Set up

```
# Du code pour vérifier la VM sur laquelle je tourne
from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 359.2 gigabytes of available RAM

You are using a high-RAM runtime!
```

```
import os
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from google.colab import drive

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score, recall_score, precision_score
from sklearn.linear_model import LogisticRegression

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense



drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

Loading dataset

Ce dataset provient de : https://data.mendeley.com/datasets/c4n7fckkz3/3 Avec l'article associé : https://link.springer.com/article/10.1007/s10207-023-00723-w

```
folder_path="/content/drive/MyDrive/DNS_Exfiltration_Dataset /DNS_Exfiltration_Dataset/"
csv_file_path = os.path.join(folder_path, 'dataset.csv')
data = pd.read_csv(csv_file_path)


# Load the English words from file
with open(os.path.join(folder_path, 'english_words.txt'), 'r') as f:
    english_words = set(f.read().splitlines())

print(english_words)
```

```
{'deplanes', 'lined', 'entomology', 'germs', 'pointless', 'crystallizes', 'introspection', 'ones', 'preordaining', 'plan
```

```
# Adjust display options to ensure full content is shown
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)


data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35074150 entries, 0 to 35074149
Data columns (total 22 columns):
 #   Column               Dtype
---  ------               -----
 0   186.169.253.58       object
 1   surbl.org            object
 2   1624438272607        int64
 3   False                bool
 4   h.surbl.org          object
 5   1                    int64
 6   1.1                  int64
```

```
 7   0                   int64
 8   0.1                 int64
 9   -0.0                float64
 10  0.0                 float64
 11  0.0.1               float64
 12  0.0.2               float64
 13  0.0.3               float64
 14  3.4444444444444446  float64
 15  9.59311095410544    float64
 16  1.5                 float64
 17  1.5811388300841898  float64
 18  468.75              float64
 19  0.4444444444444444  float64
 20  0.25849625007211563 float64
 21  0.81743691684035    float64
dtypes: bool(1), float64(13), int64(5), object(3)
memory usage: 5.5+ GB
```

```
# On nomme les colonnes (d'après la documentation)
data.columns = [
    'user_ip', 'domain', 'timestamp', 'attack', 'request', 'len',
    'subdomains_count', 'w_count', 'w_max', 'entropy', 'w_max_ratio',
    'w_count_ratio', 'digits_ratio', 'uppercase_ratio', 'time_avg',
    'time_stdev', 'size_avg', 'size_stdev', 'throughput', 'unique',
    'entropy_avg', 'entropy_stdev'
]

# Vérification
print(data.columns)
data.info()
```

```
Index(['user_ip', 'domain', 'timestamp', 'attack', 'request', 'len',
       'subdomains_count', 'w_count', 'w_max', 'entropy', 'w_max_ratio',
       'w_count_ratio', 'digits_ratio', 'uppercase_ratio', 'time_avg',
       'time_stdev', 'size_avg', 'size_stdev', 'throughput', 'unique',
       'entropy_avg', 'entropy_stdev'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35074150 entries, 0 to 35074149
Data columns (total 22 columns):
 #   Column            Dtype
---  ------            -----
 0   user_ip           object
 1   domain            object
 2   timestamp         int64
 3   attack            bool
 4   request           object
 5   len               int64
 6   subdomains_count  int64
 7   w_count           int64
 8   w_max             int64
 9   entropy           float64
 10  w_max_ratio       float64
 11  w_count_ratio     float64
 12  digits_ratio      float64
 13  uppercase_ratio   float64
 14  time_avg          float64
 15  time_stdev        float64
 16  size_avg          float64
 17  size_stdev        float64
 18  throughput        float64
 19  unique            float64
 20  entropy_avg       float64
 21  entropy_stdev     float64
dtypes: bool(1), float64(13), int64(5), object(3)
memory usage: 5.5+ GB
```

```
# Check the first few rows of the dataset
print(data.head())
```

```
          user_ip        domain      timestamp  attack  \
0  186.169.253.58     surbl.org  1624438272607   False
1  186.169.253.58  spamhaus.org  1624438273058   False
2  186.169.253.58  spamhaus.org  1624438273058   False
3  186.169.253.58  spamhaus.org  1624438273059   False
4  186.169.253.58  spamhaus.org  1624438273059   False

                          request  len  subdomains_count  w_count  w_max  \
0                     f.surbl.org    1                 1        0      0
1  118.141.11.106.sbl.spamhaus.org   18                 5        0      0
2  118.141.11.106.zen.spamhaus.org   18                 5        1      3
3  128.141.11.106.sbl.spamhaus.org   18                 5        0      0
4  128.141.11.106.zen.spamhaus.org   18                 5        1      3

    entropy  w_max_ratio  w_count_ratio  digits_ratio  uppercase_ratio  \
0 -0.000000     0.000000       0.000000      0.000000              0.0
```

```
1  2.633731     0.000000       0.000000      0.611111              0.0
2  2.633731     0.166667       0.055556      0.611111              0.0
3  2.863826     0.000000       0.000000      0.611111              0.0
4  2.863826     0.166667       0.055556      0.611111              0.0

    time_avg  time_stdev  size_avg  size_stdev    throughput    unique  \
0   0.222222    0.440959       1.0    0.000000   3333.333333  0.555556
1  55.555556  165.542375      17.2    0.421637    343.313373  0.000000
2   0.333333    0.500000      17.2    0.421637  43000.000000  0.000000
3   0.333333    0.500000      17.3    0.483046  43250.000000  0.000000
4   0.333333    0.500000      17.4    0.516398  43500.000000  0.000000

   entropy_avg  entropy_stdev
0     0.000000       0.000000
1     3.048277       0.177285
2     2.983547       0.199622
3     2.959741       0.198131
4     2.935936       0.193400
```

```
# Check for any missing values
print(data.isnull().sum())
```

```
user_ip             0
domain              0
timestamp           0
attack              0
request             0
len                 0
subdomains_count    0
w_count             0
w_max               0
entropy             0
w_max_ratio         0
w_count_ratio       0
digits_ratio        0
uppercase_ratio     0
time_avg            0
time_stdev          0
size_avg            0
size_stdev          0
throughput          0
unique              0
entropy_avg         0
entropy_stdev       0
dtype: int64
```

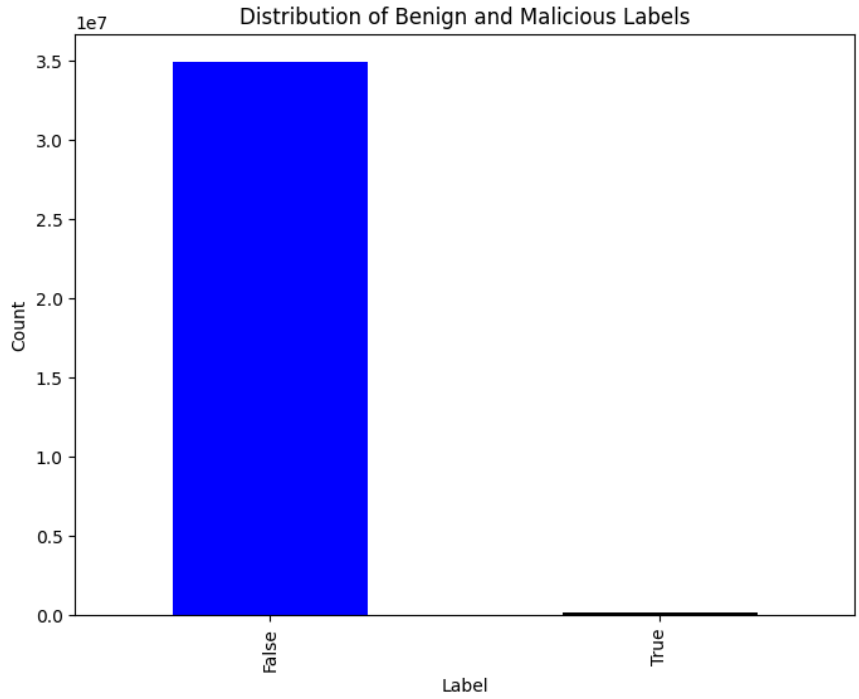
Visualisation du dataset

