PROJETO DE REDES NEURAIS ARTIFICIAIS: "PREDICTIVE MAINTENANCE"

IMPORTANDO BASE DE DADOS

Foi importado o *dataset* "Machine Predictive Maintenance Classification", disponível em: https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification. Esse banco de dados contém informações de operação de um certo equipamento industrial e informa se, com tais parâmetros, tal equipamento falhou ou continuou em pleno funcionamento. Desse modo, a aplicação de Redes Neurais Artificiais torna-se de grande valia nesse cenário com o objetivo de predizer se, em determinadas situações, o equipamento está ou não suscetível à falhas, facilitando a atuação preventiva.

In [1]:
```
import pandas as pd
dataset = pd.read_csv("predictive_maintenance.csv")
dataset
```

Out[1]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
...	...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0	No Failure
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0	No Failure
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0	No Failure
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0	No Failure
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0	No Failure

10000 rows × 10 columns

PRÉ-PROCESSAMENTO

Primeiramente, retirou-se as linhas de dados que contenham valores nulos, a fim de que a rede neural tivesse uma aprendizagem correta.

In [2]:
```
dataset = dataset.dropna()
dataset
```

Out[2]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
...	...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0	No Failure
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0	No Failure
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0	No Failure
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0	No Failure
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0	No Failure

10000 rows × 10 columns

Em seguida, separou-se os dados das variáveis preditivas (x) e os dados referentes as variáveis de resposta (y).

In [3]:
```
x = dataset.iloc[:,2:-2]
y = dataset.iloc[:,-2:-1]
```

In [4]:
```
x
```

Out[4]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
0	M	298.1	308.6	1551	42.8	0
1	L	298.2	308.7	1408	46.3	3
2	L	298.1	308.5	1498	49.4	5
3	L	298.2	308.6	1433	39.5	7
4	L	298.2	308.7	1408	40.0	9
...	...	...	...	...	...	...
9995	M	298.8	308.4	1604	29.5	14
9996	H	298.9	308.4	1632	31.8	17
9997	M	299.0	308.6	1645	33.4	22
9998	H	299.0	308.7	1408	48.5	25
9999	M	299.0	308.7	1500	40.2	30

10000 rows × 6 columns

In [5]:
```
y
```

Out[5]:

	Target
0	0
1	0
2	0
3	0
4	0
...	...
9995	0
9996	0
9997	0
9998	0
9999	0

10000 rows × 1 columns

Para que a rede neural possa atuar, é necessário que todas as varáveis envolvidas tenham valores numéricos. Desse modo, importou-se o *OneHotEncoder* a fim de se atribuir valores numéricos à varáveis categóricas. Outrossim, importou-se o *ColumTransformer* a fim de se auxiliar o trabalho do *OneHotEncoder*. Os valores numéricos atribuidos constam nas colunas "ohe1", "ohe2" e "ohe3".

In [6]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), ['Type'])],
                       remainder='passthrough')
x = ct.fit_transform(x)

colunas = ['ohe1', 'ohe2','ohe3','Air temperature [K]','Process temperature [K]','Rotational speed [rpm]','Torq
pd.DataFrame(x, columns=colunas)
```

Out[6]:

	ohe1	ohe2	ohe3	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
0	0.0	0.0	1.0	298.1	308.6	1551.0	42.8	0.0
1	0.0	1.0	0.0	298.2	308.7	1408.0	46.3	3.0
2	0.0	1.0	0.0	298.1	308.5	1498.0	49.4	5.0
3	0.0	1.0	0.0	298.2	308.6	1433.0	39.5	7.0
4	0.0	1.0	0.0	298.2	308.7	1408.0	40.0	9.0
...	...	...	...	...	...	...	...	...
9995	0.0	0.0	1.0	298.8	308.4	1604.0	29.5	14.0
9996	1.0	0.0	0.0	298.9	308.4	1632.0	31.8	17.0
9997	0.0	0.0	1.0	299.0	308.6	1645.0	33.4	22.0
9998	1.0	0.0	0.0	299.0	308.7	1408.0	48.5	25.0
9999	0.0	0.0	1.0	299.0	308.7	1500.0	40.2	30.0

10000 rows × 8 columns

Em seguida, dividiu-se os dados de treino e de teste da rede neural, no qual 70% destes foram utilizados para treino e 30% para teste.

In [7]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3)
```

Para que a rede neural aprenda pela importância de cada variável preditiva na variável de resposta, sem levar em conta a ordem de grandeza das suas variáves, realizou-se a normalização destas por meio da importação do *StandardScaler.* Com isso, os valores ficaram na mesma faixa de ordem de grandeza.

