Etude de cas - Captation de chocs sur les voies d'un tramway

Descriptions des données

L'étude de cas porte sur la surveillance des rails de tramways et de métro.

Le client souhaite un système automatique de surveillance des voies, qui lui permette de repérer en quasi temps réel l'apparition ou l'aggravation majeure de défauts des rails pouvant entrainer notamment des risques pour la sécurité des passagers.

A l'heure actuelle les dispositifs de surveillance sont coûteux (instrumentations lourdes) ou peu fiables (tournées visuelles). L'instrumentation non intrusive vise donc à proposer un dispositif complémentaire à ces tournées en identifiant en amont les zones où les défauts potentiels sont susceptible de se trouver.

Nous disposons de données récoltées sur une semaine d'expérience de captation de chocs sur les voies d'un tramway d'une grande métropole. L'émission de données est en continue, toutes les 5 secondes, sauf en cas d'arrêt prolongé des rames. Un enregistrement est effectué en cas de détection de choc. Le choc est détecté quand la vibration atteint un certain seuil.

Voici quelques informations essentielles sur la ligne instrumentée :

```
- est une ligne de metro aérien d’une grande capitale
- un seul train a été équipé
- le train ne change pas de sens
- le train circule sur deux tracks dédiés (deux directions possible : PANLAP ou LAPPAN), sans “fourche”.
- les données disponibles sont des données du 1er semestre 2021
```

In [8]:
```
#librairies
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
#import seaborn as sns
#sns.set_theme()

import time
import datetime
```

In [9]:
```
#Tables
data_gps = pd.read_csv("C:/Users/valentin/Documents/University/M2/Etude_de_cas/2023MIS/d
data_located = pd.read_csv("C:/Users/valentin/Documents/University/M2/Etude_de_cas/2023M
data_journeys_detail = pd.read_csv("C:/Users/valentin/Documents/University/M2/Etude_de_c
#data_signal = pd.read_csv("C:/Users/cloee.LAPTOP-4MUN60P0/OneDrive/Bureau/etudecas/2023
```

Données des trajets

In [10]:
```
data_journeys_detail
```

Out[10]:

	id_serie	direction	sub_subseq_start	sub_subseq_end	pk_min	pk_max	nb_points	area	su

0	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:25:00.022	2021-02-01 00:29:10.020	10360.805360	11699.673374	19	_002	
1	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:29:30.022	2021-02-01 00:31:05.021	11892.421504	12972.885931	19	_003	1
2	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:31:10.019	2021-02-01 00:33:10.019	13043.234435	14482.103059	21	_004	1
3	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:33:30.020	2021-02-01 00:35:10.019	14739.138865	15805.048502	20	_005	1
4	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:35:30.021	2021-02-01 00:38:10.017	16041.106610	18279.755237	29	_006	1
...	...	...	...	...	...	...	...	...	
48756	2021-07-22_23:11:51-23:38:51	LAPPAN	2021-07-22 23:19:51.017	2021-07-22 23:24:06.016	18348.305195	19808.073443	44	_007	1
48757	2021-07-22_23:11:51-23:38:51	LAPPAN	2021-07-22 23:18:06.017	2021-07-22 23:19:46.017	19883.728224	20992.005237	11	_008	
48758	2021-07-22_23:11:51-23:38:51	LAPPAN	2021-07-22 23:15:26.014	2021-07-22 23:18:01.017	21090.923887	22955.548596	29	_009	1
48759	2021-07-22_23:11:51-23:38:51	LAPPAN	2021-07-22 23:12:06.015	2021-07-22 23:15:21.014	23040.657940	25100.873221	31	_010	1
48760	2021-07-22_23:11:51-23:38:51	LAPPAN	2021-07-22 23:11:51.014	2021-07-22 23:12:01.013	25215.920880	25256.238625	2	_011	

48761 rows × 11 columns

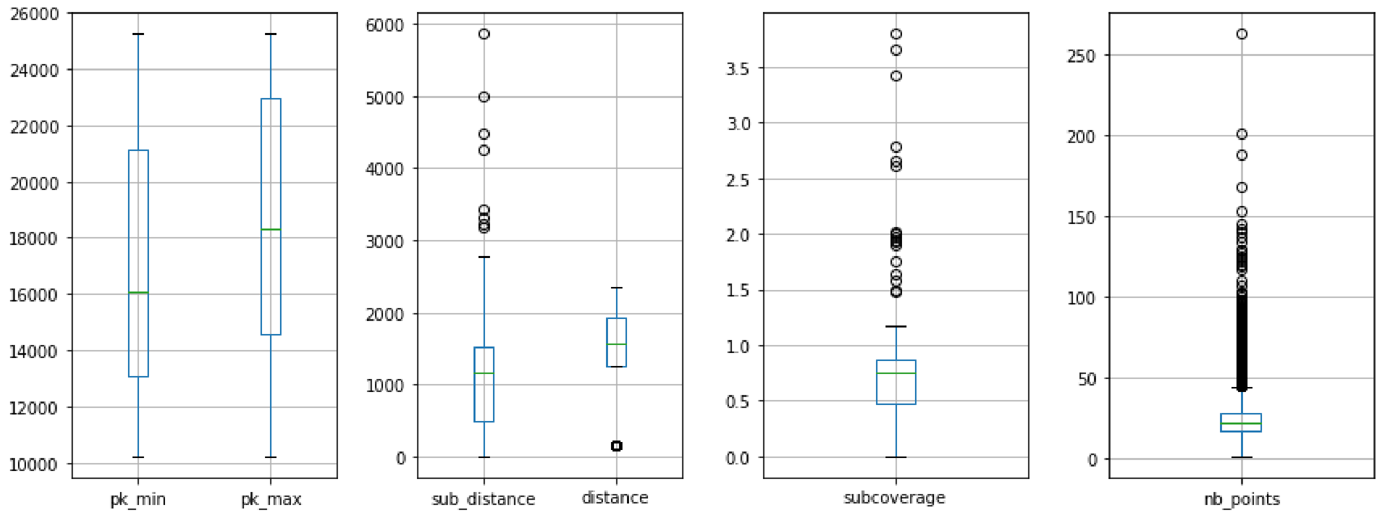
Ce premier jeu de données de taille 48 761 observations et 11 variables ne comportent pas de valeurs manquantes. Dans cette table on y trouve pour chaque portion de trajet effectuée par un tramway, la date et l'heure de départ, la direction du tramway, la date et l'heure du début de de la fin de la portion enregistrée (5 secondes environ). Cette portion enregistre le numéro de la station, la distance effectuée, le minimum et le maximum des points kilométriques, le rapport entre la distance parcourue et celle enregistrée, 'subcoverage'.

In [11]:

```
figure = plt.figure(figsize = (14, 5))
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,
                       right = 0.9, top = 0.9, wspace = 0.3, hspace = 0.1)
axes = figure.add_subplot(1, 4, 1)
data_journeys_detail.boxplot(column=['pk_min','pk_max'])
axes = figure.add_subplot(1, 4, 2)
data_journeys_detail.boxplot(column=['sub_distance','distance'])
axes = figure.add_subplot(1, 4, 3)
data_journeys_detail.boxplot(column=['subcoverage'])
axes = figure.add_subplot(1, 4, 4)
data_journeys_detail.boxplot(column=['nb_points'])
```

Out[11]:

```
<AxesSubplot: >
```



In [12]:
```
data_journeys_detail.describe(include="float64").transpose()
```

Out[12]:

	count	mean	std	min	25%	50%	75%	
pk_min	48761.0	17310.165712	4695.605532	10223.001931	13064.208598	16093.573204	21112.457418	25264.9
pk_max	48761.0	18684.535719	4623.466960	10237.768796	14583.843541	18294.827997	22959.216529	25265.9
sub_distance	48761.0	1050.195651	603.228492	0.000000	485.169297	1167.756651	1511.373205	5866.3
distance	48761.0	1512.295310	564.333498	163.025851	1250.000000	1559.000000	1933.000000	2356.0
subcoverage	48761.0	0.672357	0.272373	0.000000	0.471934	0.759122	0.864151	3.8

Données GPS

In [13]:
```
data_gps
```

Out[13]:

	timestamp	sensor	pk	timestamp_dt	id_serie	direction	nearest_pk	speed_prec	spe
0	1617235275019	XXX_3	10245.018825	2021-04-01 00:01:15.019	2021-04-01 00:01:10.020	PANLAP	10245.28	3.740910	5
1	1617235280020	XXX_3	10253.188120	2021-04-01 00:01:20.020	2021-04-01 00:01:10.020	PANLAP	10255.28	5.880716	6
2	1617235285020	XXX_3	10262.140325	2021-04-01 00:01:25.020	2021-04-01 00:01:10.020	PANLAP	10265.28	6.445588	7
3	1617235290020	XXX_3	10271.950702	2021-04-01 00:01:30.020	2021-04-01 00:01:10.020	PANLAP	10275.28	7.063472	5
4	1617235295020	XXX_3	10279.218242	2021-04-01 00:01:35.020	2021-04-01 00:01:10.020	PANLAP	10275.28	5.232628	3
...	...	...	...	...	...	...	...	...	
583779	1625055538022	XXX_3	23655.005060	2021-06-30 12:18:58.022	2021-06-30 12:18:53.022	LAPPAN	23655.13	60.276973	
583780	1625055558023	XXX_3	23370.757261	2021-06-30 12:19:18.023	2021-06-30 12:19:13.022	LAPPAN	23375.13	47.277533	
583781	1625055578022	XXX_3	23197.077074	2021-06-30 12:19:38.022	2021-06-30 12:19:33.021	LAPPAN	23195.13	19.900385	
583782	1625055618022	XXX_3	23166.869811	2021-06-30 12:20:18.022	2021-06-30 12:20:13.022	LAPPAN	23165.13	1.042909	

583783	1625055658023	XXX_3	22670.777278	2021-06-30 12:20:58.023	2021-06-30 12:20:53.023	LAPPAN	22675.13	77.592979	

583784 rows × 9 columns

In [14]:
```
data_gps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583784 entries, 0 to 583783
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   timestamp     583784 non-null  int64  
 1   sensor        583784 non-null  object 
 2   pk            583784 non-null  float64
 3   timestamp_dt  583784 non-null  object 
 4   id_serie      583784 non-null  object 
 5   direction     583784 non-null  object 
 6   nearest_pk    583784 non-null  float64
 7   speed_prec    583784 non-null  float64
 8   speed_next    544571 non-null  float64
dtypes: float64(4), int64(1), object(4)
memory usage: 40.1+ MB
```

In [15]:
```
print(f"La variable 'speed_next' contient {100*(583784-544571)/583784:.2f} % de valeurs 
```

```
La variable 'speed_next' contient 6.72 % de valeurs manquantes.
```

La table de données GPS est composée de 9 variables et 583 784 observations. Nous disposons de 4 variables de type 'object', 4 variables de type 'float' et une variable de type 'int'.

Analyse des variables

In [16]:
```
# Variables de type 'object'

print('Valeurs de la variable \'sensor\':', data_gps.sensor.value_counts())
print('Valeurs de la variable \'timestamp_dt\':', data_gps.timestamp_dt.value_counts())
print('Valeurs de la variable \'id_serie\':', data_gps.id_serie.value_counts())
print('Valeurs de la variable \'direction\':', data_gps.direction.value_counts())
```

```
Valeurs de la variable 'sensor': XXX_3    583784
Name: sensor, dtype: int64
Valeurs de la variable 'timestamp_dt': 2021-04-01 00:01:15.019    1
2021-05-31 22:08:45.023    1
2021-05-31 22:08:15.022    1
2021-05-31 22:08:20.023    1
2021-05-31 22:08:25.023    1
                          ..
2021-04-28 23:03:22.019    1
2021-04-28 23:03:17.019    1
2021-04-28 23:03:12.019    1
2021-04-28 23:03:07.019    1
2021-06-30 12:20:58.023    1
Name: timestamp_dt, Length: 583784, dtype: int64
Valeurs de la variable 'id_serie': 2021-05-09 12:19:05.018    281
2021-04-03 23:13:09.018    269
2021-04-12 20:01:45.017    239
2021-04-03 14:10:50.019    234
2021-04-05 10:08:41.021    232
                          ...
2021-04-30 01:38:13.018      1
2021-04-30 01:22:43.019      1
2021-04-30 01:18:33.017      1
```

```
2021-04-30 01:17:18.017      1
2021-06-30 12:20:53.023      1
Name: id_serie, Length: 35200, dtype: int64
Valeurs de la variable 'direction': PANLAP    302662
LAPPAN    281122
Name: direction, dtype: int64
```

La variable 'sensor' est l'identifiant du capteur GPS, elle peut être négligée dans cette table car toutes les observations ont été enregistrées avec ce même capteur GPS.
La variable 'timestamp_dt' est une variable temporelle correspondant à la date et heure de chaque enregistrement et donc à peu près tous les 5 secondes.
La variable 'id_serie' est une variable temporelle également. Elle contient des valeurs dupliquées.
Pour finir nous avons la variable 'direction' qui indique le sens du trajet. Il y a deux choix possibles: direction PANLAP ou direction LAPPAN.

In [17]:
```python
# Variables de type 'float'

print('Valeurs de la variable \'pk\':', data_gps.pk.value_counts())
print('Valeurs de la variable \'nearest_pk\':', data_gps.nearest_pk.value_counts())
print('Valeurs de la variable \'speed_prec\':', data_gps.speed_prec.value_counts())
print('Valeurs de la variable \'speed_next\':', data_gps.speed_next.value_counts())
```

```
Valeurs de la variable 'pk': 10365.280000    60
10375.130000    58
19975.280000    31
10355.280000    25
25245.280000    21
                ..
24989.867344     1
24624.152280     1
24542.273078     1
24459.022639     1
22670.777278     1
Name: pk, Length: 581966, dtype: int64
Valeurs de la variable 'nearest_pk': 11935.13    8194
14785.13    8128
18455.13    7876
10365.28    7840
16095.13    7583
            ...
10415.13       1
20505.13       1
11285.13       1
20685.13       1
11485.28       1
Name: nearest_pk, Length: 2800, dtype: int64
Valeurs de la variable 'speed_prec': 0.000000     30
0.093616      2
0.062909      2
7.200000      2
1.270048      1
             ..
0.399357      1
0.506815      1
0.920747      1
0.549890      1
77.592979     1
Name: speed_prec, Length: 583752, dtype: int64
Valeurs de la variable 'speed_next': 0.000000     29
0.093616      2
0.062909      2
11.297639     2
55.711057     1
```

```
            ..
0.412412      1
0.582026      1
0.371081      1
0.759022      1
0.337420      1
Name: speed_next, Length: 544540, dtype: int64
```

La variable 'pk' contient des valeurs non uniques même si cela correspond à des données continues car des points kilométriques sont déja prédéfinis au préalable.
La variable 'nearest_pk' est construite de la même manière et représente le point de référence le plus proche de la position du GPS.
Les variables 'speed_next' et 'speed_prec' représentent respectivement la vitesse enregistrée du point kilométrique suivant et précédent.

In [18]:
```
# Variables de type 'int'

print('Valeurs de la variable \'timestamp\':', data_gps.timestamp.value_counts())
```

```
Valeurs de la variable 'timestamp': 1617235275019    1
1622498925023    1
1622498895022    1
1622498900023    1
1622498905023    1
                ..
1619651002019    1
1619650997019    1
1619650992019    1
1619650987019    1
1625055658023    1
Name: timestamp, Length: 583784, dtype: int64
```

La variable 'timestamp' est la seule variable discrète dans notre jeu de données et elle correspond à l'instant de l'enregistrement.

In [19]:
```
data_gps.describe(include="float64").transpose()
```

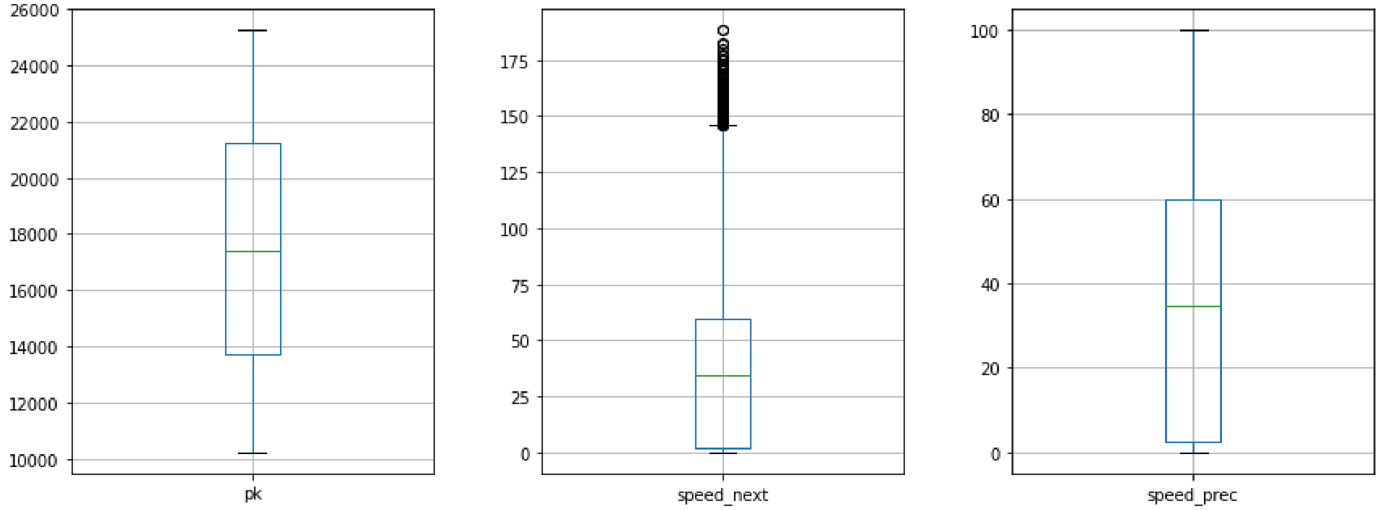
Out[19]:

	count	mean	std	min	25%	50%	75%	
pk	583784.0	17575.654954	4414.369398	10223.285841	13705.668359	17371.609620	21233.510027	25265.9
nearest_pk	583784.0	17575.673265	4414.426085	10225.130000	13705.130000	17375.130000	21235.130000	25268.7
speed_prec	583784.0	34.521597	28.243035	0.000000	2.433499	34.838479	59.732193	99.9
speed_next	544571.0	34.438940	28.430914	0.000000	2.277984	34.349877	59.737663	188.3

Il ne semble pas avoir de valeurs aberrantes dans les observations de la table de données GPS.