```
# Count the number of attacks
print(data['attack'].value_counts())

num_benign, num_attacks = data['attack'].value_counts()
ratio = num_attacks / (num_attacks + num_benign)
print("ratio of attacks in the dataset: ", ratio*100,"%")
```

```
attack
False    34899371
True       174779
Name: count, dtype: int64
ratio of attacks in the dataset:  0.49831286004079933 %
```

```
plt.figure(figsize=(8, 6))
data["attack"].value_counts().plot(kind='bar', color=['blue', 'black'])
plt.title('Distribution of Benign and Malicious Labels')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```

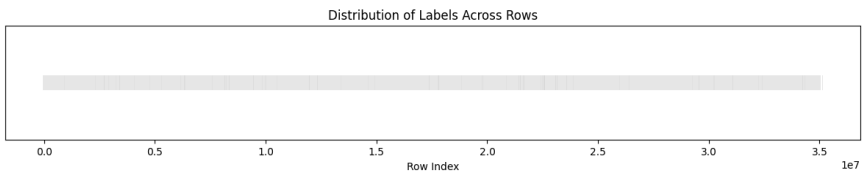



```
# Create a color map: True will be green, False will be red
color_map = data['attack'].map({True: 'black', False: 'white'})

# Plotting
plt.figure(figsize=(15, 2))

# Scatter plot where each point's color represents the label
plt.scatter(data.index, [1]*len(data), c=color_map, marker='|', s=200)

plt.xlabel('Row Index')
plt.yticks([])
plt.title('Distribution of Labels Across Rows')
plt.show()
```




```
#Printing some random rows
random_rows = data.sample(n=10, random_state=42)
print(random_rows)
```

```
                    src_ip                domain   timestamp  attack  \
24577790       186.169.157.176           spamhaus.org  1624517077166   False
33959090          130.9.48.128       root-servers.net  1624531070154   False
29079969           130.9.40.66            local.local  1624523951775   False
32988913        186.169.134.52                bg.ac.rs  1624529568649   False
18346957           130.9.40.66            local.local  1624491628548   False
26706171          130.9.48.179  googlesyndication.com  1624520513843   False
15605255  6c47:8c4e:2567::62e7                bg.ac.rs  1624479125869   False
27688005           130.9.47.63       root-servers.net  1624521973857   False
24907918           130.9.40.66            local.local  1624517631226   False
17026505           130.9.40.66            local.local  1624485232008   False

                             request  len  subdomains_count  w_count  \
24577790  7.22.142.174.zen.spamhaus.org   16                 5        1
33959090             a.root-servers.net    1                 1        0
29079969              samba.local.local    5                 1        4
32988913                  www.fpu.bg.ac.rs    7                 2        0
18346957              samba.local.local    5                 1        4
26706171      tpc.googlesyndication.com    3                 1        0
15605255                ns.rcub.bg.ac.rs    7                 2        1
```

```
24907918              samba.local.local    5                1        4
17026505              samba.local.local    5                1        4

          w_max   entropy  w_max_ratio  w_count_ratio  digits_ratio  \
24577790      3  2.827820     0.187500       0.062500        0.5625
33959090      0 -0.000000     0.000000       0.000000        0.0000
29079969      5  1.921928     1.000000       0.800000        0.0000
32988913      0  2.128085     0.000000       0.000000        0.0000
18346957      5  1.921928     1.000000       0.800000        0.0000
26706171      0  1.584963     0.000000       0.000000        0.0000
15605255      3  2.807355     0.428571       0.142857        0.0000
27688005      0 -0.000000     0.000000       0.000000        0.0000
24907918      5  1.921928     1.000000       0.800000        0.0000
17026505      5  1.921928     1.000000       0.800000        0.0000

          uppercase_ratio       time_avg    time_stdev  size_avg  size_stdev  \
24577790              0.0       3.777778  3.270236e+00      18.3    3.301515
33959090              0.0   31031.111111  3.333333e-01       1.0    0.000000
29079969              0.0      11.888889  1.105039e+01       5.0    0.000000
32988913              0.0  989073.555556  1.964106e+06       7.7    0.948683
18346957              0.0       9.888889  9.157571e+00       5.0    0.000000
26706171              0.0   11525.000000  1.291751e+04       8.9   11.798776
15605255              0.0  242450.777778  4.553647e+05       5.4    2.221111
27688005              0.0   10374.777778  9.619658e+03       1.0    0.000000
24907918              0.0       8.222222  9.297550e+00       5.0    0.000000
17026505              0.0      11.111111  1.037358e+01       5.0    0.000000

           throughput    unique  entropy_avg  entropy_stdev
24577790  5228.571429  0.000000     3.323672       0.365614
33959090     0.035806  1.000000     0.000000       0.000000
29079969   462.962963  1.000000     1.921928       0.000000
32988913     0.008650  0.777778     2.358491       0.365534
18346957   555.555556  1.000000     1.921928       0.000000
26706171     0.858030  0.777778     2.303968       0.781771
15605255     0.024747  0.444444     2.274853       0.526870
27688005     0.107096  1.000000     0.000000       0.000000
24907918   666.666667  1.000000     1.921928       0.000000
17026505   495.049505  1.000000     1.921928       0.000000
```

```
# Printing the first 10 attacks
attack_rows = data[data['attack'] == True].head(10)
print(attack_rows)
```

```
                 user_ip  domain      timestamp  attack  \
3942  186.169.146.147  e5.sk  1624438294225    True
4297  186.169.146.147  e5.sk  1624438295586    True
4590  186.169.146.147  e5.sk  1624438296656    True
6096   186.169.127.58  e5.sk  1624438302237    True
6187  186.169.146.147  e5.sk  1624438302672    True
6495   186.169.127.58  e5.sk  1624438303710    True
6724   186.169.127.58  e5.sk  1624438304691    True
6968   186.169.127.58  e5.sk  1624438305372    True
7721   186.169.127.58  e5.sk  1624438308174    True
7722   186.169.127.58  e5.sk  1624438308181    True

                                          request  len  subdomains_count  \
3942  sebubx76xk4erpp3rwehoo3ubmbqeaqbaeaq.a.e.e5.sk   40                 3
4297  4az3kiecotwu3okbtvfm7pdpcabqeaqbaeaq.a.e.e5.sk   40                 3
4590  x3i2wbqsiucuviqyfaaoxz3lzybqeaqbaeaq.a.e.e5.sk   40                 3
6096  ez2vzwchw3ce5m6wz6cw3nnc2ibqeaqbaeaq.a.e.e5.sk   40                 3
6187  htm7xrligq2enc4lsjhkzdnd6mbqeaqbaeaq.a.e.e5.sk   40                 3
6495  f4clwtzqaonejfevfnc3vnm334bqeaqbaeaq.a.e.e5.sk   40                 3
6724  hshm7dgsfuvungjbsgjocfazoibqeaqbaeaq.a.e.e5.sk   40                 3
6968  uk7xg4v2usyupazkwfjietmf3ybqeaqbaeaq.a.e.e5.sk   40                 3
7721  ijjuunvalweehk2jgbquu2atwabqeaqbaeaq.a.e.e5.sk   40                 3
7722  mnwmw2m3timeblpdxzjqmnvf3ibqeaqbaeaq.a.e.e5.sk   40                 3

      w_count  w_max   entropy  w_max_ratio  w_count_ratio  digits_ratio  \
3942        3      3  3.975071        0.075          0.075         0.125
4297        5      3  4.146439        0.075          0.125         0.100
4590        1      3  3.987326        0.075          0.025         0.075
6096        1      3  3.893943        0.075          0.025         0.175
6187        3      3  4.371928        0.075          0.075         0.100
6495        2      3  3.934830        0.075          0.050         0.125
6724        3      3  4.137326        0.075          0.075         0.025
6968        2      3  4.343943        0.075          0.050         0.100
7721        6      3  3.752656        0.075          0.150         0.050
7722        3      4  4.062815        0.100          0.075         0.075

      uppercase_ratio     time_avg   time_stdev  size_avg  size_stdev  \
3942              0.0  2197.222222  2875.261022      48.2   53.370404
4297              0.0  2348.444444  2779.448601      48.2   53.370404
4590              0.0  2460.111111  2695.151964      51.8   51.228898
6096              0.0  1799.222222  1935.781934      44.0   27.712813
6187              0.0  3105.444444  2782.422466      51.8   51.228898
6495              0.0  1382.111111  1447.797417      44.0   27.712813
6724              0.0  1327.555556  1453.227538      44.0   27.712813
6968              0.0   852.555556   514.388256      44.0   27.712813
7721              0.0  1163.888889   734.667347      47.6   24.033310
7722              0.0  1006.777778   819.147389      40.0    0.000000
```

