```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
from datetime import timedelta
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Importation et familiarisation avec le jeu de données

```
# df = pd.read_csv('./SWaT_dataset_Jul_19_v2.csv')
df = pd.read_csv('drive/MyDrive/PDD/SWaT_dataset_Jul_19_v2.csv')
#df =
pd.read_csv('drive/MyDrive/ProtectionDeDonnees/SWaT_dataset_Jul_19_v2.
csv')
```

```
df.head()
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/
interactiveshell.py:2718: DtypeWarning: Columns
(1,2,3,4,5,6,7,8,9,10,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
31,32,33,34,35,36,37,38,39,40,41,43,44,45,46,47,48,49,50,51,52,53,54,5
5,56,57,58,59,60,61,62,63,64,65,66,67,74,75,76,77) have mixed
types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

	Unnamed: 0	P1	...	Unnamed: 76	Unnamed: 77
0	GMT +0	FIT 101	...	P602 Status	P603 Status
1	timestamp	value	...	value	value
2	2019-07-20T04:30:00Z	0	...	1	1
3	2019-07-20T04:30:01Z	0	...	1	1
4	2019-07-20T04:30:02.004013Z	0	...	1	1

```
[5 rows x 78 columns]
```

```
# On met les bons noms de colones
df.columns = df.loc[0]

# On supprime les lignes inutiles
df = df.drop([0,1], axis=0)

# Mise a jour des index
df.reset_index(inplace=True, drop=True)
```

```
# On modifie la colone temps en ne gardant que l'heure, les minutes et
les secondes
# et on stocke sous format timedelta
df = df.rename(columns={"GMT +0":"timestamp"})

date_pattern = '[0-9]{2}:[0-9]{2}:[0-9]*'
i=0
for date in df['timestamp']:
    hour = re.findall(date_pattern, date)
    h = str(hour)[2:10]
    n = h.split(':')
    df['timestamp'][i] = timedelta(hours=int(n[0]), minutes=int(n[1]),
seconds=int(n[2]))
    i+=1

df.head()
```

```
0 timestamp FIT 101   LIT 101  ... P601 Status P602 Status P603 Status
0   4:30:00       0  729.8658  ...           1           1           1
1   4:30:01       0   729.434  ...           1           1           1
2   4:30:02       0    729.12  ...           1           1           1
3   4:30:03       0  728.6882  ...           1           1           1
4   4:30:04       0  727.7069  ...           1           1           1

[5 rows x 78 columns]
```

Data Observation

```
print("La taille du dataset est: " + str(len(df)))
nbCols = len(df.columns)
print("Nombre de colonnes dans le dataset: " + str(nbCols))
```

```
La taille du dataset est: 14996
Nombre de colonnes dans le dataset: 78
```

```
# Noms des colonnes et leur type
for i in df.columns:
  print(i + ", type: " + str(df.dtypes[i]))
```

Quels sont les attributs catégoriques?

Dans le jeu de données original, toutes les colonnes avaient le type 'object' Mais en examinant les différentes valeurs possibles dans chacune des colonnes il nous semble que certaines colonnes sont bien catégoriques mais qu'il en existe qui sont numériques

```
# Liste de colonnes qui nous semblent être catégoriques
cat_list = ['MV 101','P1_STATE','P101 Status','P102 Status','LS
201','LS 202',\
            'LSL 203','LSLL 203','MV201','P2_STATE','P201
Status','P202 Status',\
            'P203 Status','P204 Status','P205 Status','P206
```

```
Status','P207 Status',\
            'P208 Status','MV 301','MV 302','MV 303','MV
304','P3_STATE',\
            'P301 Status','P302 Status','LS 401','P4_STATE','P401
Status',\
            'P402 Status','P403 Status','P404 Status','UV401','MV
501','MV 502',\
            'MV 503','MV 504','P5_STATE','P501 Status','P502
Status','LSH 601',\
            'LSH 602','LSH 603','LSL 601','LSL 602','LSL 603','P6
STATE',\
            'P601 Status','P602 Status','P603 Status']

# Liste de colonnes numériques
num_list = list(set(list(df.columns)) - set(cat_list))

# On enleve de la liste de colonnes numériques la colonne décrivant le
temps
num_list.remove('timestamp')

# Pour chaque colonne catégorique on affiche les valeurs possibles
for i in cat_list:
  df[i] = df[i].astype(str).astype(object)
  print("Valeurs possibles de "+ str(i) + ": " + str(df[i].unique()))
```

```
Valeurs possibles de MV 101: ['1' '0' '2']
Valeurs possibles de P1_STATE: ['3' '2']
Valeurs possibles de P101 Status: ['2' '1']
Valeurs possibles de P102 Status: ['1']
Valeurs possibles de LS 201: ['Inactive']
Valeurs possibles de LS 202: ['Inactive']
Valeurs possibles de LSL 203: ['Inactive']
Valeurs possibles de LSLL 203: ['Inactive']
Valeurs possibles de MV201: ['2' '0' '1']
Valeurs possibles de P2_STATE: ['2']
Valeurs possibles de P201 Status: ['1']
Valeurs possibles de P202 Status: ['1']
Valeurs possibles de P203 Status: ['2' '1']
Valeurs possibles de P204 Status: ['1']
Valeurs possibles de P205 Status: ['2' '1']
Valeurs possibles de P206 Status: ['1']
Valeurs possibles de P207 Status: ['1']
Valeurs possibles de P208 Status: ['1']
Valeurs possibles de MV 301: ['1' '0' '2']
Valeurs possibles de MV 302: ['1' '0' '2']
Valeurs possibles de MV 303: ['1' '0' '2']
Valeurs possibles de MV 304: ['1' '0' '2']
Valeurs possibles de P3_STATE: ['99' '2' '4' '5' '6' '7' '9' '10' '14'
'15' '16']
Valeurs possibles de P301 Status: ['1' '2']
```

```
Valeurs possibles de P302 Status: ['1']
Valeurs possibles de LS 401: ['Inactive']
Valeurs possibles de P4_STATE: ['4']
Valeurs possibles de P401 Status: ['2' '1']
Valeurs possibles de P402 Status: ['1']
Valeurs possibles de P403 Status: ['1']
Valeurs possibles de P404 Status: ['1']
Valeurs possibles de UV401: ['2' '1']
Valeurs possibles de MV 501: ['2' '0' '1']
Valeurs possibles de MV 502: ['2']
Valeurs possibles de MV 503: ['1']
Valeurs possibles de MV 504: ['1']
Valeurs possibles de P5_STATE: ['12']
Valeurs possibles de P501 Status: ['2']
Valeurs possibles de P502 Status: ['1']
Valeurs possibles de LSH 601: ['Active' 'Inactive']
Valeurs possibles de LSH 602: ['Active']
Valeurs possibles de LSH 603: ['Inactive']
Valeurs possibles de LSL 601: ['Inactive']
Valeurs possibles de LSL 602: ['Inactive']
Valeurs possibles de LSL 603: ['Active']
Valeurs possibles de P6 STATE: ['2']
Valeurs possibles de P601 Status: ['1' '2']
Valeurs possibles de P602 Status: ['1']
Valeurs possibles de P603 Status: ['1']
```

Parmi les colonnes categoriques, nous allons enlever les colonnes ayant uniquement une valeur

```
const_cat = []
for i in cat_list:
  l = len(df[i].unique())
  if (l == 1):
    print(str(i) + ", et la valeur est: " + str(df[i].unique()))
    df = df.drop(i, axis=1)
    const_cat.append(i)

#On enleve de la liste de colonnes categoriques toutes les colonnes
constantes
cat_list = list(set(cat_list) - set(const_cat))
```

```
P102 Status, et la valeur est: ['1']
LS 201, et la valeur est: ['Inactive']
LS 202, et la valeur est: ['Inactive']
LSL 203, et la valeur est: ['Inactive']
LSLL 203, et la valeur est: ['Inactive']
P2_STATE, et la valeur est: ['2']
P201 Status, et la valeur est: ['1']
P202 Status, et la valeur est: ['1']
P204 Status, et la valeur est: ['1']
P206 Status, et la valeur est: ['1']
```

```
P207 Status, et la valeur est: ['1']
P208 Status, et la valeur est: ['1']
P302 Status, et la valeur est: ['1']
LS 401, et la valeur est: ['Inactive']
P4_STATE, et la valeur est: ['4']
P402 Status, et la valeur est: ['1']
P403 Status, et la valeur est: ['1']
P404 Status, et la valeur est: ['1']
MV 502, et la valeur est: ['2']
MV 503, et la valeur est: ['1']
MV 504, et la valeur est: ['1']
P5_STATE, et la valeur est: ['12']
P501 Status, et la valeur est: ['2']
P502 Status, et la valeur est: ['1']
LSH 602, et la valeur est: ['Active']
LSH 603, et la valeur est: ['Inactive']
LSL 601, et la valeur est: ['Inactive']
LSL 602, et la valeur est: ['Inactive']
LSL 603, et la valeur est: ['Active']
P6 STATE, et la valeur est: ['2']
P602 Status, et la valeur est: ['1']
P603 Status, et la valeur est: ['1']
```

Ensuite pour les colonnes numériques nous aimerions afficher pour chaque colonne, le minmum, le maximum, la moyenne, la mediane et l'ecart-type. Pour faire cela, nous devons d'abord changer le type de la colonne en float

```
for i in num_list:
    df[i] = df[i].astype(float)
```

```
for i in num_list:
    print(df[i].describe())
```

```
count    14996.000000
mean         6.510505
std          5.556797
min          0.918753
25%          1.206863
50%          1.498175
75%         12.520008
max         12.913759
Name: DPIT 301, dtype: float64
count    14996.000000
mean       283.695239
std          8.547831
min        256.431274
25%        277.303284
50%        281.656952
75%        292.285034
max        299.020416
```

```
Name: AIT 302, dtype: float64
count    14996.000000
mean         9.210022
std          0.175812
min          8.768457
25%          9.090170
50%          9.233082
75%          9.345873
max          9.490067
Name: AIT 202, dtype: float64
count    14996.000000
mean         4.673115
std         18.183883
min          2.450902
25%          2.851376
50%          2.883414
75%          2.963509
max        192.371765
Name: PIT 502, dtype: float64
count    14996.000000
mean         7.710572
std          0.059549
min          7.483530
25%          7.696296
50%          7.733146
75%          7.749808
max          7.768393
Name: AIT 501, dtype: float64
count    14996.000000
mean       886.224353
std         63.100612
min        756.921300
25%        832.132800
50%        881.274048
75%        940.230069
max       1003.550350
Name: LIT 401, dtype: float64
count    14996.000000
mean         0.740876
std          1.634632
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          4.403484
Name: FIT 101, dtype: float64
count    14996.000000
mean         0.611743
std          0.015349
min          0.583043
```

```
25%          0.608264
50%          0.610441
75%          0.611465
max          0.769324
Name: FIT 503, dtype: float64
count    14996.000000
mean         0.340819
std          0.041552
min          0.001793
25%          0.327270
50%          0.346868
75%          0.365441
max          0.515307
Name: FIT 502, dtype: float64
count    14996.000000
mean       115.157048
std          5.324942
min        111.654060
25%        114.233528
50%        114.714172
75%        115.002563
max        170.565247
Name: PIT 503, dtype: float64
count    14996.000000
mean         8.718149
std          0.102093
min          8.339394
25%          8.665143
50%          8.745566
75%          8.790512
max          8.895800
Name: AIT 301, dtype: float64
count    14996.000000
mean         0.794251
std          0.040510
min          0.462386
25%          0.797746
50%          0.799411
75%          0.801972
max          0.818490
Name: FIT 401, dtype: float64
count    14996.000000
mean        69.268478
std          8.878605
min         61.394516
25%         65.957440
50%         68.982315
75%         69.982056
max        147.398100
Name: AIT 502, dtype: float64
```

```
count    14996.000000
mean       946.637019
std         92.168566
min        730.702100
25%        858.103317
50%        981.158700
75%       1009.156310
max       1112.775630
Name: LIT 301, dtype: float64
count    14996.000000
mean         0.000402
std          0.003317
min          0.000000
25%          0.000256
50%          0.000320
75%          0.000320
max          0.137315
Name: FIT 601, dtype: float64
count    14996.000000
mean       733.960251
std        110.960185
min        491.169769
25%        640.595184
50%        819.636841
75%        820.971436
max        825.092957
Name: LIT 101, dtype: float64
count    14996.000000
mean         0.869760
std          1.121283
min          0.000000
25%          0.000384
50%          0.000513
75%          2.320187
max          2.342357
Name: FIT 201, dtype: float64
count    14996.000000
mean      1016.263990
std          0.028943
min       1016.053590
25%       1016.277890
50%       1016.277890
75%       1016.277890
max       1016.438110
Name: AIT 503, dtype: float64
count    14996.000000
mean        25.231166
std          1.862370
min         23.647783
25%         24.609074
```

```
50%         24.839785
75%         25.032044
max         47.103306
Name: AIT 504, dtype: float64
count    14996.000000
mean        14.032767
std         14.145882
min          3.332479
25%          5.075622
50%          9.741092
75%         15.790823
max         87.951805
Name: AIT 402, dtype: float64
count    14996.000000
mean         0.824523
std          0.856954
min          0.000000
25%          0.000512
50%          0.000641
75%          1.718220
max          1.856452
Name: FIT 301, dtype: float64
count    14996.000000
mean         0.798153
std          0.033274
min          0.457703
25%          0.799539
50%          0.801589
75%          0.802871
max          0.827736
Name: FIT 501, dtype: float64
count    14996.000000
mean       160.706744
std          5.707502
min        158.132523
25%        159.526400
50%        160.231354
75%        160.695969
max        219.014359
Name: PIT 501, dtype: float64
count    14996.000000
mean       247.985162
std         11.806186
min        198.077423
25%        239.887200
50%        246.218918
75%        257.190460
max        272.289154
Name: AIT 203, dtype: float64
count    14996.000000
```

```
mean       138.766501
std          8.265845
min        113.849014
25%        131.536789
50%        143.713150
75%        144.033585
max        146.821335
Name: AIT 201, dtype: float64
count    14996.000000
mean       142.095907
std          3.166724
min        134.131500
25%        142.710800
50%        143.383057
75%        143.863235
max        153.787048
Name: AIT 303, dtype: float64
count    14996.000000
mean         0.211526
std          0.006174
min          0.198059
25%          0.209909
50%          0.211126
75%          0.211895
max          0.275694
Name: FIT 504, dtype: float64
```

Parmi les colonnes numériques, nous allons enlever les colonnes ayant uniquement une valeur

```
const_num = []
for i in num_list:
  l = len(df[i].unique())
  if (l == 1):
    print(str(i) + ", et la valeur est: " + str(df[i].unique()))
    df = df.drop(i, axis=1)
    const_num.append(i)

#On enleve de la liste de colonnes numeriques toutes les colonnes
constantes
num_list = list(set(num_list) - set(const_num))
```

```
AIT 401, et la valeur est: [0.]
```

On vérifie qu'il n'y ait pas de NaN dans les colonnes pour eventuellement les éliminer.

```
nan = False
cols = list(df.columns)
for i in cols:
    nan_number = df[i].isna().sum()
    if (nan_number > 0):
```

```
        print(str((nan_number/df.shape[0])*100)+'%')
        nan = True

if (nan == False):
    print('Il n\'y a pas de données manquantes')
```

```
Il n'y a pas de données manquantes
```

```
print('Le jeu de données comporte maintenant ' + str(len(df.columns))
+ ' colonnes')
```

```
Le jeu de données comporte maintenant 45 colonnes
```

Labelisation des données representant les 6 attaques selon pdf

On déclare les debuts et fins des differentes attaques A noter que les heures indiqué dans le pdf expliquant le déroulement des attaques sont à GMT +8 de celles du csv

On crée les colonnes qui serviront pour labéliser les attaques Pour l'instant que des 0

```
df['FIT401_attack'] = pd.Series(np.zeros(len(df)).astype(int))
df['LIT301_attack'] = pd.Series(np.zeros(len(df)).astype(int))
df['P601_attack'] = pd.Series(np.zeros(len(df)).astype(int))
df['MultiPoint_attack'] = pd.Series(np.zeros(len(df)).astype(int))
df['MV501_attack'] = pd.Series(np.zeros(len(df)).astype(int))
df['P301_attack'] = pd.Series(np.zeros(len(df)).astype(int))

df.head()
```

```
0 timestamp  FIT 101   LIT 101  ... MultiPoint_attack MV501_attack
P301_attack
0   4:30:00      0.0  729.8658  ...                 0            0
0
1   4:30:01      0.0  729.4340  ...                 0            0
0
2   4:30:02      0.0  729.1200  ...                 0            0
0
3   4:30:03      0.0  728.6882  ...                 0            0
0
4   4:30:04      0.0  727.7069  ...                 0            0
0

[5 rows x 51 columns]
```

```
deb_FIT401 = timedelta(hours=7, minutes=8, seconds=46)
fin_FIT401 = timedelta(hours=7, minutes=10, seconds=31)

deb_LIT301 = timedelta(hours=7, minutes=15, seconds=0)
fin_LIT301 = timedelta(hours=7, minutes=19, seconds=32)
```

```
deb_P601 = timedelta(hours=7, minutes=26, seconds=57)
fin_P601 = timedelta(hours=7, minutes=30, seconds=48)

deb_MultiPoint = timedelta(hours=7, minutes=38, seconds=50)
fin_MultiPoint = timedelta(hours=7, minutes=46, seconds=20)

deb_MV501 = timedelta(hours=7, minutes=54, seconds=0)
fin_MV501 = timedelta(hours=7, minutes=56, seconds=0)

deb_P301 = timedelta(hours=8, minutes=2, seconds=56)
fin_P301 = timedelta(hours=8, minutes=16, seconds=18)
```