In [20]:
```
figure = plt.figure(figsize = (14, 5))
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,
                       right = 0.9, top = 0.9, wspace = 0.3, hspace = 0.1)
axes = figure.add_subplot(1, 3, 1)
data_gps.boxplot(column=['pk'])
axes = figure.add_subplot(1, 3, 2)
data_gps.boxplot(column=['speed_next'])
axes = figure.add_subplot(1, 3, 3)
data_gps.boxplot(column=['speed_prec'])
```

Out[20]:
```
<AxesSubplot: >
```



Données de chocs

In [21]:
```
data_located
```

Out[21]:

	timestamp	pk	sensor	direction	x	y	
0	1617235693516	12104.077518	XXX_2	PANLAP	[0.8668750000000001, -0.43312500000000004, 0.0...	[-3.035, -5.535, 0.6649999999999999, -0.835000...	[-1.034375, - 0.565625, -
1	1617235693516	12104.077518	XXX_2	PANLAP	[0.8668750000000001, -0.43312500000000004, 0.0...	[-3.035, -5.535, 0.6649999999999999, -0.835000...	[-1.034375, - 0.565625, -
2	1617235693516	12104.077518	XXX_2	PANLAP	[0.8668750000000001, -0.43312500000000004, 0.0...	[-3.035, -5.535, 0.6649999999999999, -0.835000...	[-1.034375, - 0.565625, -
3	1617235693516	12104.077518	XXX_2	PANLAP	[0.8668750000000001, -0.43312500000000004, 0.0...	[-3.035, -5.535, 0.6649999999999999, -0.835000...	[-1.034375, - 0.565625, -
4	1617235799669	13361.726185	XXX_2	PANLAP	[0.78625, 0.38625000000000004, -0.41375, 0.286...	[-1.888125, 1.511875, 0.511875, 2.411875, 1.81...	[0.360625, - -0.539374999
...	...	...	...	...	...	...	
2995	1617666408598	23543.877808	XXX_2	PANLAP	[-1.084375, 0.415625, 0.315625, -0.984375, 0.9...	[-1.50875, -3.7087499999999998, -3.80875, -2.3...	[0.698125, 1.298125, -
2996	1617666408768	23547.636918	XXX_1	PANLAP	[-0.34125, -0.041249999999999995, 0.55875, -0....	[-4.734375, -5.434375, -2.134375, 0.5656249999...	[0.311249999 2.11125
2997	1617666450296	24518.120466	XXX_2	PANLAP	[0.985, 1.385, -0.715, -0.915, 0.885, 0.485, 0...	[1.050625, -1.449375, 1.650625, 4.450625000000...	[-1.806875, 0.293124999
2998	1617668752278	11620.193952	XXX_1	PANLAP	[0.266875, 0.06687500000000002, 0.166875000000...	[-0.571875, -0.471875, 2.428125, 1.728125, -0....	[-0.09875, 0.30125
2999	1617668753096	11634.654126	XXX_1	PANLAP	[-0.395, -0.195, -0.395, 0.505, 0.305, 0.405, ...	[-0.546875, -0.646875, 1.6531250000000002, -0....	[0.3275 -1.0725 -0

3000 rows × 17 columns

In [22]:
```
data_located.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   timestamp               3000 non-null   int64
 1   pk                      3000 non-null   float64
 2   sensor                  3000 non-null   object
 3   direction               3000 non-null   object
 4   x                       3000 non-null   object
 5   y                       3000 non-null   object
 6   z                       3000 non-null   object
 7   pk_merge_prev           3000 non-null   float64
 8   speed_prec_merge_prev   3000 non-null   float64
 9   speed_next_merge_prev   3000 non-null   float64
 10  pk_merge_next           3000 non-null   float64
 11  speed_prec_merge_next   3000 non-null   float64
 12  speed_next_merge_next   2732 non-null   float64
 13  timeframe_gps_prev      3000 non-null   float64
 14  timeframe_gps_next      3000 non-null   float64
 15  average_speed_km_h_prev 3000 non-null   float64
 16  average_speed_km_h_next 3000 non-null   float64
dtypes: float64(11), int64(1), object(5)
memory usage: 398.6+ KB
```

In [23]:
```
print(f"La variable 'speed_next_merge' contient {100*(3000-2732)/3000:.2f} % de valeurs
```

```
La variable 'speed_next_merge' contient 8.93 % de valeurs manquantes.
```

La table de données des chocs est composée de 17 variables et 3000 observations. Nous avons 11 variables de type 'float', 1 variable de type 'int' et 5 variables de type 'object'.

Analyse des variables

In [24]:
```
# Variables de type 'object'

print('Valeurs de la variable \'x\':')
data_located.x
```

Out[24]:
```
Valeurs de la variable 'x':
0       [0.8668750000000001, -0.43312500000000004, 0.0...
1       [0.8668750000000001, -0.43312500000000004, 0.0...
2       [0.8668750000000001, -0.43312500000000004, 0.0...
3       [0.8668750000000001, -0.43312500000000004, 0.0...
4       [0.78625, 0.38625000000000004, -0.41375, 0.286...
                              ...
2995    [-1.084375, 0.415625, 0.315625, -0.984375, 0.9...
2996    [-0.34125, -0.041249999999999995, 0.55875, -0....
2997    [0.985, 1.385, -0.715, -0.915, 0.885, 0.485, 0...
2998    [0.266875, 0.06687500000000002, 0.166875000000...
2999    [-0.395, -0.195, -0.395, 0.505, 0.305, 0.405, ...
Name: x, Length: 3000, dtype: object
```

Les variables 'x', 'y' et 'z' correspondent aux positions des accélérations enregistrées par les 2 accéléromètres (avec une fréquence d'échantillonage de 400Hz et un buffer de 80 points). Ces variables doivent être transformées en vecteurs numériques.
Contrairement au jeu de données précédent, la direction est unique pour toutes les observations: direction PANLAP.

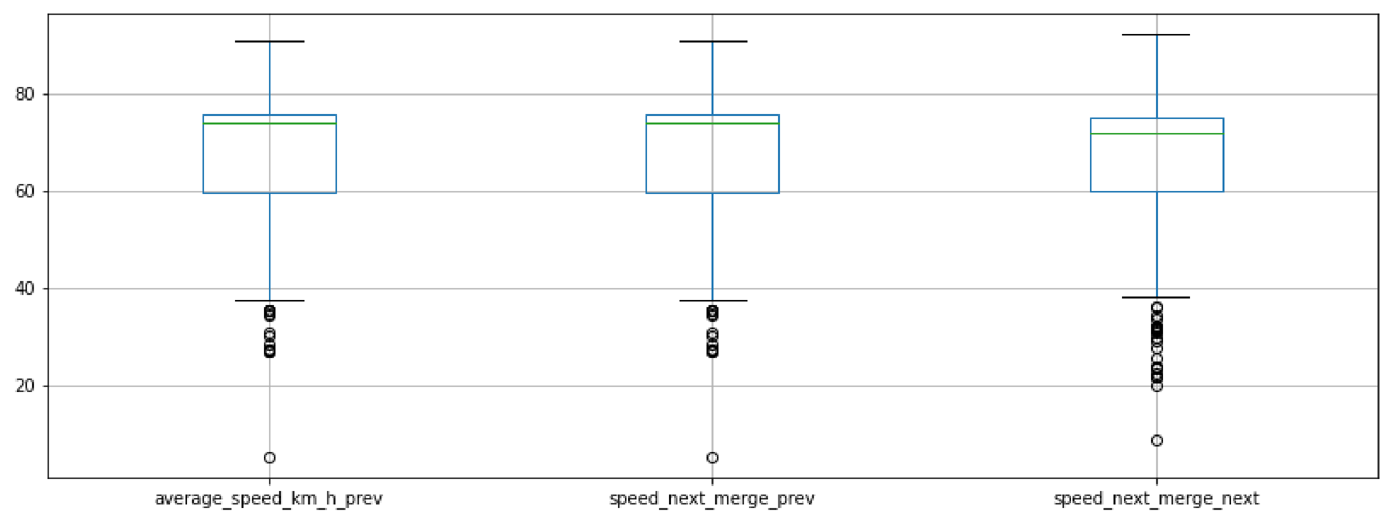
In [25]:
```
data_located.describe(include="float64").transpose()
```

Out[25]:

	count	mean	std	min	25%	50%	7
pk	3000.0	16884.945285	3639.180137	11603.559162	13641.147836	16611.700745	19163.282
pk_merge_prev	3000.0	16838.064461	3638.742279	11567.125941	13599.359300	16550.045215	19102.907
speed_prec_merge_prev	3000.0	66.970730	11.972519	6.903873	59.384810	69.698323	75.094
speed_next_merge_prev	3000.0	69.020333	9.863546	5.333708	59.674090	73.800451	75.584
pk_merge_next	3000.0	16933.924292	3639.763815	11637.962938	13704.456299	16648.267001	19206.442
speed_prec_merge_next	3000.0	69.020333	9.863546	5.333708	59.674090	73.800451	75.584
speed_next_merge_next	2732.0	68.790062	10.239396	8.726046	59.781855	71.906317	74.848
timeframe_gps_prev	3000.0	2.438827	1.420935	0.007000	1.205000	2.431000	3.648
timeframe_gps_next	3000.0	2.561081	1.420934	0.001000	1.351750	2.569500	3.796
average_speed_km_h_prev	3000.0	69.020333	9.863546	5.333708	59.674090	73.800451	75.584
average_speed_km_h_next	3000.0	69.020333	9.863546	5.333708	59.674090	73.800451	75.584

On remarque que les deux dernières variables 'average_speed_km_h_prev' et 'average_speed_km_h_next' ont les mêmes statistiques descriptives et donc probablement les mêmes observations dans la table.

In [26]:
```
figure = plt.figure(figsize = (14, 5))
boxplot = data_located.boxplot(column=['average_speed_km_h_prev','speed_next_merge_prev'
```



In [27]:
```
data_located.corr()
```

```
C:\Users\valentin\AppData\Local\Temp\ipykernel_15960\2006538106.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it w
ill default to False. Select only valid columns or specify the value of numeric_only to
silence this warning.
  data_located.corr()
```

Out[27]:

	timestamp	pk	pk_merge_prev	speed_prec_merge_prev	speed_next_merge_prev
timestamp	1.000000	-0.011724	-0.012288	0.133121	0.130460
pk	-0.011724	1.000000	0.999970	0.088361	0.074757
pk_merge_prev	-0.012288	0.999970	1.000000	0.086968	0.072695
speed_prec_merge_prev	0.133121	0.088361	0.086968	1.000000	0.880271
speed_next_merge_prev	0.130460	0.074757	0.072695	0.880271	1.000000

pk_merge_next	-0.011794	0.999970	0.999993	0.090257	0.076439
speed_prec_merge_next	0.130460	0.074757	0.072695	0.880271	1.000000
speed_next_merge_next	0.118923	0.044225	0.041892	0.639088	0.905708
timeframe_gps_prev	0.039873	0.008677	0.001164	-0.041683	0.031568
timeframe_gps_next	-0.039870	-0.008674	-0.001161	0.041687	-0.031566
average_speed_km_h_prev	0.130460	0.074757	0.072695	0.880271	1.000000
average_speed_km_h_next	0.130460	0.074757	0.072695	0.880271	1.000000

In [28]:

```
data_signal
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [28], in <cell line: 1>()
----> 1 data_signal

NameError: name 'data_signal' is not defined
```

In [ ]:

```
data_signal.info()
```

La table de données des signaux, de taille 3000 observations et 5 variables, ne contient pas de valeurs manquantes.
Elle est composées d'une variable de type 'int' et 4 variables de type 'object'. Ces cinq variables sont les mêmes que celles utilisées dans la table précédente.

Analyse des observations

In [29]:

```
data_gps.iloc[1:272,[2,5,6,7,8]]
```

Out[29]:

	pk	direction	nearest_pk	speed_prec	speed_next
1	10253.188120	PANLAP	10255.28	5.880716	6.445588
2	10262.140325	PANLAP	10265.28	6.445588	7.063472
3	10271.950702	PANLAP	10275.28	7.063472	5.232628
4	10279.218242	PANLAP	10275.28	5.232628	3.760407
5	10284.441030	PANLAP	10285.28	3.760407	8.906976
...	...	...	...	...	...
267	25264.810303	PANLAP	25268.75	0.489006	0.654619
268	25265.719496	PANLAP	25268.75	0.654619	2.181526
269	25260.047157	LAPPAN	25255.13	7.628656	6.291195
270	25251.309387	LAPPAN	25255.13	6.291195	5.483355
271	25243.693616	LAPPAN	25245.13	5.483355	8.982713

271 rows × 5 columns

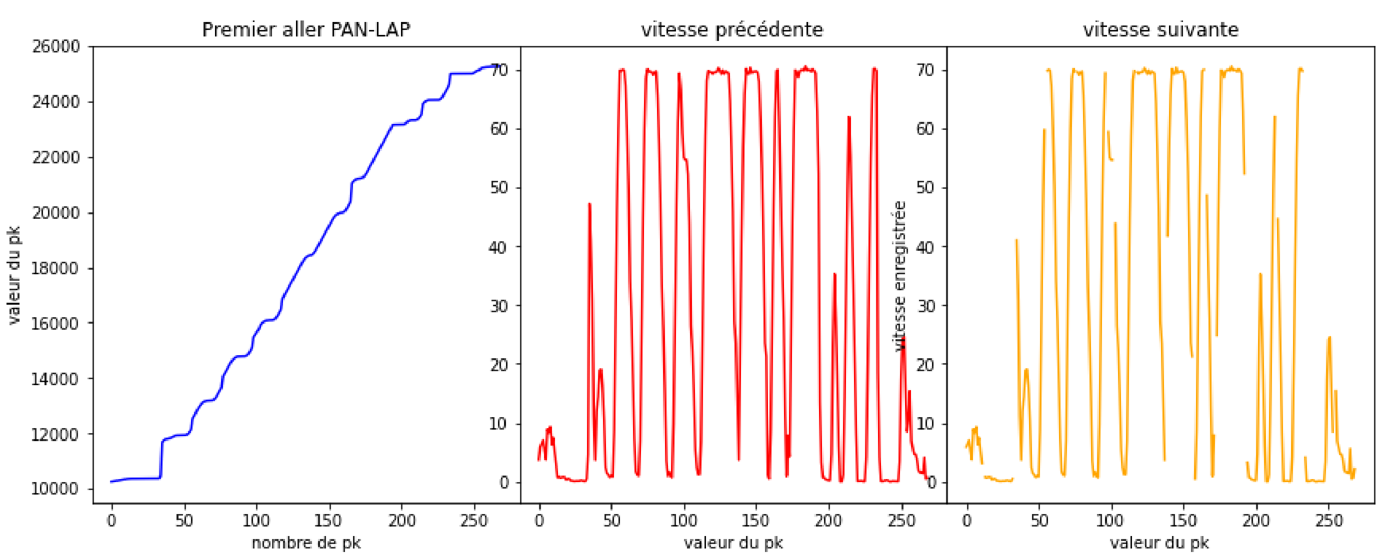
Le trajet est découpé en plusieurs portions. Tout d'abord la direction 'PAN-LAP' correspondant aux observations 1 à 268, puis la direction LAP-PAN correspondant aux observations 269 à 475. Puis de nouveau la direction 'PAN-LAP' à partir de l'observations 476, etc.