In [8]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [9]:
```
x_train
```

Out[9]:
```
array([[-0.33280398, -1.22474487,  1.52648673, ...,  0.03992117,
        -0.16291183, -0.3294887 ],
       [-0.33280398, -1.22474487,  1.52648673, ...,  0.98858919,
        -1.12972648, -0.29796169],
       [-0.33280398, -1.22474487,  1.52648673, ..., -1.17044838,
         1.08298334, -1.41717026],
       ...,
       [-0.33280398,  0.81649658, -0.65509904, ..., -0.28175362,
         0.7839685 , -1.36987976],
       [-0.33280398,  0.81649658, -0.65509904, ...,  0.20893673,
        -0.30245209,  0.39563235],
       [ 3.00477175, -1.22474487, -0.65509904, ..., -0.85967782,
         0.46501934,  1.19957089]])
```

IMPLEMENTAÇÃO DAS REDES NEURAIS ARTIFICIAIS (MLP)

Para o treinamento desta rede neural artificial MLP, utilizou-se 1 camada intermediária com 6 neurônios, além de um neurônio na camada de saída. Segundo estudos de Jeff Heaton relatados no livro "Introduction to Neural Networks for Java", o número de neurônios da camada intermediária deve ser igual à 2/3 do número de neurônios da camada de entrada (8 variáveis de entrada nesse caso) mais o número de neurônios da camada de saída (1 neurônio para este estudo). Ademais, vide literatura, é aceitável que uma camada intermediária já é suficiente para aproximar o modelo de qualquer relação não-linear.

In [10]:
```
import tensorflow as tf

ann = tf.keras.models.Sequential()

ann.add (tf.keras.layers.Dense (units=6, activation='relu'))
ann.add (tf.keras.layers.Dense (units=1, activation='sigmoid'))

ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

ann.fit(x_train, y_train, batch_size=64, epochs=40)
```

```
Epoch 1/40
110/110 [==============================] - 1s 2ms/step - loss: 0.4783 - accuracy: 0.9590
Epoch 2/40
110/110 [==============================] - 0s 2ms/step - loss: 0.3329 - accuracy: 0.9659
Epoch 3/40
110/110 [==============================] - 0s 2ms/step - loss: 0.2410 - accuracy: 0.9659
Epoch 4/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1937 - accuracy: 0.9659
Epoch 5/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1702 - accuracy: 0.9659
Epoch 6/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1574 - accuracy: 0.9659
Epoch 7/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1495 - accuracy: 0.9659
Epoch 8/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1437 - accuracy: 0.9659
Epoch 9/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1392 - accuracy: 0.9659
Epoch 10/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1353 - accuracy: 0.9659
Epoch 11/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1315 - accuracy: 0.9659
Epoch 12/40
110/110 [==============================] - 0s 1ms/step - loss: 0.1279 - accuracy: 0.9659
Epoch 13/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1245 - accuracy: 0.9659
Epoch 14/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1212 - accuracy: 0.9659
Epoch 15/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1182 - accuracy: 0.9660
Epoch 16/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1156 - accuracy: 0.9660
Epoch 17/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1130 - accuracy: 0.9664
Epoch 18/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1107 - accuracy: 0.9667
Epoch 19/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1088 - accuracy: 0.9671
Epoch 20/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1071 - accuracy: 0.9673
Epoch 21/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1058 - accuracy: 0.9679
Epoch 22/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1045 - accuracy: 0.9679
Epoch 23/40
110/110 [==============================] - 0s 3ms/step - loss: 0.1033 - accuracy: 0.9680
Epoch 24/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1024 - accuracy: 0.9690
Epoch 25/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1017 - accuracy: 0.9686
Epoch 26/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1010 - accuracy: 0.9691
Epoch 27/40
110/110 [==============================] - 0s 2ms/step - loss: 0.1004 - accuracy: 0.9693
Epoch 28/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0999 - accuracy: 0.9693
Epoch 29/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0995 - accuracy: 0.9693
Epoch 30/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0992 - accuracy: 0.9693
Epoch 31/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0989 - accuracy: 0.9694
Epoch 32/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0987 - accuracy: 0.9699
Epoch 33/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0985 - accuracy: 0.9696
Epoch 34/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0983 - accuracy: 0.9696
Epoch 35/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0981 - accuracy: 0.9700
Epoch 36/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0980 - accuracy: 0.9691
Epoch 37/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0977 - accuracy: 0.9697
Epoch 38/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0977 - accuracy: 0.9696
Epoch 39/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0975 - accuracy: 0.9694
Epoch 40/40
110/110 [==============================] - 0s 2ms/step - loss: 0.0976 - accuracy: 0.9696
```

