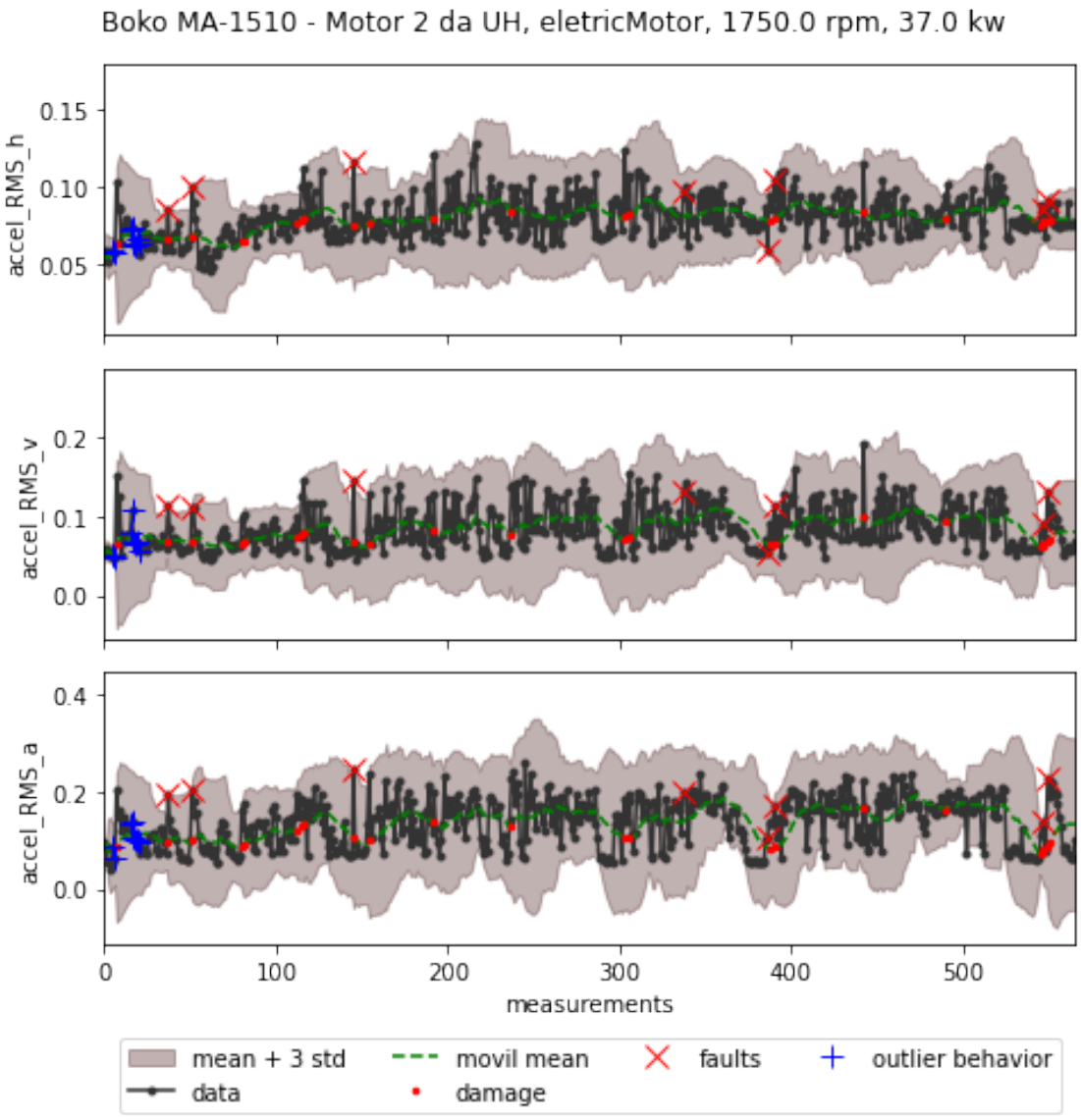

RDF-61.1, heaterFurnace, 1740.0 rpm, 0.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
0.03
0.02
0.01
0.06
0.04
0.02
0.00
−0.02
0
500
1000
1500
2000
2500
measurements
mean + 3 std
data
movil mean
damage
faults
outlier behavior