In [30]:

```
figure = plt.figure(figsize = (14, 5))
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,
                        right = 0.9, top = 0.9, wspace = 0, hspace = 0.1)
axes = figure.add_subplot(1, 3, 1)
axes.set_xlabel('nombre de pk')
axes.set_ylabel('valeur du pk')
axes.set_title("Premier aller PAN-LAP")
axes.plot(range(269),data_gps.iloc[0:269,2], color = 'blue')
axes = figure.add_subplot(1, 3, 2)
axes.set_xlabel('valeur du pk')
axes.set_title("vitesse précédente")
axes.plot(range(269), data_gps.iloc[0:269,7], color = 'red')
axes = figure.add_subplot(1, 3, 3)
axes.set_xlabel('valeur du pk')
axes.set_ylabel('vitesse enregistrée')
axes.set_title("vitesse suivante")
axes.plot(range(269), data_gps.iloc[0:269,8], color = 'orange')
#---------------------------------------------------------------------

fig, axs = plt.subplots(2, 4, figsize=(16, 8))

axs = axs.flatten()
for i in range(8):
    sns.kdeplot(df.iloc[:, i], ax=axs[i])
    axs[i].set_xlabel("")
    axs[i].set_ylabel("")
    axs[i].set_title(f"{df.columns[i]}, mean={df.iloc[:,i].mean():.2}, sd={df.iloc[:,i].

plt.show()
#----------------------------------------------------------------------
fig,axe=plt.Subplot(1,2,figsize=())
axe=axe.flatten()

for i in ra,ge():
    plo(,axe=axe)
    axe.se_titl(f"mean={.mean():.2},st={:.3}")


j=plt.figure(fi)
j.add_subplot(1,2,)
```

Out[30]:

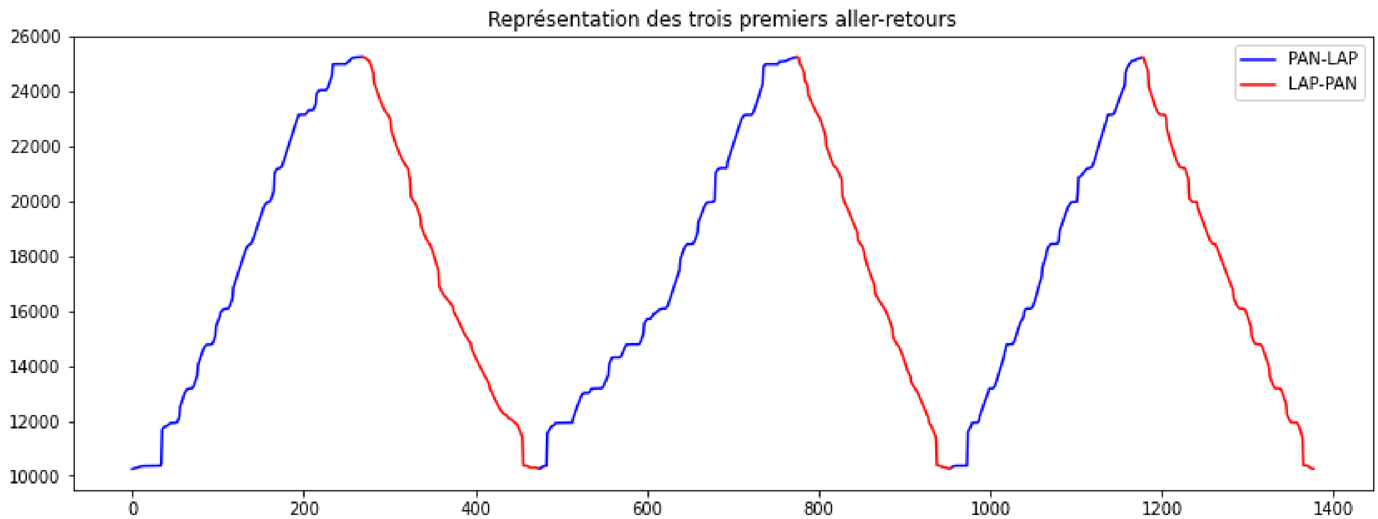
```
[<matplotlib.lines.Line2D at 0x2840bf2f520>]
```



Sur ces graphes nous pouvons observer tout d'abord l'évolution des points kilométrique pour le premier aller PAN-LAP, puis la vitesse enregistrée au point précédent et au point suivant. On distingue des coupures auu niveau de la vitesse sur le troisième graphe.

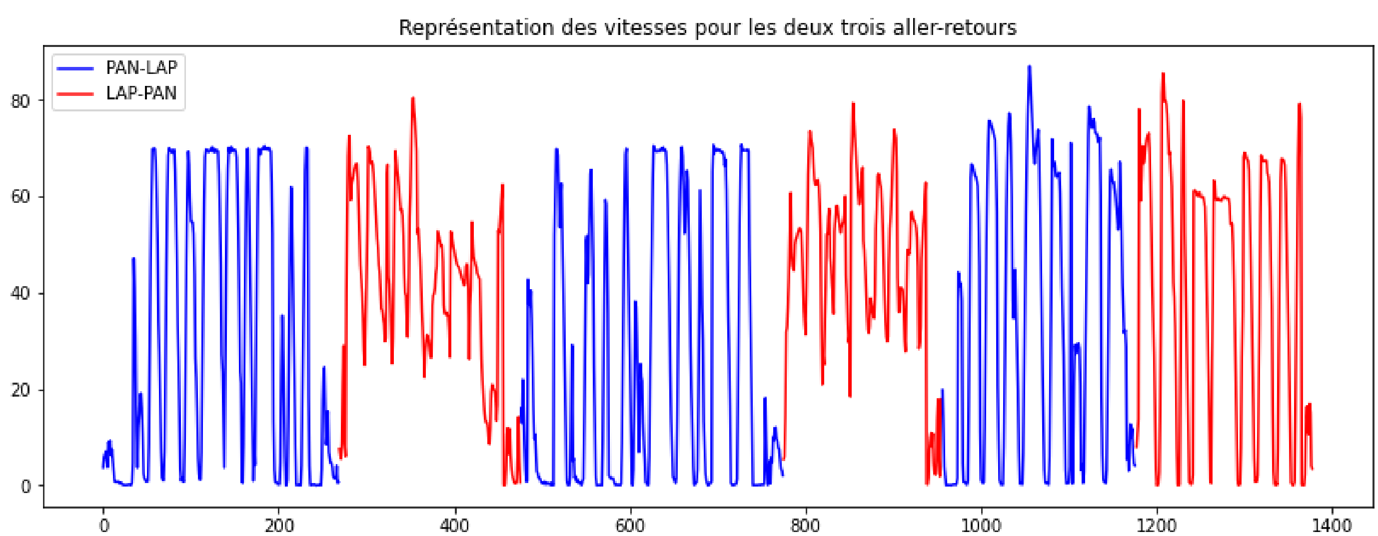
In [31]:
```
figure = plt.figure(figsize = (14, 5))
plt.plot(data_gps.iloc[0:269,2],color="b",label="PAN-LAP")
plt.plot(data_gps.iloc[269:476,2],color="r",label="LAP-PAN")
plt.plot(data_gps.iloc[476:775,2],color="b")
plt.plot(data_gps.iloc[775:955,2],color="r")
plt.plot(data_gps.iloc[956:1176,2],color="b")
plt.plot(data_gps.iloc[1177:1378,2],color="r")
plt.title("Représentation des trois premiers aller-retours")
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x28407a2f1f0>



In [32]:
```
figure = plt.figure(figsize = (14, 5))
plt.plot(data_gps.iloc[0:269,7],color="b",label="PAN-LAP")
plt.plot(data_gps.iloc[269:476,7],color="r",label="LAP-PAN")
plt.plot(data_gps.iloc[476:775,7],color="b")
plt.plot(data_gps.iloc[775:955,7],color="r")
plt.plot(data_gps.iloc[956:1176,7],color="b")
plt.plot(data_gps.iloc[1177:1378,7],color="r")
plt.title("Représentation des vitesses pour les deux trois aller-retours")
plt.legend()
```

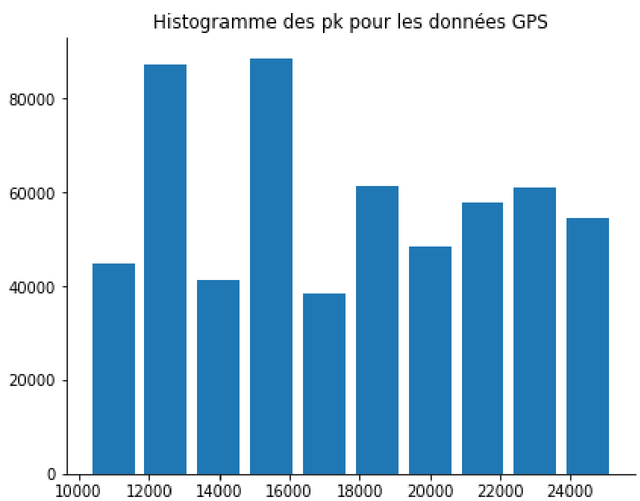
Out[32]: <matplotlib.legend.Legend at 0x28407801720>

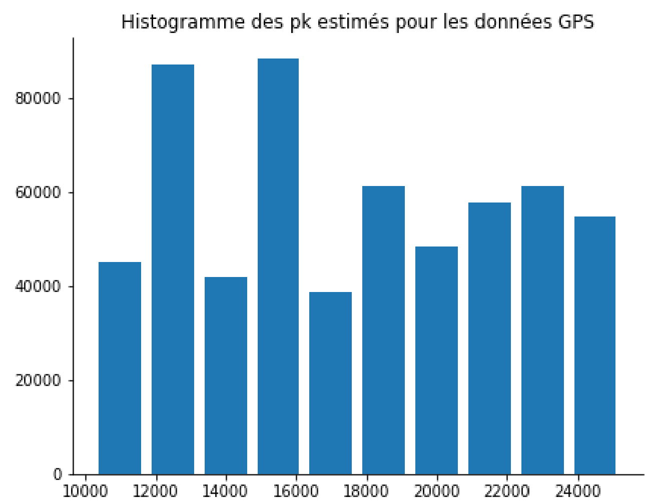


On remarque que les retours paraissent plus courts que les aller, et la vitesse des retours est en moyenne plus rapide.

In [33]:
```
figure = plt.figure(figsize = (15, 5))
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,
                       right = 0.9, top = 0.9, wspace = 0.3, hspace = 0.1)
axes = figure.add_subplot(1, 2, 1)
axes.spines['top'].set_visible(False)
axes.spines['right'].set_visible(False)
axes.hist(data_gps.pk, rwidth=0.8)
axes.set_title("Histogramme des pk pour les données GPS")
axes = figure.add_subplot(1,2, 2)
axes.spines['top'].set_visible(False)
axes.spines['right'].set_visible(False)
axes.hist(data_gps.nearest_pk, rwidth=0.8)
axes.set_title("Histogramme des pk estimés pour les données GPS")
```

Out[33]:
```
Text(0.5, 1.0, 'Histogramme des pk estimés pour les données GPS')
```





La distribution est relativement similaire.

Clustering

Dans cette partie nous cherchons à effectuer un clustering des points kilométrques à partir des données de chocs. L'objectif est double:

- Détecter les zones de défauts de voie potentiels
- Distinguer parmi les chocs lesquels sont du bruit et lesquels sont de rées défauts

Tout d'abord, nous récupérons les données de trajets pour lesquels la valeur de 'subcoverage' est supérieure à 0.8. Cette valeur caractérise le bon fonctionnement du GPS sur la séquence commençant à 'sub_subseq_start' et finissant à 'sub_subseq_end'.

In [34]:
```
data_choc = data_located.copy()

data_PANLAP = data_journeys_detail[data_journeys_detail.direction == 'PANLAP']
```

In [35]:
```
# Table des trajets dont la couverture est strictement inférieure à 80% pour la directio

uncovered_journeys = data_PANLAP.iloc[np.where(data_PANLAP['subcoverage']<0.8)]
uncovered_journeys
```

Out[35]:

	id_serie	direction	sub_subseq_start	sub_subseq_end	pk_min	pk_max	nb_points	area	su
0	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:25:00.022	2021-02-01 00:29:10.020	10360.805360	11699.673374	19	_002	

4	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:35:30.021	2021-02-01 00:38:10.017	16041.106610	18279.755237	29	_006	1
5	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:38:15.016	2021-02-01 00:40:15.016	18343.459553	19811.417355	21	_007	1
6	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:40:20.016	2021-02-01 00:42:00.018	19874.354061	21027.225663	12	_008	
8	2021-02-01_00:25:00-00:51:10	PANLAP	2021-02-01 00:44:30.018	2021-02-01 00:49:10.016	23026.672991	25090.908426	44	_010	1
...	...	...	...	...	...	...	...	...	
24447	2021-07-22_23:41:41-23:59:51	PANLAP	2021-07-22 23:41:41.016	2021-07-22 23:49:36.018	10244.847805	11745.941333	40	_002	
24448	2021-07-22_23:41:41-23:59:51	PANLAP	2021-07-22 23:49:41.018	2021-07-22 23:51:36.018	11811.548113	12991.147561	21	_003	
24449	2021-07-22_23:41:41-23:59:51	PANLAP	2021-07-22 23:51:41.018	2021-07-22 23:53:56.019	13055.884349	14598.991860	25	_004	1
24450	2021-07-22_23:41:41-23:59:51	PANLAP	2021-07-22 23:54:01.018	2021-07-22 23:56:06.019	14664.126368	15938.938194	23	_005	
24451	2021-07-22_23:41:41-23:59:51	PANLAP	2021-07-22 23:56:11.018	2021-07-22 23:59:16.018	15995.702581	18266.111654	32	_006	1

14857 rows × 11 columns

In [36]:

```
# Fonctions qui convertissent et sélectionne le moment de début et de fin de l'enregistr

def pdtotimestampdeb(i):
    string = uncovered_journeys.sub_subseq_start.values[i]
    element = datetime.datetime.strptime(string,"%Y-%m-%d %H:%M:%S.%f")
    timestamp = datetime.datetime.timestamp(element)
    return(timestamp)

def pdtotimestampfin(i):
    string = uncovered_journeys.sub_subseq_end.values[i]
    element = datetime.datetime.strptime(string,"%Y-%m-%d %H:%M:%S.%f")
    timestamp = datetime.datetime.timestamp(element)
    return(timestamp)

debut = []
for i in range(len(uncovered_journeys)):
    debut.append(pdtotimestampdeb(i))

fin = []
for i in range(len(uncovered_journeys)):
    fin.append(pdtotimestampfin(i))
```

In [37]:

```
# Pour chaque choc data_located, nous regardons si le 'timestamp' associé à chaque obser
# à un intervalle de temps où l'enregistrement à mal fonctionné

toexclude=np.zeros(len(data_located))
```

```python
for i in range(len(data_located)):
    ts = data_located['timestamp'][i]
    for j in range(len(debut)):
        if debut[j]<ts<fin[j]:
            print('exclude it')
            toexclude[i]=1
            break
```

In [38]:
```python
print(np.where(toexclude == 1))
```

```
(array([], dtype=int64),)
```

Nous remarquons que tous les chocs enregistrés proviennent d'enregistrements fiables.
Pour la suite, nous séparons les chocs enregistrés selon le trajet qui est caractérisé par la variable 'id_serie'.

In [39]:
```python
data_trajet = data_PANLAP[data_PANLAP.subcoverage >=0.8].copy()

data_trajet.sub_subseq_start = pd.to_datetime(data_trajet.sub_subseq_start)
data_trajet.sub_subseq_end = pd.to_datetime(data_trajet.sub_subseq_end)

data_choc.timestamp = pd.to_datetime(data_choc.timestamp, unit="ms")

id_serie_to_shocks = dict()

for id_serie, traj in data_trajet.groupby("id_serie"):
    shocks_per_id_serie = list()

    for idx, row in traj.iterrows():
        mask = data_choc.timestamp.between(row.sub_subseq_start, row.sub_subseq_end)
        if mask.any():
            shocks_per_id_serie.append(data_choc[mask])

    if len(shocks_per_id_serie) > 0:
        id_serie_to_shocks[id_serie] = pd.concat(shocks_per_id_serie, axis=0)
```

In [40]:
```python
# Visualisation du premier trajet
id_serie_list = list(id_serie_to_shocks.keys())
print("On a ainsi, après découpage, ",len(id_serie_to_shocks)," trajets.")
id_serie_to_shocks[id_serie_list[0]]
```