Out[10]: `<keras.callbacks.History at 0x7fa064549f40>`

RESULTADOS

Na seção de resultados, realizou-se a predição por meio das variáveis preditivas de teste.

In [11]:
```
y_pred = ann.predict(x_test)
y_pred = (y_pred > 0.5)

pred_array = 1 * y_pred.reshape(len(y_pred), 1)
test_array = y_test.values.reshape(len(y_test), 1)
```

```
94/94 [==============================] - 0s 1ms/step
```

Posteriormente, comparou-se os resultados obtidos da predição com os valores reais por meio de uma Matriz de Confusão. Também verificou-se a acuracidade da aprendizagem de máquina da rede neural por meio da importação do *accuracy_score*.

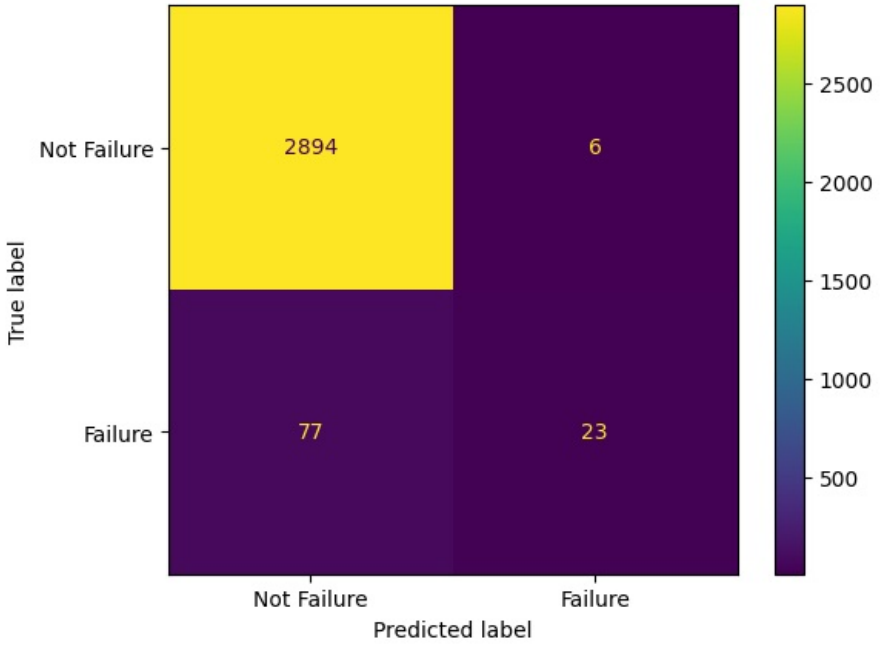
In [12]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(test_array, pred_array)

cm_display = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = ["Not Failure", "Failure"])

cm_display.plot()
plt.show()

print("Acuracidade da rede neural: ", (accuracy_score(y_test, y_pred)))
```




```
Acuracidade da rede neural:  0.9723333333333334
```

Obserou-se que a rede neural em questão apresentou uma acuracidade de 97,23%. Além disso, percebe-se no Gráfico de Confusão anterior que, nos dados de teste, há uma discrepância considerável entre os volumes de casos de falha e de casos de não-falha, o que ocorre também em todo o banco de dados (incluindo dados de treino e de teste). O valor de acuracidade poderia ser maior se o banco de dados possuisse mais cenários de falhas para melhor treinar a rede neural artificial nesses contextos. A quantidade de casos de não-falha (0) e de falha (1) estão presentes a seguir.

In [13]:
```
dataset["Target"].value_counts()
```

Out[13]:
```
0    9661
1     339
Name: Target, dtype: int64
```