Parcourt du dataframe pour labeliser les données selon les attaques

```
i = 0
for date in df['timestamp']:
    if(date > deb_FIT401 and date < fin_FIT401):
        df['FIT401_attack'][i] = 1
    if(date > deb_LIT301 and date < fin_LIT301):
        df['LIT301_attack'][i] = 1
    if(date > deb_P601 and date < fin_P601):
        df['P601_attack'][i] = 1
    if(date > deb_MultiPoint and date < fin_MultiPoint):
        df['MultiPoint_attack'][i] = 1
    if(date > deb_MV501 and date < fin_MV501):
        df['MV501_attack'][i] = 1
    if(date > deb_P301 and date < fin_P301):
        df['P301_attack'][i] = 1

    i += 1
```

```
print("Nombre de données avec attaque sur FIT401 : " +
str(len(df.loc[df['FIT401_attack'] == 1])))
print("Nombre de données avec attaque sur LIT301 : " +
str(len(df.loc[df['LIT301_attack'] == 1])))
print("Nombre de données avec attaque sur P601 : " +
str(len(df.loc[df['P601_attack'] == 1])))
print("Nombre de données avec attaque multi point : " +
str(len(df.loc[df['MultiPoint_attack'] == 1])))
print("Nombre de données avec attaque sur MV501 : " +
str(len(df.loc[df['MV501_attack'] == 1])))
print("Nombre de données avec attaque sur P301 : " +
str(len(df.loc[df['P301_attack'] == 1])))
```

```
Nombre de données avec attaque sur FIT401 : 104
Nombre de données avec attaque sur LIT301 : 271
Nombre de données avec attaque sur P601 : 230
Nombre de données avec attaque multi point : 449
Nombre de données avec attaque sur MV501 : 119
Nombre de données avec attaque sur P301 : 801
```

Plot des graphiques avec affichages des timezones d'attaques (données décalées)

Nous créons un dossier ou les graphiques décalés seront stockés

```
if not os.path.exists('./bad_timezones'):
  os.mkdir('./bad_timezones')

folderImg = './bad_timezones'

import matplotlib.dates as mdates
_deb_FIT401 = deb_FIT401
_fin_FIT401 = fin_FIT401

_deb_LIT301 = deb_LIT301
_fin_LIT301 = fin_LIT301

_deb_P601 = deb_P601
_fin_P601 = fin_P601

_deb_MultiPoint = deb_MultiPoint
_fin_MultiPoint = fin_MultiPoint

_deb_MV501 = deb_MV501
_fin_MV501 = fin_MV501

_deb_P301 = deb_P301
_fin_P301 = fin_P301

# specify a date to use for the times
zero = pd.datetime(1,1,1)
time = [zero + t for t in df['timestamp']]


_deb_FIT401 += zero
_fin_FIT401 += zero

_deb_LIT301 += zero
_fin_LIT301 += zero

_deb_P601 += zero
_fin_P601 += zero

_deb_MultiPoint += zero
_fin_MultiPoint += zero

_deb_MV501 += zero
_fin_MV501 += zero
```

```
_deb_P301 += zero
_fin_P301 += zero

# convert datetimes to numbers
# Mise en commentaires de cette partie à cause d'erreurs apparaissant
seulement sur Google Colab, sûrement dues à l'utilisation de Python
3.6.9 et non de la version 3.8?
# zero = mdates.date2num(zero)
# time = [t-zero for t in mdates.date2num(time)]
# _deb_FIT401 = mdates.date2num(_deb_FIT401) - zero
# _fin_FIT401 = mdates.date2num(_fin_FIT401) - zero

# _deb_LIT301 = mdates.date2num(_deb_LIT301) - zero
# _fin_LIT301 = mdates.date2num(_fin_LIT301) - zero

# _deb_P601 = mdates.date2num(_deb_P601) - zero
# _fin_P601 = mdates.date2num(_fin_P601) - zero

# _deb_MultiPoint = mdates.date2num(_deb_MultiPoint) - zero
# _fin_MultiPoint = mdates.date2num(_fin_MultiPoint) - zero

# _deb_MV501 = mdates.date2num(_deb_MV501) - zero
# _fin_MV501 = mdates.date2num(_fin_MV501) - zero

# _deb_P301 = mdates.date2num(_deb_P301) - zero
# _fin_P301 = mdates.date2num(_fin_P301) - zero

print(len(num_list))
#add data to plots
for i in range(0,len(num_list)):
    fig, axs = plt.subplots()
    axs.title.set_text(num_list[i])
    axs.plot_date(time, df[num_list[i]],'b-')
    axs.axvspan(_deb_FIT401,_fin_FIT401, facecolor='pink',alpha=0.5)#
FIT401 atk
    axs.axvspan(_deb_LIT301,_fin_LIT301, facecolor='pink',alpha=0.5)#
LIT301_atk
    axs.axvspan(_deb_P601,_fin_P601, facecolor='pink',alpha=0.5)#
P601_atk
    axs.axvspan(_deb_MultiPoint,_fin_MultiPoint,
facecolor='pink',alpha=0.5)# Multipoint_atk
    axs.axvspan(_deb_MV501,_fin_MV501, facecolor='pink',alpha=0.5)#
MV501_atk
    axs.axvspan(_deb_P301,_fin_P301, facecolor='pink',alpha=0.5)#
P301_atk
    plt.xticks(rotation=-35, ha='left', rotation_mode='anchor')
    plt.savefig('./bad_timezones/' + num_list[i] + '.jpg', dpi=150,
```

```
bbox_inches='tight')
    plt.close(fig)
```

```
27
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21:
FutureWarning: The pandas.datetime class is deprecated and will be
removed from pandas in a future version. Import from datetime module
instead.
```

Correction des labels avec determination du bon debut et fin de chaque attaque

Les vrais débuts et fins des attaques ont été trouvées à la main

```
# Attaque sur FIT 401
# Debut ligne 9416, fin ligne 9520
real_deb_FIT401 = timedelta(hours=7, minutes=6, seconds=59)
real_fin_FIT401 = timedelta(hours=7, minutes=8, seconds=45)

# Attaque sur LIT 301
# Debut ligne 9803, fin ligne 10063
real_deb_LIT301 = timedelta(hours=7, minutes=13, seconds=26)
real_fin_LIT301 = timedelta(hours=7, minutes=17, seconds=48)

# Attaque sur P601
# Debut ligne 10509, fin ligne 10738
real_deb_P601 = timedelta(hours=7, minutes=25, seconds=12)
real_fin_P601 = timedelta(hours=7, minutes=29, seconds=3)

# Attaque Multi Point
# Debut ligne 11228, fin ligne 11691
real_deb_MultiPoint = timedelta(hours=7, minutes=37, seconds=11)
real_fin_MultiPoint = timedelta(hours=7, minutes=44, seconds=56)

# Attaque sur MV 501
# Debut ligne 12141, fin ligne 12291
real_deb_MV501 = timedelta(hours=7, minutes=52, seconds=24)
real_fin_MV501 = timedelta(hours=7, minutes=54, seconds=56)

# Attaque sur P301
# Debut ligne 12662, fin ligne 13503
real_deb_P301 = timedelta(hours=8, minutes=1, seconds=5)
real_fin_P301 = timedelta(hours=8, minutes=15, seconds=8)
```

On relabelise les données, cette fois ci correctement

```
cat_attack = ['FIT401_attack','LIT301_attack','P601_attack',\
              'MultiPoint_attack','MV501_attack','P301_attack']
```

```
# On transforme chaque colonne XXX_attack en colonne de 0
for c in cat_attack:
  df[c] = pd.Series(np.zeros(len(df)).astype(int))


# On relabelise
i = 0
for date in df['timestamp']:
    if(date > real_deb_FIT401 and date < real_fin_FIT401):
        df['FIT401_attack'][i] = 1
    if(date > real_deb_LIT301 and date < real_fin_LIT301):
        df['LIT301_attack'][i] = 1
    if(date > real_deb_P601 and date < real_fin_P601):
        df['P601_attack'][i] = 1
    if(date > real_deb_MultiPoint and date < real_fin_MultiPoint):
        df['MultiPoint_attack'][i] = 1
    if(date > real_deb_MV501 and date < real_fin_MV501):
        df['MV501_attack'][i] = 1
    if(date > real_deb_P301 and date < real_fin_P301):
        df['P301_attack'][i] = 1

    i += 1


# Pour finir, on change le type de colonnes de int vers object
for c in cat_attack:
  df[c] = df[c].astype(str).astype(object)
```

Matrices de correlation

Nous créons un dossier ou les matrices de correlation seront stockées

```
if not os.path.exists('./corrMat'):
  os.mkdir('./corrMat')

folderImg = './corrMat'

pre_attack = df.loc[df['timestamp'] < real_deb_FIT401]

pre_attack_correlation_mat = pre_attack.corr()
plt.figure(figsize=(11,9))
sns.heatmap(pre_attack_correlation_mat)

img = 'preAttack.png'
plt.title('Correlation Matrix before attacks')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
```

```
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')

# Pour ne pas afficher directement la matrice de correlation, on fait:
plt.close()

# Matrice de correlation pendant l'attaque sur FIT 401
pdtFit401Attack = df.loc[(df['timestamp'] > real_deb_FIT401) &\
                         (df['timestamp'] < real_fin_FIT401)]

pdtFit401Attack_corr = pdtFit401Attack.corr()
plt.figure(figsize=(11,9))

sns.heatmap(pdtFit401Attack_corr)
img = 'pdtFit401Attack.png'
plt.title('Correlation Matrix during FIT401 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation entre l'attaque sur FIT 401 et LIT 301
entreFit401_Lit301 = df.loc[(df['timestamp'] > real_fin_FIT401) &\
                            (df['timestamp'] < real_deb_LIT301)]

entreFit401_Lit301_corr = entreFit401_Lit301.corr()
plt.figure(figsize=(11,9))
sns.heatmap(entreFit401_Lit301_corr)

img = 'entreFit401_Lit301.png'
plt.title('Correlation Matrix between FIT401 attack and LIT301
Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation pendant l'attaque sur LIT 301
pdtLit301Attack = df.loc[(df['timestamp'] > real_deb_LIT301) &\
                         (df['timestamp'] < real_fin_LIT301)]

pdtLit301Attack_corr = pdtLit301Attack.corr()
plt.figure(figsize=(11,9))
sns.heatmap(pdtLit301Attack_corr)

img = 'pdtLit301Attack.png'
```

```
plt.title('Correlation Matrix during LIT301 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation entre l'attaque sur LIT 301 et P601
entreLit301_P601 = df.loc[(df['timestamp'] > real_fin_LIT301) &\
                          (df['timestamp'] < real_deb_P601)]

entreLit301_P601_corr = entreLit301_P601.corr()
plt.figure(figsize=(11,9))
sns.heatmap(entreLit301_P601_corr)

img = 'entreLit301_P601.png'
plt.title('Correlation Matrix between Lit301 attack and P601 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation pendant l'attaque sur P601
pdtP601Attack = df.loc[(df['timestamp'] > real_deb_P601) &\
                       (df['timestamp'] < real_fin_P601)]

pdtP601Attack_corr = pdtP601Attack.corr()
plt.figure(figsize=(11,9))
sns.heatmap(pdtP601Attack_corr)

img = 'pdtP601Attack.png'
plt.title('Correlation Matrix during P601 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation entre l'attaque sur P601 et l'ataque
MultiPoint
entreP601_Mp = df.loc[(df['timestamp'] > real_fin_P601) &\
                          (df['timestamp'] < real_deb_MultiPoint)]

entreP601_Mp_corr = entreP601_Mp.corr()
plt.figure(figsize=(11,9))
sns.heatmap(entreP601_Mp_corr)
```

```
img = 'entreP601_Mp.png'
plt.title('Correlation Matrix between P601 attack and multipoint
Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation pendant l'attaque MultiPoint
pdtMultiPointAttack = df.loc[(df['timestamp'] > real_deb_MultiPoint)
&\
                         (df['timestamp'] < real_fin_MultiPoint)]

pdtMultiPointAttack_corr = pdtMultiPointAttack.corr()
plt.figure(figsize=(11,9))
sns.heatmap(pdtMultiPointAttack_corr)

img = 'pdtMultiPointAttack.png'
plt.title('Correlation Matrix during Multipoint Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation entre l'attaque Multi Point et l'attaque sur
MV501
entreMp_MV501 = df.loc[(df['timestamp'] > real_fin_MultiPoint) &\
                           (df['timestamp'] < real_deb_MV501)]

entreMp_MV501_corr = entreMp_MV501.corr()
plt.figure(figsize=(11,9))
sns.heatmap(entreMp_MV501_corr)

img = 'entreMp_MV501.png'
plt.title('Correlation Matrix between multipoint attack and MV501
Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation pendant l'attaque MV501
pdtMV501Attack = df.loc[(df['timestamp'] > real_deb_MV501) &\
```

```
                          (df['timestamp'] < real_fin_MV501)]

pdtMV501Attack_corr = pdtMV501Attack.corr()
plt.figure(figsize=(11,9))
sns.heatmap(pdtMV501Attack_corr)

img = 'pdtMV501Attack.png'
plt.title('Correlation Matrix during MV501 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation entre l'attaque sur MV501 et sur P301
entreMV501_P301 = df.loc[(df['timestamp'] > real_fin_MV501) &\
                             (df['timestamp'] < real_deb_P301)]

entreMV501_P301_corr = entreMV501_P301.corr()
plt.figure(figsize=(11,9))
sns.heatmap(entreMV501_P301_corr)

img = 'entreMV501_P301.png'
plt.title('Correlation Matrix between MV501 and P301 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()

# Matrice de correlation pendant l'attaque sur P301
pdtP301Attack = df.loc[(df['timestamp'] > real_deb_P301) &\
                          (df['timestamp'] < real_fin_P301)]

pdtP301Attack_corr = pdtP301Attack.corr()
plt.figure(figsize=(11,9))
sns.heatmap(pdtP301Attack_corr)

img = 'pdtP301Attack.png'
plt.title('Correlation Matrix during P301 Attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')
plt.close()
```

```
# Matrice de correlation apres toutes les attaques
apresAttack = df.loc[df['timestamp']> real_fin_P301]

apresAttack_corr = apresAttack.corr()
plt.figure(figsize=(11,9))
sns_plot=sns.heatmap(apresAttack_corr)

img = 'apresAttack.png'
plt.title('Correlation Matrix after all attack')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')

plt.close()

# Matrice de correlation, toutes les données avec les labels
d'attaques
bla = df.copy()
bla['FIT401_attack'] = bla['FIT401_attack'].astype(float)
bla['LIT301_attack'] = bla['LIT301_attack'].astype(float)
bla['P601_attack'] = bla['P601_attack'].astype(float)
bla['MultiPoint_attack'] = bla['MultiPoint_attack'].astype(float)
bla['MV501_attack'] = bla['MV501_attack'].astype(float)
bla['P301_attack'] = bla['P301_attack'].astype(float)

print(bla['FIT401_attack'].dtypes)
bla_corr = bla.corr()
plt.figure(figsize=(11,9))
sns.heatmap(bla_corr)

img = 'toutesDonnesAvecLabel.png'
plt.title('Correlation Matrix on all datas and on attacks ')
plt.xlabel('')
plt.ylabel('')
plt.savefig(os.path.join(folderImg,
img),facecolor='lightgrey',edgecolor='black',
transparent=False,bbox_inches='tight')


plt.close()
```

```
float64
```

Plot des graphiques avec affichages des timezones d'attaques (données correctes)

Nous créons un dossier ou les bonnes timezones seront stockées

```
if not os.path.exists('./good_timezones'):
  os.mkdir('./good_timezones')

folderImg = './good_timezones'

_deb_FIT401 = real_deb_FIT401
_fin_FIT401 = real_fin_FIT401

_deb_LIT301 = real_deb_LIT301
_fin_LIT301 = real_fin_LIT301

_deb_P601 = real_deb_P601
_fin_P601 = real_fin_P601

_deb_MultiPoint = real_deb_MultiPoint
_fin_MultiPoint = real_fin_MultiPoint

_deb_MV501 = real_deb_MV501
_fin_MV501 = real_fin_MV501

_deb_P301 = real_deb_P301
_fin_P301 = real_fin_P301

# specify a date to use for the times
zero = pd.datetime(1,1,1)
time = [zero + t for t in df['timestamp']]


_deb_FIT401 += zero
_fin_FIT401 += zero

_deb_LIT301 += zero
_fin_LIT301 += zero

_deb_P601 += zero
_fin_P601 += zero

_deb_MultiPoint += zero
_fin_MultiPoint += zero

_deb_MV501 += zero
_fin_MV501 += zero
```

```
_deb_P301 += zero
_fin_P301 += zero

# convert datetimes to numbers
# Mise en commentaires de cette partie à cause d'erreurs apparaissant
seulement sur Google Colab, sûrement dues à l'utilisation de Python
3.6.9 et non de la version 3.8?
# zero = mdates.date2num(zero)
# time = [t-zero for t in mdates.date2num(time)]
# _deb_FIT401 = mdates.date2num(_deb_FIT401) - zero
# _fin_FIT401 = mdates.date2num(_fin_FIT401) - zero

# _deb_LIT301 = mdates.date2num(_deb_LIT301) - zero
# _fin_LIT301 = mdates.date2num(_fin_LIT301) - zero

# _deb_P601 = mdates.date2num(_deb_P601) - zero
# _fin_P601 = mdates.date2num(_fin_P601) - zero

# _deb_MultiPoint = mdates.date2num(_deb_MultiPoint) - zero
# _fin_MultiPoint = mdates.date2num(_fin_MultiPoint) - zero

# _deb_MV501 = mdates.date2num(_deb_MV501) - zero
# _fin_MV501 = mdates.date2num(_fin_MV501) - zero

# _deb_P301 = mdates.date2num(_deb_P301) - zero
# _fin_P301 = mdates.date2num(_fin_P301) - zero

#add data to plots
print(len(num_list))
for i in range(0,len(num_list)):
    fig, axs = plt.subplots()
    axs.title.set_text(num_list[i])
    axs.plot_date(time, df[num_list[i]],'b-')
    axs.axvspan(_deb_FIT401,_fin_FIT401, facecolor='pink',alpha=0.5)#
FIT401 atk
    axs.axvspan(_deb_LIT301,_fin_LIT301, facecolor='pink',alpha=0.5)#
LIT301_atk
    axs.axvspan(_deb_P601,_fin_P601, facecolor='pink',alpha=0.5)#
P601_atk
    axs.axvspan(_deb_MultiPoint,_fin_MultiPoint,
facecolor='pink',alpha=0.5)# Multipoint_atk
    axs.axvspan(_deb_MV501,_fin_MV501, facecolor='pink',alpha=0.5)#
MV501_atk
    axs.axvspan(_deb_P301,_fin_P301, facecolor='pink',alpha=0.5)#
P301_atk

    plt.xticks(rotation=-35, ha='left', rotation_mode='anchor')
    plt.savefig('./good_timezones/' + num_list[i] + '.jpg', dpi=150,
```

```
bbox_inches='tight')
    plt.close(fig)
```