```
On a ainsi, après découpage,  70  trajets.
```

Out[40]:

	timestamp	pk	sensor	direction	x	y	
17	2021-04-01 00:13:54.178	16128.006392	XXX_2	PANLAP	[-0.39, -0.79, 0.51, 0.31, -0.29, 0.0099999999...	[0.246875, -0.653125, -2.353125, -1.353125, 0....	[-0.22625, -0 -0.12625, 0 -
18	2021-04-01 00:14:09.161	16356.841050	XXX_2	PANLAP	[0.155, 1.155, 0.455, -0.34500000000000003, -0...	[0.325, -0.17500000000000002, -0.275, -0.07500...	[1.424375, 0.7 0.22437500000
19	2021-04-01 00:14:09.588	16364.923015	XXX_1	PANLAP	[0.511875, 0.111875, 0.011875000000000004, -0....	[-0.881875, 0.018125000000000006, 0.318125, 1....	[0.232499999999 -0.8675, -0
20	2021-04-01 00:14:09.971	16372.172177	XXX_2	PANLAP	[0.47125, 0.17125, -0.32875, 0.27125, -0.02875...	[-0.851875, 0.948125, 1.248125, 2.548125, 0.74...	[-0.68124999999 -1.08125, 0
21	2021-04-01 00:15:34.629	18008.476951	XXX_1	PANLAP	[-0.579375, 0.520625, 0.420625, 0.120625000000...	[-0.185, 0.11500000000000002, -0.7849999999999...	[-0.843125, 0.3 1.256875, 1.4
22	2021-04-01 00:15:34.746	18010.736114	XXX_2	PANLAP	[-0.8456250000000001, 0.254375, 0.854375, 0.05...	[1.3099999999999998, 1.6099999999999999, -0.99...	[-0.61125, -1 1.78875, 0 0.8

23	2021-04-01 00:15:36.771	18049.760665	XXX_2	PANLAP	[-0.052500000000000005, 1.3475000000000001, -0...	[1.26625, 0.46625, 0.86625, -1.23375, -1.43374...	[-0.80312500000 -0.203125, 0.2
24	2021-04-01 00:15:57.521	18362.382579	XXX_1	PANLAP	[0.526875, -0.37312500000000004, -0.0731250000...	[0.66625, 2.06625, -0.23374999999999999, 1.066...	[-0.6 0.09062500000 -1.0093
25	2021-04-01 00:16:55.979	18637.957134	XXX_1	PANLAP	[-0.7, 0.7, 0.4, -0.8, 0.8, 0.9, 0.3, 0.2, -0....	[-2.014375, -1.414375, -0.31437499999999996, 2...	[-1.22937499999 1.170625, -0.6
26	2021-04-01 00:16:56.550	18648.402642	XXX_2	PANLAP	[0.48875, 0.08875, -0.011250000000000003, -0.3...	[-0.771875, -0.37187499999999996, 1.828125, 0....	[-0.07750000000 -0.9775, 0.9225, 0
27	2021-04-01 00:17:00.600	18723.182167	XXX_2	PANLAP	[0.218125, 0.518125, 1.4181249999999999, 0.218...	[1.6468749999999999, 1.946875, 1.046875, 0.246...	[-0.795, 0.405, 0.505, -0.295, 0
28	2021-04-01 00:17:11.133	18926.872719	XXX_2	PANLAP	[-1.94, 0.45999999999999996, 0.86, -0.74, 0.86...	[-1.26875, -1.56875, 0.031250000000000014, -0....	[-0.40375, 0 -1.50375, 0 1.3
29	2021-04-01 00:17:24.904	19193.449975	XXX_2	PANLAP	[1.12, -0.48, -0.78, 0.72, -1.28, 0.52, 1.02, ...	[-3.7137499999999997, -1.71375, -2.31374999999...	[-0.67562499999 -0.175625, -0.9
30	2021-04-01 00:17:43.943	19561.646724	XXX_2	PANLAP	[-1.840625, 0.059375000000000004, 1.159375, -0...	[0.121875, -4.478125, -1.778125, 0.621875, 4.5...	[-0.23062499999 1.069375, -2.0306
31	2021-04-01 00:17:47.899	19636.805652	XXX_1	PANLAP	[0.5525, 0.052500000000000005, 0.4525, -0.5475...	[0.6968749999999999, 0.496875, -0.103125, 1.09...	[0.05687500000 -0.9 0.156875
32	2021-04-01 00:20:49.011	21393.627289	XXX_1	PANLAP	[-0.95625, -0.25625, 0.54375, -1.25625, 0.0437...	[-1.06875, -0.46875, -1.8687500000000001, -0.7...	[0.68062499999 -2.0 -0.019374
33	2021-04-01 00:20:49.057	21394.387560	XXX_2	PANLAP	[0.19312500000000002, -0.6068749999999999, -0....	[1.7975, 0.1975, -0.6024999999999999, -0.70249...	[-0.405, 0.395 -0.00500000000
34	2021-04-01 00:20:50.646	21422.460800	XXX_1	PANLAP	[0.5225, -0.4775, 0.5225, 0.7224999999999999, ...	[0.224375, -2.7756250000000002, -1.775625, -1....	[0.576875, -1.5 2.47687499999
35	2021-04-01 00:21:20.641	22002.538990	XXX_2	PANLAP	[-0.22187500000000002, 0.178125, -0.0218750000...	[0.19937500000000002, 1.599375, -1.10062500000...	[-1.793125, -1.1 1.3 -0.19312
36	2021-04-01 00:21:45.340	22482.762700	XXX_2	PANLAP	[-0.14, -0.24, -0.44, 1.16, -0.039999999999999...	[-1.4024999999999999, 0.5975, -0.2025, 0.3975,...	[1.445, -0.055, -0.155, 1.145, -2

Cette table nous montre les informations pour le premier trajet. Comme nous avons 70 trajets, nous avons donc 70 clusterings à effectuer pour analyser le type des chocs.

In [41]:
```
data_choc_cols = pd.Series(data_choc.columns)
```

Nature des chocs

Le clustering que nous cherchons à faire est basé sur la position des chocs enregistrés par les accéléromètres. Pour cela, nous sélectionnons la variable des points kilométriques et réalisons un clustering DBSCAN pour chaque trajet isolé.

In [2]:
```
# plot dbscan 70 trajets
from sklearn.cluster import DBSCAN
```

In [221…
```
liste_trajet = list(id_serie_to_shocks.keys())
print("Nombre de trajets : ", len(liste_trajet))
print('Le premier trajet : ',liste_trajet[0],' et le dernier trajet :',liste_trajet[-1])
```

```
Nombre de trajets :  70
Le premier trajet :  2021-04-01_00:01:15-00:29:55  et le dernier trajet : 2021-04-05_23:
25:58-23:49:08
```

- On remarque que les trajets se situent sur 5 jours différents, du 1er Avril 2021 au 5 Avril 2021, soit 5 jours.
  $\Rightarrow$ On entreprend donc de séparer nos trajets par jours de voyage puisque des travaux ont pu être effectués la nuit pour réparer des défauts de voies qui auraient été détectés (par le conducteur par exemple).

In [222…
```
day1,day2,day3,day4,day5 = [], [], [], [], []
day1=[trajet for trajet in liste_trajet if trajet[8:10] == '01']
day2=[trajet for trajet in liste_trajet if trajet[8:10] == '02']
day3=[trajet for trajet in liste_trajet if trajet[8:10] == '03']
day4=[trajet for trajet in liste_trajet if trajet[8:10] == '04']
day5=[trajet for trajet in liste_trajet if trajet[8:10] == '05']
days = [day1,day2,day3,day4,day5]
```

In [223…
```
for i in range(5):
    print(f'Il y a eu {len(days[i])} trajets {i+1} Avril 2021.')
```

```
Il y a eu 15 trajets 1 Avril 2021.
Il y a eu 9 trajets 2 Avril 2021.
Il y a eu 11 trajets 3 Avril 2021.
Il y a eu 17 trajets 4 Avril 2021.
Il y a eu 18 trajets 5 Avril 2021.
```

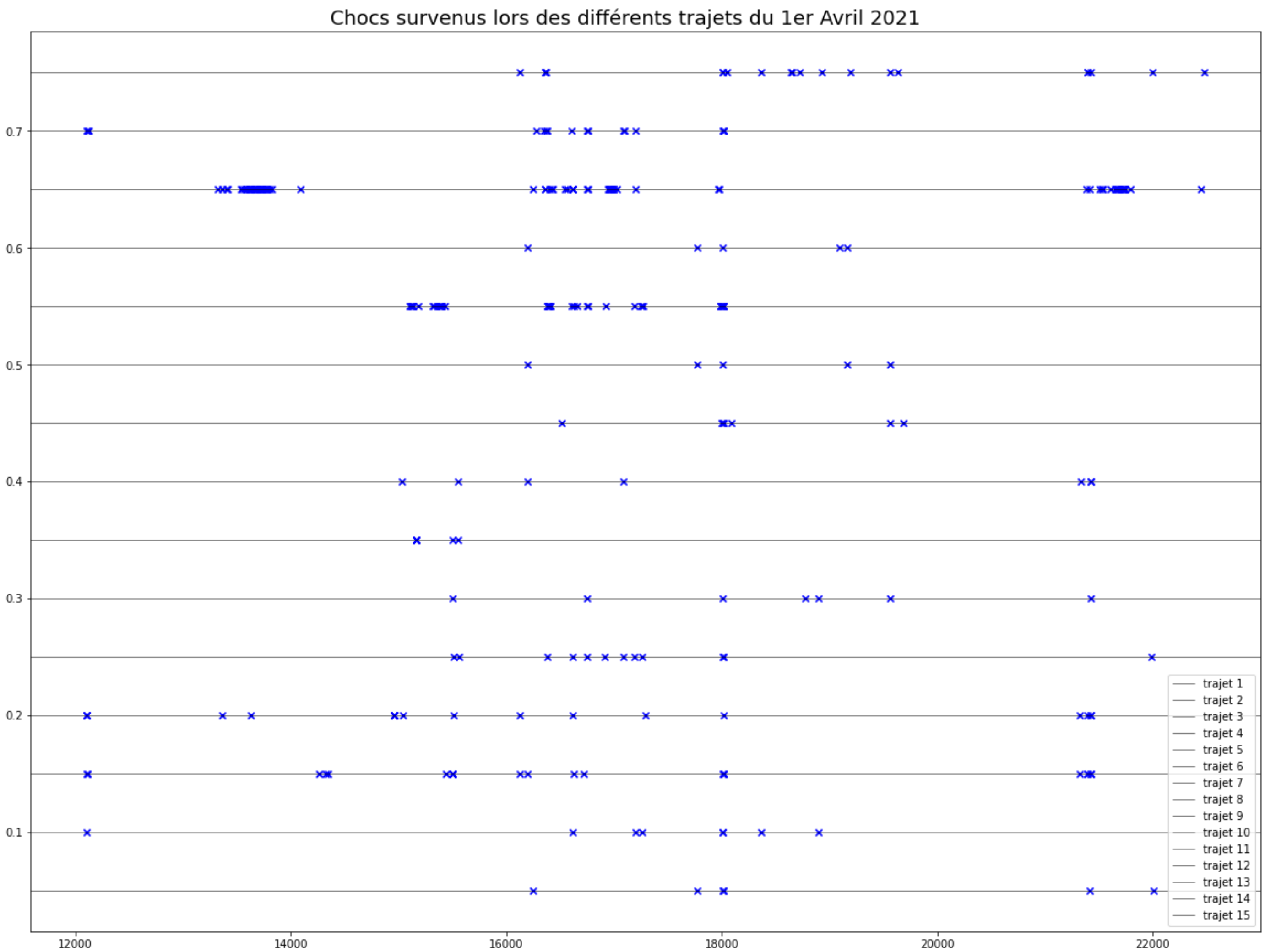
In [224…
```
from palettable.colorbrewer.qualitative import Paired_12
from palettable.tableau import Tableau_20
```

In [426…
```
#Shocks located dataframe pour le premier trajet du jour 1
#id_serie_to_shocks[days[0][0]].reset_index(drop=True)
```

Chocs survenus 1er Avril 2021 :

In [428…
```
plt.figure(figsize=(20,len(days[0])))
plt.subplots_adjust(hspace=1.5)
plt.suptitle("Chocs survenus lors des différents trajets du 1er Avril 2021", fontsize=18

trajets_jour1 = []
ys = np.arange(len(days[0])*0.05,0,-0.05)

for i in range(len(days[0])):
    trajet_i = id_serie_to_shocks[days[0][i]].reset_index(drop=True) #on récupère le tra
    clustering = DBSCAN(eps=10, min_samples=1,metric='euclidean',algorithm = 'auto') #db
    clustering.fit(trajet_i[['pk']].to_numpy().reshape(-1,1))
    trajet_i['clusterlabs']=clustering.labels_ #On ajoute les labels des chocs du trajet

    #on récupère les dfs de chaque trajet, au cas où l'on voudrait regarder les labels p
    trajets_jour1.append(trajet_i[['pk','clusterlabs']])
```

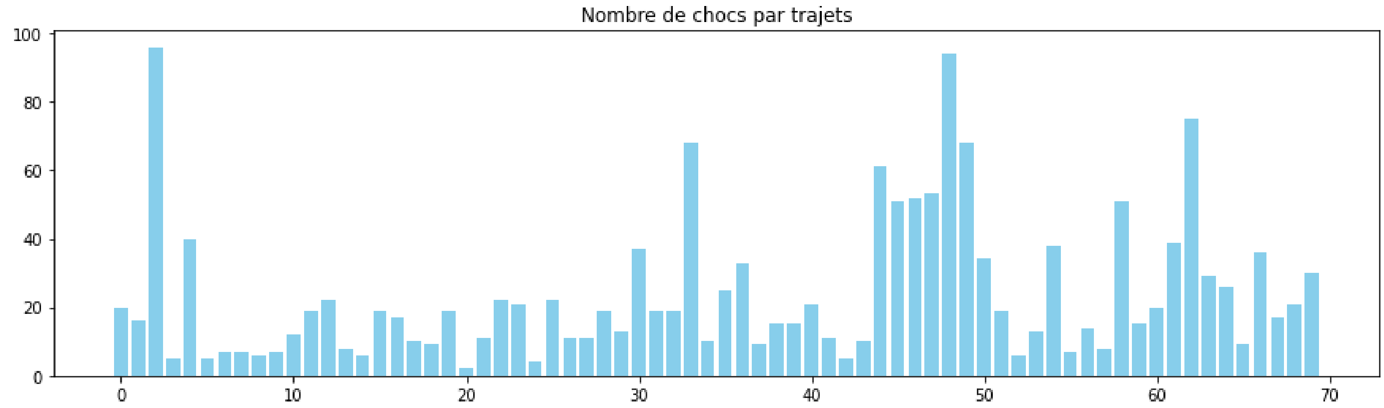
```
    plt.scatter(trajet_i[['pk']].to_numpy(), y=np.repeat(ys[i],len(trajet_i)),
            color="blue",
            #c = trajet_i['clusterlabs'].astype(float),
            #cmap = Paired_12.mpl_colormap,
            marker = 'x')
    plt.axhline(y=ys[i], linewidth = 0.5, color='black', label = f'trajet {i+1}')
    #for j in range(len(trajet_i)):
     #   plt.axvline(trajet_i['pk'][j],color='skyblue',alpha=0.5)
    plt.legend()
plt.savefig('chocs_jour_1.png')
```



In [431…
```
# On peut accèder aux trajets et au groupe formés comme suit
# trajets_jour1[0] #pour le trajet n°1
```

- On a pour intuition que si, au fur et à mesure des trajets, des chocs sont enregistrés à un même endroit ou à des endroits très proches, alors ils sont dûs à un probable défaut de voie.

In [287…
```
plt.figure(figsize = (15,4))
plt.title("Nombre de chocs par trajets")
plt.bar(np.arange(0,70,1),[len(id_serie_to_shocks[id_serie_list[i]]) for i in range(70)]
```



- On remarque également qu'un nombre très élevé de chocs a été enregistré lors de certains trajets, pourtant tous de direction PANLAP et sur la même voie. Il est légitime de se demander si ces enregistrements n'aurait pas été compromis par un défaut de l'appareil, un freinage brusque etc...

⇒ On souhaite donc effectuer un premier tri des chocs, en retirant ceux qui ne se répètent pas au fur et à mesure des trajets, et ceux dont les trajets ont enregistrés un nombre anormal de chocs (ce qui pourrait venir biaiser notre étude).

Clustering de tous les chocs enregistrés sur une journée :

In [414…

```
# Création d'une liste de 5 dataframe.
# Chaque dataframe contient les chocs enregistrés pour les journées du 1 au 5 Avril 2021
shocks_by_day = []
for i in range(5):
    dayshocks = []
    for j in range(len(days[i])):
        dayshocks.append(id_serie_to_shocks[days[i][j]].reset_index(drop=True))
    shocks_by_day.append(pd.concat(dayshocks).reset_index(drop=True))
```

In [415…

```
#Chocs du 1er Avril 2021 :
choc1 = shocks_by_day[0]
len(choc1)
#clusteringday1 = DBSCAN(eps=10, min_samples=3,metric='euclidean',algorithm = 'auto').fi
```

Out[415]:

```
276
```

In [416…

```
idx = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[days[0][2]]['pk'].tolist
idx2 = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[days[0][4]]['pk'].tolis
```

In [417…

