```
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	740.0	49.583784	4.378947e+01	1.0	3.0000	43.50	94.00	117.0
nbr	740.0	150.000000	0.000000e+00	150.0	150.0000	150.00	150.00	150.0
txVaccin	740.0	0.200000	1.944204e-16	0.2	0.2000	0.20	0.20	0.2
txQuar	740.0	0.000000	0.000000e+00	0.0	0.0000	0.00	0.00	0.0
txInfect	740.0	0.521432	2.880800e-01	0.0	0.2675	0.54	0.77	1.0
tmpsInfect	740.0	2.500000	0.000000e+00	2.5	2.5000	2.50	2.50	2.5
tmpsQuar	740.0	0.000000	0.000000e+00	0.0	0.0000	0.00	0.00	0.0
tmpsVoyage	740.0	100.000000	0.000000e+00	100.0	100.0000	100.00	100.00	100.0

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

```
<seaborn.axisgrid.PairGrid at 0x7f38a12c2e90>
```

```
#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 / 5


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

print(fin)
```

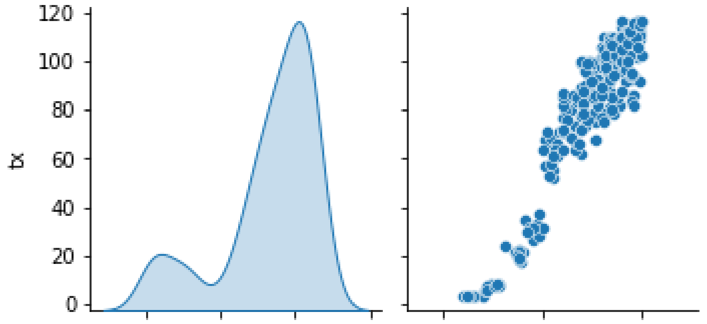
```
740
[0, 90, 241, 341, 407, 436, 593, 616, 722, 733, 71, 172, 286, 311, 339, 386, 446, 506, 689, 37, 313, 323, 442, 450, 510, 519, 6
```

```
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
txInfect	261.0	0.741073	0.222136	0.1	0.63	0.81	0.92	1.0
tx	261.0	82.402299	31.665120	3.0	72.00	92.00	106.00	117.0

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

```
<seaborn.axisgrid.PairGrid at 0x7f38a1193a50>
```



```
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
txInfect	209.0	0.731579	0.227099	0.1	0.61	0.8	0.9	1.0
tx	209.0	81.138756	32.417423	3.0	71.00	91.0	106.0	117.0

```
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()
```

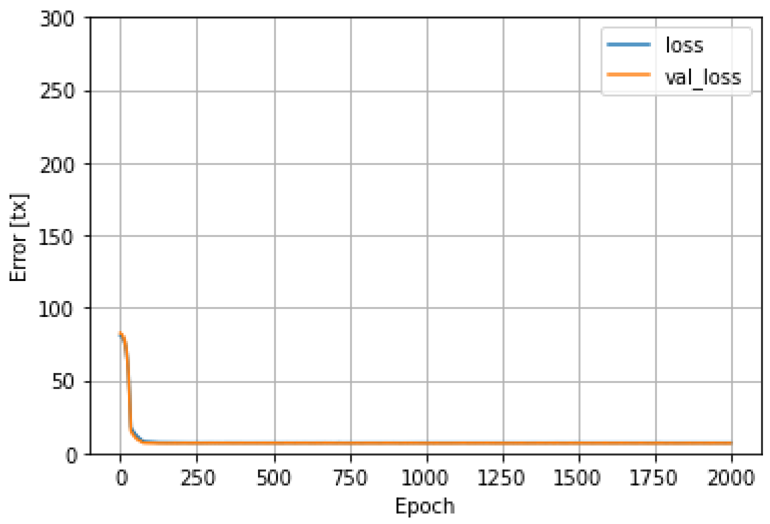
```
     Model: "sequential_13"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      normalization_26 (Normaliza  (None, 1)                3
      tion)

      dense_39 (Dense)            (None, 64)                128

      dense_40 (Dense)            (None, 64)                4160

      dense_41 (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)
```

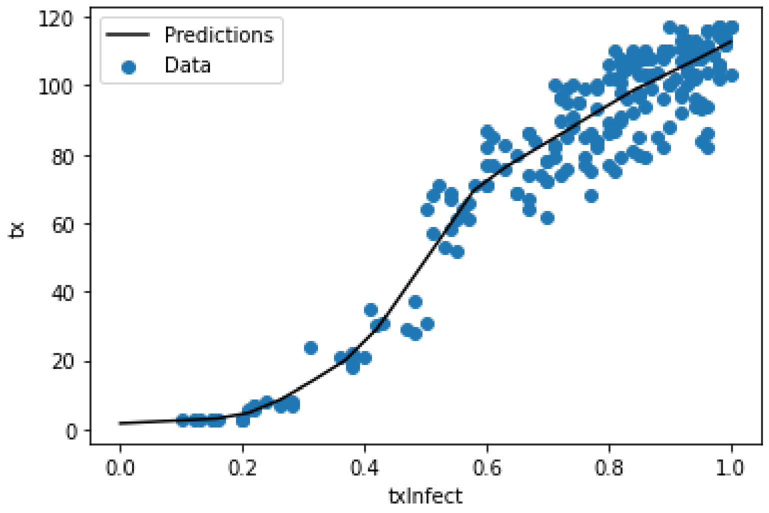
```
     CPU times: user 1min 3s, sys: 3.23 s, total: 1min 6s
     Wall time: 1min 22s
```

```
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 300])
  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)
```

```
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)

     6.367334365844727


dnn_horsepower_model.predict([0.74])

     array([[87.847]], dtype=float32)
```

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

✓ 3 s terminée à 14:56