	throughput	unique	entropy_avg	entropy_stdev
3942	24.372977	0.0	3.691242	0.910175
4297	22.803615	0.0	3.685581	0.906808
4590	23.394454	0.0	3.884313	0.687639
6096	27.170557	0.0	3.835620	0.663023
6187	18.533095	0.0	3.905225	0.700116
6495	35.369775	0.0	3.824709	0.660105
6724	36.823165	0.0	3.813797	0.653225
6968	57.336461	0.0	3.861596	0.674604
7721	45.437190	0.0	4.036861	0.192920
7722	44.140366	0.0	4.010757	0.165480

1. Training on the data provided : only "single" features

1.1 Splitting the data

```
shuffled_data_0 = data.sample(frac=1).reset_index(drop=True)
```

```
shuffled_data_0.head(10)
```

	user_ip	domain	timestamp	attack	request	len	su
0	186.169.145.58	ampproject.org	1624444275435	False	cdn-content.ampproject.org	11	
1	130.9.48.79	telephony.goog	1624521121964	False	rs-mts.rcs.telephony.goog	10	
2	186.169.123.159	kaspersky-labs.com	1624523977988	False	ksn-crypto-info-geo.kaspersky-labs.com	19	
3	186.169.114.79	pki.goog	1624448889645	False	ocsp.pki.goog	4	
4	130.9.40.66	local.local	1624515813306	False	samba.local.local	5	
5	186.169.4.59	bg.ac.rs	1624523282263	False	proxy.rcub.bg.ac.rs	10	
6	130.9.40.66	local.local	1624451279934	False	samba.local.local	5	
7	130.9.38.92	root-servers.net	1624503727627	False	a.root-servers.net	1	
8	233.132.83.180	mikrotik.com	1624453303726	False	cloud.mikrotik.com	5	
9	130.9.48.195	msftncsi.com	1624493617802	False	dns.msftncsi.com	3	

```
shuffled_data_0[shuffled_data_0['attack'] == True].head(10)
```

	user_ip	domain	timestamp	attack	
201	199.177.247.90	e5.sk	1624447625053	True	
305	186.169.150.60	e5.sk	1624457178293	True	zrpsuaacaakmktzoahbe6lwsaqa
414	186.169.175.72	e5.sk	1624459480374	True	
574	186.169.146.147	e5.sk	1624450213304	True	
590	186.169.127.58	e5.sk	1624528188340	True	
722	186.169.146.147	e5.sk	1624529184853	True	
729	199.177.247.53	e5.sk	1624442338674	True	7vgsuaacaakmi77cahbh7ywsaqaaalw
1131	186.169.4.59	e5.sk	1624530694898	True	
1260	186.169.146.147	e5.sk	1624450503676	True	
1378	186.169.139.60	e5.sk	1624521946272	True	

```
# variables
x0 = shuffled_data_0[['len','subdomains_count','w_count','w_count_ratio','w_max','w_max_ratio', 'digits_ratio','entropy']]

# target
y0 = shuffled_data_0['attack']


# splitting the data
x_train0, x_test0, y_train0, y_test0 = train_test_split(x0, y0, train_size=0.8)

print(y_train0.shape)
```

```
print(x_train0.shape)
print(x_test0.shape)
print(y_train0.shape)
print(y_test0.shape)
```

```
(28059320, 8)
(7014830, 8)
(28059320,)
(7014830,)
```

```
print(y_train0.value_counts())
print(y_test0.value_counts())
```

```
attack
False    27919212
True       140108
Name: count, dtype: int64
attack
False    6980159
True       34671
Name: count, dtype: int64
```

```
y_test0.head()
```

```
29283865    False
19184558    False
27011927    False
31828242    False
25171135    False
Name: attack, dtype: bool
```

Même ratio de attack/overall dans le train quand dans le test (0.5%)

```
x_train0.head()
```

	len	subdomains_count	w_count	w_count_ratio	w_max	w_max_ratio	dig
8721718	10	1	11	1.100000	5	0.500000	
1533542	5	1	0	0.000000	0	0.000000	
14560664	4	1	0	0.000000	0	0.000000	
33411955	20	5	4	0.200000	4	0.200000	
4161846	21	2	18	0.857143	8	0.380952	

```
y_train0.head()
```

```
8721718     False
1533542     False
14560664    False
33411955    False
4161846     False
Name: attack, dtype: bool
```

1.2 The first model : a Logistic Regression

```
model01 = LogisticRegression(max_iter=10000)
model01.fit(x_train0, y_train0)
```

```
LogisticRegression(max_iter=10000)
```

```
train_predictions = model01.predict(x_train0)
train_score = accuracy_score(y_train0, train_predictions)
print(f"Training accuracy : {train_score}")
recall = recall_score(y_train0, train_predictions)
print(f"Recall score: {recall}")
print(confusion_matrix(y_train0, train_predictions))
```

```
Training accuracy : 0.9977963471673583
Recall score: 0.7226639449567477
[[27896236    22976]
 [   38857   101251]]
```

```
predictions = model01.predict(x_test0)
test_score = accuracy_score(y_test0, predictions)
print(f"Testing accuracy : {test_score}")
recall = recall_score(y_test0, predictions)
print(f"Recall score: {recall_score}")
print(confusion_matrix(y_test0,predictions))
```

```
Testing accuracy : 0.9978073595511224
Recall score: <function recall_score at 0x7cd27966d990>
[[6974333    5826]
 [   9555   25116]]
```