Algorithmes de detection d'anomalies

- Isolation Forest

On ajoute une colonne isAttack qui servira de label pour généraliser les attaques. Si lors de l'observation, le systeme est attaqué (FIT401, LIT301 etc.), isAttack sera à 1, 0 sinon.

```
df['isAttack'] = ((df['FIT401_attack']=='1') |
(df['LIT301_attack']=='1') | \
                  (df['P601_attack']=='1') |
(df['MultiPoint_attack']=='1') |\
                  (df['MV501_attack']=='1') |
(df['P301_attack']=='1'))

df['isAttack'] = df['isAttack'].replace({True:1, False:0})
df['isAttack'] = df['isAttack'].astype(str).astype(object)

# On définit X et Y qu'on utilisera pour fit le classifieur
X =
df.drop(['timestamp','FIT401_attack','LIT301_attack','P601_attack','Mu
ltiPoint_attack',\
             'MV501_attack','P301_attack','isAttack'], axis=1)
Y = df[['isAttack']]

# On replace tous les espaces dans les noms des colonnes par des '_'
X.rename(columns = lambda x: x.replace(' ', '_'), inplace=True)

# On one-hot toutes les colonnes catégoriques de X
cat_list_X = X.select_dtypes(include=['object'])

for i in cat_list_X:
  oneHotEnc = pd.get_dummies(X[i], prefix=i)
  X = pd.concat([X, oneHotEnc], axis=1)
  X = X.drop(i, axis=1)
```

Definition des paramètres du modèle

```
from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(99)
#Dans le jeu de données, on a 2053 observations qui sont labélisées 1
sur
#'isAttack' et en tout nous avons 14996 observations donc ça fait
#13.69032% de outliers
outlier_fraction=0.1369
nb_samples = len(df)
```

```
clf = IsolationForest(max_samples = nb_samples,
contamination=outlier_fraction,\
                      random_state=rng)
```

Entrainement et prédiction du modèle

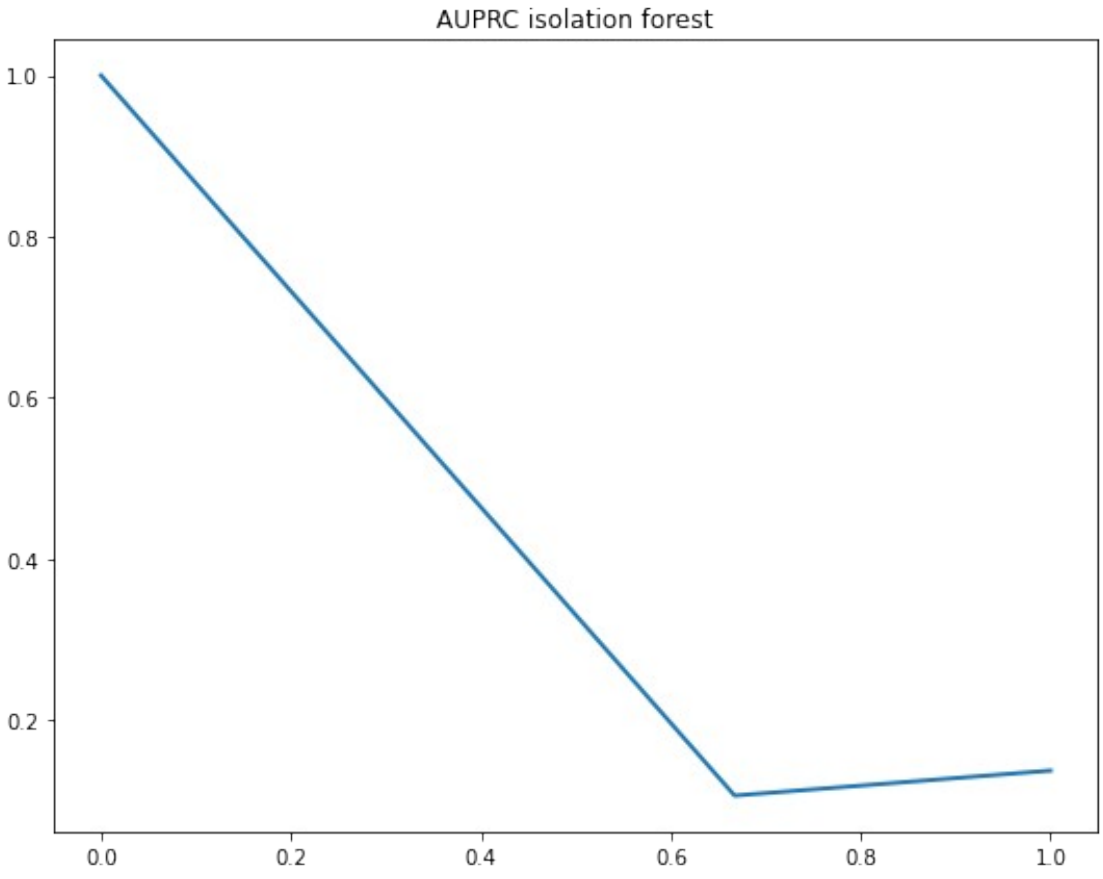
```
clf.fit(X)
ifOutliers = clf.predict(X)
```

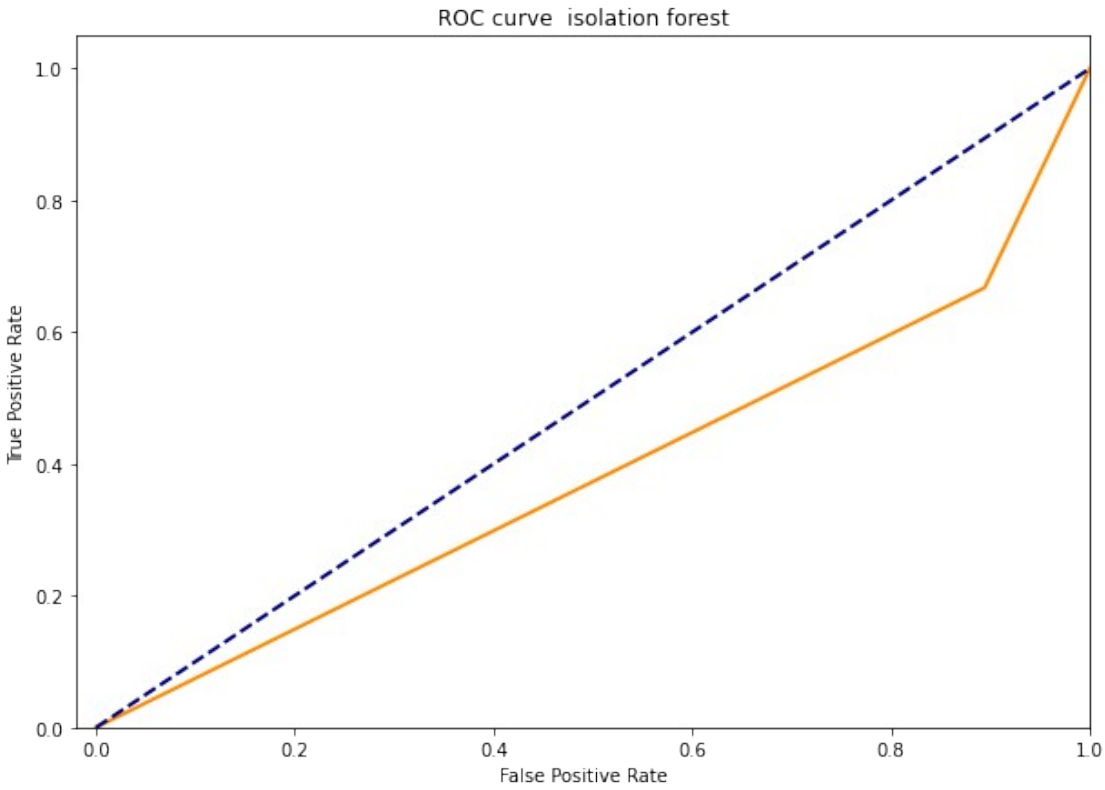
Evaluation de l'éfficacité d'isolation forest

```
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve, auc
compTab =  Y.copy()
compTab['ifOutliers'] = ifOutliers

nbAnomalies = len(compTab.loc[compTab['isAttack']=='1'])
nbTruePos = len(compTab.loc[(compTab['isAttack']=='1') &\
                            (compTab['ifOutliers']==-1)])
print("Le nombre d'anomalies au total est: " + str(nbAnomalies)+"\n")
print("Le nombre de vrais positifs est: " + str(nbTruePos)+ "\n")
print("Le taux de detections correctes est de: " + \
      str(nbTruePos*100/nbAnomalies) +"%\n")

#AUPRC (Area Under the Precision-Recall Curve)
#print([int(label) for label in RFOutliers ], y_test)
print('AUPRC =
{}'.format(average_precision_score(y_true=compTab['isAttack'],\

y_score=compTab['ifOutliers'],\
                                                  pos_label='1')))
precision, recall, thresholds =
precision_recall_curve(compTab['isAttack'],\

compTab['ifOutliers'],\
                                                        pos_label='1')

plt.figure(figsize=(9,7))
plt.title("AUPRC isolation forest")
plt.plot(recall, precision, lw=2.0)
#ROC (Receiver Operating Characteristic)
fpr, tpr, thresholds = roc_curve(compTab['isAttack'],
compTab['ifOutliers'] , pos_label='1')
roc_auc = auc(fpr, tpr)
# plt.figure(figsize=(9,7))
# plt.plot(fpr, tpr, lw=2.0)
plt.figure(figsize=(10,7))
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.02, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve  isolation forest')
plt.show()
```

```
Le nombre d'anomalies au total est: 2053

Le nombre de vrais positifs est: 683

Le taux de detections correctes est de: 33.26838772528008%

AUPRC = 0.11618003710104861
```





Local Outlier Factor (LOF)

Definition des paramètres du modèle

```
from sklearn.neighbors import LocalOutlierFactor

# On peut faire varier n_neighbors pour obtenir des taux de detection
# differents, pour 5000 le taux de detection etait le plus elevé
clf = LocalOutlierFactor(n_neighbors=5000)
```

Entrainement et prédiction du modèle

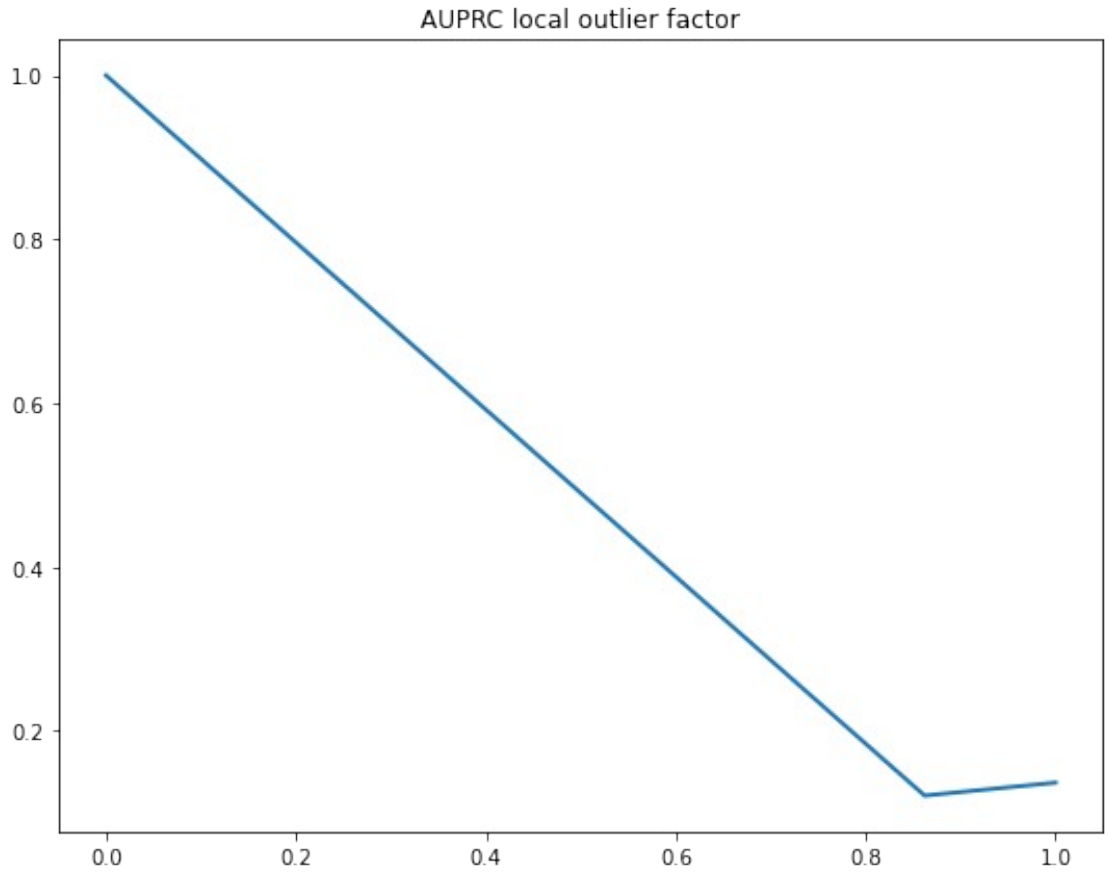
```
lofOutliers = clf.fit_predict(X)
```

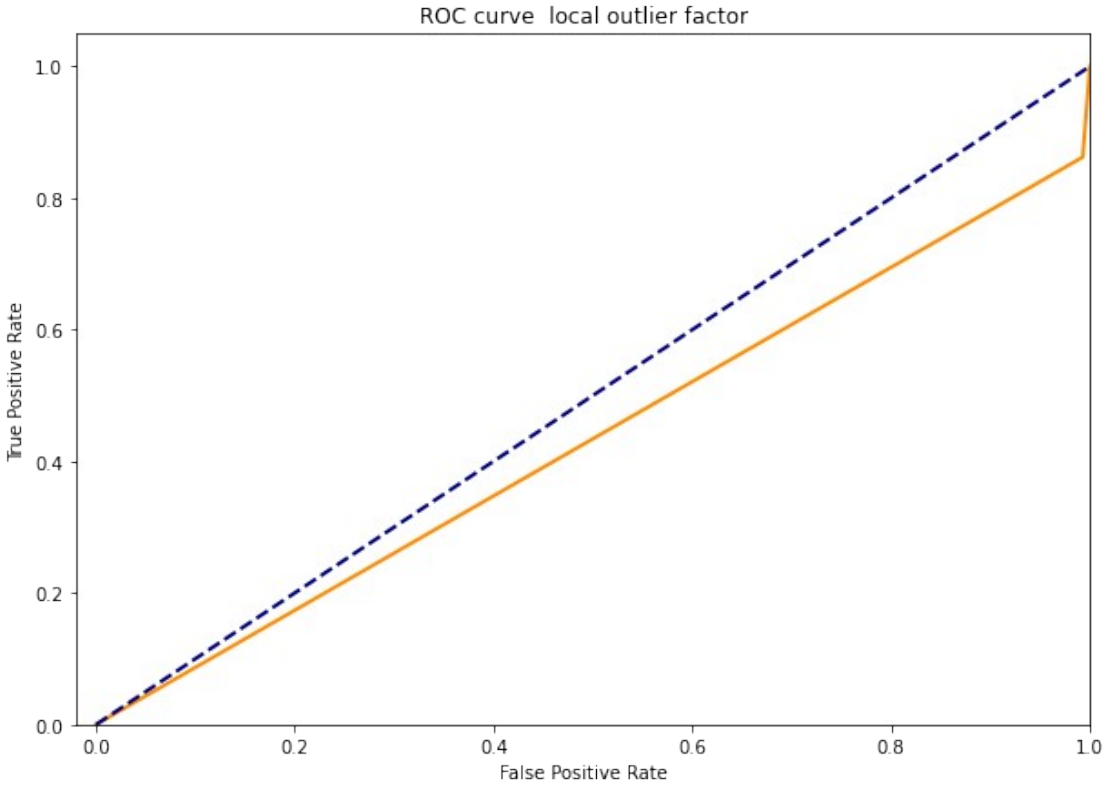
Evaluation de l'éfficacité de Local Outlier Factor

```
compTab['lofOutliers'] = lofOutliers

nbTruePos = len(compTab.loc[(compTab['isAttack']=='1') &\
                            (compTab['lofOutliers']==-1)])
print("Le nombre d'anomalies au total est: " + str(nbAnomalies)+"\n")
print("Le nombre de vrais positifs est: " + str(nbTruePos)+ "\n")
print("Le taux de detections correctes est de: " + \
      str(nbTruePos*100/nbAnomalies) +"%\n")

#AUPRC (Area Under the Precision-Recall Curve)
print('AUPRC =
{}'.format(average_precision_score(y_true=compTab['isAttack'],\
```

```
y_score=compTab['lofOutliers'],\
                                                    pos_label='1')))
precision, recall, thresholds =
precision_recall_curve(compTab['isAttack'],\

compTab['lofOutliers'],\
                                                          pos_label='1')

plt.figure(figsize=(9,7))
plt.title("AUPRC local outlier factor")
plt.plot(recall, precision, lw=2.0)
#ROC (Receiver Operating Characteristic)
fpr, tpr, thresholds = roc_curve(compTab['isAttack'],
compTab['lofOutliers'] , pos_label='1')
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(10,7))
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.02, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve  local outlier factor')
plt.show()
```

```
Le nombre d'anomalies au total est: 2053

Le nombre de vrais positifs est: 283

Le taux de detections correctes est de: 13.784705309303458%

AUPRC = 0.12324999706951417
```


