```
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/nbr.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 = 'nbr'
```

```
     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** | 346.0 | 65.936416 | 5.090874e+01 | 1.0 | 11.50 | 66.0 | 109.00 | 158.0 |
| **nbr** | 346.0 | 100.075145 | 5.418835e+01 | 10.0 | 54.25 | 98.5 | 145.25 | 200.0 |
| **txVaccin** | 346.0 | 0.200000 | 1.278606e-15 | 0.2 | 0.20 | 0.2 | 0.20 | 0.2 |
| **txQuar** | 346.0 | 0.000000 | 0.000000e+00 | 0.0 | 0.00 | 0.0 | 0.00 | 0.0 |
| **txInfect** | 346.0 | 0.700000 | 4.224957e-15 | 0.7 | 0.70 | 0.7 | 0.70 | 0.7 |
| **tmpsInfect** | 346.0 | 4.000000 | 0.000000e+00 | 4.0 | 4.00 | 4.0 | 4.00 | 4.0 |
| **tmpsQuar** | 346.0 | 1.000000 | 0.000000e+00 | 1.0 | 1.00 | 1.0 | 1.00 | 1.0 |
| **tmpsVoyage** | 346.0 | 100.000000 | 0.000000e+00 | 100.0 | 100.00 | 100.0 | 100.00 | 100.0 |

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

```
<seaborn.axisgrid.PairGrid at 0x7f3a487f9a50>
```

```
#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 (0,200):
  a = i
  b = a + 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)
```

```
450
```

```
[106, 31, 50, 109, 145, 168, 169, 215, 222, 286, 376, 417, 435, 31, 50, 109, 145, 168, 169, 205, 215, 222, 236, 286, 376, 409,
```

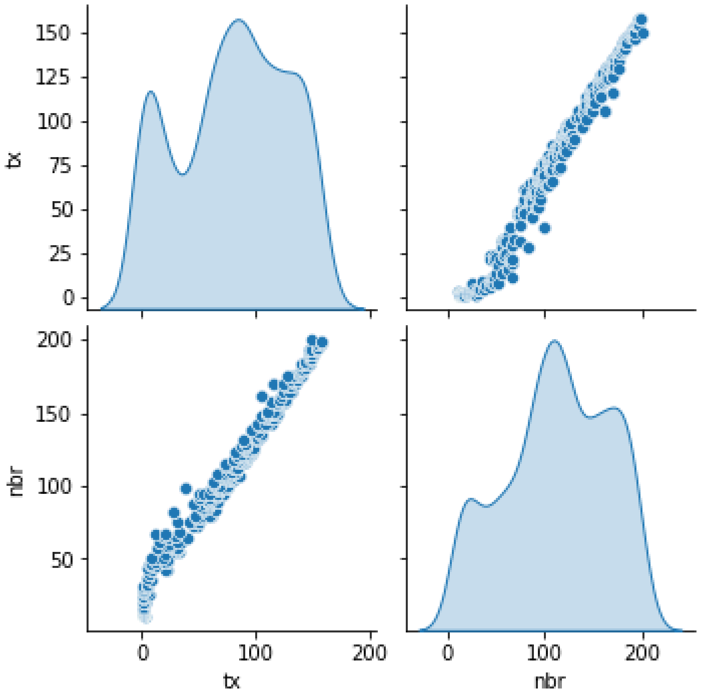
```
data = []
for j in fin:
  data.append([l1[j],l2[j]])

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

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

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **nbr** | 871.0 | 111.902411 | 52.786195 | 11.0 | 76.5 | 114.0 | 156.0 | 200.0 |
| **tx** | 871.0 | 79.448909 | 47.782186 | 1.0 | 45.5 | 84.0 | 119.0 | 158.0 |