```
indx = list(range(idx[0][0][0],idx[len(idx)-1][0][0]+1))
indx2 = list(range(idx2[0][0][0],idx2[len(idx2)-1][0][0]+1))

choc1.drop(indx+indx2, axis=0, inplace=True)
```

In [418…

```
print(len(choc1))
choc1
```

```
140
```

Out[418]:

	timestamp	pk	sensor	direction	x	y	
0	2021-04-01 00:13:54.178	16128.006392	XXX_2	PANLAP	[-0.39, -0.79, 0.51, 0.31, -0.29, 0.0099999999...	[0.246875, -0.653125, -2.353125, -1.353125, 0....	[-0.22625, -0.(-0.12625, 0.: -(
1	2021-04-01 00:14:09.161	16356.841050	XXX_2	PANLAP	[0.155, 1.155, 0.455, -0.34500000000000003,	[0.325, -0.17500000000000002,	[1.424375, 0.7: 0.224375000000(

					-0...	-0.275, -0.07500...	
2	2021-04-01 00:14:09.588	16364.923015	XXX_1	PANLAP	[0.511875, 0.111875, 0.011875000000000004, -0....	[-0.881875, 0.018125000000000006, 0.318125, 1....	[0.2324999999999 -0.8675, 0 -0.
3	2021-04-01 00:14:09.971	16372.172177	XXX_2	PANLAP	[0.47125, 0.17125, -0.32875, 0.27125, -0.02875...	[-0.851875, 0.948125, 1.248125, 2.548125, 0.74...	[-0.6812499999999 -1.08125, 0.
4	2021-04-01 00:15:34.629	18008.476951	XXX_1	PANLAP	[-0.579375, 0.520625, 0.420625, 0.120625000000...	[-0.185, 0.11500000000000002, -0.7849999999999...	[-0.843125, 0.3 1.256875, 1.4
...	...	...	...	...	...	...	
271	2021-04-01 23:13:21.655	17775.240368	XXX_2	PANLAP	[0.495, 0.995, 1.0950000000000002, -0.705, 0.2...	[-1.01125, 0.7887500000000001, 1.58875, 0.1887...	[0.773749999999 -2.72625, -3.
272	2021-04-01 23:13:35.822	18008.675968	XXX_2	PANLAP	[-0.36250000000000004, -0.1625, 0.6375, -0.662...	[-1.07625, -1.2762499999999999, -0.67625, -0.5...	[-0.131875, 0.4 -0.1 -1.331875
273	2021-04-01 23:13:36.227	18015.369067	XXX_2	PANLAP	[0.23125, -4.06875, -1.06875, 0.13125, -1.6687...	[-2.029375, 13.070625, 4.070625, 4.470625, -0....	[-1.62437500000 1.675625, 2.8
274	2021-04-01 23:19:13.909	21419.964504	XXX_2	PANLAP	[0.46375, -0.33625, 0.06375, -0.13625, 0.16375...	[-1.3587500000000001, -1.9587500000000002, -1....	[0.135, 0.835, -1.464999999
275	2021-04-01 23:19:49.153	22004.547631	XXX_2	PANLAP	[-0.244375, 0.955625, -0.444375, -1.1443750000...	[-1.2037499999999999, -0.90375, 0.59625, 2.196...	[1.10312500000 -0.9 -0.5968749

140 rows × 18 columns

In [419…

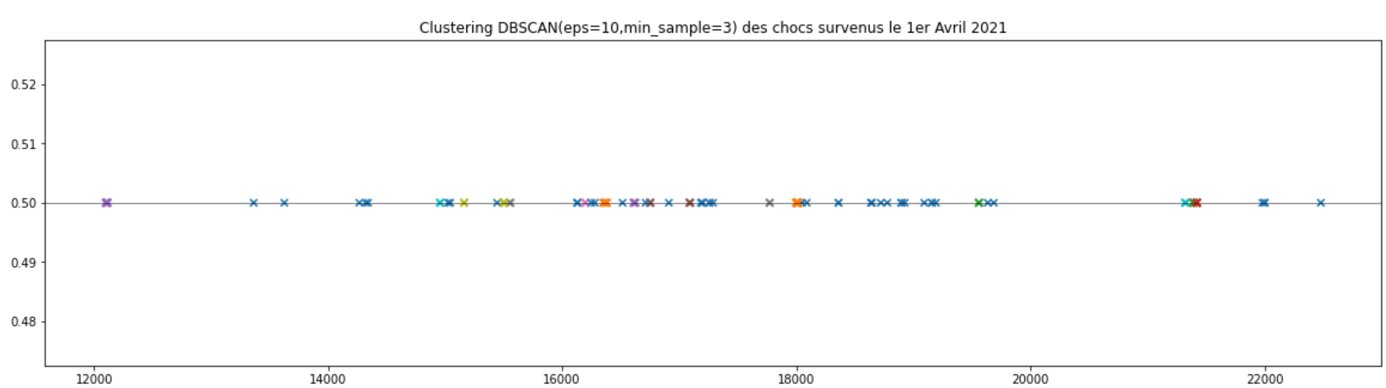
```
clusteringday1 = DBSCAN(eps=10, min_samples=3,metric='euclidean',algorithm = 'auto').fit
choc1['clusterlabs']= clusteringday1.labels_
```

In [433…

```
plt.figure(figsize=(20,5))

plt.title("Clustering DBSCAN(eps=10,min_sample=3) des chocs survenus le 1er Avril 2021")

plt.scatter(choc1[['pk']].to_numpy(), y=np.repeat(0.5,len(choc1)),
                c = choc1['clusterlabs'].astype(float),
            cmap = 'tab10',
            marker = 'x')
plt.axhline(y=0.5, linewidth = 0.5,color='black', label = 'track1')
```

Out[433]:

```
<matplotlib.lines.Line2D at 0x2841b418a00>
```



In [432…

```
print(set(choc1['clusterlabs'])) #17 groupes plus du bruit (groupe -1).
choc1[['pk','clusterlabs']]
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, -1}
```

Out[432]:

	pk	clusterlabs
0	16128.006392	0
1	16356.841050	1
2	16364.923015	1
3	16372.172177	1
4	18008.476951	2
...	...	...
271	17775.240368	11
272	18008.675968	2
273	18015.369067	2
274	21419.964504	5
275	22004.547631	-1

140 rows × 2 columns

Retirons les pks associés au groupe -1 et voyons si les chocs qui ne se répètent pas au fur et à mesure des trajets ont bien été enlevés

In [437…

```
gp_moins1 = choc1[['pk','clusterlabs']].loc[choc1['clusterlabs']== -1]
choc1.drop(list(gp_moins1.index), axis=0, inplace=True)
```

```
40
[6, 7, 8, 9, 10, 11, 12, 14, 18, 19, 23, 32, 135, 136, 180, 182, 186, 188, 189, 205, 20
6, 214, 216, 217, 220, 224, 225, 230, 234, 242, 243, 244, 245, 253, 264, 265, 268, 269,
270, 275]
```

On réeffectue le clustering :

In [440…

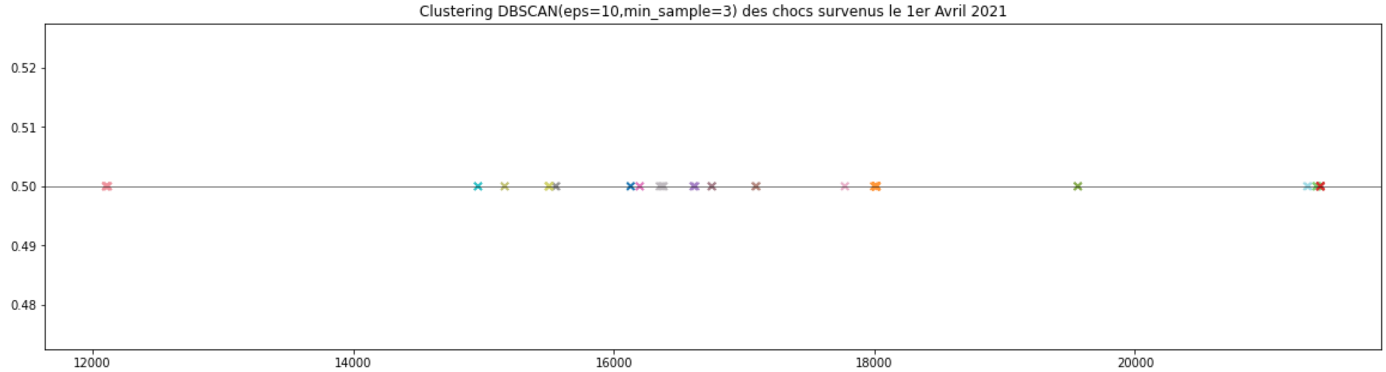
```
clusteringday1 = DBSCAN(eps=10, min_samples=3,metric='euclidean',algorithm = 'auto').fit
choc1['clusterlabs']= clusteringday1.labels_
```

In [546…

```
plt.figure(figsize=(20,5))

plt.title("Clustering DBSCAN(eps=10,min_sample=3) des chocs survenus le 1er Avril 2021")

plt.scatter(choc1[['pk']].to_numpy(), y=np.repeat(0.5,len(choc1)),
                c = choc1['clusterlabs'].astype(float),
            cmap = Tableau_20.mpl_colormap,
            marker = 'x')
plt.axhline(y=0.5, linewidth = 0.5,color='black', label = 'track1')
```

Out[546]:

```
<matplotlib.lines.Line2D at 0x28427fa5d20>
```



In [442…

```
print(set(choc1['clusterlabs']))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

On voit bien qu'il n'y a plus de bruit. Essayons maintenant de voir ce que ça donne si l'on sépare les chocs du jour 1 par trajets

On re-sépare les chocs du jours 1 grâce aux 2 cellules ci-dessous.

In [484…

```
tr1idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[0]]['p
tr2idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[1]]['p
tr4idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[3]]['p
tr6idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[5]]['p
tr7idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[6]]['p
tr8idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[7]]['p
tr9idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[8]]['p
tr10idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[9]]['
tr11idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[10]][
tr12idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[11]][
tr13idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[12]][
tr14idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[13]][
tr15idx_ = [np.where(choc1['pk'] == pk) for pk in id_serie_to_shocks[id_serie_list[14]][
```

In [496…

```
tr1idx,tr2idx,tr4idx,tr6idx,tr7idx,tr8idx,tr9idx, = [],[],[],[],[],[],[],
tr10idx,tr11idx,tr12idx,tr13idx,tr14idx,tr15idx = [],[],[],[],[],[]


for i in range(len(tr1idx_)):
    if tr1idx_[i][0].shape[0] != 0:
        tr1idx.append(tr1idx_[i][0][0])

for i in range(len(tr2idx_)):
    if tr2idx_[i][0].shape[0] != 0:
        tr2idx.append(tr2idx_[i][0][0])

for i in range(len(tr4idx_)):
    if tr4idx_[i][0].shape[0] != 0:
        tr4idx.append(tr4idx_[i][0][0])

for i in range(len(tr6idx_)):
    if tr6idx_[i][0].shape[0] != 0:
        tr6idx.append(tr6idx_[i][0][0])

for i in range(len(tr7idx_)):
    if tr7idx_[i][0].shape[0] != 0:
        tr7idx.append(tr7idx_[i][0][0])

for i in range(len(tr8idx_)):
    if tr8idx_[i][0].shape[0] != 0:
        tr8idx.append(tr8idx_[i][0][0])
```

```
for i in range(len(tr9idx_)):
    if tr9idx_[i][0].shape[0] != 0:
        tr9idx.append(tr9idx_[i][0][0])

for i in range(len(tr10idx_)):
    if tr10idx_[i][0].shape[0] != 0:
        tr10idx.append(tr10idx_[i][0][0])

for i in range(len(tr11idx_)):
    if tr11idx_[i][0].shape[0] != 0:
        tr11idx.append(tr11idx_[i][0][0])

for i in range(len(tr12idx_)):
    if tr12idx_[i][0].shape[0] != 0:
        tr12idx.append(tr12idx_[i][0][0])

for i in range(len(tr13idx_)):
    if tr13idx_[i][0].shape[0] != 0:
        tr13idx.append(tr13idx_[i][0][0])

for i in range(len(tr14idx_)):
    if tr14idx_[i][0].shape[0] != 0:
        tr14idx.append(tr14idx_[i][0][0])

for i in range(len(tr15idx_)):
    if tr15idx_[i][0].shape[0] != 0:
        tr15idx.append(tr15idx_[i][0][0])
```

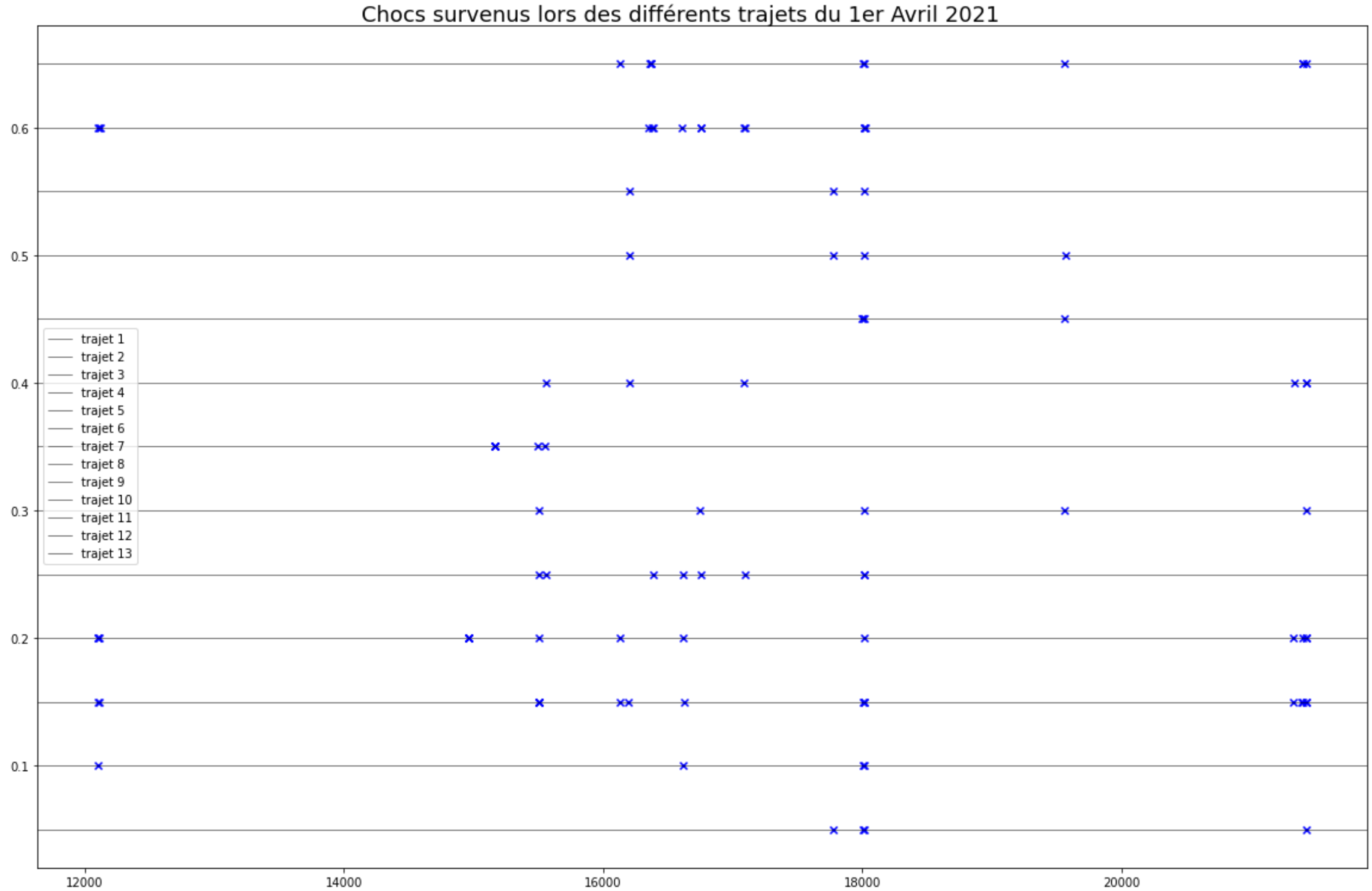
Les chocs du jours 1 par trajets sont stocké dans la liste de dataframe suivante :

In [511…

```
choc1_tr = [choc1.iloc[tr1idx],choc1.iloc[tr2idx],choc1.iloc[tr4idx],choc1.iloc[tr6idx],
            choc1.iloc[tr8idx],choc1.iloc[tr9idx],choc1.iloc[tr10idx],choc1.iloc[tr11idx
            choc1.iloc[tr13idx],choc1.iloc[tr14idx],choc1.iloc[tr15idx]]
```

In [529…

```
plt.figure(figsize=(20,len(choc1_tr)))
plt.subplots_adjust(hspace=1.5)
plt.suptitle("Chocs survenus lors des différents trajets du 1er Avril 2021", fontsize=18

trajets_jour1_clean = []
ys = np.arange(len(choc1_tr)*0.05,0,-0.05)

for i in range(len(choc1_tr)):
    trajet_i = choc1_tr[i].reset_index(drop=True) #on récupère le trajet
    clustering = DBSCAN(eps=10, min_samples=1,metric='euclidean',algorithm = 'auto') #db
    clustering.fit(trajet_i[['pk']].to_numpy().reshape(-1,1))
    trajet_i['clusterlabs']=clustering.labels_ #On ajoute les labels des chocs du trajet

    #on récupère les dfs de chaque trajet, au cas où l'on voudrait regarder les labels p
    trajets_jour1_clean.append(trajet_i[['pk','clusterlabs']])

    plt.scatter(trajet_i[['pk']].to_numpy(), y=np.repeat(ys[i],len(trajet_i)),
                color="blue",
                #c = trajet_i['clusterlabs'].astype(float),
                #cmap = Paired_12.mpl_colormap,
                marker = 'x')
    plt.axhline(y=ys[i], linewidth = 0.5, color='black', label = f'trajet {i+1}')
    #for j in range(len(trajet_i)):
     #   plt.axvline(trajet_i['pk'][j],color='skyblue',alpha=0.5)
    plt.legend()
plt.savefig('chocs_jour_1_clean.png')
```



In [536…

```
centroids = choc1.groupby('clusterlabs')['pk'].mean()
mins = choc1.groupby('clusterlabs')['pk'].min()
maxs = choc1.groupby('clusterlabs')['pk'].max()
```

In [543…

```
DBclusters = pd.concat([mins,centroids,maxs],axis=1)
DBclusters.columns = ['pk_min','centroid','pk_max']
DBclusters
```