AUPRC local outlier factor
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve local outlier factor
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

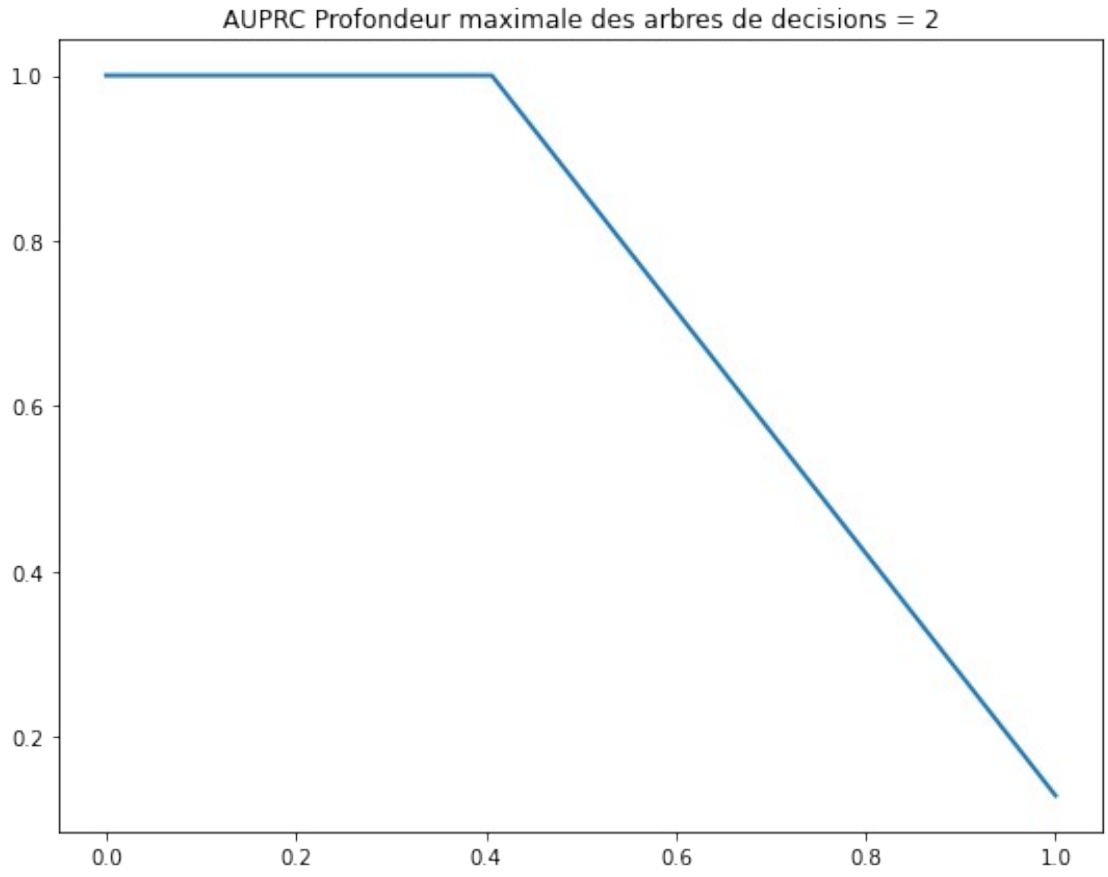
Random Forest

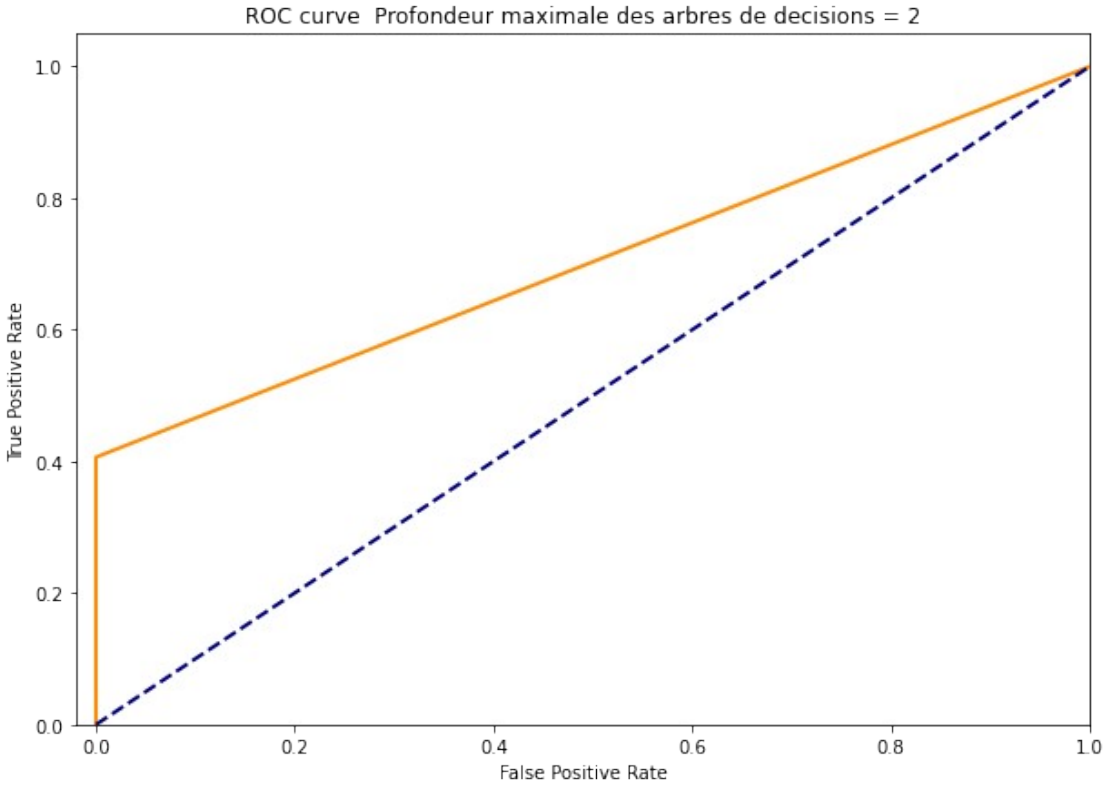
On modifie la profondeur maximale des arbres de decisions.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.2,\
                                                    random_state=5)
for i in range(2, 6):
      clf = RandomForestClassifier(max_depth=i, random_state=0)
      clf.fit(X_train, y_train)

      RFOutliers = clf.predict(X_test)
      compTab =  y_test.copy()
      compTab['RFOutliers'] = RFOutliers

      nbAnomaliesTest = len(y_test.loc[y_test['isAttack']=='1'])

      nbTruePos = len(compTab.loc[(compTab['isAttack']=='1') &\
                                  (compTab['RFOutliers']=='1')])
      print("----- Profondeur maximale des arbres de decisions = " +
str(i) + " -----")
      print("Le nombre d'anomalies au total est: " +
str(nbAnomaliesTest)+"\n")
      print("Le nombre de vrais positifs est: " + str(nbTruePos)+ "\
n")
      print("Le taux de detections correctes est de: " + \
            str(nbTruePos*100/nbAnomaliesTest) +"%\n")
      print("Le score est de: " + \
            str(clf.score(X_test, y_test)))

      #AUPRC (Area Under the Precision-Recall Curve)
      print('AUPRC =
{}'.format(average_precision_score(y_true=y_test,\

y_score=[int(label) for label in RFOutliers ],\
                                                      pos_label='1')))
      precision, recall, thresholds = precision_recall_curve(y_test,\

[int(label) for label in RFOutliers ],\

pos_label='1')
      print('ROC = ' + str(roc_auc))

      plt.figure(figsize=(9,7))
      plt.title("AUPRC Profondeur maximale des arbres de decisions = "
```

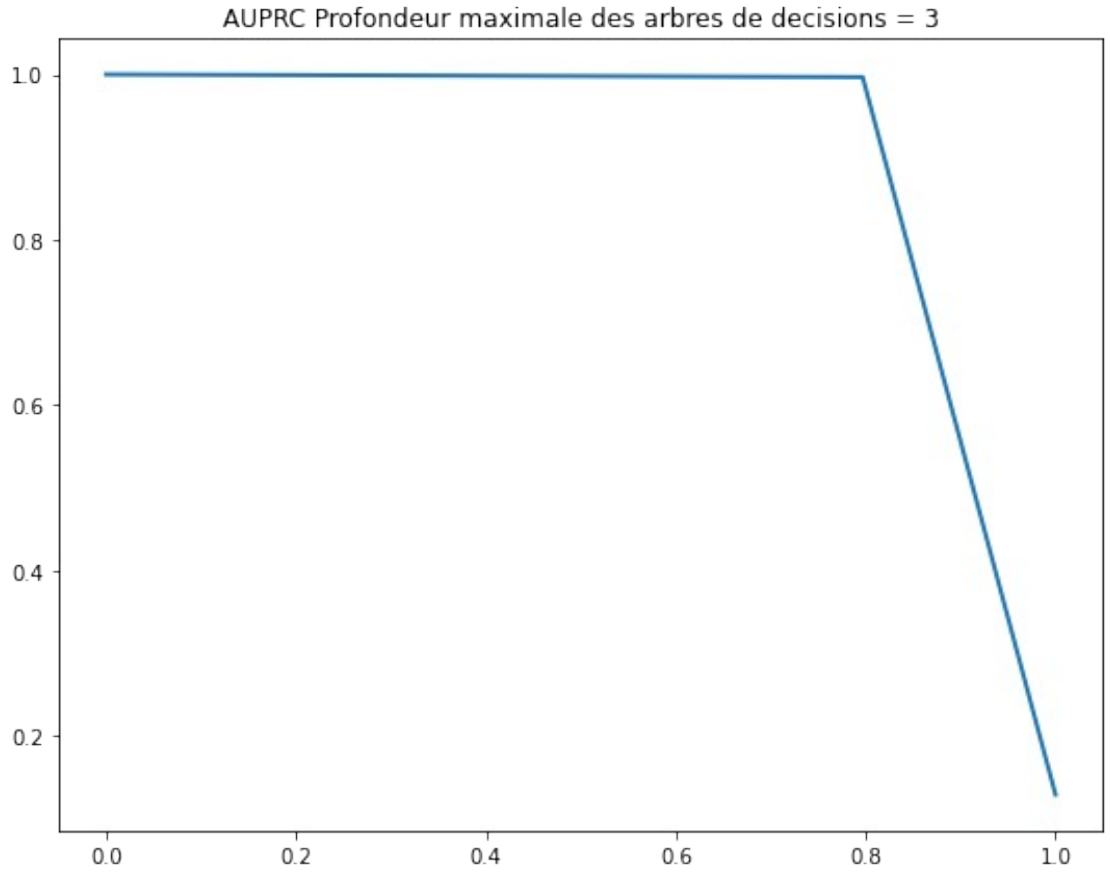
```
+ str(i))
      plt.plot(recall, precision, lw=2.0)
      #ROC (Receiver Operating Characteristic)
      fpr, tpr, thresholds = roc_curve(y_test, [int(label) for label
in RFOutliers ] , pos_label='1')
      roc_auc = auc(fpr, tpr)
      # plt.figure(figsize=(9,7))
      # plt.plot(fpr, tpr, lw=2.0)
      plt.figure(figsize=(10,7))
      lw = 2
      plt.plot(fpr, tpr, color='darkorange', lw=lw)
      plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
      plt.xlim([-0.02, 1.0])
      plt.ylim([0.0, 1.05])
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('ROC curve  Profondeur maximale des arbres de
decisions = ' + str(i))
      plt.show()
```

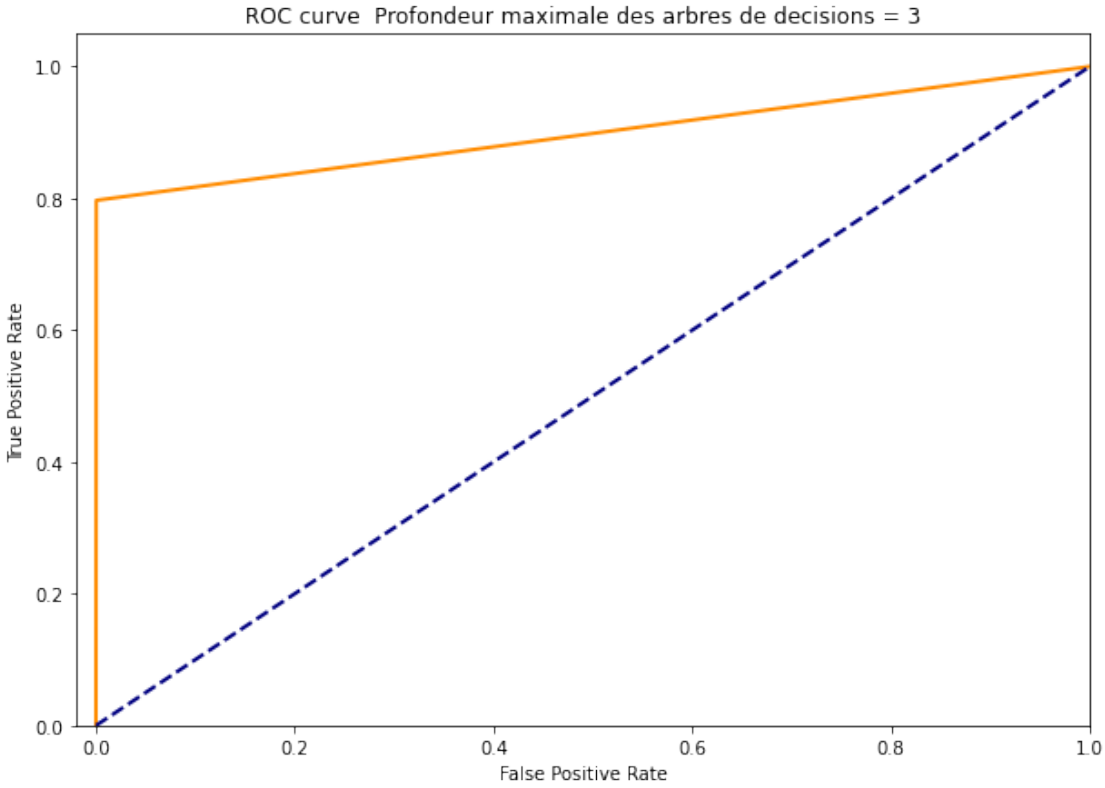
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().


----- Profondeur maximale des arbres de decisions = 2 -----
Le nombre d'anomalies au total est: 384

Le nombre de vrais positifs est: 156

Le taux de detections correctes est de: 40.625%

Le score est de: 0.924
AUPRC = 0.48225
ROC = 0.999987058804791
```


AUPRC Profondeur maximale des arbres de decisions = 2
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0