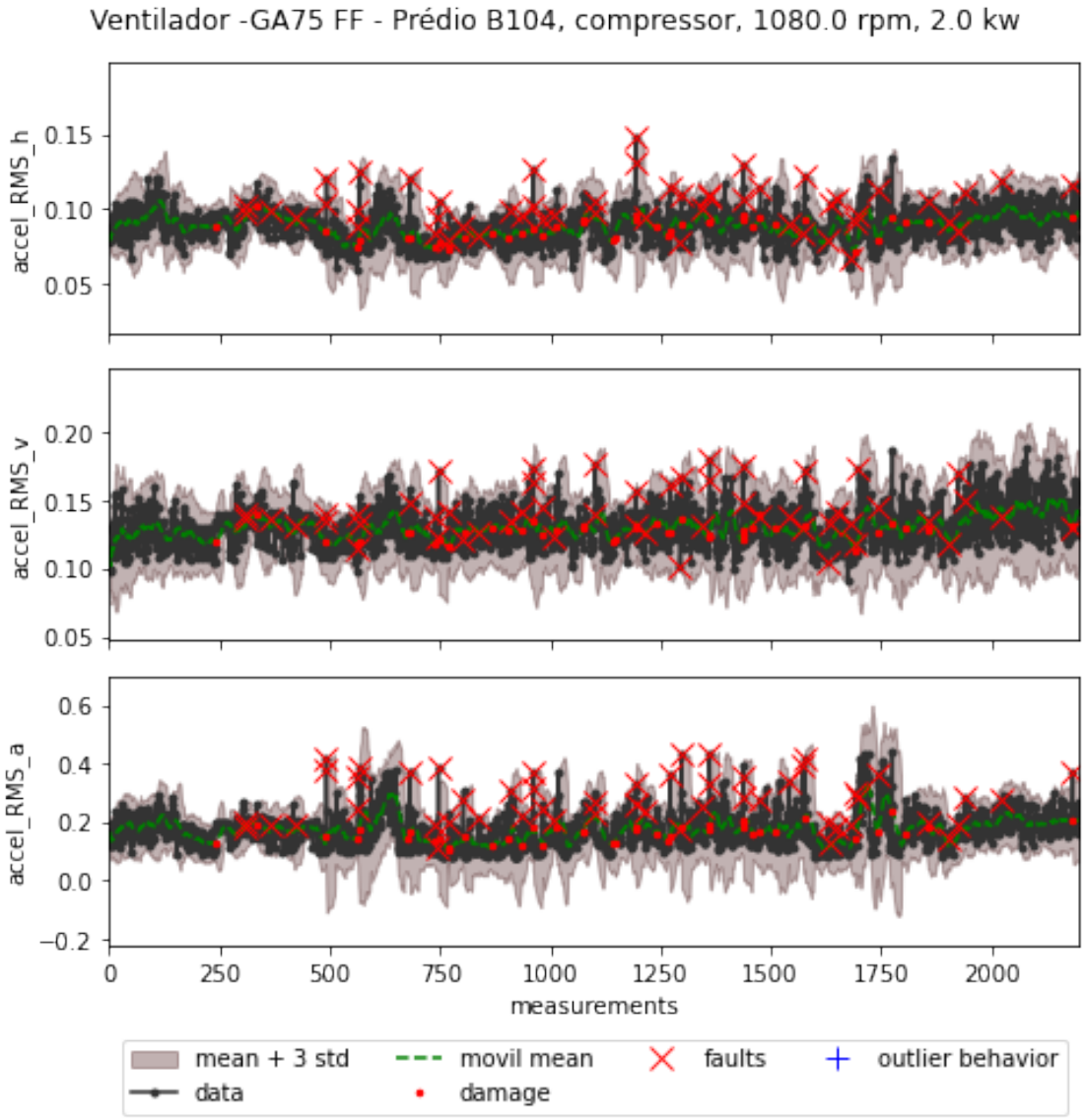

TRANSFORMADOR 500 KVA N°1, transformer, 0.0 rpm, 0.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
0.04
0.02
0.04
0.03
0.02
0.01
0.00
0.08
0.06
0.04
0.02
0.00
0
250
500
750
1000
1250
1500
1750
2000
measurements
mean + 3 std
data
movil mean
damage
faults
outlier behavior

