```
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/tmpsQuar.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 = 'tmpsQuar'
```

```
     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	1858.0	88.418192	4.135926e+01	1.00	81.00	109.00	116.00	120.0
nbr	1858.0	150.000000	0.000000e+00	150.00	150.00	150.00	150.00	150.0
txVaccin	1858.0	0.200000	6.774184e-15	0.20	0.20	0.20	0.20	0.2
txQuar	1858.0	0.500000	0.000000e+00	0.50	0.50	0.50	0.50	0.5
txInfect	1858.0	0.700000	2.443148e-14	0.70	0.70	0.70	0.70	0.7
tmpsInfect	1858.0	5.000000	0.000000e+00	5.00	5.00	5.00	5.00	5.0
tmpsQuar	1858.0	2.457605	1.427869e+00	0.01	1.25	2.45	3.68	5.0
tmpsVoyage	1858.0	100.000000	0.000000e+00	100.00	100.00	100.00	100.00	100.0

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

```
<seaborn.axisgrid.PairGrid at 0x7f3a5bf8cf50>
```

```
#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 (0,50):
  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 / 20
    for k in l:
      if (l2[k] < sumi + error) and (l2[k] > sumi - error):
        fin.append(k)

print(fin)
```

```
1858
[1222, 1102, 1685, 320, 552, 1487, 1187, 452, 1692, 1207, 696, 806, 160, 162, 1125, 1795, 1812, 44, 284, 367, 513, 680, 852, 87
```

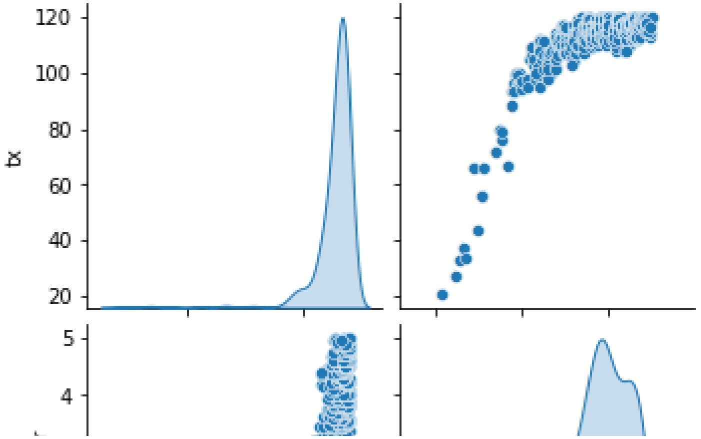
```
data = []
for j in fin:
  data.append([l1[j],l2[j]])

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

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

	count	mean	std	min	25%	50%	75%	max
tmpsQuar	772.0	3.711839	0.861771	0.15	3.21	3.83	4.385	5.0
tx	772.0	113.154145	10.247182	21.00	112.00	116.00	118.000	120.0

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

```
<seaborn.axisgrid.PairGrid at 0x7f3a50487810>
```



```
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
tmpsQuar	618.0	3.721634	0.862833	0.15	3.2525	3.84	4.38	5.0
tx	618.0	113.119741	10.841808	21.00	112.0000	116.00	118.00	120.0

```
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
```

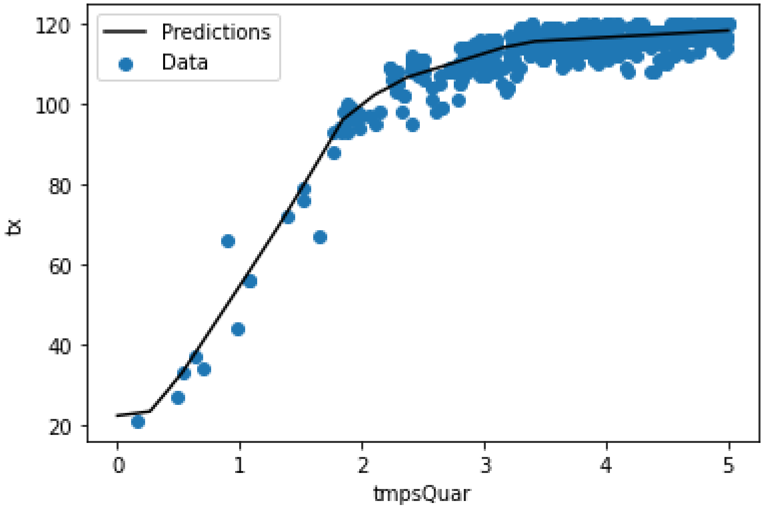
```
Non-trainable params: 3
_________________________________________________________________
```

```
%%time
history = dnn_horsepower_model.fit(
    train_features[unit],
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=500)
```

```
CPU times: user 23.7 s, sys: 1.35 s, total: 25 s
Wall time: 41.6 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, 5, 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)
```

```
     2.3207743167877197
```

```
dnn_horsepower_model.predict([1.5])
```

```
     array([[78.634]], dtype=float32)
```

```
dnn_horsepower_model.save('/content/model.h5')
```

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