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

np.set_printoptions(precision=3, suppress=True)

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)

url = '/content/txQuar.csv'
column_names = ['tx', 'nbr', 'txVaccin', 'txQuar', 'txInfect',
                'tmpsInfect', 'tmpsQuar', 'tmpsVoyage']

raw_dataset = pd.read_csv(url, names=column_names,
                          na_values='?', comment='\t',
                          sep=',', skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()

unit = 'txQuar'
```

```
     2.7.0
```

```
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

train_dataset.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
tx	662.0	81.767372	60.033301	1.0	7.00	106.00	139.00	150.0
nbr	662.0	150.000000	0.000000	150.0	150.00	150.00	150.00	150.0
txVaccin	662.0	0.000000	0.000000	0.0	0.00	0.00	0.00	0.0
txQuar	662.0	0.498807	0.296450	0.0	0.23	0.51	0.77	1.0
txInfect	662.0	0.500000	0.000000	0.5	0.50	0.50	0.50	0.5
tmpsInfect	662.0	4.000000	0.000000	4.0	4.00	4.00	4.00	4.0
tmpsQuar	662.0	1.000000	0.000000	1.0	1.00	1.00	1.00	1.0
tmpsVoyage	662.0	100.000000	0.000000	100.0	100.00	100.00	100.00	100.0

```
sns.pairplot(train_dataset[['tx', unit]], diag_kind='kde')
```

```
<seaborn.axisgrid.PairGrid at 0x7f02726ccb90>
```

```
#coder fct pour supprimer les valeurs trop absurdes
train_features = train_dataset.copy()
test = train_features.pop(unit)
toto = train_features.pop('tx')
l1 = test.values.tolist()
l2 = toto.values.tolist()

print(len(l1))

fin = []

for i in range (0,10):
  a = i/10
  b = a + 0.1

  l = []
  for j in range(len(l1)):
    p = l1[j]
    if (p >= a) and (p <= b):
      l.append(j)


  sumi = 0

  for u in l:
    sumi = sumi + l2[u]

  if len(l) == 0:
    print()
  else:
    sumi = sumi / len(l)
    #print(sumi)
    error = sumi / 3


    for k in l:
      if (l2[k] < sumi + error) and (l2[k] > sumi - error):
        fin.append(k)

print(fin)
```

```
662
[2, 10, 32, 36, 40, 53, 69, 71, 75, 80, 82, 86, 98, 99, 110, 113, 135, 156, 162, 167, 174, 182, 184, 187, 226, 251, 267, 269, 2
```

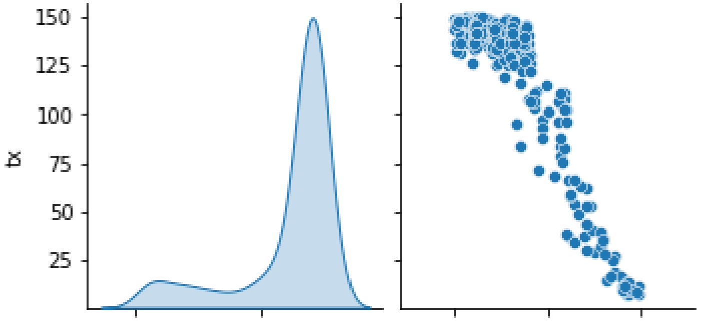
```
data = []
for j in fin:
  data.append([l1[j],l2[j]])

df = pd.DataFrame(data, columns = [unit, 'tx'])
```

```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
txQuar	353.0	0.297394	0.241280	0.0	0.11	0.23	0.38	0.99
tx	353.0	122.135977	38.554178	8.0	126.00	139.00	144.00	150.00

```
sns.pairplot(df[['tx', unit]], diag_kind='kde')
```

```
<seaborn.axisgrid.PairGrid at 0x7f0267363190>
```



```
dataset = df.copy()
dataset.tail()

train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

train_dataset.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
txQuar	282.0	0.291631	0.239858	0.0	0.1	0.23	0.37	0.99
tx	282.0	122.751773	38.459695	8.0	129.0	139.00	144.00	150.00

```
train_features = train_dataset.copy()
test_features = test_dataset.copy()

train_labels = train_features.pop('tx')
test_labels = test_features.pop('tx')


normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(np.array(train_features))

horsepower = np.array(train_features[unit])
```

```
horsepower_normalizer = layers.Normalization(input_shape=[1,], axis=None)
horsepower_normalizer.adapt(horsepower)
```

```
def build_and_compile_model(norm):
  model = keras.Sequential([
      norm,
      layers.Dense(64, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(1)
  ])

  model.compile(loss='mean_absolute_error',
                optimizer=tf.keras.optimizers.Adam(0.001))
  return model
```