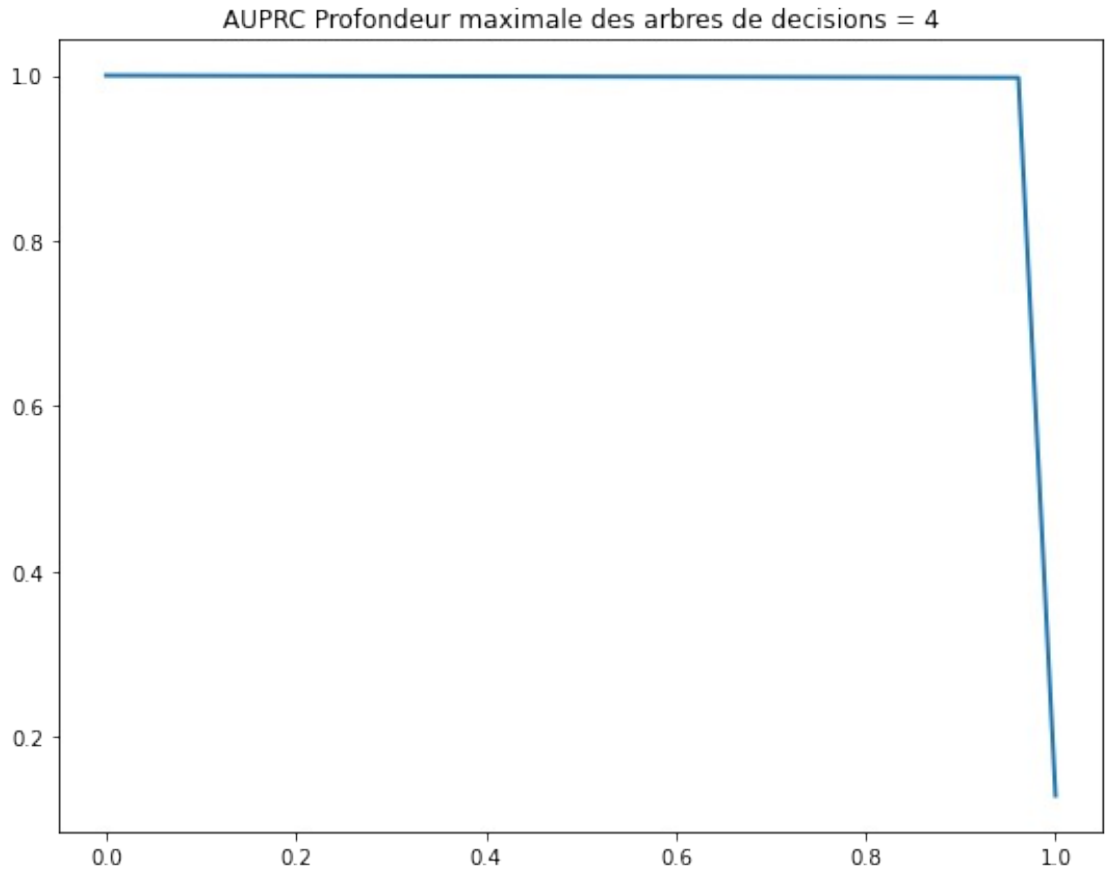

ROC curve Profondeur maximale des arbres de decisions = 2
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

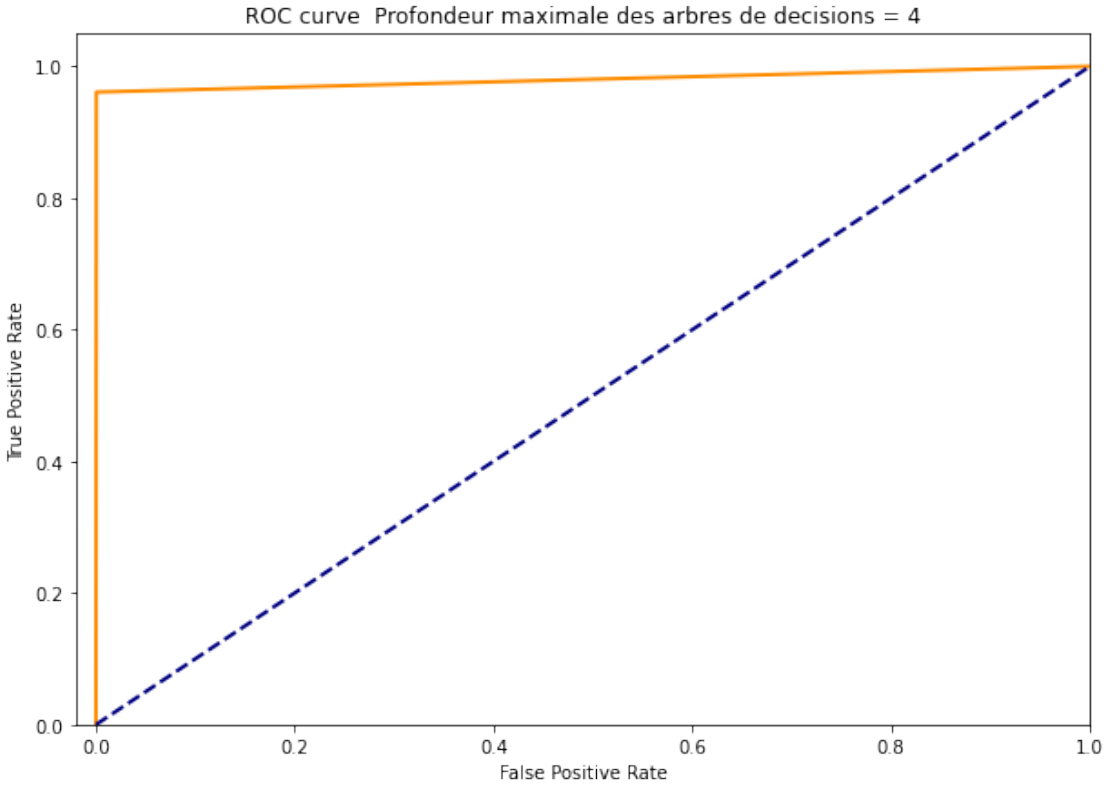
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().


----- Profondeur maximale des arbres de decisions = 3 -----
Le nombre d'anomalies au total est: 384

Le nombre de vrais positifs est: 306

Le taux de detections correctes est de: 79.6875%

Le score est de: 0.9736666666666667
AUPRC = 0.8202793159609121
ROC = 0.703125
```



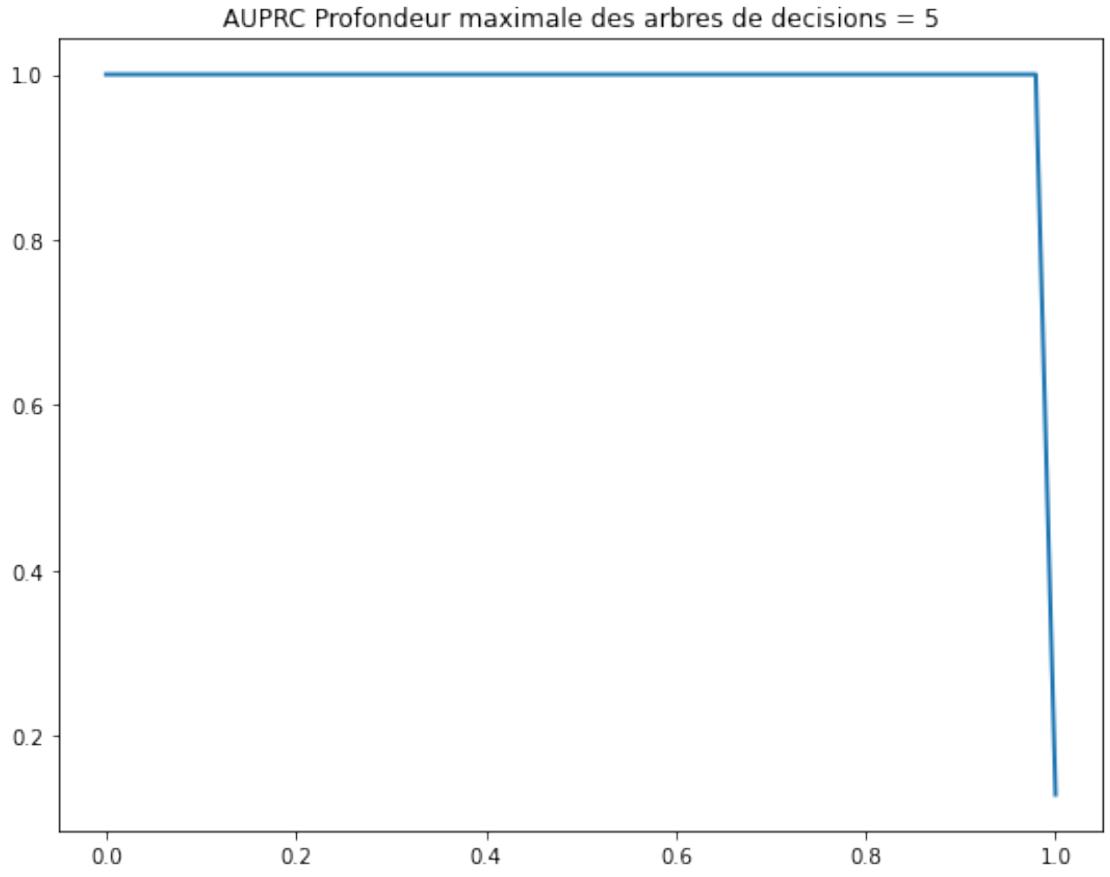


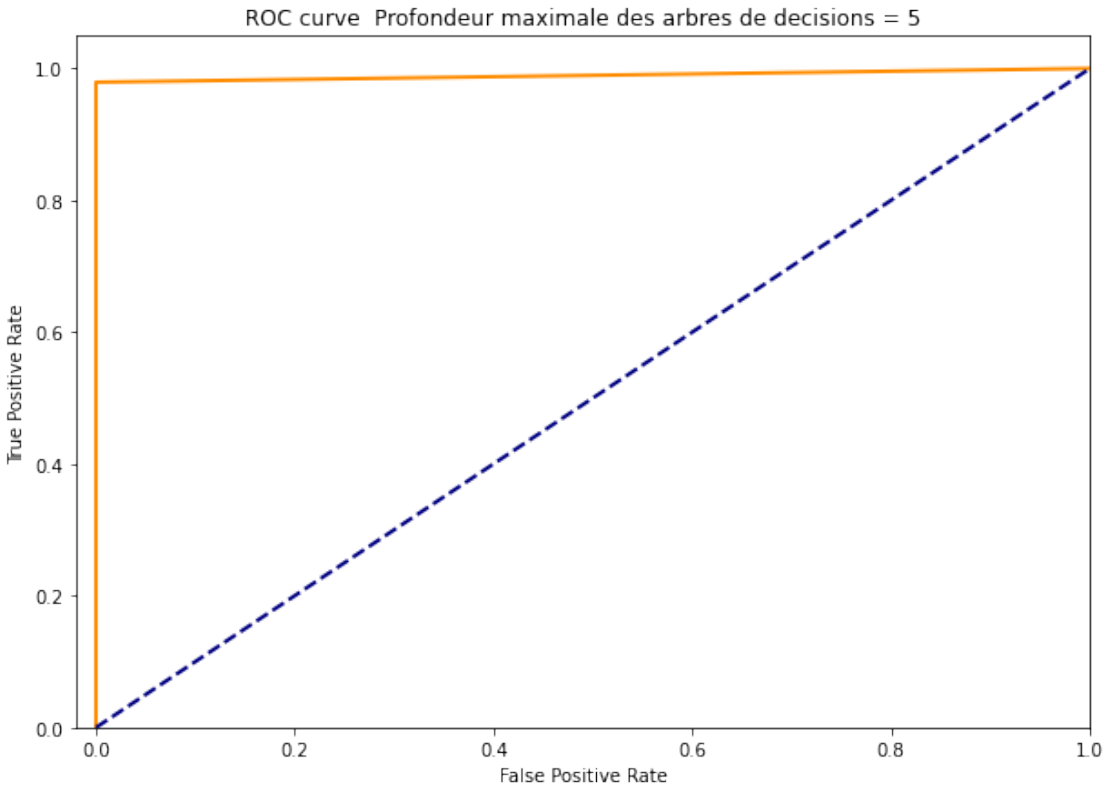

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().


----- Profondeur maximale des arbres de decisions = 4 -----
Le nombre d'anomalies au total est: 384

Le nombre de vrais positifs est: 369

Le taux de detections correctes est de: 96.09375%

Le score est de: 0.9946666666666667
AUPRC = 0.9633403716216217
ROC = 0.8982463685015291
```


AUPRC Profondeur maximale des arbres de decisions = 4
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve Profondeur maximale des arbres de decisions = 4
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().


----- Profondeur maximale des arbres de decisions = 5 -----
Le nombre d'anomalies au total est: 384

Le nombre de vrais positifs est: 376

Le taux de detections correctes est de: 97.91666666666667%

Le score est de: 0.9973333333333333
AUPRC = 0.9818333333333333
ROC = 0.9802776185015291
```





XGBoost

Préparation des données

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.2,\
                                                    random_state=5)
```

Définition des paramétres du modèle

On ajoute un poids pour équilibrer les données car dans notre jeu de donnés nous avons 2053 observations qui ont isAttack=1 et 12943 observations qui ont isAttack = 0.

```
from xgboost import XGBClassifier
# nbr d'observations non attaque divisé par le nombre d'observations
lors d'une attaque
weights = (Y['isAttack']=='0').sum() / (1.0 *
(Y['isAttack']=='1').sum())
clf = XGBClassifier(max_depth = 3, scale_pos_weight=weights, n_jobs =
4)
xgbTestOutliers = clf.fit(X_train,y_train).predict(X_test)
probabilities = clf.predict_proba(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/
_label.py:235: DataConversionWarning: A column-vector y was passed
```

```
when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py
:268: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

Evaluation de l'éfficacité d'xgboost

```
compTab2 = y_test.copy()
compTab2['xgbTestOutliers'] = xgbTestOutliers

nbElts = len(y_test)
nbAnomaliesTest = len(y_test.loc[y_test['isAttack']=='1'])
nbTruePos = len(compTab2.loc[(compTab2['isAttack']=='1') &\
                             (compTab2['xgbTestOutliers']=='1')])
print("Le nombre d'elements dans y_test est: " + str(nbElts) +"\n")
print("Le nombre d'anomalies dans y_test est: " + str(nbAnomaliesTest)
+ "\n")
print("Le nombre d'anomalies detectées par XGBoost: " + str(nbTruePos)
+ "\n")
```

```
Le nombre d'elements dans y_test est: 3000

Le nombre d'anomalies dans y_test est: 384

Le nombre d'anomalies detectées par XGBoost: 382
```

```
print("Pourcentage d'anomalies détectées par XGBoost par rapport au
nombre réel d'anomalie : " + str((nbTruePos/nbAnomaliesTest)*100) +
"%")
```

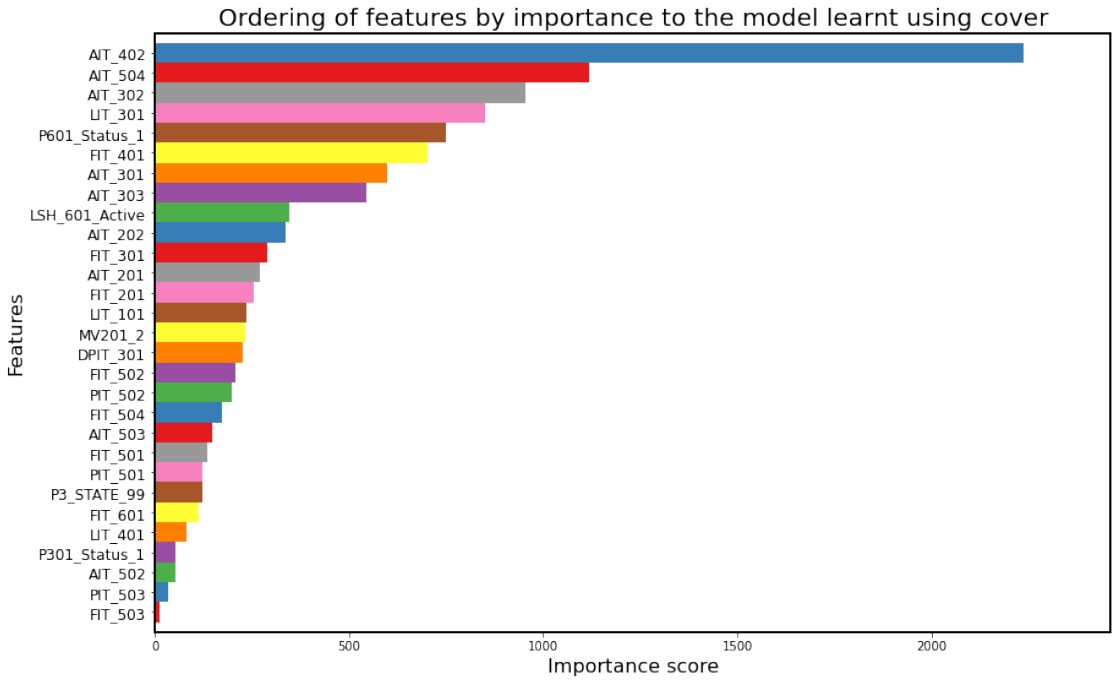
```
Pourcentage d'anomalies détectées par XGBoost par rapport au nombre
réel d'anomalie : 99.47916666666666%
```

Visualisation des resultats et interpretation

Cover