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

```
<seaborn.axisgrid.PairGrid at 0x7f3a482fedd0>
```



```
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 |
|---|---|---|---|---|---|---|---|---|
| **nbr** | 697.0 | 112.649928 | 52.443190 | 11.0 | 79.0 | 115.0 | 156.0 | 200.0 |
| **tx** | 697.0 | 80.116212 | 47.459505 | 1.0 | 46.0 | 85.0 | 120.0 | 158.0 |

```
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
```

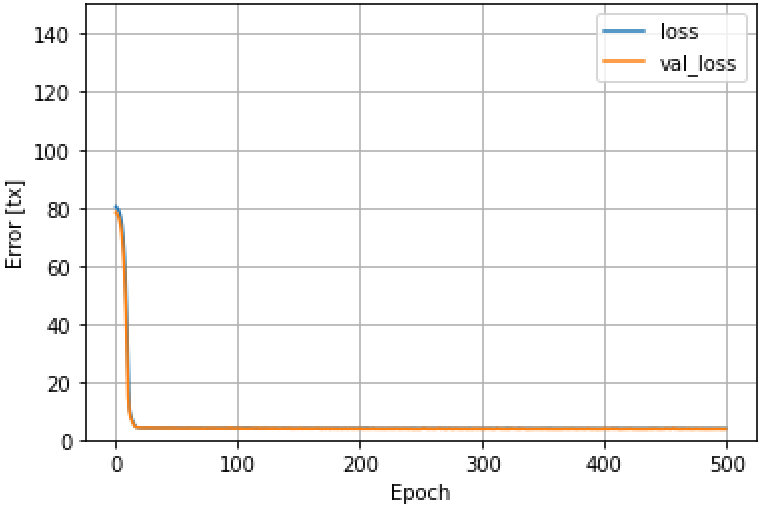
```
dnn_horsepower_model = build_and_compile_model(horsepower_normalizer)
dnn_horsepower_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_3 (Normalizat  (None, 1)                3
 ion)

 dense_3 (Dense)             (None, 64)                128

 dense_4 (Dense)             (None, 64)                4160

 dense_5 (Dense)             (None, 1)                 65

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

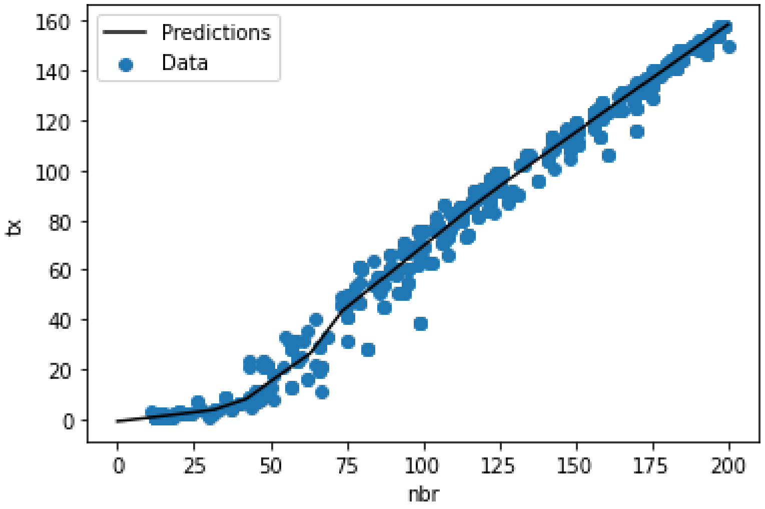
```
%%time
history = dnn_horsepower_model.fit(
    train_features[unit],
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=500)
```

```
CPU times: user 26.7 s, sys: 1.44 s, total: 28.1 s
Wall time: 26.4 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, 200, 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)
```

```
     3.367340326309204
```

```
dnn_horsepower_model.predict([80])
```

```
     array([[49.694]], dtype=float32)
```

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

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

✓ 0 s terminée à 08:35 ● ×

Impossible d'établir une connexion avec le service reCAPTCHA. Veuillez vérifier votre connexion Internet, puis actualiser la page pour afficher une image reCAPTCHA.