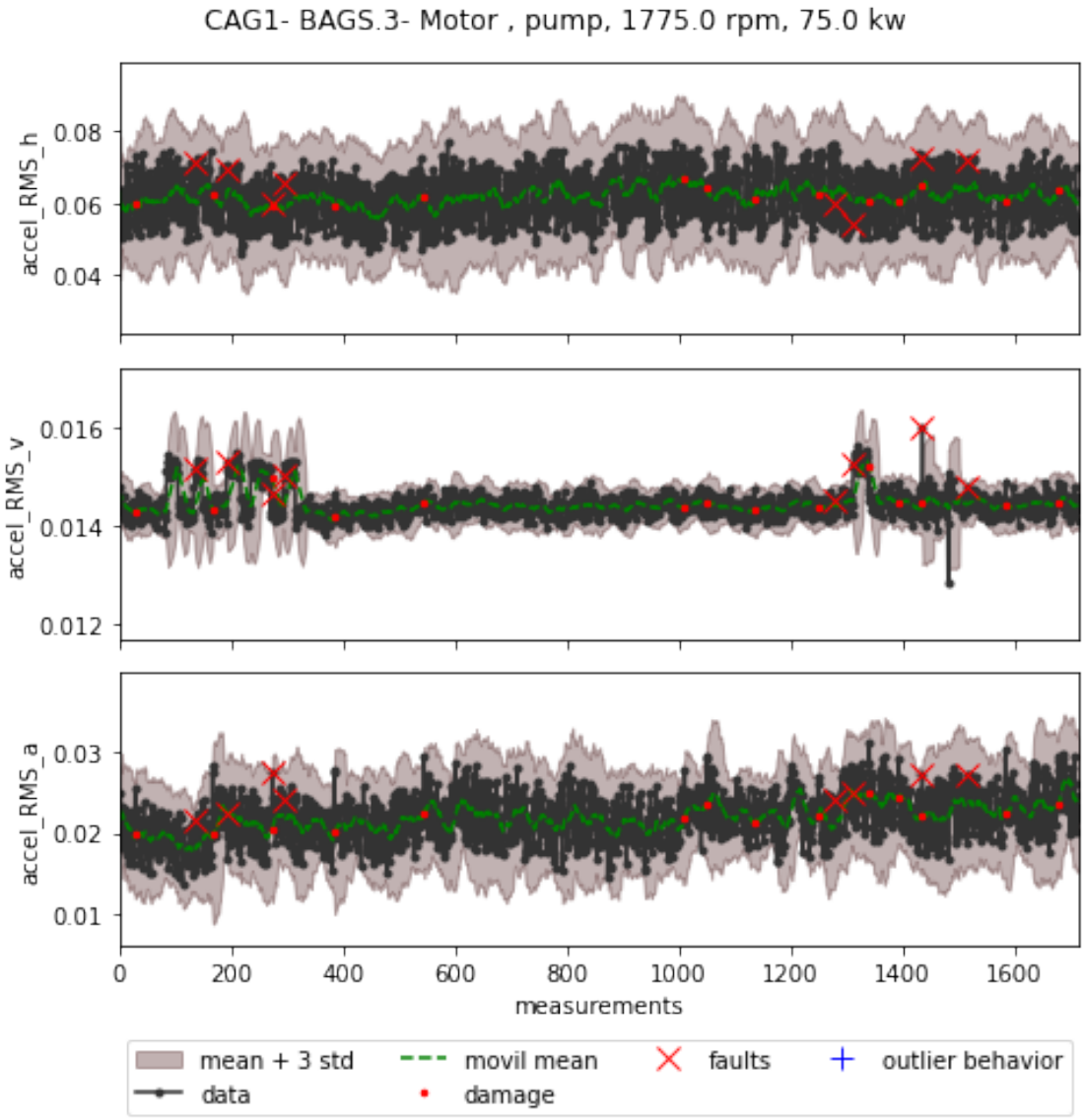

Motor Bomba - Tanque de Expansão Tubo Verde , pump, 3525.0 rpm, 7.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
0.1
0.0
0.10
0.05
0.00
0
200
400
600
800
1000
measurements
mean + 3 std
data
movil mean
damage
faults
outlier behavior