```
from xgboost import plot_importance,  plot_tree

# Affichage de l'importance des colonnes selon le cover
fig = plt.figure(figsize=(14,9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0,1,9))
ax = plot_importance(clf, height=1, color=colours, grid=False, \
                     show_values=False, importance_type='cover',
```

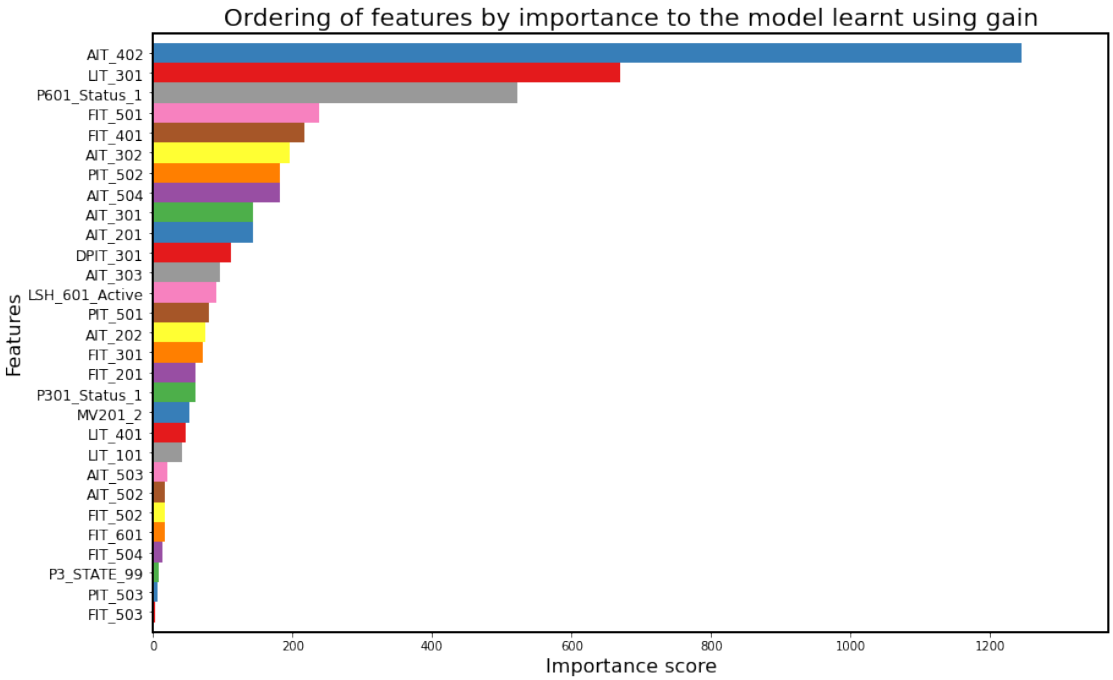
```
ax=ax)

for axis in ['top', 'bottom', 'left', 'right']:
  ax.spines[axis].set_linewidth(2)

ax.set_xlabel('Importance score', size=16);
ax.set_ylabel('Features', size=16);
ax.set_yticklabels(ax.get_yticklabels(), size=12);
ax.set_title('Ordering of features by importance to the model learnt '
\
              'using cover', size=20);
```



Gain

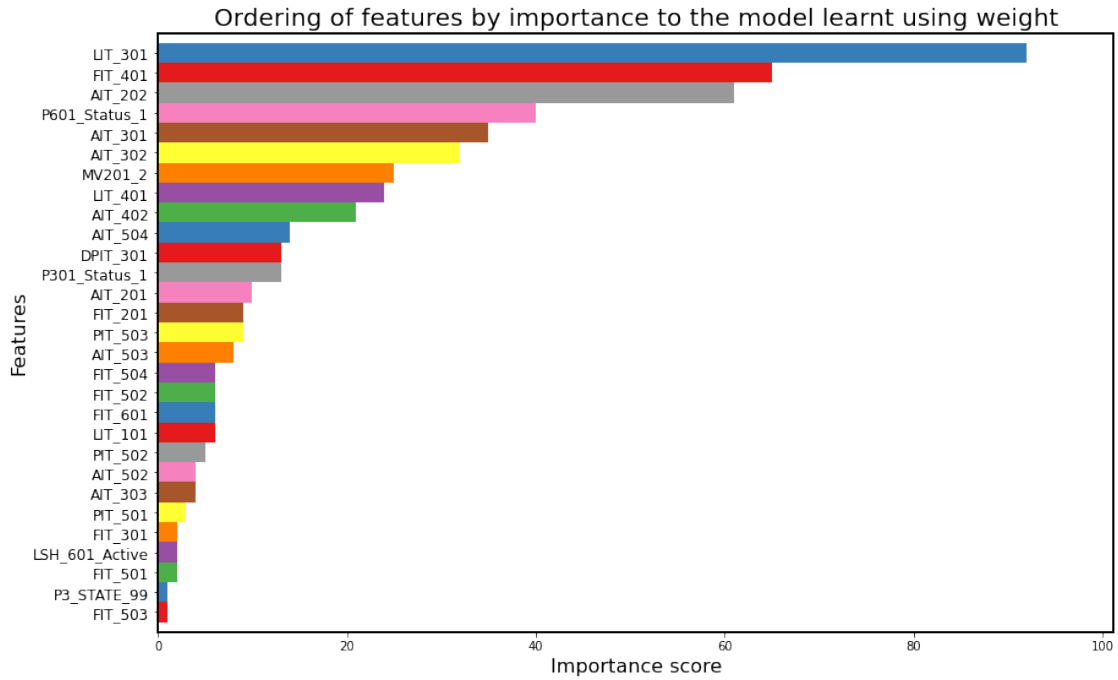
```
# Affichage de l'importance des colonnes selon le gain
fig = plt.figure(figsize=(14,9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0,1,9))
ax = plot_importance(clf, height=1, color=colours, grid=False, \
                     show_values=False, importance_type='gain', ax=ax)

for axis in ['top', 'bottom', 'left', 'right']:
  ax.spines[axis].set_linewidth(2)

ax.set_xlabel('Importance score', size=16);
ax.set_ylabel('Features', size=16);
ax.set_yticklabels(ax.get_yticklabels(), size=12);
ax.set_title('Ordering of features by importance to the model learnt '
```

```
\
              'using gain', size=20);
```



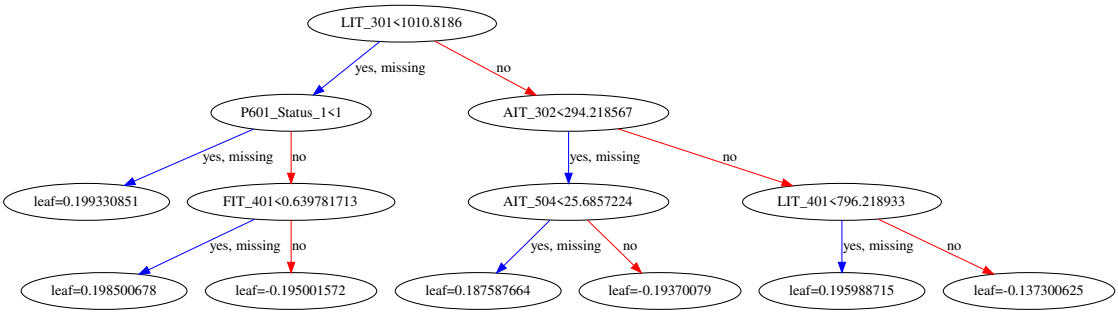
Weight

```
# Affichage de l'importance des colonnes selon le weight
fig = plt.figure(figsize=(14,9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0,1,9))
ax = plot_importance(clf, height=1, color=colours, grid=False, \
                     show_values=False, importance_type='weight',
ax=ax)

for axis in ['top', 'bottom', 'left', 'right']:
  ax.spines[axis].set_linewidth(2)

ax.set_xlabel('Importance score', size=16);
ax.set_ylabel('Features', size=16);
ax.set_yticklabels(ax.get_yticklabels(), size=12);
ax.set_title('Ordering of features by importance to the model learnt '
\
              'using weight', size=20);
```



Arbre du classifieur

```
from xgboost import to_graphviz
to_graphviz(clf)
```

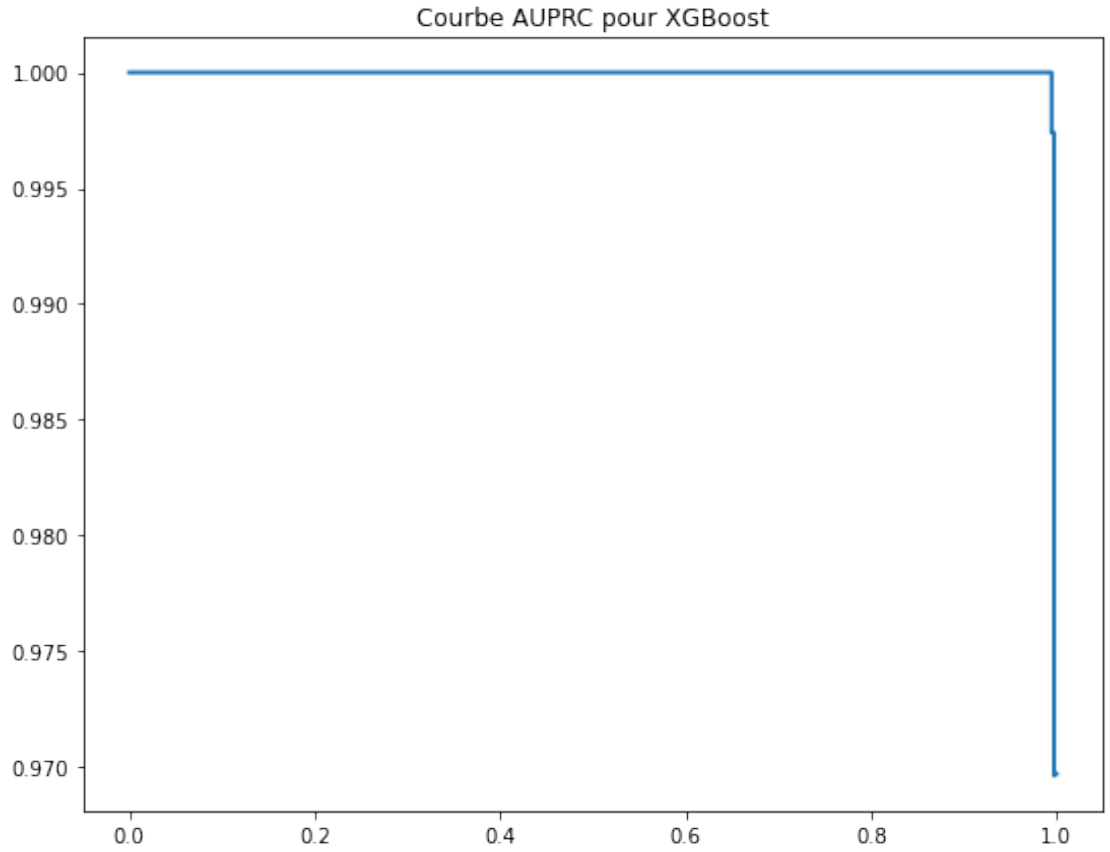


Courbe AUPRC

```
#AUPRC (Area Under the Precision-Recall Curve)
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

print('AUPRC = {}'.format(average_precision_score(y_true=y_test,\

y_score=probabilities[:,1],\
                                                  pos_label='1')))
print("\n\n\n")
precision, recall, thresholds = precision_recall_curve(y_test,\

probabilities[:,1],\
```

```
                                                        pos_label='1')

plt.figure(figsize=(9,7))
plt.title('Courbe AUPRC pour XGBoost')
plt.plot(recall, precision, lw=2.0)

AUPRC = 0.9999143041745581
```

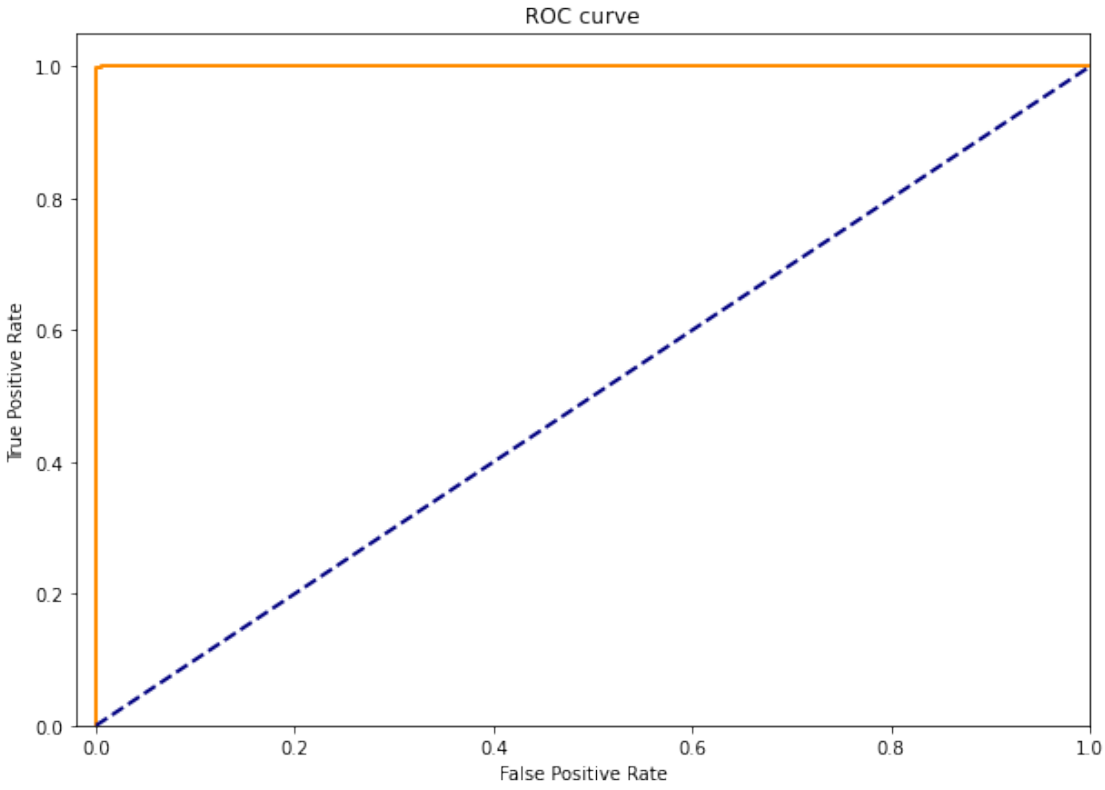
```
[<matplotlib.lines.Line2D at 0x7f1d8b7a0c50>]
```



Courbe ROC

```
#ROC (Receiver Operating Characteristic)
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, probabilities[:,1],
pos_label='1')
roc_auc = auc(fpr, tpr)
# plt.figure(figsize=(9,7))
# plt.plot(fpr, tpr, lw=2.0)
plt.figure(figsize=(10,7))
lw = 2
```

```
plt.plot(fpr, tpr, color='darkorange', lw=lw)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.02, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.show()
```



Interpretations

D'après ces résultats, nous pouvons conclure que l'attribut qui influence le plus grand nombre de prédictions (cover) est AIT 402 qui correspond à l'analyseur de NaHSO3 et de NaOCl dans la partie P4 du système de traitement.

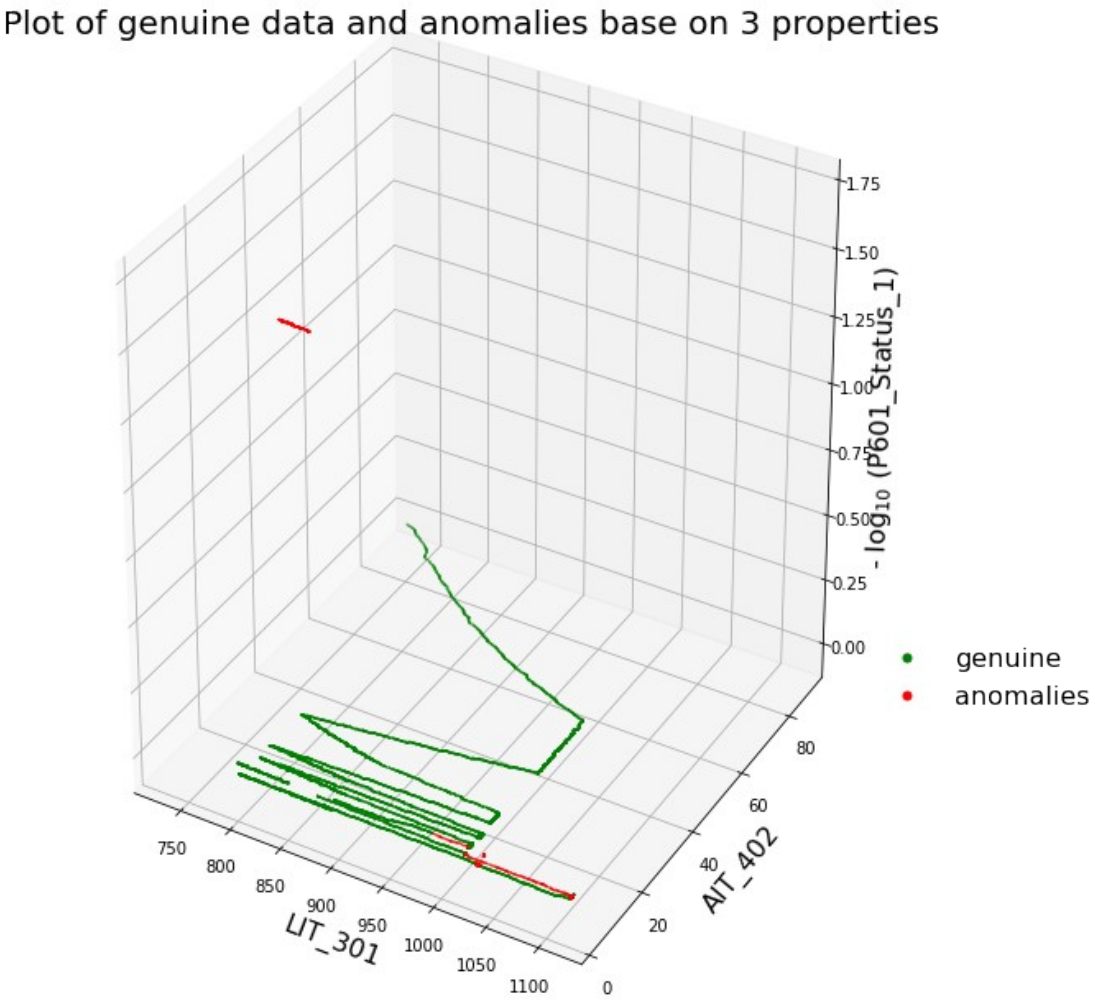
L'attribut le plus important et le plus discriminant (gain) pour les prédictions est aussi AIT 402.

L'attribut qui est utilisé dans le plus grand nombre de splits dans les arbres de décision créés par XGBoost (weight) est LIT 301, qui correspond au détecteur de niveau d'eau dans le UF feed tank de la partie P3 du système de traitement.

Donc pour améliorer la sécurité du système, il faudrait idéalement regarder de plus près et avec plus de soin les attributs AIT 402 et LIT 301.

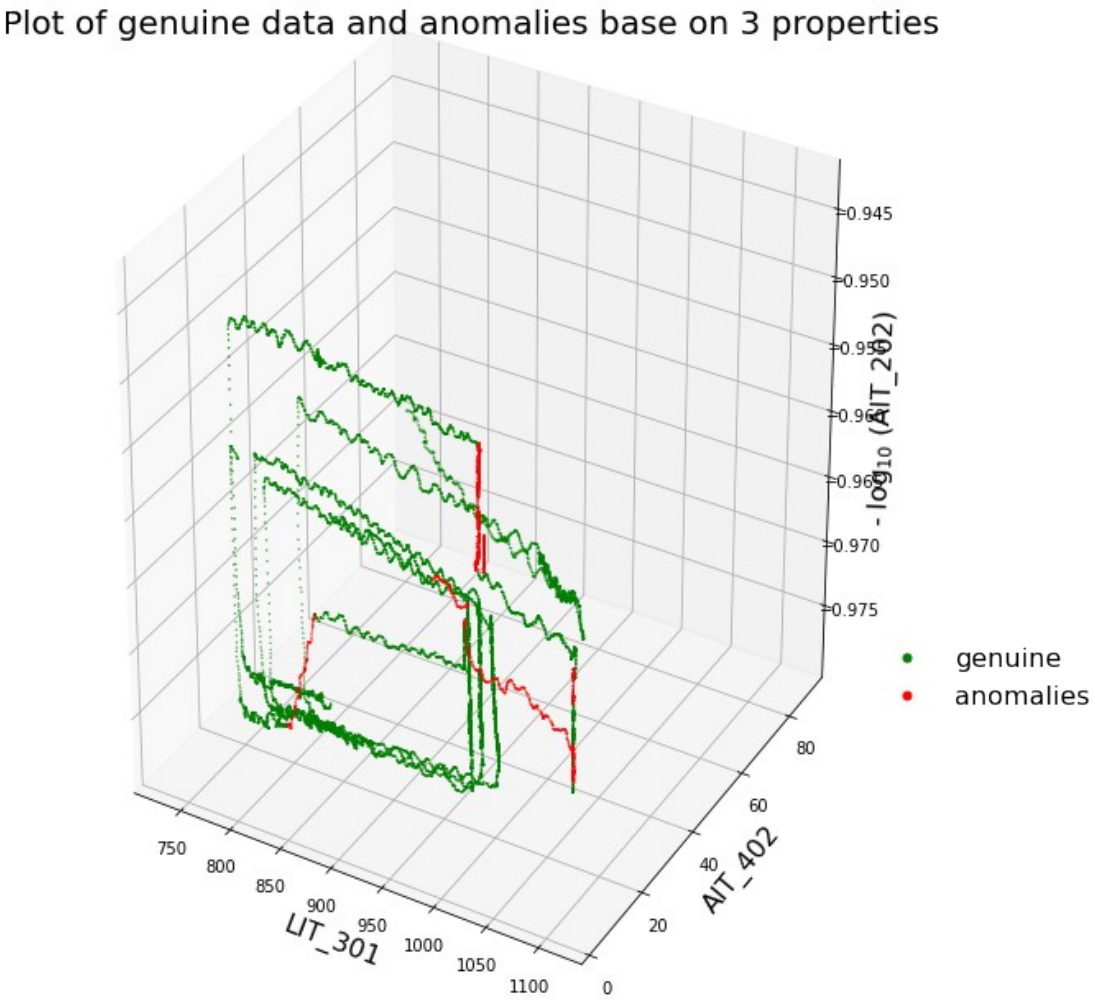
Plot des données en 3D