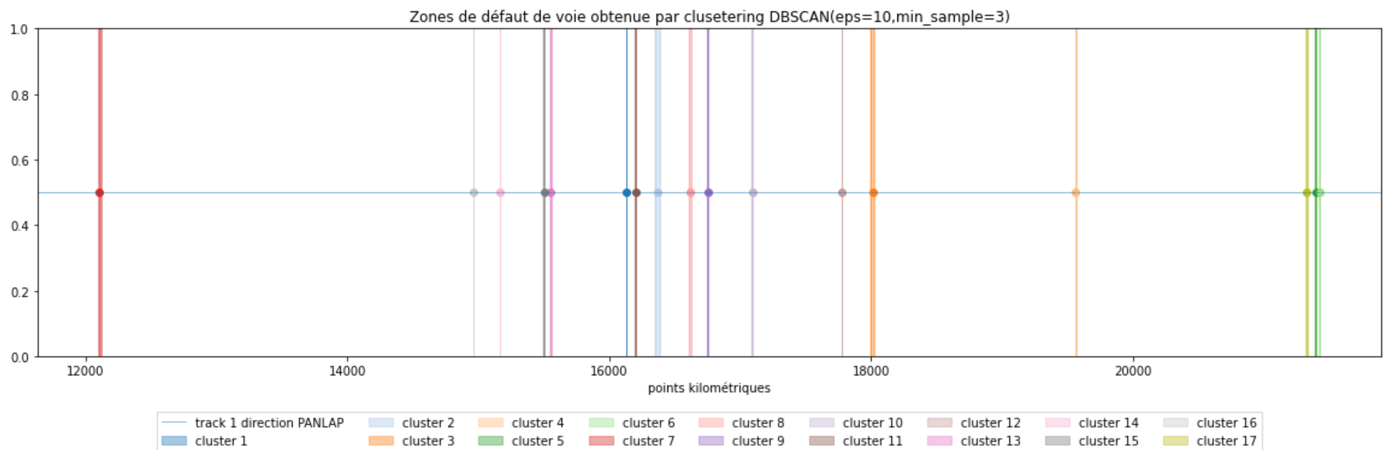
Out[543]:

	pk_min	centroid	pk_max
clusterlabs			
0	16128.006392	16129.841857	16130.792796
1	16350.861639	16369.538718	16383.733825
2	17995.461683	18011.234280	18019.820138
3	19558.655338	19561.264553	19563.114609
4	21389.663828	21394.578248	21398.703600
5	21419.964504	21423.038097	21426.652514
6	12102.823539	12109.207930	12120.374033
7	16610.742632	16618.118040	16628.074697
8	16749.900751	16752.459386	16756.717694
9	17087.701659	17091.190357	17096.783250
10	16197.853654	16199.184899	16199.948788
11	17771.301484	17773.852007	17775.240368
12	15549.283121	15555.461621	15559.363958
13	15163.358015	15163.358015	15163.358015

14	15497.979314	15504.315632	15507.769225
15	14958.326083	14958.326083	14958.326083
16	21319.375410	21325.422849	21331.740889

In [571…

```
plt.figure(figsize=(20,5))
plt.title('Zones de défaut de voie obtenue par clusetering DBSCAN(eps=10,min_sample=3)')
plt.ylim([0,1])
plt.xlabel('points kilométriques')

plt.axhline(y=0.5,linewidth = 0.5, label = 'track 1 direction PANLAP')
for i in range(len(DBclusters)):
    plt.scatter(x = DBclusters.centroid[i],y=0.5,color = Tableau_20.mpl_colors[i])
    deb = DBclusters.pk_min[i]
    fin = DBclusters.pk_max[i]
    x = np.arange(deb, fin, 0.01)
    plt.axvline(x =deb,linewidth = 0.5,color=Tableau_20.mpl_colors[i])
    plt.axvline(x =fin,linewidth = 0.5,color=Tableau_20.mpl_colors[i])
    plt.fill_between(x, -1, 2,color=Tableau_20.mpl_colors[i],alpha=0.4,label=f'cluster {
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15),fancybox=False, shadow=False,
#plt.savefig('clustering_chocs_jour_1.png')
```



Analyse du signal des chocs

Après avoir identifié les zones de probables défauts de voie, on se propose de compléter notre étude en analysant les signaux renvoyés par les chocs de nos différents clusters.
Voici comment nous allons orienter notre étude :

1. On se propose tout d'abord de calculer différentes métriques pour plusieurs signaux de chocs, et de regarder s'il y a une cohérence entre les métriques des chocs appartenant à un même cluster et une différence entre les chocs appartenant à deux clusters différents.
2. On se proposera ensuite, pour les chocs d'un seul cluster dans un premier temps, d'observer l'évolution des déplacement relatifs sur les 3 axes au fil des jours (jour 1, jour 2 etc...) afin de pouvoir caractériser la sévérité du choc

1. Comparaison des chocs

On décide de tenter de caractériser nos chocs par leurs pic d'accélération, la RMS de l'accélération, et le déplacement relatif de chaque axe (accélération intégrée 2 fois).

In [694…

```
#voici les dataframes du jour 1 déjà regroupés par cluster
chocs1_by_groups = choc1.groupby('clusterlabs')[['timestamp','pk','clusterlabs','sensor'
```

```
chocs1_by_groups.apply(display)
```

	timestamp	pk	clusterlabs	sensor	x	y	
0	2021-04-01 00:13:54.178	16128.006392	0	XXX_2	[-0.39, -0.79, 0.51, 0.31, -0.29, 0.0099999999...	[0.246875, -0.653125, -2.353125, -1.353125, 0....	[-0.22625, -0. -0.12625, 0. -
232	2021-04-01 19:54:24.114	16130.726384	0	XXX_2	[-0.9125, -0.0125, 0.1875, -0.1125, -0.2125000...	[1.4124999999999999, 0.6124999999999999, 0.312...	[-0.22875, 0. 0. -0.0287500000
250	2021-04-01 21:00:27.156	16130.792796	0	XXX_2	[-0.09, -0.09, -0.09, 0.010000000000000002, 0....	[0.774375, 0.774375, 0.774375, -0.525625, -1.8...	[-0.351875000000 0.448125, -0.1

	timestamp	pk	clusterlabs	sensor	x	y	
1	2021-04-01 00:14:09.161	16356.841050	1	XXX_2	[0.155, 1.155, 0.455, -0.34500000000000003, -0...	[0.325, -0.17500000000000002, -0.275, -0.07500...	[1.424375, 0 0.22437500000
2	2021-04-01 00:14:09.588	16364.923015	1	XXX_1	[0.511875, 0.111875, 0.011875000000000004, -0....	[-0.881875, 0.018125000000000006, 0.318125, 1....	[0.23249999999 -0.8675 -
3	2021-04-01 00:14:09.971	16372.172177	1	XXX_2	[0.47125, 0.17125, -0.32875, 0.27125, -0.02875...	[-0.851875, 0.948125, 1.248125, 2.548125, 0.74...	[-0.68124999999 -1.08125,
24	2021-04-01 01:21:15.464	16350.861639	1	XXX_2	[-0.148125, -0.148125, -0.7481249999999999, 0....	[-2.2737499999999997, -4.37375, -2.57375, -0.4...	[-0.91, 0.49, 0.4 -0.51, 0.5
25	2021-04-01 01:21:16.678	16374.643433	1	XXX_2	[0.77625, -0.22375, -0.22375, -0.22375, 0.0762...	[-1.8218750000000001, 1.1781249999999999, 2.77...	[-0.629375, -0 -0.7293750000
26	2021-04-01 01:21:17.135	16383.595888	1	XXX_1	[0.395, -0.005, -0.305, -0.805, 0.095, -0.1050...	[0.68125, -0.21875, 0.28125, 0.4812499999999999...	[1.6531250000 -1.3468749999
211	2021-04-01 18:56:53.704	16383.733825	1	XXX_2	[0.164375, 0.064375, 0.26437499999999997, -0.8...	[-1.845625, -0.245625, -0.645625, 1.554375, 1....	[-0.28562499999 0.01437500000

	timestamp	pk	clusterlabs	sensor	x	y	
4	2021-04-01 00:15:34.629	18008.476951	2	XXX_1	[-0.579375, 0.520625, 0.420625, 0.120625000000...	[-0.185, 0.11500000000000002, -0.7849999999999...	[-0.84312 1.25687
5	2021-04-01 00:15:34.746	18010.736114	2	XXX_2	[-0.8456250000000001, 0.254375, 0.854375, 0.05...	[1.3099999999999998, 1.6099999999999999, -0.99...	[-0.6112 1.788
33	2021-04-01 01:22:41.316	18012.021986	2	XXX_2	[0.113125, -0.48687499999999995, -0.686875, 0....	[-0.11125000000000002, 0.8887499999999999, 0.6...	[-0.1812 -0.08124999
34	2021-04-01 01:22:41.520	18015.949944	2	XXX_1	[0.1925, -0.3075, 1.1925, 0.0925, -0.207500000...	[-0.905, 0.295, 0.695, 0.695, -0.2050000000000...	[-0.13187500 1.668
35	2021-04-01 01:22:41.721	18019.820138	2	XXX_2	[0.420625, 1.0206250000000001, 0.220625, -0.27...	[0.231875, -0.268125, 0.231875, 1.131875000000...	[0.1625, -1.63 -0.2375,

134	2021-04-01 03:19:50.364	18012.436653	2	XXX_2	[-0.50125, 0.09875, 0.39875000000000005, -0.50...	[-0.52625, -2.02625, -1.52625, 1.3737499999999...	[-0.6906249 -0.49062
179	2021-04-01 05:29:41.777	18009.332251	2	XXX_2	[0.29874999999999996, -0.10125, 0.298749999999...	[2.3162499999999997, -0.28375, -2.183750000000...	0.3199999 -0.38,
183	2021-04-01 13:19:41.988	17995.461683	2	XXX_2	[-0.516875, 0.183125, -0.216875, 0.38312500000...	[3.6550000000000002, 3.1550000000000002, 0.355...	[-0.05562 1.044375
184	2021-04-01 13:19:42.787	18008.515859	2	XXX_1	[0.36062500000000003, 0.260625, 0.360625000000...	[-1.079375, 2.2206249999999996, 2.320624999999...	[-0.0806249 -0.7806249
185	2021-04-01 13:19:43.202	18015.301301	2	XXX_2	[0.84, -0.26, -0.46, 1.04, 1.54, -0.8600000000...	[2.47625, 2.27625, 1.7762499999999999, -0.5237...	[0.86437 2.164375
204	2021-04-01 16:15:29.425	18009.693834	2	XXX_2	[0.21125, -0.08875000000000001, -0.28875, -0.0...	[-2.868125, -0.568125, 1.631875, 2.131875, 2.7...	[-0.74, 0.16, -0.14, 0.
218	2021-04-01 18:58:32.068	18010.175835	2	XXX_2	[0.26875, -0.23125, 0.26875, -1.33125, -1.0312...	[0.16499999999999998, 0.9650000000000001, 0.96...	[-0.3575000 -1.2575, -0.5
219	2021-04-01 18:58:32.460	18016.663066	2	XXX_1	[0.843125, -0.15687500000000001, -0.2568749999...	[1.235625, 1.935625, -0.06437500000000002, -0....	[1.15562 -0.0443750000
235	2021-04-01 19:56:19.878	18014.511776	2	XXX_2	[-0.043125000000000004, -0.743125, 0.856875, 0...	[1.36625, 2.26625, 2.96625, 2.1662500000000002...	[-0.491 0.20875000
254	2021-04-01 21:02:22.536	18007.883240	2	XXX_2	[0.17875000000000002, 0.27875, 1.47875, -0.421...	[-1.783125, -0.583125, -0.583125, -0.783125000...	[-0.611875 1.88
255	2021-04-01 21:02:22.873	18013.458646	2	XXX_1	[0.14125000000000001, -0.75875, -0.05875000000...	[-1.3, -2.9, -1.0, 0.3, 1.5, 3.6, 0.7, -0.5, -...	[0.2149999 2.315, 0.315,
256	2021-04-01 21:02:22.941	18014.583654	2	XXX_2	[-0.149375, 0.8506250000000001, -1.74937500000...	[-2.83, -5.43, -6.13, 2.87, 3.9699999999999998...	[-0.90687 4.393125
266	2021-04-01 22:09:17.254	18005.437253	2	XXX_1	[-0.71875, 0.9812500000000001, -1.11875, 0.181...	[1.443125, 0.34312499999999996, 1.443125, -2.0...	0.3993750 -0.80
267	2021-04-01 22:09:17.616	18011.414652	2	XXX_2	[-0.043125000000000004, -0.043125000000000004,...	[1.90375, 2.50375, 3.00375, 1.80375, -1.59625,...	-1.363125 0.5
272	2021-04-01 23:13:35.822	18008.675968	2	XXX_2	[-0.36250000000000004, -0.1625, 0.6375, -0.662...	[-1.07625, -1.2762499999999999, -0.67625, -0.5...	[-0.13187 -1.33
273	2021-04-01 23:13:36.227	18015.369067	2	XXX_2	[0.23125, -4.06875, -1.06875, 0.13125, -1.6687...	[-2.029375, 13.070625, 4.070625, 4.470625, -0....	[-1.624375 1.67562

	timestamp	**pk**	**clusterlabs**	**sensor**	**x**	**y**	
13	2021-04-01 00:17:43.943	19561.646724	3	XXX_2	[-1.840625, 0.059375000000000004, 1.159375, -0...	[0.121875, -4.478125, -1.778125, 0.621875, 4.5...	[-0.2306249999 1.069375, -2.030

181	2021-04-01 05:31:54.585	19563.114609	3	XXX_1	[0.7837500000000001, 0.7837500000000001, -0.41...	[0.143125, -0.0568750000000001, -0.0568750000...	[-1.36375, 0.93625, 0.0
187	2021-04-01 13:21:57.902	19561.641541	3	XXX_2	[0.0275, -0.4725, 0.0275, 0.7274999999999999, ...	[0.904375, 0.904375, 0.404375, -3.195625, -0.5...	[0.34125000000 0.44125000000
207	2021-04-01 16:17:43.484	19558.655338	3	XXX_1	[-1.0693750000000002, 0.8306250000000001, 0.03...	[2.086875, -0.9131250000000001, -0.613125, -0....	[0.483125, -1. 0. 0.383125

	timestamp	pk	clusterlabs	sensor	x	y	
15	2021-04-01 00:20:49.011	21393.627289	4	XXX_1	[-0.95625, -0.25625, 0.54375, -1.25625, 0.0437...	[-1.06875, -0.46875, -1.8687500000000001, -0.7...	[0.6806249999 -2. -0.019374
16	2021-04-01 00:20:49.057	21394.387560	4	XXX_2	[0.19312500000000002, -0.6068749999999999, -0....	[1.7975, 0.1975, -0.6024999999999999, -0.70249...	[-0.405, 0.39 -0.0050000000000
237	2021-04-01 20:01:34.575	21396.508965	4	XXX_1	[0.15500000000000003, -0.8450000000000001, -0....	[0.16749999999999998, -2.0325, -2.2325, -1.532...	[0.35625, -0.44375000000
258	2021-04-01 21:07:50.125	21389.663828	4	XXX_2	[-0.561875, -0.16187500000000002, -0.661875, 0...	[0.18250000000000002, 2.6825, 1.4825, -0.01749...	[1.2037499999 0.10375000000
259	2021-04-01 21:07:50.688	21398.703600	4	XXX_1	[0.01625, 0.21625, 0.21625, -0.383750000000000...	[0.95, 2.75, -1.55, -1.05, -1.45, -2.15, 0.25,...	[0. 1.1993749999 0.499375

	timestamp	pk	clusterlabs	sensor	x	y	
17	2021-04-01 00:20:50.646	21422.460800	5	XXX_1	[0.5225, -0.4775, 0.5225, 0.7224999999999999, ...	[0.224375, -2.7756250000000002, -1.775625, -1....	[0.576875, -1 2.4768749999
194	2021-04-01 14:26:05.656	21422.592413	5	XXX_1	[0.39625000000000005, -0.40375, -0.30374999999...	[2.1350000000000002, 1.2349999999999999, -0.26...	-8.6736173798 19, 1.2, -1
195	2021-04-01 14:26:05.903	21426.652514	5	XXX_2	[-0.26437499999999997, -0.364375, 0.735625, 0....	[-2.7424999999999997, -1.2425, 0.2575, 1.9575,...	[0.3112499999 0.41125, -
208	2021-04-01 16:21:03.764	21423.507719	5	XXX_2	[-0.209375, -0.609375, 0.890625, 0.19062500000...	[-0.6956249999999999, -1.6956250000000002, -1....	[-0.0706250000 0.12937500000
238	2021-04-01 20:01:36.045	21420.302964	5	XXX_2	[-0.12312500000000001, -0.22312500000000002, 0...	[0.229375, 1.2293749999999999, 2.229375, 1.029...	[0.489375, 0 -0.610625, -0
239	2021-04-01 20:01:36.213	21423.098852	5	XXX_1	[-0.21625, -0.21625, 0.28375, 0.08375, -0.3162...	[0.6849999999999999, -0.515, -2.7150000000000...	[0.2775, 0.1775 0.7775, 0.3775
260	2021-04-01 21:07:52.148	21422.893743	5	XXX_2	[0.13124999999999998, 0.13124999999999998, -0....	[0.761875, 0.161875, 0.661875, 0.461875, -1.23...	[-0.268125, -0 0.431875, -0.668
261	2021-04-01 21:07:52.326	21425.869367	5	XXX_1	[0.4975, -0.3025, -0.10250000000000001, 0.1975...	[0.03437499999999999, 0.9343750000000001, -1.6...	0.1900000000 -0.00999999