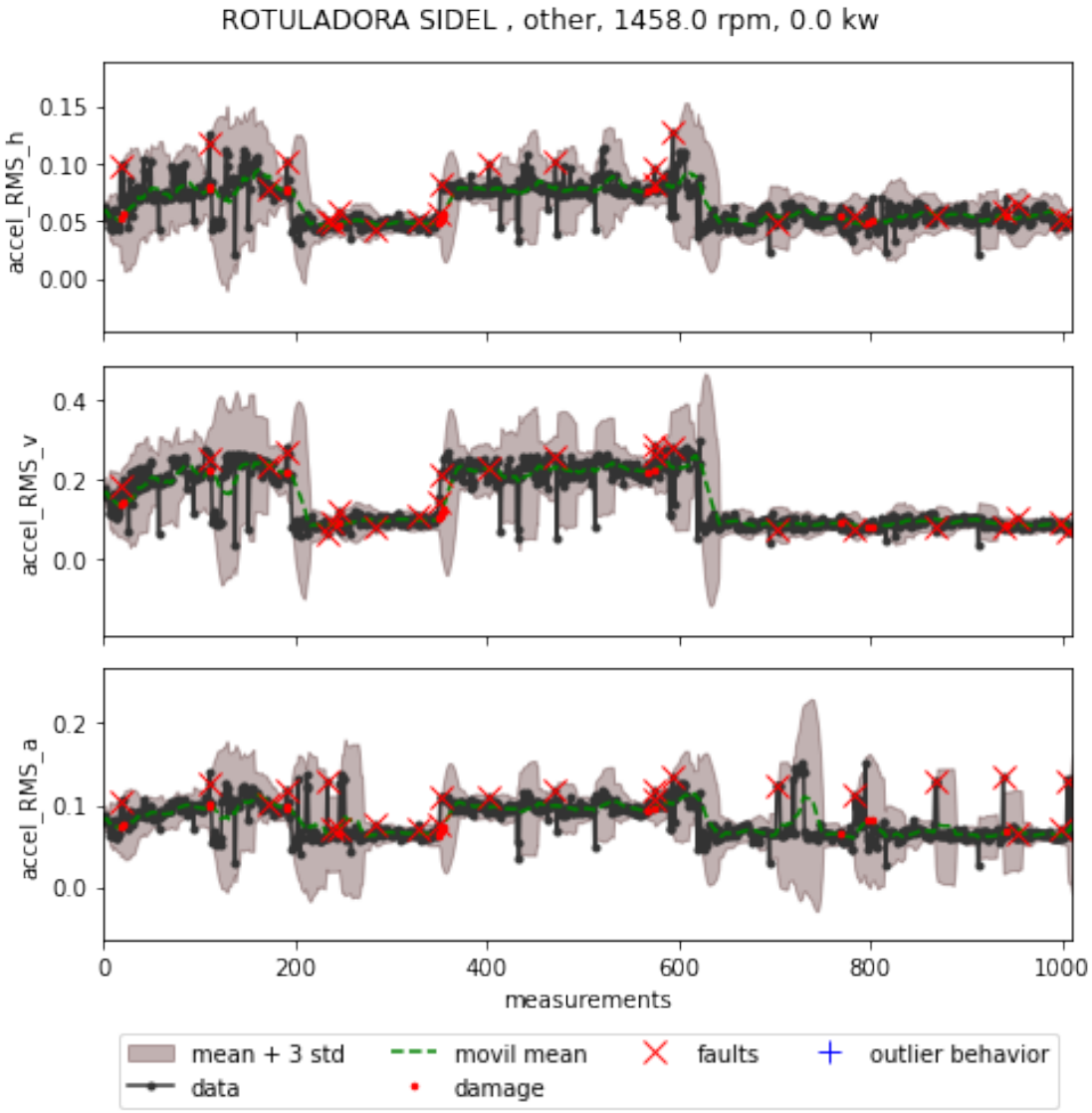

CAG1- BAGS.12- Motor, pump, 1765.0 rpm, 30.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
measurements
mean + 3 std
movil mean
faults
outlier behavior
data
damage


VTF-61.1, heaterFurnace, 894.0 rpm, 1.5 kw
accel_RMS_h
0.10
0.08
accel_RMS_v
0.125
0.100
0.075
0.050
0.025
accel_RMS_a
0.080
0.075
0.070
0.065
0
1000
2000
3000
4000
measurements
mean + 3 std
movil mean
faults
outlier behavior
data
damage


Boko MA-1510 - Motor 2 da UH, eletricMotor, 1750.0 rpm, 37.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
0.15
0.10
0.05
0.2
0.1
0.0
0.4
0.2
0.0
0
100
200
300
400
500
measurements
mean + 3 std
data
movil mean
damage
faults
outlier behavior


Ventilador -GA75 FF - Prédio B104, compressor, 1080.0 rpm, 2.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
0.15
0.10
0.05
0.20
0.15
0.10
0.05
0.6
0.4
0.2
0.0
−0.2
0
250
500
750
1000
1250
1500
1750
2000
measurements
mean + 3 std
data
movil mean
damage
faults
outlier behavior


CAG1- BAGS.3- Motor , pump, 1775.0 rpm, 75.0 kw
accel_RMS_h
0.08
0.06
0.04
accel_RMS_v
0.016
0.014
0.012
accel_RMS_a
0.03
0.02
0.01
0
200
400
600
800
1000
1200
1400
1600
measurements
mean + 3 std
movil mean
faults
outlier behavior
data
damage


ROTULADORA SIDEL , other, 1458.0 rpm, 0.0 kw
accel_RMS_h
accel_RMS_v
accel_RMS_a
0.15
0.10
0.05
0.00
0.4
0.2
0.0
0.2
0.1
0.0
0
200
400
600
800
1000
measurements
mean + 3 std
movil mean
faults
outlier behavior
data
damage