Vraiment pas terrible. C'est d'ailleurs les résultats du papier, donc pas de grosse surprise. L'accuracy est vraiment pas un bon représentant de la performance étant donnée la petite proportion de "true". Là, on détecte 72% des attaques

1.3 The second model: a MLP

```
# Define the model architecture
model02 = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(x_train0.shape[1],)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model02.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Precision()])

# Print the model summary
model02.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 64)                576

 dropout_6 (Dropout)         (None, 64)                0

 dense_10 (Dense)            (None, 64)                4160

 dropout_7 (Dropout)         (None, 64)                0

 dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 4801 (18.75 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the model
history = model02.fit(x_train0, y_train0, epochs=4, batch_size=2048, validation_data=(x_test0, y_test0))
```

```
Epoch 1/4
13701/13701 [==============================] - 79s 6ms/step - loss: 0.0080 - accuracy: 0.9981 - precision_3: 0.7928 - va
Epoch 2/4
13701/13701 [==============================] - 75s 5ms/step - loss: 0.0023 - accuracy: 0.9996 - precision_3: 0.9673 - va
Epoch 3/4
13701/13701 [==============================] - 74s 5ms/step - loss: 0.0021 - accuracy: 0.9996 - precision_3: 0.9707 - va
Epoch 4/4
13701/13701 [==============================] - 75s 6ms/step - loss: 0.0020 - accuracy: 0.9996 - precision_3: 0.9744 - va
```

```
# Make predictions on the test set
y_pred = model02.predict(x_test0)
y_pred_classes = (y_pred > 0.5).astype("int32")  # Assuming a binary classification with a threshold of 0.5
```

```
219214/219214 [==============================] - 266s 1ms/step
```

```
# Compute the confusion matrix
cm = confusion_matrix(y_test0, y_pred_classes)
print(cm)
```

```
[[6979991     168]
 [   1538   33133]]
```

Super résultats !!!! C'est quasi parfait sur les false, et c'est pas si mal sur les true : 96% des attaques ont été détectées !

2. Training with data from scratch

```
# Going from scratch
data_ini = data[['request', 'attack']]
```

2.1 Preprocessing

```
# Calculate the length of the request (excluding the TLD)
def calculate_len(request):
    parts = request.split('.')
    tld_length = len(parts[-1]) + len(parts[-2])
    return len(request) - tld_length - 2  # exclude the 2 dots

# Calculate number of subdomains
def calculate_subdomains(request):
    return request.count('.') - 1

# Count the number of English words in the request
def calculate_w_count(request):
    delimiters = ['.', '-', '_']
    words = request
    for delimiter in delimiters:
        words = words.replace(delimiter, ' ')
    words = words.split()
    #print(words)

    count = 0
    for word in words:
        if word.lower() in english_words:
            count += 1
            #print(word)
    return count


# Calculate the length of the longest English word in the request
def calculate_longest_word_length(request):
    delimiters = ['.', '-', '_']
    words = request
    for delimiter in delimiters:
        words = words.replace(delimiter, ' ')
    words = words.split()

    longest_length = 0
    for word in words:
        if word.lower() in english_words:
            if len(word) > longest_length:
                longest_length = len(word)

    return longest_length


# Calculate the percentage of digits in the request
def count_digits(text):
    return sum(char.isdigit() for char in text)


# Calculate the entropy of the request
def calculate_entropy(request):
    probability = [float(request.count(c)) / len(request) for c in dict.fromkeys(list(request))]
    entropy = - sum([p * math.log(p) / math.log(2.0) for p in probability])
    return entropy

# Function to calculate entropy of a string
def calculate_entropy(text):
    if len(text) == 0:
        return 0.0

    # Calculate frequency of each character
    freq = {}
    for char in text:
        if char in freq:
            freq[char] += 1
        else:
            freq[char] = 1

    # Calculate entropy
    entropy = 0.0
    text_length = len(text)
    for count in freq.values():
        probability = count / text_length
        entropy -= probability * math.log2(probability)

    return entropy



data_test=data_ini.head(30)


data_test['len'] = data_test['request'].apply(calculate_len)
```

```
data_test['subdomains_count'] = data_test['request'].apply(calculate_subdomains)
data_test['w_count'] = data_test['request'].apply(calculate_w_count)
data_test['w_count_ratio'] = data_test['w_count'] / data_test['len']
data_test['w_max'] = data_test['request'].apply(calculate_longest_word_length)
data_test['w_max_ratio'] = data_test['w_max'] / data_test['len']
data_test['digit_count'] = data_test['request'].apply(count_digits)
data_test['digit_ratio'] = data_test['digit_count'] / data_test['len']
data_test['entropy'] = data_test['request'].apply(calculate_entropy)

data_test.head(30)
```

```
<ipython-input-113-adc5d2d8e2c1>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['len'] = data_test['request'].apply(calculate_len)
<ipython-input-113-adc5d2d8e2c1>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['subdomains_count'] = data_test['request'].apply(calculate_subdomains)
<ipython-input-113-adc5d2d8e2c1>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['w_count'] = data_test['request'].apply(calculate_w_count)
<ipython-input-113-adc5d2d8e2c1>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['w_count_ratio'] = data_test['w_count'] / data_test['len']
<ipython-input-113-adc5d2d8e2c1>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['w_max'] = data_test['request'].apply(calculate_longest_word_length)
<ipython-input-113-adc5d2d8e2c1>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['w_max_ratio'] = data_test['w_max'] / data_test['len']
<ipython-input-113-adc5d2d8e2c1>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['digit_count'] = data_test['request'].apply(count_digits)
<ipython-input-113-adc5d2d8e2c1>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['digit_ratio'] = data_test['digit_count'] / data_test['len']
<ipython-input-113-adc5d2d8e2c1>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_test['entropy'] = data_test['request'].apply(calculate_entropy)
```

	request	attack	len	subdomains_count	w_count	w_count_ratio	w_max	w_max_ratio	digit_count	dig
0	f.surbl.org	False	1	1	0	0.000000	0	0.000000	0	
1	118.141.11.106.sbl.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11	
2	118.141.11.106.zen.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11	
3	128.141.11.106.sbl.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11	
4	128.141.11.106.zen.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11	
5	68.211.11.106.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
6	68.211.11.106.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
7	28.41.205.140.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
8	28.41.205.140.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
9	17.41.205.140.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
10	17.41.205.140.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
11	127.141.11.106.sbl.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11	
12	127.141.11.106.zen.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11	
13	67.211.11.106.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
14	67.211.11.106.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
15	57.211.11.106.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
16	57.211.11.106.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
17	27.41.205.140.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	
18	27.41.205.140.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10	

19	17.81.205.140.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10
20	17.81.205.140.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10
21	117.141.11.106.sbl.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11
22	117.141.11.106.zen.spamhaus.org	False	18	5	0	0.000000	0	0.000000	11
23	27.81.205.140.sbl.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10
24	27.81.205.140.zen.spamhaus.org	False	17	5	0	0.000000	0	0.000000	10
25	20.177.89.47.sbl.spamhaus.org	False	16	5	0	0.000000	0	0.000000	9
26	20.177.89.47.zen.spamhaus.org	False	16	5	0	0.000000	0	0.000000	9
27	ksn-crypto-url-geo.kas-labs.com	False	18	1	1	0.055556	4	0.222222	0
28	ksn-crypto-url-geo.kaspersky-labs.com	False	18	1	1	0.055556	4	0.222222	0