274	2021-04-01 23:19:13.909	21419.964504	5	XXX_2	[0.46375, -0.33625, 0.06375, -0.13625, 0.16375...	[-1.3587500000000001, -1.9587500000000002, -1....	[0.135, 0.83 -1.46499999

	timestamp	pk	clusterlabs	sensor	x	y	
20	2021-04-01 01:10:35.653	12103.529972	6	XXX_2	[0.49625, -0.70375, -0.10375000000000001, 0.79...	[-0.05875000000000001, -0.15875, -0.75875, -1....	[-1.031875, -0.1318
21	2021-04-01 01:10:36.463	12117.466325	6	XXX_2	[-0.35375, -0.053750000000000006, -0.75375, 0....	[0.374375, -0.025624999999999998, -1.025625, -0...	[-1.108125, - 2.79187499
22	2021-04-01 01:10:36.632	12120.374033	6	XXX_1	[1.4206249999999998, -0.079375, -0.979375, -0....	[-3.428125, -3.728125, -2.728125, -0.528125, 2...	[0.16125 -0.13875
221	2021-04-01 19:48:43.234	12102.823539	6	XXX_2	[0.0037500000000000007, 0.30374999999999996, -...	[2.828125, 0.42812500000000003, 0.328125, -2.7...	[0.674375, -0.325625, -
222	2021-04-01 19:48:43.336	12104.585724	6	XXX_1	[-0.135625, 0.564375, 2.464375, -0.235625, 1.7...	[-2.7581249999999997, -4.758125000000001, -1.9...	[2.795625, - -0.904375, -
223	2021-04-01 19:48:43.638	12109.803173	6	XXX_2	[0.7062499999999999, -1.1937499999999999, -0.8...	[3.02, -0.28, 0.8200000000000001, 4.02, -2.18,...	[-1.36125, 1.03875
240	2021-04-01 20:54:17.369	12107.048661	6	XXX_2	[-0.085625, -0.285625, -0.185625, 1.014375, 0....	[-1.690625, -2.1906250000000003, -0.3906250000...	-0.19562500001 1.704
241	2021-04-01 20:54:17.774	12114.161754	6	XXX_2	[0.0031249999999999984, 0.303125, -1.996875, 1...	[8.4025, 5.2025, 6.9025, 0.8025, -12.2975, -1....	[2.580625, -0.81937500
262	2021-04-01 22:00:52.232	12103.078189	6	XXX_2	[0.6224999999999999, 0.022500000000000003, -0....	[1.541875, 0.841875, 1.141875, -1.058125, -2.9...	[-1.83562499 0.76437499

	timestamp	pk	clusterlabs	sensor	x	y	
27	2021-04-01 01:21:28.824	16610.742632	7	XXX_2	[0.461875, 0.461875, -0.6381249999999999, -0.6...	[2.1568750000000003, 0.656875, -0.343125, 0.35...	-0.7675000000 -0.0675, -
212	2021-04-01 18:57:08.015	16620.755367	7	XXX_1	[0.500625, -0.999375, 0.300625, 0.500625, -0.2...	[-1.3243749999999999, 0.275625, -0.52437499999...	[1.0593750000 -0.040625, -0.
233	2021-04-01 19:54:55.284	16614.252746	7	XXX_2	[0.72375, -0.5762499999999999, 0.02375, 0.1237...	[-0.41625, 1.3837499999999998, 2.38375, 2.5837...	[0.153125, -0. 0.053125000000
252	2021-04-01 21:00:59.137	16628.074697	7	XXX_2	[0.335625, 0.335625, 0.7356250000000001, 0.035...	[0.45125000000000004, -0.44875, 1.55125, 2.051...	[0.39875000000 0.69875, -
263	2021-04-01 22:07:53.333	16616.764756	7	XXX_1	[0.38625000000000004, -0.31375, -0.41375, 0.78...	[-1.69, 1.51, 2.11, 0.81, 0.41000000000000003,...	[-1.3924999999 0.00749999

	timestamp	pk	clusterlabs	sensor	x	y	
28	2021-04-01 01:21:36.115	16751.480164	8	XXX_2	[-0.165, 0.135, -0.665, -0.165,	[0.11, 0.21000000000000002,	[1.46, 1.26, 0.3 -1.54, -2.739999

	timestamp	pk	clusterlabs	sensor	x	y	
					0.534999999999...	-0.090000000000000...	
29	2021-04-01 01:21:36.387	16756.717694	8	XXX_1	[0.306875, -0.393125, -0.393125, -0.2931249999...	[0.017499999999999998, -0.4825, -0.9825, 0.317...	[0.14125000000 0.44125000000
203	2021-04-01 16:14:13.302	16749.900751	8	XXX_2	[0.18875, 0.48875, 0.18875, -0.01125, -0.51125...	[-2.52, -1.12, 1.88, 1.48, 1.38, 0.18000000000...	[(0.08875000000 -1.31125
213	2021-04-01 18:57:15.968	16751.738937	8	XXX_2	[0.25687499999999996, 0.656875, -0.04312500000...	[-1.014375, -2.4143749999999997, 0.385625, 1.6...	[0.6875, - -0.21250000000

	timestamp	pk	clusterlabs	sensor	x	y	z
30	2021-04-01 01:21:53.533	17087.701659	9	XXX_2	[0.40125, -0.19875, 0.20125, -0.29875, 0.30124...	[3.705, 0.405, 0.505, 1.905, -4.995, -0.795, 0...	[0.298125, -2.2018750000000002, 0.698125, -0.5...
31	2021-04-01 01:21:54.002	17096.783250	9	XXX_1	[1.626875, 0.626875, -0.473125, 0.826875, -0.1...	[1.026875, -2.073125, -0.37312500000000004, -1...	[-2.5275, 0.27249999999999996, 0.9725, 0.17250...
192	2021-04-01 14:19:40.060	17087.912781	9	XXX_2	[0.080625, -0.419375, 0.480625, -0.719375, 0.2...	[1.5756249999999998, 4.875625, 2.875625, 0.275...	[-0.245, -0.245, 0.655, 0.155, 0.155, -0.245, ...
215	2021-04-01 18:57:36.616	17092.363738	9	XXX_2	[-0.258125, 1.041875, 0.041875, -1.858125, 0.8...	[0.4, 3.9999999999999996, 4.5, 0.8, -1.9000000...	[-0.5187499999999999, -1.51875, -0.11875000000...

	timestamp	pk	clusterlabs	sensor	x	y	
132	2021-04-01 03:18:00.607	16199.948788	10	XXX_2	[0.66375, 0.96375, -0.33625, -0.73625, 0.56375...	[1.98625, -0.11375000000000002, 0.78625, 0.486...	[-2.03625, 1.76375, 1
177	2021-04-01 05:27:52.052	16199.598490	10	XXX_2	[-0.20625000000000002, -0.20625000000000002, 0...	[1.1243750000000001, 0.424375, 0.524375, -1.07...	[-0.38375000000 0.21625, -0.683
191	2021-04-01 14:18:45.851	16199.338663	10	XXX_2	[-0.5349999999999999, -0.235, 0.7649999999999...	[-0.22875, 0.6712499999999999, 0.8712500000000...	[-0.18000000000 0.82000000000
251	2021-04-01 21:00:32.823	16197.853654	10	XXX_2	[0.89125, -0.30874999999999997, -0.40875, 0.39...	[-1.05, -0.6499999999999999, -0.6499999999999...	[0.10375000000 -0.09625, -

	timestamp	pk	clusterlabs	sensor	x	y	
133	2021-04-01 03:19:35.781	17771.301484	11	XXX_2	[0.10187500000000001, -0.7981250000000001, 0.3...	[0.238125, 2.538125, 2.438125, 0.538125, -1.16...	[-0. -0.20187500000 -0.1018
178	2021-04-01 05:29:27.608	17775.014170	11	XXX_2	[1.2293749999999999, 0.229375, -0.570624999999...	[-0.08187500000000002, 0.118125, 0.118125, -0....	[-3.06875, -3 -0.76875, 2 2

271	2021-04-01 23:13:21.655	17775.240368	11	XXX_2	[0.495, 0.995, 1.0950000000000002, -0.705, 0.2...	[-1.01125, 0.7887500000000001, 1.58875, 0.1887...	[0.77374999999 -2.72625, -3

	timestamp	pk	clusterlabs	sensor	x	y	
190	2021-04-01 14:17:24.935	15557.737784	12	XXX_2	[-0.39687500000000003, -0.496875, 0.0031249999...	[-0.883125, -1.283125, -2.183125, 0.916875, 3....	[0.11125000000 -0.88875, 0
201	2021-04-01 15:11:08.828	15549.283121	12	XXX_2	[0.45062500000000005, -0.24937499999999999, -0...	[-0.07375, 0.82625, 0.42625, 2.42625, -0.57375...	[1.26875, 0 -0.93125, -0 0.
210	2021-04-01 18:55:11.667	15559.363958	12	XXX_2	[0.579375, -0.420625, -0.420625, 0.979375, 0.9...	[-2.094375, -0.39437500000000003, 0.2056250000...	[0.68624999999 0.68624999999

	timestamp	pk	clusterlabs	sensor	x	y	
196	2021-04-01 15:10:45.349	15163.358015	13	XXX_2	[-0.135625, 0.16437500000000002, -0.535625, -0...	[-3.0900000000000003, -2.5900000000000003, 0.7...	[-0.11875000000 -0.31875, -0.2187
197	2021-04-01 15:10:45.349	15163.358015	13	XXX_2	[-0.135625, 0.16437500000000002, -0.535625, -0...	[-3.0900000000000003, -2.5900000000000003, 0.7...	[-0.11875000000 -0.31875, -0.2187
198	2021-04-01 15:10:45.349	15163.358015	13	XXX_2	[-0.135625, 0.16437500000000002, -0.535625, -0...	[-3.0900000000000003, -2.5900000000000003, 0.7...	[-0.11875000000 -0.31875, -0.2187
199	2021-04-01 15:10:45.349	15163.358015	13	XXX_2	[-0.135625, 0.16437500000000002, -0.535625, -0...	[-3.0900000000000003, -2.5900000000000003, 0.7...	[-0.11875000000 -0.31875, -0.2187

	timestamp	pk	clusterlabs	sensor	x	y	
200	2021-04-01 15:11:05.590	15497.979314	14	XXX_2	[-0.315625, -0.115625, -0.215625, -0.015624999...	[-3.2556249999999998, -0.455625, 0.144375, 1.6...	[0.180625, 0 -0 -0.219375
202	2021-04-01 16:11:12.378	15505.400911	14	XXX_2	[-0.12250000000000001, 0.27749999999999997, -1...	[-0.1975, -0.3975, 0.5025, -0.7975000000000001...	[-0.721875, -0 0.978125, 1.178
209	2021-04-01 18:55:08.426	15507.769225	14	XXX_2	[0.7174999999999999, 0.5175, 0.117500000000000...	[-0.949375, 3.050625, 2.550625, 1.750625, 1.15...	[-0.46437500000 -0.46437500000
231	2021-04-01 19:53:10.835	15507.353901	14	XXX_2	[-0.025, 0.375, -0.625, 0.07500000000000001, 0...	[0.15687500000000001, -0.243125, -0.943125, -1...	[-0.60749999999 -0.1075
246	2021-04-01 20:59:03.350	15504.005426	14	XXX_2	[0.279375, 0.279375, -1.720625, 1.179374999999...	[-1.4049999999999998, 0.095, 0.595, 0.795, -0....	[0.39875000000 1.19875,
247	2021-04-01 20:59:03.350	15504.005426	14	XXX_2	[0.279375, 0.279375, -1.720625, 1.179374999999...	[-1.4049999999999998, 0.095, 0.595, 0.795, -0....	[0.39875000000 1.19875,
248	2021-04-01 20:59:03.350	15504.005426	14	XXX_2	[0.279375, 0.279375, -1.720625, 1.179374999999...	[-1.4049999999999998, 0.095, 0.595, 0.795, -0....	[0.39875000000 1.19875,
249	2021-04-01	15504.005426	14	XXX_2	[0.279375, 0.279375,	[-1.4049999999999998,	[0.39875000000

	20:59:03.350				-1.720625, 1.179374999999...	0.095, 0.595, 0.795, -0....	1.19875,

	timestamp	pk	clusterlabs	sensor	x	y	
226	2021-04-01 19:52:37.631	14958.326083	15	XXX_2	[-0.174375, 0.42562500000000003, -0.174375, -1...	[1.83375, 1.53375, 0.033749999999999995, -1.16...	[-1.02125, -0.4 0.1787500000000 (
227	2021-04-01 19:52:37.631	14958.326083	15	XXX_2	[-0.174375, 0.42562500000000003, -0.174375, -1...	[1.83375, 1.53375, 0.033749999999999995, -1.16...	[-1.02125, -0.4 0.1787500000000 (
228	2021-04-01 19:52:37.631	14958.326083	15	XXX_2	[-0.174375, 0.42562500000000003, -0.174375, -1...	[1.83375, 1.53375, 0.033749999999999995, -1.16...	[-1.02125, -0.4 0.1787500000000 (
229	2021-04-01 19:52:37.631	14958.326083	15	XXX_2	[-0.174375, 0.42562500000000003, -0.174375, -1...	[1.83375, 1.53375, 0.033749999999999995, -1.16...	[-1.02125, -0.4 0.1787500000000 (

	timestamp	pk	clusterlabs	sensor	x	y	
193	2021-04-01 14:25:59.834	21331.740889	16	XXX_2	[-0.87875, -0.7787499999999999, 0.52125, -0.57...	[-1.115, 1.885, 1.2850000000000001, 2.685, 2.9...	[-0.293124999999 -0.493125, -0.9931
236	2021-04-01 20:01:29.971	21325.152247	16	XXX_2	[0.07875, -0.72125, 0.17875000000000002, 0.378...	[-0.975, -0.975, -1.5750000000000002, 0.025000...	[-0.553125, -1.2 1.546875, 0.24687
257	2021-04-01 21:07:45.673	21319.375410	16	XXX_2	[-0.16625, -0.46625, 0.53375, -0.0662499999999...	[-2.1700000000000004, 0.33, 0.0300000000000000...	[0.0787500000000 -0.62125, 0.4 (

Out[694]: —

In [588…
```
from ast import literal_eval
```

In [720…
```
#Analyse des valeurs ponctuelles dans un premier temps
def day1_sig_analysis(s,window_size = 8): #160 points donc 140 valeurs

    x_s = np.array(literal_eval(choc1["x"][s]))
    y_s = np.array(literal_eval(choc1["y"][s]))
    z_s = np.array(literal_eval(choc1["z"][s]))

    # Acceleration peak
    peak_x_s = np.max(abs(x_s))
    peak_y_s = np.max(abs(y_s))
    peak_z_s = np.max(abs(z_s))

    # Global RMS Acceleration
    rms_x_s = np.sqrt((x_s**2).sum())
    rms_y_s = np.sqrt((y_s**2).sum())
    rms_z_s = np.sqrt((z_s**2).sum())

    # Windowed RMS Acceleration
    wrms_x_s = np.zeros(len(x_s))
    wrms_y_s = np.zeros(len(x_s))
    wrms_z_s = np.zeros(len(x_s))

    for i in range(1,len(x_s)):
        if i <= len(x_s)-window_size:
            start = i
            end = i + window_size
```

```
            wrms_x_s[i] = np.sqrt((x_s[start:end]**2).sum())
            wrms_y_s[i] = np.sqrt((y_s[start:end]**2).sum())
            wrms_z_s[i] = np.sqrt((z_s[start:end]**2).sum())

        else:
            break

    sigs = [x_s,y_s,z_s]
    peaks = [peak_x_s, peak_y_s, peak_z_s]
    RMS = [rms_x_s,rms_y_s,rms_z_s]
    wRMS = [wrms_x_s,wrms_y_s,wrms_z_s]

    return(sigs,peaks,RMS,wRMS)
```

