```
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/exemple3.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 = 'txInfect'
```

```
     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	334.0	56.209581	54.083167	1.0	3.0	36.00	112.00	171.00
nbr	334.0	150.000000	0.000000	150.0	150.0	150.00	150.00	150.00
txVaccin	334.0	0.475898	0.291852	0.0	0.2	0.47	0.74	0.99
txQuar	334.0	0.000000	0.000000	0.0	0.0	0.00	0.00	0.00
txInfect	334.0	0.697605	0.021788	0.5	0.7	0.70	0.70	0.70
tmpsInfect	334.0	3.968563	0.215188	2.5	4.0	4.00	4.00	4.00
tmpsQuar	334.0	0.000000	0.000000	0.0	0.0	0.00	0.00	0.00
tmpsVoyage	334.0	100.000000	0.000000	100.0	100.0	100.00	100.00	100.00

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

```
<seaborn.axisgrid.PairGrid at 0x7ff5eed25d90>
```

```
#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 / 2


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

print(fin)
```

```
122
[0, 4, 5, 15, 16, 21, 30, 34, 38, 42, 44, 47, 50, 71, 74, 75, 78, 81, 86, 92, 97, 101, 103, 105, 109, 111, 115, 118, 2, 3, 6, 9
```

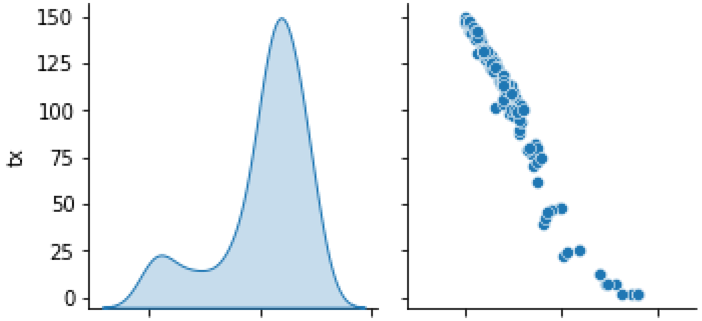
```
data = []
for j in fin:
  data.append([l1[j],l2[j]])

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

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

	count	mean	std	min	25%	50%	75%	max
txVaccin	138.0	0.249203	0.208302	0.0	0.1225	0.2	0.2875	0.9
tx	138.0	102.565217	40.006196	2.0	95.5000	114.5	128.0000	150.0

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

```
<seaborn.axisgrid.PairGrid at 0x7ff5db7c5ad0>
```



```
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
txVaccin	110.0	0.250545	0.209577	0.0	0.1125	0.2	0.3	0.9
tx	110.0	102.072727	40.233883	2.0	88.5000	115.0	130.5	150.0

```
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
```

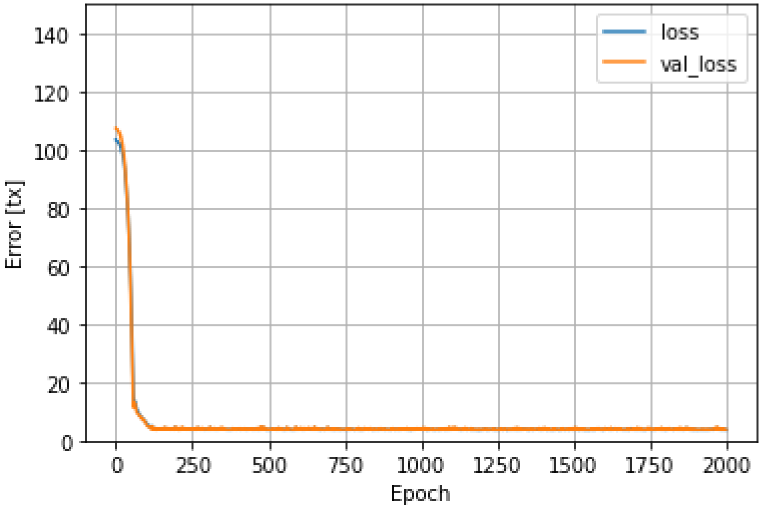
```
dnn_horsepower_model = build_and_compile_model(horsepower_normalizer)
dnn_horsepower_model.summary()
```

```
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
_________________________________________________________________
```

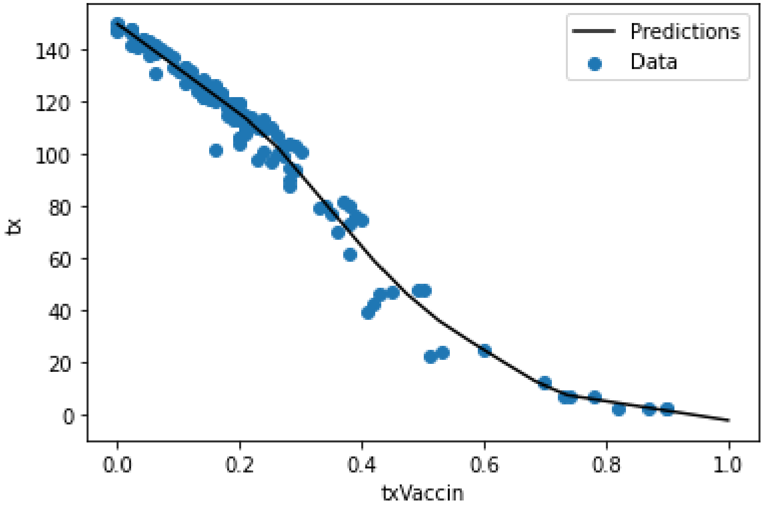
```
%%time
history = dnn_horsepower_model.fit(
    train_features[unit],
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=2000)
```

```
CPU times: user 50.3 s, sys: 2.52 s, total: 52.8 s
Wall time: 48.3 s
```

```
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)
```

```
3.0833375453948975
```

```
dnn_horsepower_model.predict([0.74])
```

```
    array([[7.353]], dtype=float32)

#lancer avec txInfect entre 0.2 et 0.4
```

✓ 0 s terminée à 15:02 ● ×