Étapes suivantes : Générer du code avec `data_test` Afficher les graphiques recommandés

```
# Pour comparer avec les features initiales
data.head(30)
```

	user_ip	domain	timestamp	attack	request	len	subdomains_count	w_count	w_max	ent
0	186.169.253.58	surbl.org	1624438272607	False	f.surbl.org	1	1	0	0	-0.0
1	186.169.253.58	spamhaus.org	1624438273058	False	118.141.11.106.sbl.spamhaus.org	18	5	0	0	2.6
2	186.169.253.58	spamhaus.org	1624438273058	False	118.141.11.106.zen.spamhaus.org	18	5	1	3	2.6
3	186.169.253.58	spamhaus.org	1624438273059	False	128.141.11.106.sbl.spamhaus.org	18	5	0	0	2.8
4	186.169.253.58	spamhaus.org	1624438273059	False	128.141.11.106.zen.spamhaus.org	18	5	1	3	2.8
5	186.169.253.58	spamhaus.org	1624438273060	False	68.211.11.106.sbl.spamhaus.org	17	5	0	0	2.8
6	186.169.253.58	spamhaus.org	1624438273060	False	68.211.11.106.zen.spamhaus.org	17	5	1	3	2.8
7	186.169.253.58	spamhaus.org	1624438273060	False	28.41.205.140.sbl.spamhaus.org	17	5	0	0	3.1
8	186.169.253.58	spamhaus.org	1624438273061	False	28.41.205.140.zen.spamhaus.org	17	5	1	3	3.1
9	186.169.253.58	spamhaus.org	1624438273062	False	17.41.205.140.sbl.spamhaus.org	17	5	0	0	3.1
10	186.169.253.58	spamhaus.org	1624438273062	False	17.41.205.140.zen.spamhaus.org	17	5	1	3	3.1
11	186.169.253.58	spamhaus.org	1624438273063	False	127.141.11.106.sbl.spamhaus.org	18	5	0	0	2.8
12	186.169.253.58	spamhaus.org	1624438273063	False	127.141.11.106.zen.spamhaus.org	18	5	1	3	2.8
13	186.169.253.58	spamhaus.org	1624438273063	False	67.211.11.106.sbl.spamhaus.org	17	5	0	0	2.8
14	186.169.253.58	spamhaus.org	1624438273064	False	67.211.11.106.zen.spamhaus.org	17	5	1	3	2.8
15	186.169.253.58	spamhaus.org	1624438273064	False	57.211.11.106.sbl.spamhaus.org	17	5	0	0	2.9
16	186.169.253.58	spamhaus.org	1624438273064	False	57.211.11.106.zen.spamhaus.org	17	5	1	3	2.9
17	186.169.253.58	spamhaus.org	1624438273065	False	27.41.205.140.sbl.spamhaus.org	17	5	0	0	3.1
18	186.169.253.58	spamhaus.org	1624438273065	False	27.41.205.140.zen.spamhaus.org	17	5	1	3	3.1
19	186.169.253.58	spamhaus.org	1624438273066	False	17.81.205.140.sbl.spamhaus.org	17	5	0	0	3.2
20	186.169.253.58	spamhaus.org	1624438273066	False	17.81.205.140.zen.spamhaus.org	17	5	1	3	3.2
21	186.169.253.58	spamhaus.org	1624438273066	False	117.141.11.106.sbl.spamhaus.org	18	5	0	0	2.6
22	186.169.253.58	spamhaus.org	1624438273067	False	117.141.11.106.zen.spamhaus.org	18	5	1	3	2.6
23	186.169.253.58	spamhaus.org	1624438273067	False	27.81.205.140.sbl.spamhaus.org	17	5	0	0	3.2
24	186.169.253.58	spamhaus.org	1624438273067	False	27.81.205.140.zen.spamhaus.org	17	5	1	3	3.2
25	186.169.253.58	spamhaus.org	1624438273100	False	20.177.89.47.sbl.spamhaus.org	16	5	0	0	3.2
26	186.169.253.58	spamhaus.org	1624438273101	False	20.177.89.47.zen.spamhaus.org	16	5	1	3	3.2
27	186.169.123.159	kas-labs.com	1624438273201	False	ksn-crypto-url-geo.kas-labs.com	18	1	4	6	3.6
28	186.169.123.159	kaspersky-labs.com	1624438273204	False	ksn-crypto-url-geo.kaspersky-labs.com	18	1	4	6	3.6
29	186.169.123.159	kas-labs.com	1624438273206	False	ksn-crypto-verdict-geo.kas-labs.com	22	1	10	7	3.7

```
data_1 = data_ini
```

```
data_1['len'] = data_1['request'].apply(calculate_len)
data_1['subdomains_count'] = data_1['request'].apply(calculate_subdomains)
data_1['w_count'] = data_1['request'].apply(calculate_w_count)
data_1['w_count_ratio'] = data_1['w_count'] / data_1['len']
data_1['w_max'] = data_1['request'].apply(calculate_longest_word_length)
data_1['w_max_ratio'] = data_1['w_max'] / data_1['len']
data_1['digit_count'] = data_1['request'].apply(count_digits)
data_1['digit_ratio'] = data_1['digit_count'] / data_1['len']
data_1['entropy'] = data_1['request'].apply(calculate_entropy)
```

```
<ipython-input-116-ce7c54dcc958>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['len'] = data_1['request'].apply(calculate_len)
<ipython-input-116-ce7c54dcc958>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['subdomains_count'] = data_1['request'].apply(calculate_subdomains)
<ipython-input-116-ce7c54dcc958>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['w_count'] = data_1['request'].apply(calculate_w_count)
<ipython-input-116-ce7c54dcc958>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['w_count_ratio'] = data_1['w_count'] / data_1['len']
<ipython-input-116-ce7c54dcc958>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['w_max'] = data_1['request'].apply(calculate_longest_word_length)
<ipython-input-116-ce7c54dcc958>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['w_max_ratio'] = data_1['w_max'] / data_1['len']
<ipython-input-116-ce7c54dcc958>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['digit_count'] = data_1['request'].apply(count_digits)
<ipython-input-116-ce7c54dcc958>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['digit_ratio'] = data_1['digit_count'] / data_1['len']
<ipython-input-116-ce7c54dcc958>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  data_1['entropy'] = data_1['request'].apply(calculate_entropy)
```

```
data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35074150 entries, 0 to 35074149
Data columns (total 11 columns):
 #   Column            Dtype
---  ------            -----
 0   request           object
 1   attack            bool
 2   len               int64
 3   subdomains_count  int64
 4   w_count           int64
 5   w_count_ratio     float64
 6   w_max             int64
 7   w_max_ratio       float64
 8   digit_count       int64
 9   digit_ratio       float64
 10  entropy           float64
dtypes: bool(1), float64(4), int64(5), object(1)
memory usage: 2.6+ GB
```