```
import matplotlib.lines as mlines

def show3D_data(X, Y, x_axis_name, y_axis_name, z_axis_name):
  x=x_axis_name
  y=y_axis_name
  z=z_axis_name
  zOffset = 0.02
  limit=len(X)

  sns.reset_orig()

  fig = plt.figure(figsize=(10,12))
  ax = fig.add_subplot(111,projection='3d')
  ax.scatter(X.loc[Y['isAttack']=='0',x][:limit],\
             X.loc[Y['isAttack']=='0',y][:limit], \
             -np.log10(X.loc[Y['isAttack']=='0',z][:limit]+zOffset),\
             c='g', marker='.', s=1, label='genuine')
  ax.scatter(X.loc[Y['isAttack']=='1',x][:limit],\
             X.loc[Y['isAttack']=='1',y][:limit],\
             -np.log10(X.loc[Y['isAttack']=='1',z][:limit] +
zOffset),\
             c='r', marker='.', s=1, label='anomalies')
  ax.set_xlabel(x,size=16);
  ax.set_ylabel(y, size=16);
  ax.set_zlabel('- log$_{10}$ (' +z+ ')', size=16)
  ax.set_title('Plot of genuine data and anomalies base on 3
properties',\
               size=20)

  plt.axis('tight')
  ax.grid(1)

  noAnomalyMaker = mlines.Line2D([], [], linewidth=0, color='g',
marker='.',\
                                 markersize=10, label='genuine')
  anomalyMaker = mlines.Line2D([], [], linewidth=0, color='r',
marker='.',\
                               markersize=10, label='anomalies')

  plt.legend(handles=[noAnomalyMaker, anomalyMaker], \
             bbox_to_anchor=(1.20, 0.38), frameon=False,
prop={'size':16});

show3D_data(X,Y,'LIT_301','AIT_402', 'P601_Status_1')
```



Plot d'autres attributs importants

```
show3D_data(X,Y,'LIT_301','AIT_402', 'AIT_202')
```



PCA

On crée un nouveau dataframe qui differencie les differentes attaques pour l'affichage.

```
new_y = pd.DataFrame(columns=['is_attack_detail'])

for i in range(len(df)):
    if (df.at[i,'FIT401_attack'] == '1'):
        new_row = {'is_attack_detail':'1'}
        new_y = new_y.append(new_row, ignore_index=True)
    elif (df.at[i,'LIT301_attack'] == '1'):
        new_row = {'is_attack_detail':'2'}
        new_y = new_y.append(new_row, ignore_index=True)
    elif (df.at[i,'P601_attack'] == '1'):
        new_row = {'is_attack_detail':'3'}
        new_y = new_y.append(new_row, ignore_index=True)
```

```
    elif (df.at[i,'MultiPoint_attack'] == '1'):
        new_row = {'is_attack_detail':'4'}
        new_y = new_y.append(new_row, ignore_index=True)
    elif (df.at[i,'MV501_attack'] == '1'):
        new_row = {'is_attack_detail':'5'}
        new_y = new_y.append(new_row, ignore_index=True)
    elif (df.at[i,'P301_attack'] == '1'):
        new_row = {'is_attack_detail':'6'}
        new_y = new_y.append(new_row, ignore_index=True)
    else:
        new_row = {'is_attack_detail':'0'}
        new_y = new_y.append(new_row, ignore_index=True)
```

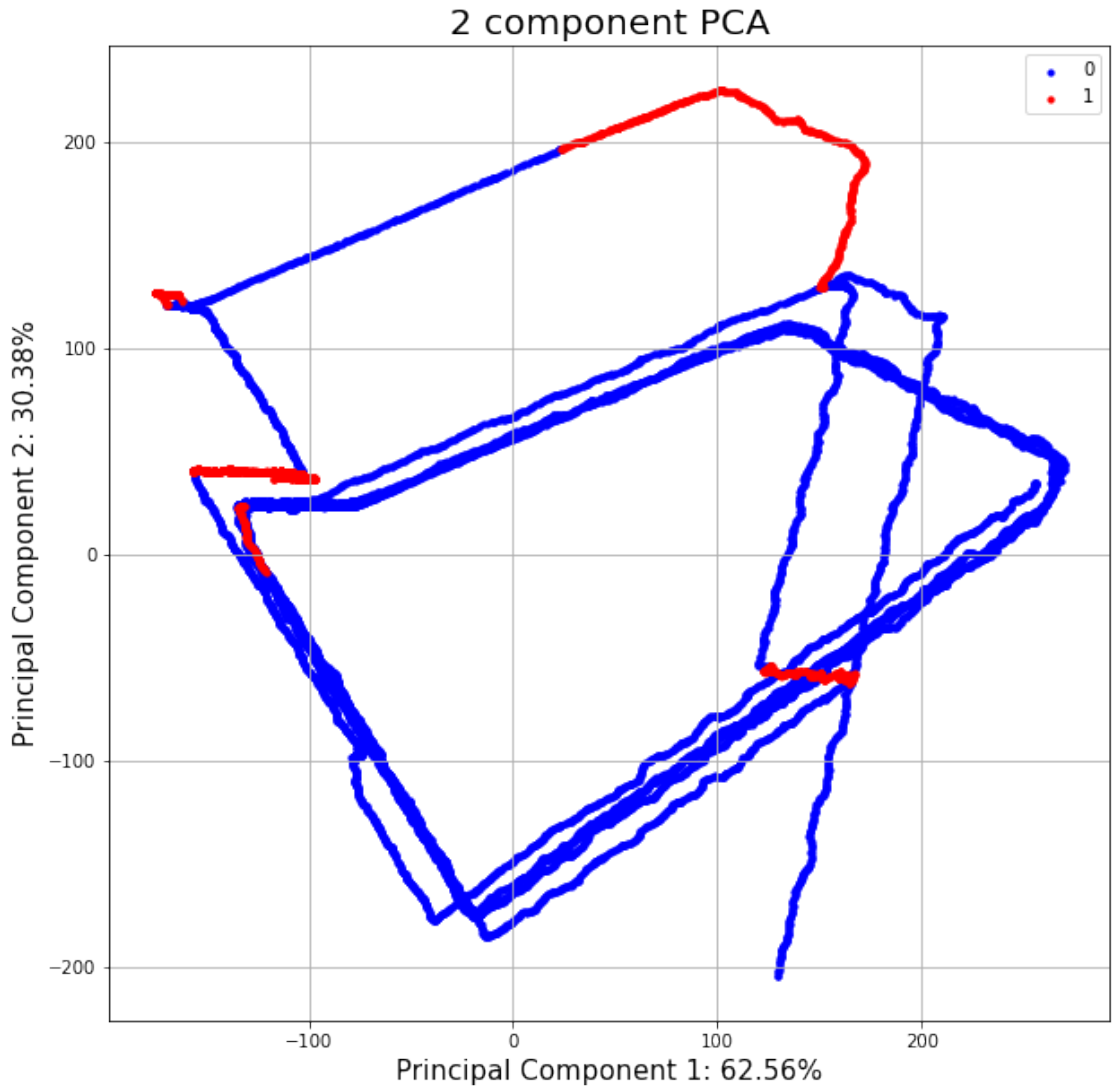
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

pca_bis = pca.fit(X)

principal_components = pca.fit_transform(X)

principalDf = pd.DataFrame(data = principal_components, columns =
['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, Y], axis = 1)
finalDf = pd.concat([finalDf, new_y], axis = 1)
finalDf.head()
```

```
   principal component 1  principal component 2 isAttack
is_attack_detail
0             129.264241            -204.826144        0
0
1             129.530638            -204.452694        0
0
2             129.180631            -203.004077        0
0
3             129.624511            -202.490752        0
0
4             130.275124            -202.057385        0
0
```

On affiche les données sur le plan de l'ACP en les différenciant selon 2 possibilités (attaque ou non)

```
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[0]*100)+'%',
fontsize = 15)
ax.set_ylabel('Principal Component 2:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[1]*100)+'%',
```

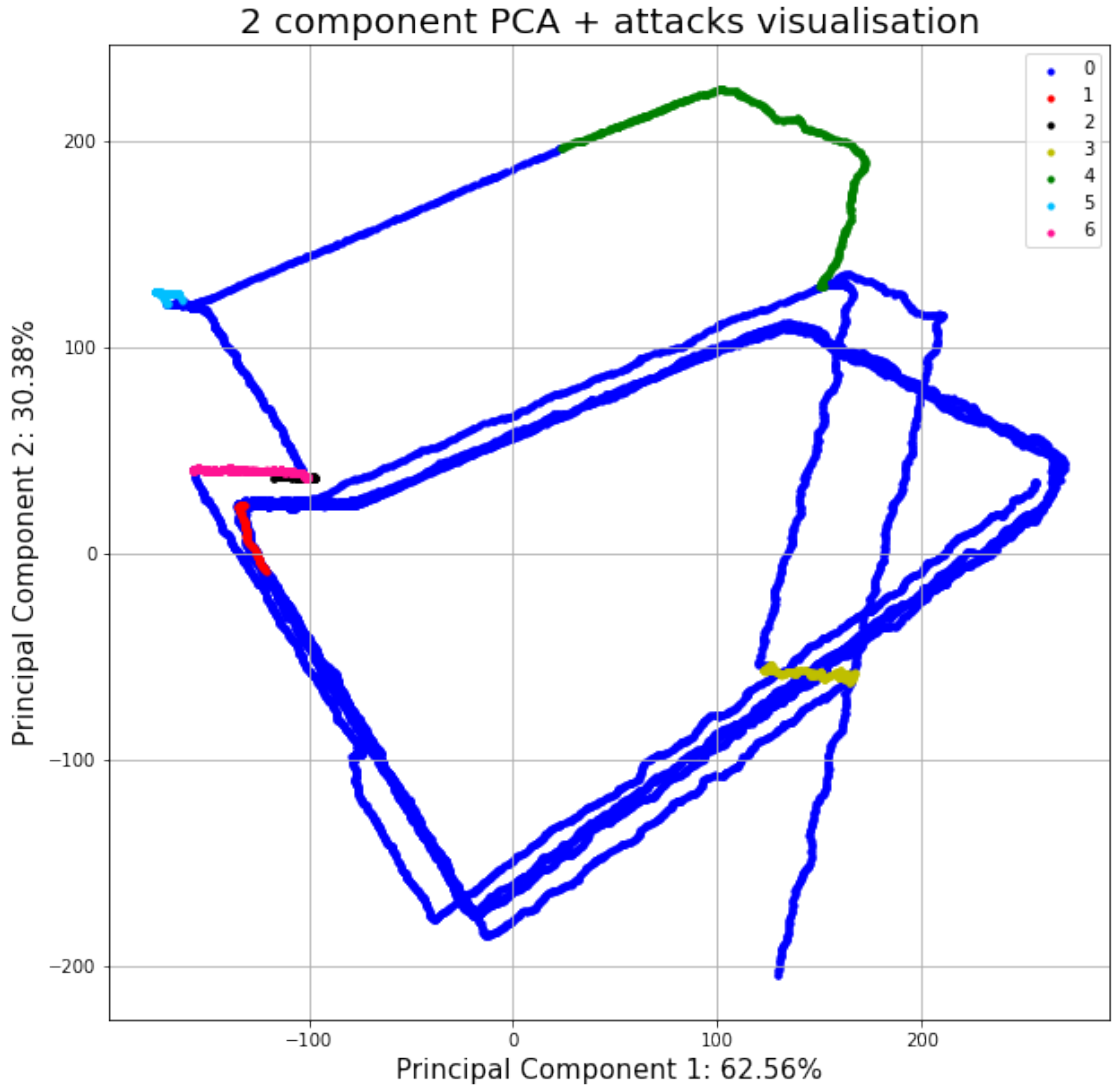
```
fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
labels = ['0', '1']
colors = ['b', 'r']
for target, color in zip(labels,colors):
    indicesToKeep = finalDf['isAttack'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 10)
ax.legend(labels)
ax.grid()
```



Ici on affiche les 6 attaques d'une couleur différente

```
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1:
```

```
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[0]*100)+'%',
fontsize = 15)
ax.set_ylabel('Principal Component 2:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[1]*100)+'%',
fontsize = 15)
ax.set_title('2 component PCA + attacks visualisation', fontsize = 20)
labels = ['0', '1', '2', '3', '4', '5', '6']
colors = ['b', 'r', 'black', 'y', 'g', 'deepskyblue', 'deeppink']
for target, color in zip(labels,colors):
    indicesToKeep = finalDf['is_attack_detail'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 10)
ax.legend(labels)
ax.grid()
```