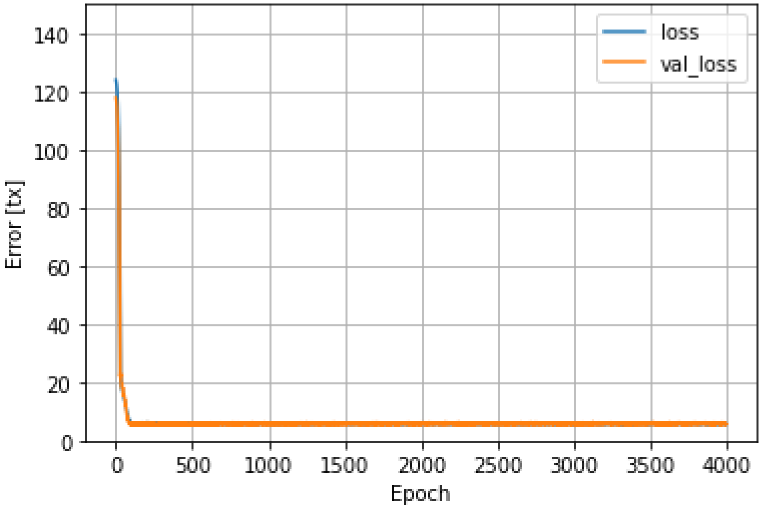
```
dnn_horsepower_model = build_and_compile_model(horsepower_normalizer)
dnn_horsepower_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normalizat  (None, 1)                3
 ion)

 dense (Dense)               (None, 64)                128

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 4,356
Trainable params: 4,353
Non-trainable params: 3
_________________________________________________________________
```

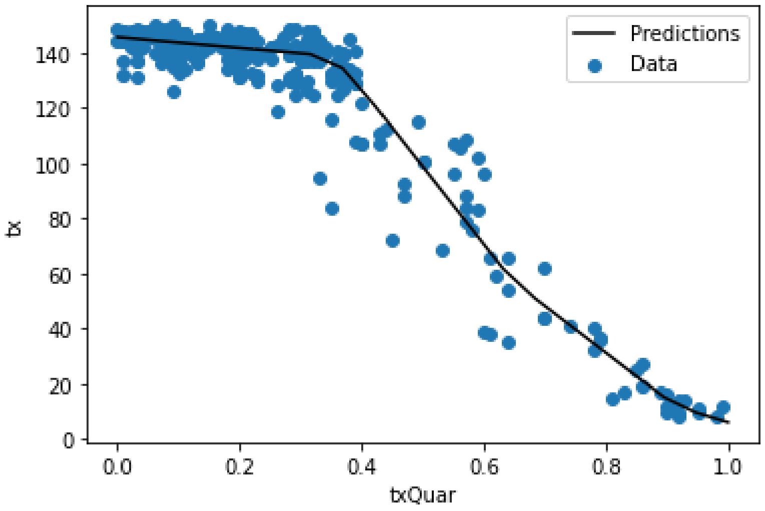
```
%%time
history = dnn_horsepower_model.fit(
    train_features[unit],
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=4000)
```

```
CPU times: user 2min 18s, sys: 7.16 s, total: 2min 25s
Wall time: 2min 22s
```

```
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 150])
  plt.xlabel('Epoch')
  plt.ylabel('Error [tx]')
  plt.legend()
  plt.grid(True)

plot_loss(history)
```



```
x = tf.linspace(0.0, 1, 20)
y = dnn_horsepower_model.predict(x)

def plot_horsepower(x, y):
  plt.scatter(train_features[unit], train_labels, label='Data')
  plt.plot(x, y, color='k', label='Predictions')
  plt.xlabel(unit)
  plt.ylabel('tx')

  plt.legend()

plot_horsepower(x, y)
```



```
dnn_horsepower_model.evaluate(
    test_features[unit], test_labels,
    verbose=0)
```

```
     5.158871650695801
```

```
dnn_horsepower_model.predict([0.24])
```

```
array([[108.224]], dtype=float32)
```

```
#lancer avec txInfect entre 0.2 et 0.4
```

✓ 0 s terminée à 21:06 ● ×