2.2 Splitting the data

```
shuffled_data_1 = data_1.sample(frac=1).reset_index(drop=True)
```

```
shuffled_data_1.head(10)
```

	request	attack	len	subdomains_count	w_count	w_count_ratio
0	i-43.b-46010.ut.bench.utorrent.com	False	21	4	1	0.04761
1	samba.local.local	False	5	1	3	0.60000
2	mail.ffh.bg.ac.rs	False	11	3	1	0.09090
3	web.facebook.com	False	3	1	1	0.33333
4	www.google.com	False	3	1	0	0.00000
5	samba.local.local	False	5	1	3	0.60000
6	teams-events-data.trafficmanager.net	False	17	1	4	0.23529
7	samba.local.local	False	5	1	3	0.60000
8	ksn-cinfo-geo.kas-labs.com	False	13	1	1	0.07692

```
shuffled_data_1[shuffled_data_1['attack'] == True].head(10)
```

	request	attack	l
95	efysfpxsydxufcp7tihtyse5gebqeaqbaeaq.a.e.e5.sk	True	
255	5lw5dg6ythiuja7egdpgsr2tjibqeaqbaeaq.a.e.e5.sk	True	
480	AIXsV_4iZR.dnsresearch.ml	True	
700	ScrQtP1j6MfXEGsF82KxzIwVBO4OOuaX9EFZ-9W5s2UbVHwCMy5D-XTb0WpQJaW.oEMG5RB.dnsresearch.ml	True	
1363	ag6cuaacaakityrgaeeaaaaaaaaabveinbki3cpceyqqm3k7aeaaaaa3aaaaa3s.yomnacaawaajqb57r7wr26nlopvkqryhstj2mxsrgefzs2ai.a.j.e5.sk	True	1
2311	n764ciz43evu7prz7dfkga2744bqeaqbaeaq.a.e.e5.sk	True	
2341	hyesuaacaakjngaaaeaaaaaaaaaabr5u4g2ml4pefekfv5acaeaaaaa3aaaabus.aktvacaawaajqbib4wdfwkx2btrccv4c5hpozlrsamqm3aai.a.j.e5.sk	True	1
2625	iluTZ0bfyQV-gGhDcxBN7XlvApM8De.m4FEt9cR-GII_-EEq0Wuzsod5Jh751.izV65jgwi.dnsresearch.ml	True	
2794	ixdo7aoqgeburlcyyqrphtflombqeaqbaeaq.a.e.e5.sk	True	
2815	o76n2dfeldzuzni44dmvbhe5xmbqeaqbaeaq.a.e.e5.sk	True	

```
# variables
x1 = shuffled_data_1[['len','subdomains_count','w_count','w_count_ratio','w_max','w_max_ratio', 'digit_ratio','entropy']]

# target
y1 = shuffled_data_1['attack']


# splitting the data
x_train1, x_test1, y_train1, y_test1 = train_test_split(x1, y1, train_size=0.8)


print(x_train1.shape)
print(x_test1.shape)
print(y_train1.shape)
print(y_test1.shape)
```

```
(28059320, 8)
(7014830, 8)
(28059320,)
(7014830,)
```

```
print(y_train1.value_counts())
print(y_test1.value_counts())
```

```
attack
False    27919538
True       139782
Name: count, dtype: int64
attack
False    6979833
True       34997
Name: count, dtype: int64
```

```
# Pour obtenir des queries du test set
false_indexes = y_test1[y_test1 == False].index

ok_in_test = shuffled_data_1.loc[false_indexes]

ok_in_test.head()
```

	request	attack	len	subdomains_count	w_count	w_count_ra
21393812	star.c10r.facebook.com	False	9	2	1	0.11
27325187	ssl.gstatic.com	False	3	1	0	0.000
4858532	sirius.mwbsys.com	False	6	1	0	0.000
7501300	a2047.r.akamai.net	False	7	2	1	0.142
24000768	a.root-servers.net	False	1	1	4	4.000

```
# Pour obtenir des queries du test set
true_indexes = y_test1[y_test1].index

attacks_in_test = shuffled_data_1.loc[true_indexes]

attacks_in_test.head()
```

	request	attack	len	subdomains_coun
13556679	90dNIhWO8jgjmZbsGLT_.dnsresearch.ml	True	20	
20085422	zfw5cdrxwxcexhuvnbskuf6jcqbqeaqbaeaq.a.e.e5.sk	True	40	
21399344	ffaolftgc2euveeuhcrrr3hfoabqeaqbaeaq.a.e.e5.sk	True	40	
22693012	2hb6r2llewbulgpm2ioizgjhqubqeaqbaeaq.a.e.e5.sk	True	40	
22654698	7fozyh3ozjmerdknbct5yjb74ybqeaqbaeaq.a.e.e5.sk	True	40	

Étapes suivantes : Générer du code avec `attacks_in_test` Afficher les graphiques recommandés

2.3 The first model : a Logistic regression

```
model1 = LogisticRegression(max_iter=10000)
model1.fit(x_train1, y_train1)
```

```
LogisticRegression(max_iter=10000)
```

```
train_predictions = model1.predict(x_train1)
train_score = accuracy_score(y_train1, train_predictions)
recall = recall_score(y_train1, train_predictions)
print(f"Recall score: {recall}")
print(f"Training accuracy : {train_score}")
print(confusion_matrix(y_train1, train_predictions))
```

```
Recall score: 0.6999828304073485
Training accuracy : 0.9979170557233746
[[27903029    16509]
 [   41937    97845]]
```

```
predictions = model1.predict(x_test1)
test_score = accuracy_score(y_test1, predictions)
print(f"Testing accuracy : {test_score}")
recall = recall_score(y_test1, predictions)
print(f"Recall score: {recall}")
print(confusion_matrix(y_test1,predictions))
```

```
Testing accuracy : 0.9979127077919209
Recall score: 0.6996885447324056
[[6975701    4132]
 [  10510   24487]]
```

Soit 70% des attaques détectées C'est pas si mal.

2.4 The second model : a MLP

```
# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(x_train1.shape[1],)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Precision()])

# Print the model summary
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 64)                576

 dropout_8 (Dropout)         (None, 64)                0

 dense_13 (Dense)            (None, 64)                4160

 dropout_9 (Dropout)         (None, 64)                0

 dense_14 (Dense)            (None, 1)                 65

=================================================================
Total params: 4801 (18.75 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the model
history = model.fit(x_train1, y_train1, epochs=4, batch_size=2048, validation_data=(x_test1, y_test1))
```

```
Epoch 1/4
13701/13701 [==============================] - 79s 6ms/step - loss: 0.0074 - accuracy: 0.9985 - precision_4: 0.8449 - va
Epoch 2/4
13701/13701 [==============================] - 76s 6ms/step - loss: 0.0020 - accuracy: 0.9996 - precision_4: 0.9802 - va
Epoch 3/4
13701/13701 [==============================] - 76s 6ms/step - loss: 0.0019 - accuracy: 0.9996 - precision_4: 0.9812 - va
Epoch 4/4
13701/13701 [==============================] - 75s 5ms/step - loss: 0.0018 - accuracy: 0.9997 - precision_4: 0.9818 - va
```

```
# Make predictions on the test set
y_pred = model.predict(x_test1)
y_pred_classes = (y_pred > 0.5).astype("int32")  # Assuming a binary classification with a threshold of 0.5
```

```
219214/219214 [==============================] - 262s 1ms/step
```

```
# Compute the confusion matrix
cm = confusion_matrix(y_test1, y_pred_classes)
print(cm)
```

```
[[6979605     228]
 [    829   34168]]
```