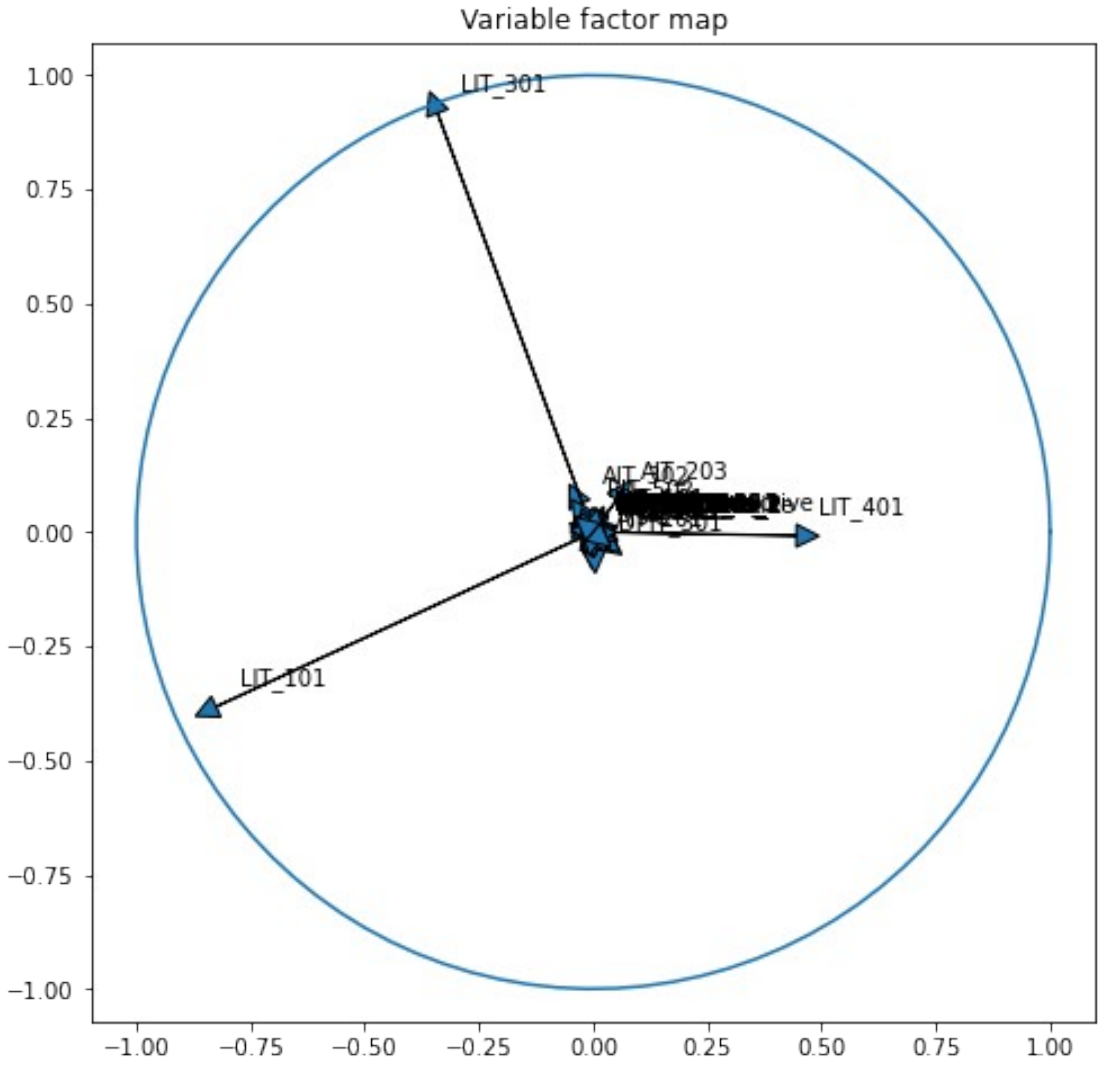
```
print(pca_bis.explained_variance_ratio_)
print(str(round(pca_bis.explained_variance_ratio_.sum() * 100, 2)) + '
%')
```

```
[0.62560855 0.30382306]
92.94 %
```

On voit que les 2 premières composantes principales gardent 92,94% de l'information ce qui nous semble correcte.

Cercle de correlation

```
(fig, ax) = plt.subplots(figsize=(8, 8))
for i in range(0, pca_bis.components_.shape[1]):
    ax.arrow(0,
             0,
             pca_bis.components_[0, i],
             pca_bis.components_[1, i],
             head_width=0.05,
             head_length=0.05)

    plt.text(pca_bis.components_[0, i] + 0.05,
             pca_bis.components_[1, i] + 0.05,
             X.columns.values[i])


an = np.linspace(0, 2 * np.pi, 100)
plt.plot(np.cos(an), np.sin(an))
plt.axis('equal')
ax.set_title('Variable factor map')
plt.show()
```



KMeans (2 clusters)

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

compTabKmeans = Y.copy()
compTabKmeans['kmeansOutliers'] = kmeans.labels_

nbElts = len(Y)
nbAnomaliesTest = len(Y.loc[Y['isAttack']=='1'])
nbTruePos = len(compTabKmeans.loc[(compTabKmeans['isAttack']=='1') &\
                                  (compTabKmeans['kmeansOutliers']==0)])
print("Le nombre d'élements dans Y est: " + str(nbElts) +"\n")
print("Le nombre d'anomalies dans Y est: " + str(nbAnomaliesTest) + "\
n")
```

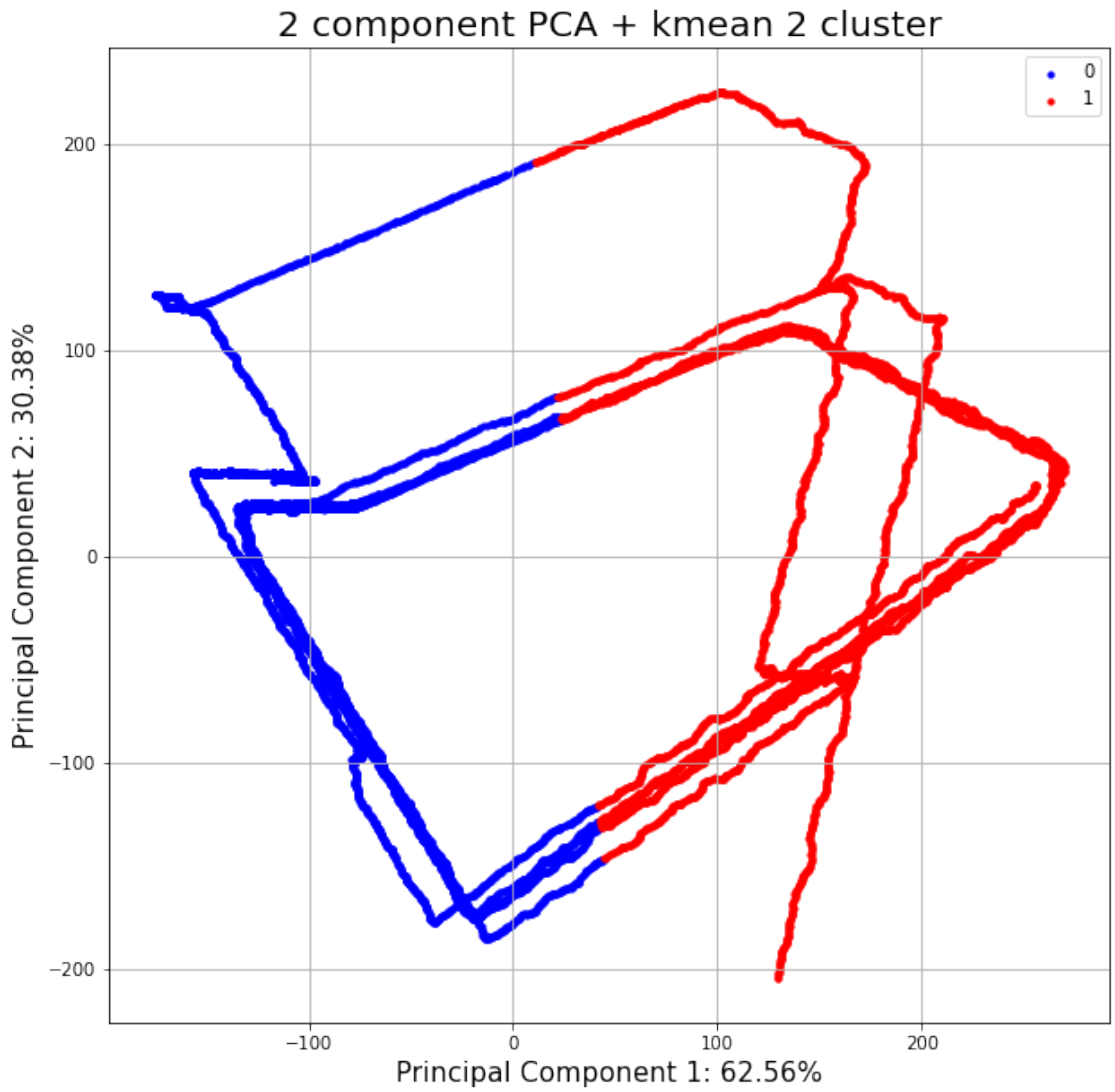
```
print("Le nombre d'anomalies detectées par Kmeans: " + str(nbTruePos)
+ "\n")
```

```
Le nombre d'élements dans Y est: 14996

Le nombre d'anomalies dans Y est: 2053

Le nombre d'anomalies detectées par Kmeans: 0
```

```
finalDf['Kmeans_2_cluster'] = kmeans.labels_.tolist()
finalDf.head()
```

```
   principal component 1  ...  Kmeans_2_cluster
0             129.264241  ...                 1
1             129.530638  ...                 1
2             129.180631  ...                 1
3             129.624511  ...                 1
4             130.275124  ...                 1

[5 rows x 5 columns]
```

```
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[0]*100)+'%',
fontsize = 15)
ax.set_ylabel('Principal Component 2:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[1]*100)+'%',
fontsize = 15)
ax.set_title('2 component PCA + kmean 2 cluster', fontsize = 20)
labels = [0, 1]
colors = ['b', 'r']
for target, color in zip(labels,colors):
    indicesToKeep = finalDf['Kmeans_2_cluster'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 10)
ax.legend(labels)
ax.grid()
```



KMeans (7 clusters)

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=7, random_state=0).fit(X)

print(kmeans.labels_)
```

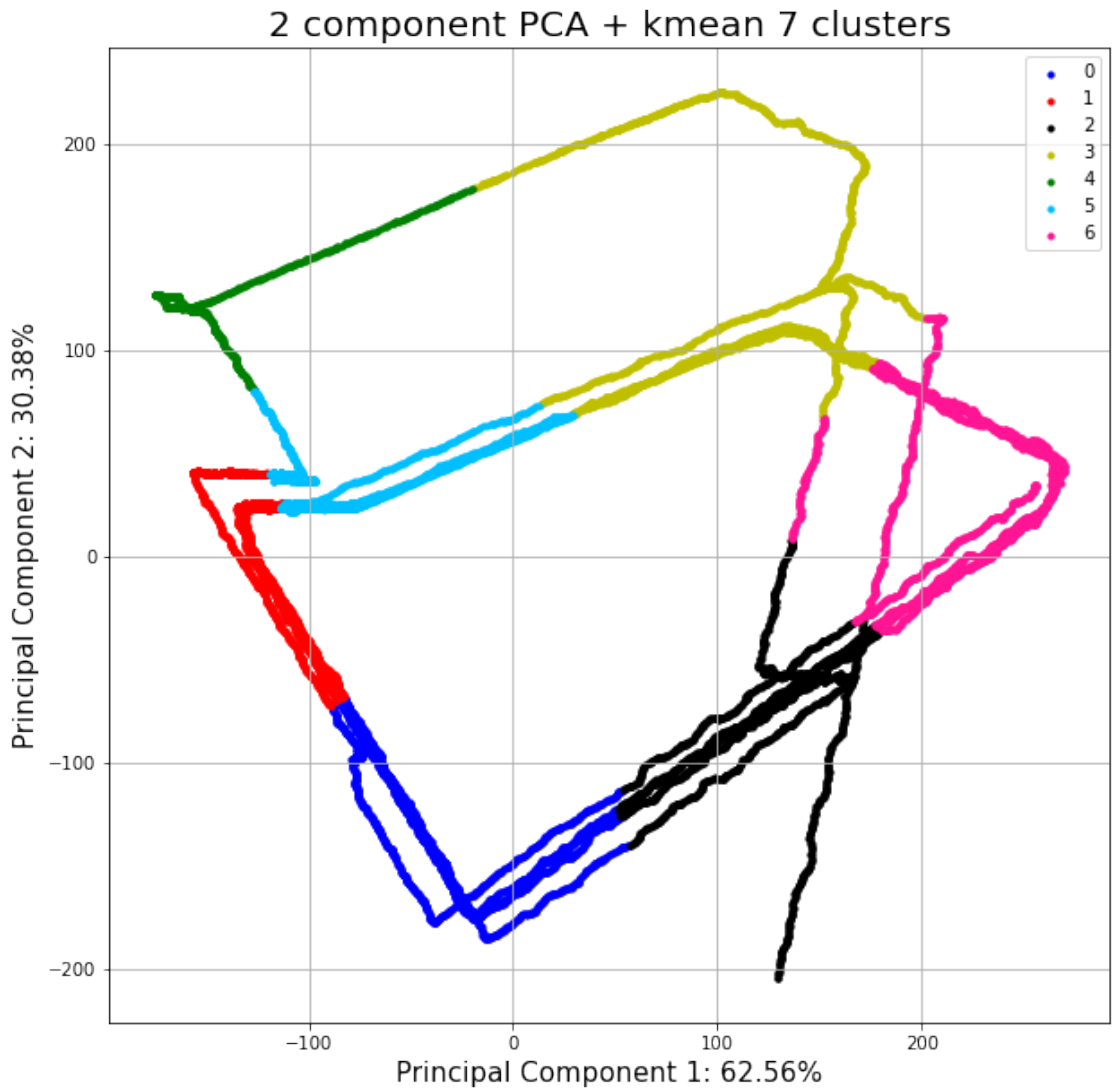
```
[2 2 2 ... 6 6 6]
```

```
finalDf['Kmeans_7_cluster'] = kmeans.labels_.tolist()
finalDf.head()
```

```
   principal component 1  ...  Kmeans_7_cluster
0             129.264241  ...                 2
1             129.530638  ...                 2
2             129.180631  ...                 2
```

```
3             129.624511  ...                 2
4             130.275124  ...                 2

[5 rows x 6 columns]
```

```
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[0]*100)+'%',
fontsize = 15)
ax.set_ylabel('Principal Component 2:
'+"{:.2f}".format(pca_bis.explained_variance_ratio_[1]*100)+'%',
fontsize = 15)
ax.set_title('2 component PCA + kmean 7 clusters', fontsize = 20)
labels = [0, 1, 2, 3, 4, 5, 6]
colors = ['b', 'r', 'black', 'y', 'g', 'deepskyblue', 'deeppink']
for target, color in zip(labels,colors):
    indicesToKeep = finalDf['Kmeans_7_cluster'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 10)
ax.legend(labels)
ax.grid()
```



LSTM

```
import plotly.express as px # to plot the time series plot
from sklearn import metrics # for the evaluation
from sklearn.preprocessing import LabelEncoder,MinMaxScaler
import tensorflow as tf

data = X.copy()

for i in data.select_dtypes('object').columns:
    le = LabelEncoder().fit(data[i])
    data[i] = le.transform(data[i])

X_scaler = MinMaxScaler()
Y_scaler = MinMaxScaler()
```

```
X_data = X_scaler.fit_transform(data)
Y_data = Y_scaler.fit_transform(Y)

def make_data_for_lstm(dataset, target, window, horizon, nb_train, n):
    # nb_test represente le nombre de données qui seront dans le jeu
d'entrainement sur n données
    # il y aura donc n - nb_train dans le jeu de test sur ces n
données

    X_train = []
    y_train = []
    X_test = []
    y_test = []

    cpt = 0

    for i in range(window, len(dataset)-window):
        indices = range(i-window, i)
        indicey = range(i+1, i+1+horizon)

        if (cpt < nb_train):
            X_train.append(dataset[indices])
            y_train.append(target[indicey])
        else:
            X_test.append(dataset[indices])
            y_test.append(target[indicey])

        cpt += 1

        if (cpt >= n):
            cpt = 0

    return np.array(X_train), np.array(X_test), np.array(y_train),
np.array(y_test)

hist_window = 30
horizon = 5
nb_train = 8
n = 10

X_train, X_test, y_train, y_test = make_data_for_lstm(X_data, Y_data,
hist_window, horizon, nb_train, n)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

(11950, 30, 77)
(11950, 5, 1)
```

```
(2986, 30, 77)
(2986, 5, 1)
```

```
batch_size = 256
buffer_size = 150
train_data = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_data =
train_data.cache().shuffle(buffer_size).batch(batch_size).repeat()
val_data = tf.data.Dataset.from_tensor_slices((X_test, y_test))
val_data = val_data.batch(batch_size).repeat()
```

```
lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(200,
return_sequences=True),
                                  input_shape=X_train.shape[-2:]),
    tf.keras.layers.Dense(20, activation='tanh'),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
    tf.keras.layers.Dense(20, activation='tanh'),
    tf.keras.layers.Dense(20, activation='tanh'),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(units=horizon),
])
lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 30, 400)           444800
_________________________________________________________________
dense (Dense)                (None, 30, 20)            8020
_________________________________________________________________
bidirectional_1 (Bidirection (None, 300)               205200
_________________________________________________________________
dense_1 (Dense)              (None, 20)                6020
_________________________________________________________________
dense_2 (Dense)              (None, 20)                420
_________________________________________________________________
dropout (Dropout)            (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 105
=================================================================
Total params: 664,565
Trainable params: 664,565
Non-trainable params: 0
_________________________________________________________________
```

```
model_path = 'Bidirectional_LSTM_Multivariate.h5'
early_stopings = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0, patience=10, verbose=1, mode='min')
```

```
checkpoint =  tf.keras.callbacks.ModelCheckpoint(model_path,
monitor='val_loss', save_best_only=True, mode='min', verbose=0)
callbacks=[early_stopings,checkpoint]

# Entraînement du modèle, environ 50 minutes pour les 50 epoques si
lancé sur colab, temps relativement long
# Avoir un meilleur processeur et/ou une carte graphique on devrait
pouvoir l'entraîner bien plus rapidement
history = lstm_model.fit(train_data,epochs=50,steps_per_epoch=100,
                         validation_data=val_data,validation_steps=50,
                         verbose=1,callbacks=callbacks)

plt.figure(figsize=(16,9))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train loss', 'validation loss'])
plt.show()

data_val = X_scaler.fit_transform(data)
val_rescaled = data_val.reshape(1, data_val.shape[0],
data_val.shape[1])
pred = lstm_model.predict(val_rescaled)
pred_Inverse = Y_scaler.inverse_transform(pred)

print(pred_Inverse)
```