Précision de 92%

2.4 Evulating manual inputs

```
def preprocessing(query):
    len_query = calculate_len(query)
    subdomains_count = calculate_subdomains(query)
    w_count = calculate_w_count(query)
    w_count_ratio = w_count / len_query if len_query != 0 else 0
    w_max = calculate_longest_word_length(query)
    w_max_ratio = w_max / len_query if len_query != 0 else 0
    digit_count = count_digits(query)
    digit_ratio = digit_count / len_query
    entropy = calculate_entropy(query)
    return np.array([len_query, subdomains_count, w_count, w_count_ratio, w_max, w_max_ratio, digit_ratio, entropy])



queries = ["mawx0cmf0zwqgb3zlcibetlmuckhlcmugaxmgysbwyxnzd29yzdoguebzc3cwcm.example.com",
           "qxmjmhckfuzcbozxjliglzigegc2vjcmv0igtletogu0vdukvus0vzmtizndu2.example.com",
           "vghpcybpcybzb21lihnlbnnpdgl2zsbkyxrhihroyxqgbmvlzhmgdg8gymugzxh.example.com",
           "samba.local.local   ",
           "a.c-0.19-a3000000.d0c0081.1838.1220.2fc9.410.0.kutu452468r6pmrknkrpzkt6lt.avqs.mcafee.com   ",
           "m27suaacaakc2obqahqeya6saqaaa4chqaeviljygcpxgaaaaeaaaaa3aaaabu6.vj7nacaawaajqaro5ttofstvahphtk3asngfy5k7xj37aaai
           "ihccuaacaakpcyjhahdkm4gsaqaaanawdsq7f4lbe4r7bjtqaeaaaaa3aaaabv5.4lmaacaawaajqbq62bnxydj5qus24g7y4bh6icer2kzuxeai
           "colab.research.google.com",
           "colab.researsh.google.com"
           ]
labels =  [True,
           True,
           True,
           False,
           True,
           True,
           True,
           False,
           True
           ]




for i in range(len(queries)):
  query = queries[i]
  input = preprocessing(query)
  input = input.reshape(1, -1)  # Reshape to add batch dimension


  pred = model.predict(input)
  print("Input query : ", query)
  print("It's an attak : ", labels[i])
  print("Prediction:", pred, "=> ",pred > 0.5)
```

```
1/1 [==============================] - 0s 71ms/step
Input query :  mawx0cmf0zwqgb3zlcibetlmuckhlcmugaxmgysbwyxnzd29yzdoguebzc3cwcm.example.com
It's an attak :  True
Prediction: [[3.7035377e-05]] =>  [[False]]
1/1 [==============================] - 0s 26ms/step
Input query :  qxmjmhckfuzcbozxjliglzigegc2vjcmv0igtletogu0vdukvus0vzmtizndu2.example.com
It's an attak :  True
Prediction: [[3.5451056e-05]] =>  [[False]]
1/1 [==============================] - 0s 24ms/step
Input query :  vghpcybpcybzb21lihnlbnnpdgl2zsbkyxrhihroyxqgbmvlzhmgdg8gymugzxh.example.com
It's an attak :  True
Prediction: [[3.9472918e-05]] =>  [[False]]
1/1 [==============================] - 0s 23ms/step
Input query :  samba.local.local
It's an attak :  False
Prediction: [[1.1201691e-15]] =>  [[False]]
1/1 [==============================] - 0s 24ms/step
Input query :  a.c-0.19-a3000000.d0c0081.1838.1220.2fc9.410.0.kutu452468r6pmrknkrpzkt6lt.avqs.mcafee.com
It's an attak :  True
Prediction: [[0.99875003]] =>  [[ True]]
1/1 [==============================] - 0s 25ms/step
Input query :  m27suaacaakc2obqahqeya6saqaaa4chqaeviljygcpxgaaaaeaaaaa3aaaabu6.vj7nacaawaajqaro5ttofstvahphtk3asngfy5k7x
It's an attak :  True
Prediction: [[0.99955285]] =>  [[ True]]
1/1 [==============================] - 0s 26ms/step
Input query :  ihccuaacaakpcyjhahdkm4gsaqaaanawdsq7f4lbe4r7bjtqaeaaaaa3aaaabv5.4lmaacaawaajqbq62bnxydj5qus24g7y4bh6icer2
It's an attak :  True
Prediction: [[0.99946713]] =>  [[ True]]
1/1 [==============================] - 0s 24ms/step
Input query :  colab.research.google.com
It's an attak :  False
Prediction: [[7.326885e-15]] =>  [[False]]
1/1 [==============================] - 0s 24ms/step
Input query :  colab.researsh.google.com
It's an attak :  True
Prediction: [[1.3066918e-05]] =>  [[False]]
```

3. Looking for a better model

Le passage de 1. à 2. a montré qu'on s'en sort bien en computant nous mêmes les features (même si y a des différences entre nos features et les features du dataset, ces différences ne sont donc pas significatives).

Maintenant, on a un problème : le modèle, en gros, dit que tout input est FALSE sauf les inputs qui ressemblent vraiment aux attaques du dataset (92% de précision quand même) D'où les résultats observés pour les manual inputs, et du coup notre modèle ne nous sert pas dans l'application

Ca, c'est dû à la quantité relativement très faibles de TRUE attacks dans le dataset. Pour parer à ça, deux approches : utiliser une nouvelle loss fonction faite pour prévoir le coup, ou bien ne retenir que très peu de rows du dataset pour un ratio 1 true pour 1 false

3.1 Focal loss function

```
# Define the model architecture
model3 = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(x_train1.shape[1],)),
    tf.keras.layers.Dropout(0.7),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.7),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model3.compile(optimizer='adam',
               loss=tf.keras.losses.BinaryFocalCrossentropy(apply_class_balancing=True, alpha=0.99, gamma=4.0, from_logits=Fa
              metrics=['accuracy', tf.keras.metrics.Precision()])

# Print the model summary
model3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 64)                576

 dropout_10 (Dropout)        (None, 64)                0

 dense_16 (Dense)            (None, 64)                4160

 dropout_11 (Dropout)        (None, 64)                0

 dense_17 (Dense)            (None, 1)                 65

=================================================================
Total params: 4801 (18.75 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the model
history = model3.fit(x_train1, y_train1, epochs=4, batch_size=4096, validation_data=(x_test1, y_test1))
```

```
Epoch 1/4
6851/6851 [==============================] - 49s 7ms/step - loss: 7.3814e-04 - accuracy: 0.9744 - precision_5: 0.1550 -
Epoch 2/4
6851/6851 [==============================] - 46s 7ms/step - loss: 7.2798e-05 - accuracy: 0.9887 - precision_5: 0.3012 -
Epoch 3/4
6851/6851 [==============================] - 46s 7ms/step - loss: 6.7300e-05 - accuracy: 0.9890 - precision_5: 0.3075 -
Epoch 4/4
6851/6851 [==============================] - 45s 7ms/step - loss: 6.8769e-05 - accuracy: 0.9879 - precision_5: 0.2874 -
```

```
# Make predictions on the test set
y_pred = model3.predict(x_test1)
y_pred_classes = (y_pred > 0.5).astype("int32")  # Assuming a binary classification with a threshold of 0.5
```

```
219214/219214 [==============================] - 262s 1ms/step
```

```
# Compute the confusion matrix
cm = confusion_matrix(y_test1, y_pred_classes)
print(cm)
```

```
for i in range(len(queries)):
  query = queries[i]
  input = preprocessing(query)
  input = input.reshape(1, -1)  # Reshape to add batch dimension


  pred = model3.predict(input)
  print("Input query : ", query)
  print("It's an attak : ", labels[i])
  print("Prediction:", pred, "=> ",pred > 0.3)
```

```
1/1 [==============================] - 0s 62ms/step
Input query :  mawx0cmf0zwqgb3zlcibetlmuckhlcmugaxmgysbwyxnzd29yzdoguebzc3cwcm.example.com
It's an attak :  True
Prediction: [[0.36274105]] =>  [[ True]]
1/1 [==============================] - 0s 25ms/step
Input query :  qxmjmhckfuzcbozxjliglzigegc2vjcmv0igtletogu0vdukvus0vzmtizndu2.example.com
It's an attak :  True
Prediction: [[0.36149856]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  vghpcybpcybzb21lihnlbnnpdgl2zsbkyxrhihroyxqgbmvlzhmgdg8gymugzxh.example.com
It's an attak :  True
Prediction: [[0.36291838]] =>  [[ True]]
1/1 [==============================] - 0s 24ms/step
Input query :  samba.local.local
It's an attak :  False
Prediction: [[0.01384288]] =>  [[False]]
1/1 [==============================] - 0s 23ms/step
Input query :  a.c-0.19-a3000000.d0c0081.1838.1220.2fc9.410.0.kutu452468r6pmrknkrpzkt6lt.avqs.mcafee.com
It's an attak :  True
Prediction: [[0.8810738]] =>  [[ True]]
1/1 [==============================] - 0s 25ms/step
Input query :  m27suaacaakc2obqahqeya6saqaaa4chqaeviljygcpxgaaaaeaaaaa3aaaabu6.vj7nacaawaajqaro5ttofstvahphtk3asngfy5k7x
It's an attak :  True
Prediction: [[0.98675406]] =>  [[ True]]
1/1 [==============================] - 0s 26ms/step
Input query :  ihccuaacaakpcyjhahdkm4gsaqaaanawdsq7f4lbe4r7bjtqaeaaaaa3aaaabv5.4lmaacaawaajqbq62bnxydj5qus24g7y4bh6icer2
It's an attak :  True
Prediction: [[0.9858124]] =>  [[ True]]
1/1 [==============================] - 0s 27ms/step
Input query :  colab.research.google.com
It's an attak :  False
Prediction: [[0.03670552]] =>  [[False]]
1/1 [==============================] - 0s 24ms/step
Input query :  colab.researsh.google.com
It's an attak :  True
Prediction: [[0.33043373]] =>  [[ True]]
```

Je trouve ça pas mal comme résultat, car si on change le threshold à 0.30, on obtient une bonne prédiction Remarque : "colab.research.google.com" est déclaré false, mais "colab.researsh.google.com" est déclaré true => impact des mots. Une faute de frappe ou d'orthographe a une grosse conséquence

3.2 Looking for the threshold appropriate to our application

J'ai réalisé une exfiltration avec un grand fichier (diary.txt) afin d'obtenir de nombreuses query générées par mes scripts. Le but est d'obtenir un set pertinent pour notre application, afin de voir les prédictions du modèles dessus, puis de déterminer un threshold adapté.

```
# Load the sample of "real" attack queries
with open(os.path.join(folder_path, 'queries.txt'), 'r') as f:
    lines = f.readlines()
    line = [line.strip() for line in lines]

real_att_queries = []
for i in range(len(line)):
  if i%2 == 0 :
    real_att_queries.append(line[i])


print(len(real_att_queries))
print(real_att_queries)
```

```
583
['tg9yzw0gaxbzdw0gzg9sb3igc2l0igftzxqsignvbnnly3rldhvyigfkaxbpc2n.example.com.', 'pbmcgzwxpdc4guhjvaw4gy29tbw9kbybzzwqgb
```

```
for i in range(50):
  query = real_att_queries[i]
  input = preprocessing(query)
  input = input.reshape(1, -1)  # Reshape to add batch dimension


  pred = model3.predict(input)
  print("Input query : ", query)
  print("It's an attak : TRUE ", )
  print("Prediction:", pred, "=> ",pred > 0.3)
```

```
Input query :  bsbwbgrjzxjnac4gqsjncybzzwwgczrnaxrbaxmgzwxpac4gtwyyymkgcg9zqwv.example.com.
It's an attak : TRUE
Prediction: [[0.36869046]] =>  [[ True]]
1/1 [==============================] - 0s 25ms/step
Input query :  yzsb1dcbyaxn1cybldsbpywn1bglzlibdcmfzihr1cnbpcybmzwxpcywgbhvjdh.example.com.
It's an attak : TRUE
Prediction: [[0.36840957]] =>  [[ True]]
1/1 [==============================] - 0s 22ms/step
Input query :  vzihnvbgxpy2l0dwrpbib0zw1wb3igc2l0igftzxqsigzhy2lsaxnpcyb2zwwgc.example.com.
It's an attak : TRUE
Prediction: [[0.3681815]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  2fwawvulibqcmflc2vudcb0zw1wdxmgc29kywxlcybzywdpdhrpcy4gu3vzcgvu.example.com.
It's an attak : TRUE
Prediction: [[0.36848196]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  zglzc2ugcg90zw50as4grxrpyw0gbmlzbcbuaxnplcb1bgxhbwnvcnblcibldcb.example.com.
It's an attak : TRUE
Prediction: [[0.36756617]] =>  [[ True]]
1/1 [==============================] - 0s 24ms/step
Input query :  kawn0dw0gywmsihzhcml1cybhbglxdwv0igvuaw0ucgpnywvjzw5hcyb1dcb2zw.example.com.
It's an attak : TRUE
Prediction: [[0.368235]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  xpdcb0aw5jawr1bnqgbgvvihrlbxb1cyb0zw1wb3iuifnlzcbzy2vszxjpc3f1z.example.com.
It's an attak : TRUE
Prediction: [[0.36834386]] =>  [[ True]]
1/1 [==============================] - 0s 22ms/step
Input query :  sbmzwxpcybtywduyswgbmvjigzyaw5nawxsysbkb2xvcibhbglxdwftiglklibt.example.com.
It's an attak : TRUE
Prediction: [[0.36997682]] =>  [[ True]]
1/1 [==============================] - 0s 22ms/step
Input query :  zwqgdmvoawn1bgesigrvbg9yighlbmryzxjpdcb2dwxwdxrhdgugc3vzy2lwaxq.example.com.
It's an attak : TRUE
Prediction: [[0.3696098]] =>  [[ True]]
1/1 [==============================] - 0s 22ms/step
Input query :  sihnhcgllbibqdxn0bybzb2rhbgvzihr1cnbpcywgdml0ywugymxhbmrpdcbuax.example.com.
It's an attak : TRUE
Prediction: [[0.36913875]] =>  [[ True]]
1/1 [==============================] - 0s 22ms/step
Input query :  npihnlbsbhyybvcmnplibtzwqgdmvuzw5hdglzig1hbgvzdwfkysbszw8gzxuga.example.com.
It's an attak : TRUE
Prediction: [[0.37018648]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  gvuzhjlcml0librdwlzcxvlihnhz2l0dglzlcblcmf0igegdgvtchvzigfsaxf1.example.com.
It's an attak : TRUE
Prediction: [[0.36845958]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  yw0sig51bgxhigxpymvybybtyxr0axmgchvydxmsigfjigvnzxn0yxmgdmvsaxq.example.com.
It's an attak : TRUE
Prediction: [[0.36854482]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  gdgvsbhvzihnlzcblcm9zlibjbiboywmgagfiaxrhc3nlihbsyxrlysbkawn0dw.example.com.
It's an attak : TRUE
Prediction: [[0.3699207]] =>  [[ True]]
1/1 [==============================] - 0s 23ms/step
Input query :  1zdc4gu2vkigfiigvvb3mgzxugbgfidxmgc2fnaxr0axmgcghhcmv0cmegbm9ui.example.com.
```

```
real_att = pd.DataFrame(real_att_queries, columns=["request"])
```

```
real_att.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
```