In [725…

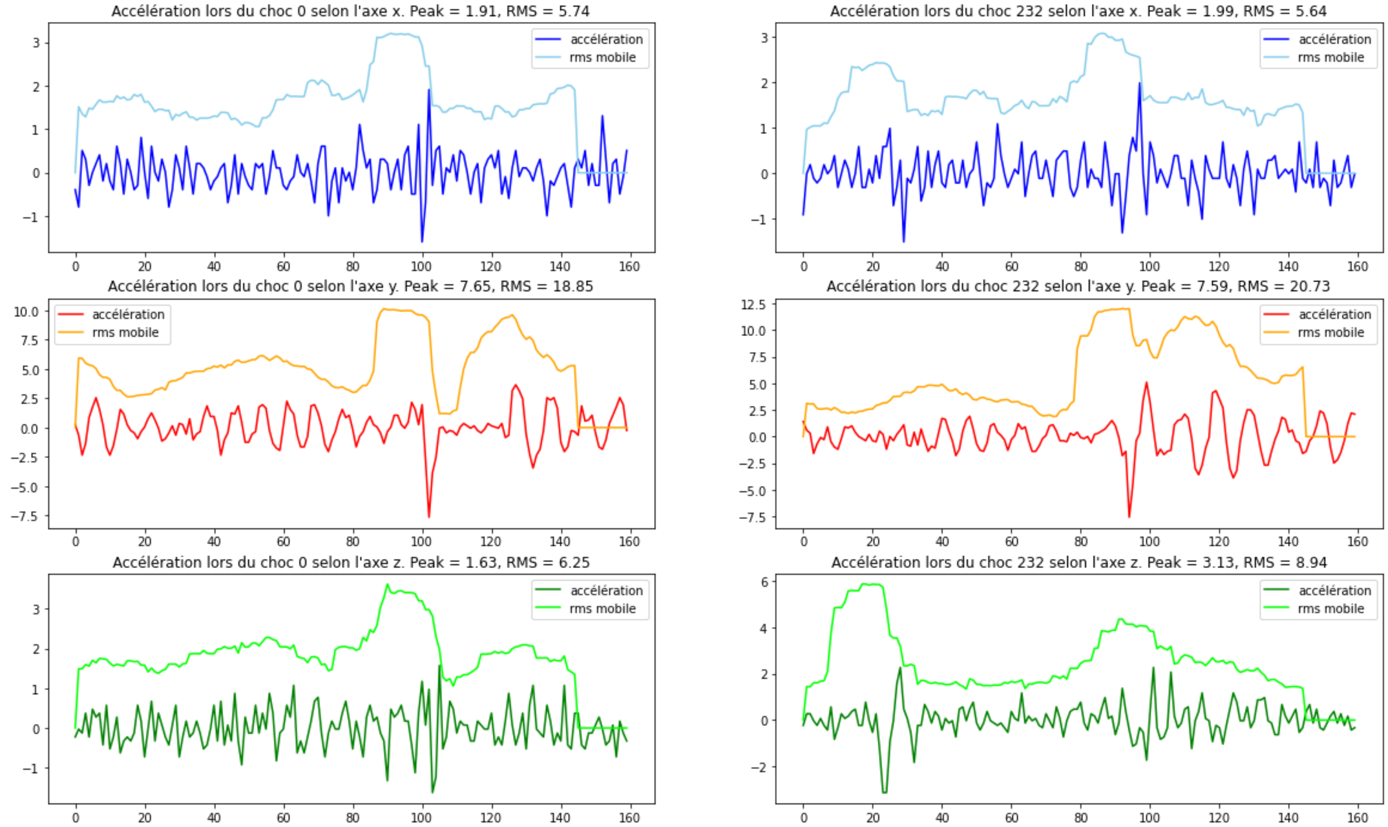
```
def comparaison_chocs(A,B,window_size):
    sigs_A,peaks_A,rms_A,wrms_A = day1_sig_analysis(A,window_size)
    sigs_B,peaks_B,rms_B,wrms_B = day1_sig_analysis(B,window_size)

    fig, ((ax1,ax4),(ax2,ax5),(ax3,ax6)) = plt.subplots(3,2,figsize = (20,12))

    ax1.plot(sigs_A[0],color = 'blue',label = 'accélération')
    ax1.plot(wrms_A[0],color = 'skyblue',label = 'rms mobile')
    ax1.set_title(f"Accélération lors du choc {A} selon l'axe x. Peak = {round(peaks_A[0
    ax1.legend()

    ax2.plot(sigs_A[1],color = 'red',label = 'accélération')
    ax2.plot(wrms_A[1],color = 'orange',label = 'rms mobile')
    ax2.set_title(f"Accélération lors du choc {A} selon l'axe y. Peak = {round(peaks_A[1
    ax2.legend()

    ax3.plot(sigs_A[2],color = 'green',label = 'accélération')
    ax3.plot(wrms_A[2],color = 'lime',label = 'rms mobile')
    ax3.set_title(f"Accélération lors du choc {A} selon l'axe z. Peak = {round(peaks_A[2
    ax3.legend()

    ax4.plot(sigs_B[0],color = 'blue',label = 'accélération')
    ax4.plot(wrms_B[0],color = 'skyblue',label = 'rms mobile')
    ax4.set_title(f"Accélération lors du choc {B} selon l'axe x. Peak = {round(peaks_B[0
    ax4.legend()

    ax5.plot(sigs_B[1],color = 'red',label = 'accélération')
    ax5.plot(wrms_B[1],color = 'orange',label = 'rms mobile')
    ax5.set_title(f"Accélération lors du choc {B} selon l'axe y. Peak = {round(peaks_B[1
    ax5.legend()

    ax6.plot(sigs_B[2],color = 'green',label = 'accélération')
    ax6.plot(wrms_B[2],color = 'lime',label = 'rms mobile')
    ax6.set_title(f"Accélération lors du choc {B} selon l'axe z. Peak = {round(peaks_B[2
    ax6.legend()

    print(f"{A} est un choc du groupe {choc1.loc[A]['clusterlabs']} et {B} est un choc d
    print(f"Différence des pics selon l'axe x : {round(abs(peaks_A[0]-peaks_B[0]),2)}, e
    print(f"Différence des pics selon l'axe y : {round(abs(peaks_A[1]-peaks_B[1]),2)}, e
    print(f"Différence des pics selon l'axe z : {round(abs(peaks_A[2]-peaks_B[2]),2)}, e
```

Lorsque les chocs appartiennent au même cluster :

In [727…

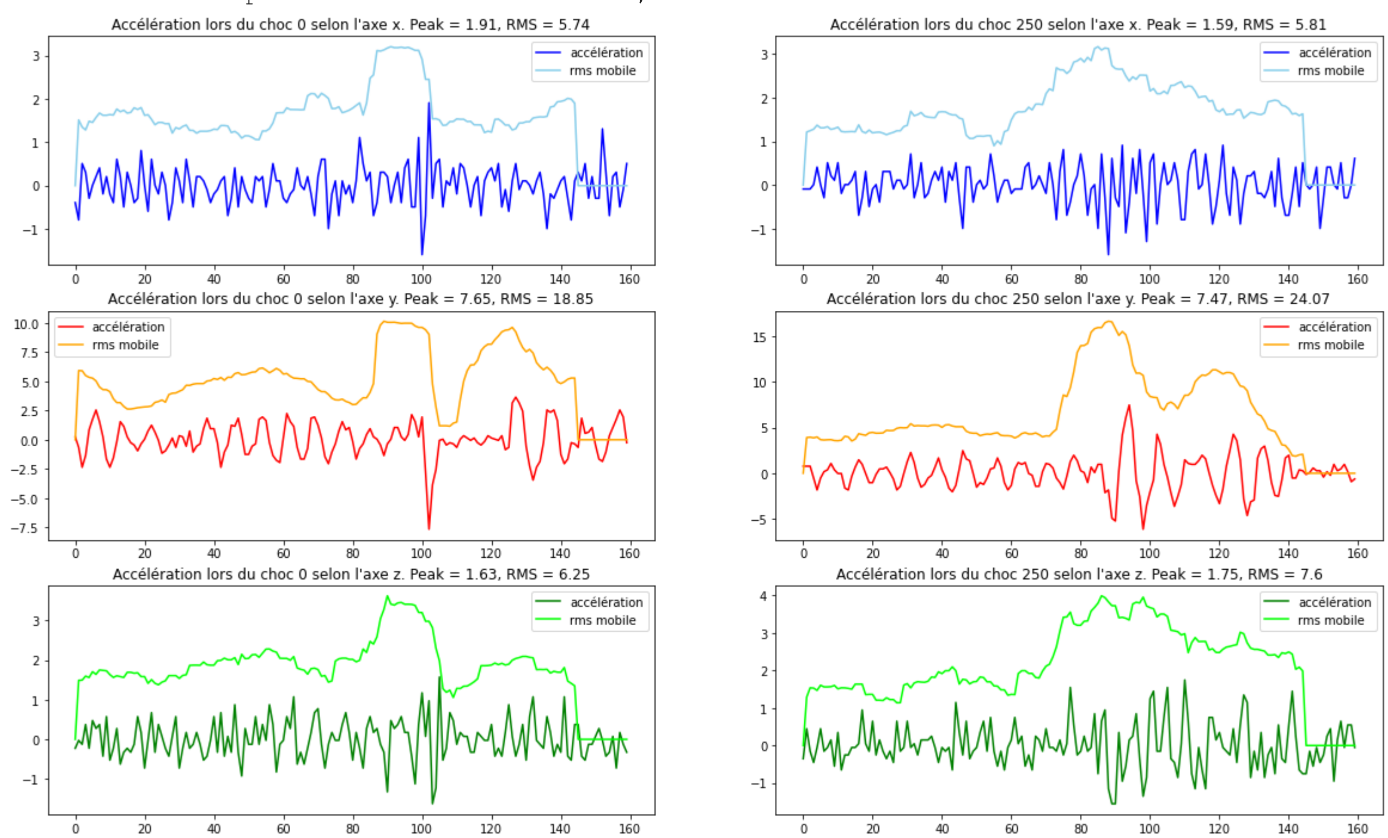
```
comparaison_chocs(0,232,window_size=16) #XXX_2
```

```
0 est un choc du groupe 0 et 232 est un choc du groupe 0
Différence des pics selon l'axe x : 0.08, et différence des RMS selon l'axe x : 0.1
Différence des pics selon l'axe y : 0.07, et différence des RMS selon l'axe y : 1.88
Différence des pics selon l'axe z : 1.5, et différence des RMS selon l'axe z : 2.68
```



In [729…
```
comparaison_chocs(0,250,window_size=16) #XXX_2
```

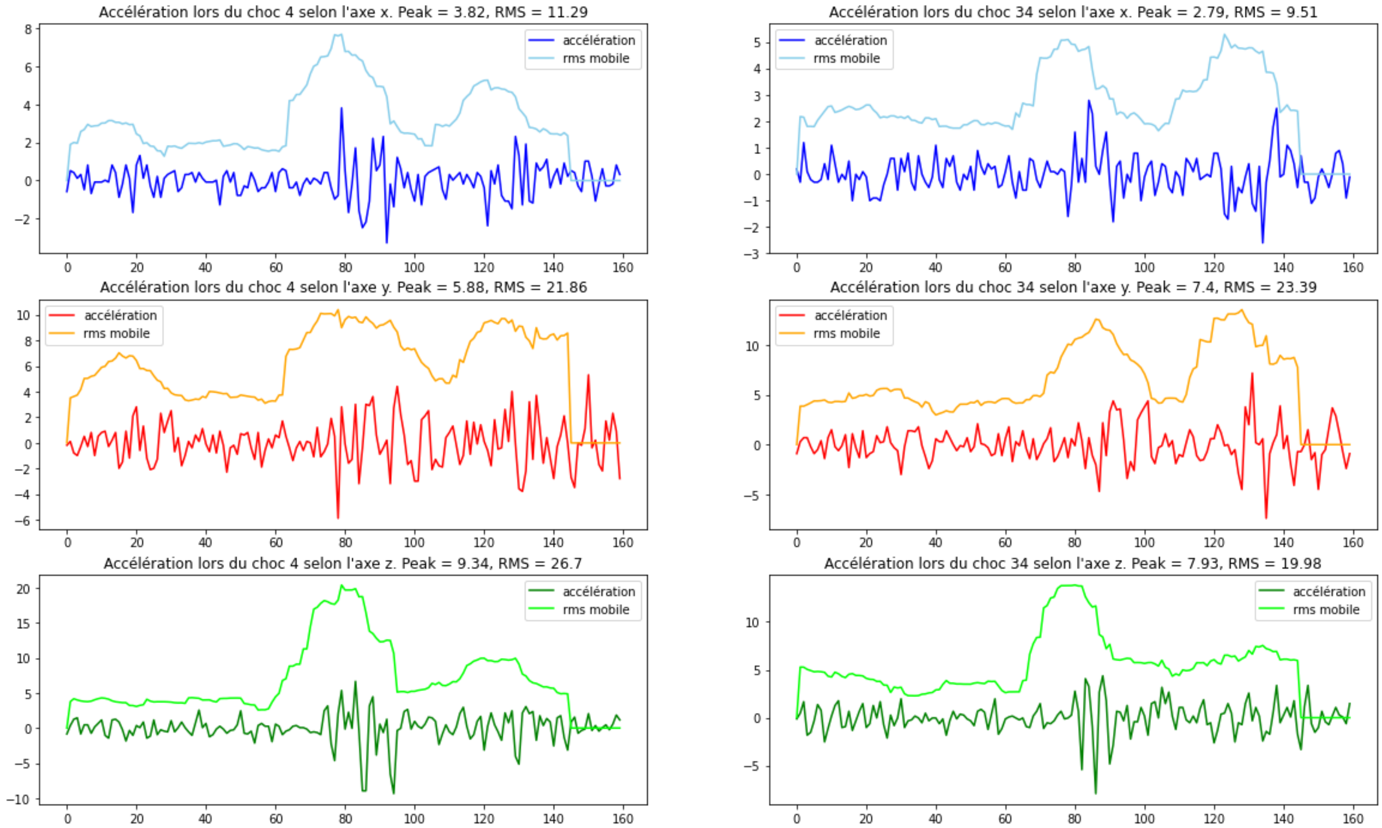
```
0 est un choc du groupe 0 et 250 est un choc du groupe 0
Différence des pics selon l'axe x : 0.32, et différence des RMS selon l'axe x : 0.07
Différence des pics selon l'axe y : 0.18, et différence des RMS selon l'axe y : 5.22
Différence des pics selon l'axe z : 0.12, et différence des RMS selon l'axe z : 1.34
```



In [730…
```
comparaison_chocs(4,34,window_size=16) #XXX_1
```

```
4 est un choc du groupe 2 et 34 est un choc du groupe 2
Différence des pics selon l'axe x : 1.03, et différence des RMS selon l'axe x : 1.78
```

```
Différence des pics selon l'axe y : 1.52, et différence des RMS selon l'axe y : 1.52
Différence des pics selon l'axe z : 1.41, et différence des RMS selon l'axe z : 6.71
```

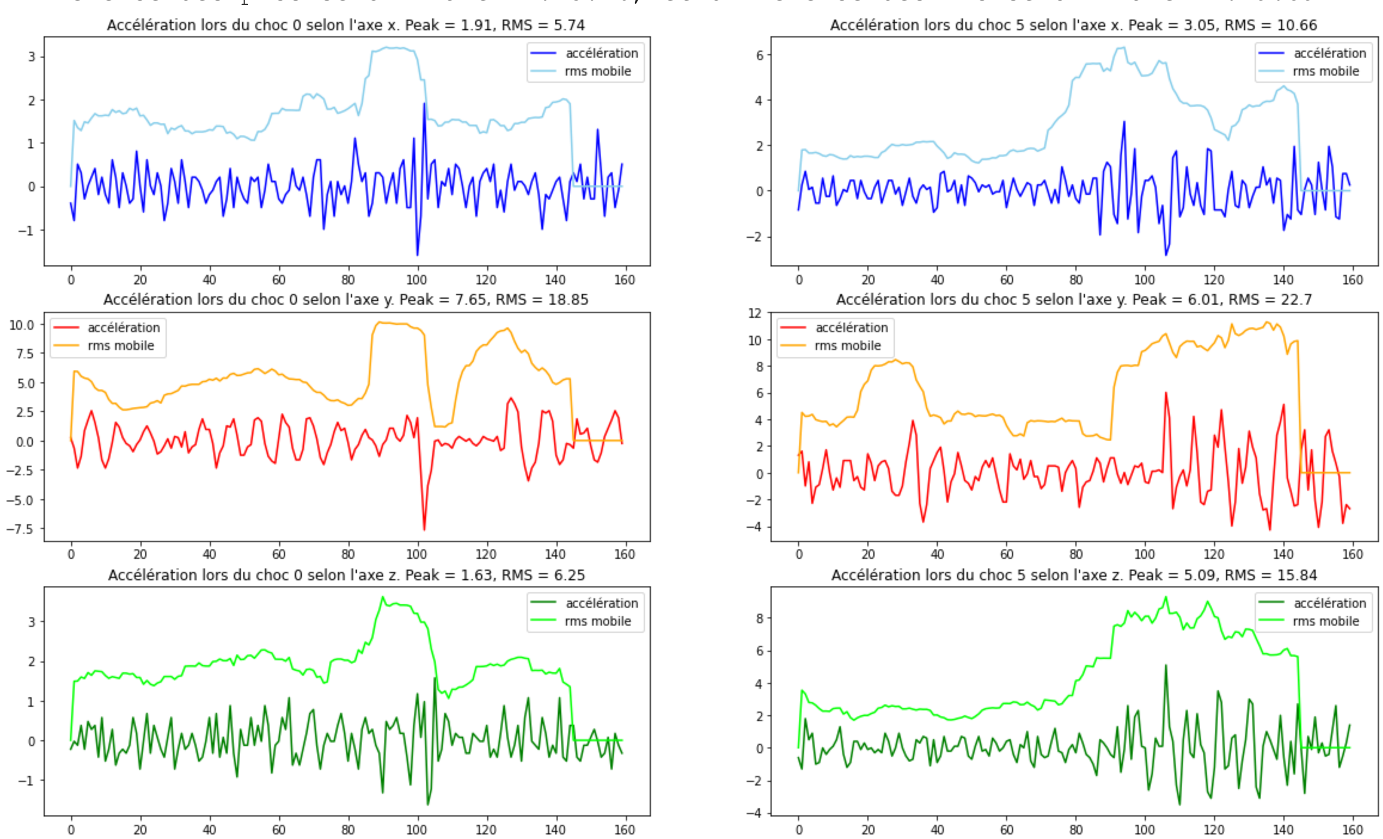


Lorsque les chocs de sont pas dans le même cluster :

In [733…

```
comparaison_chocs(0,5,window_size=16) #XXX_2
```

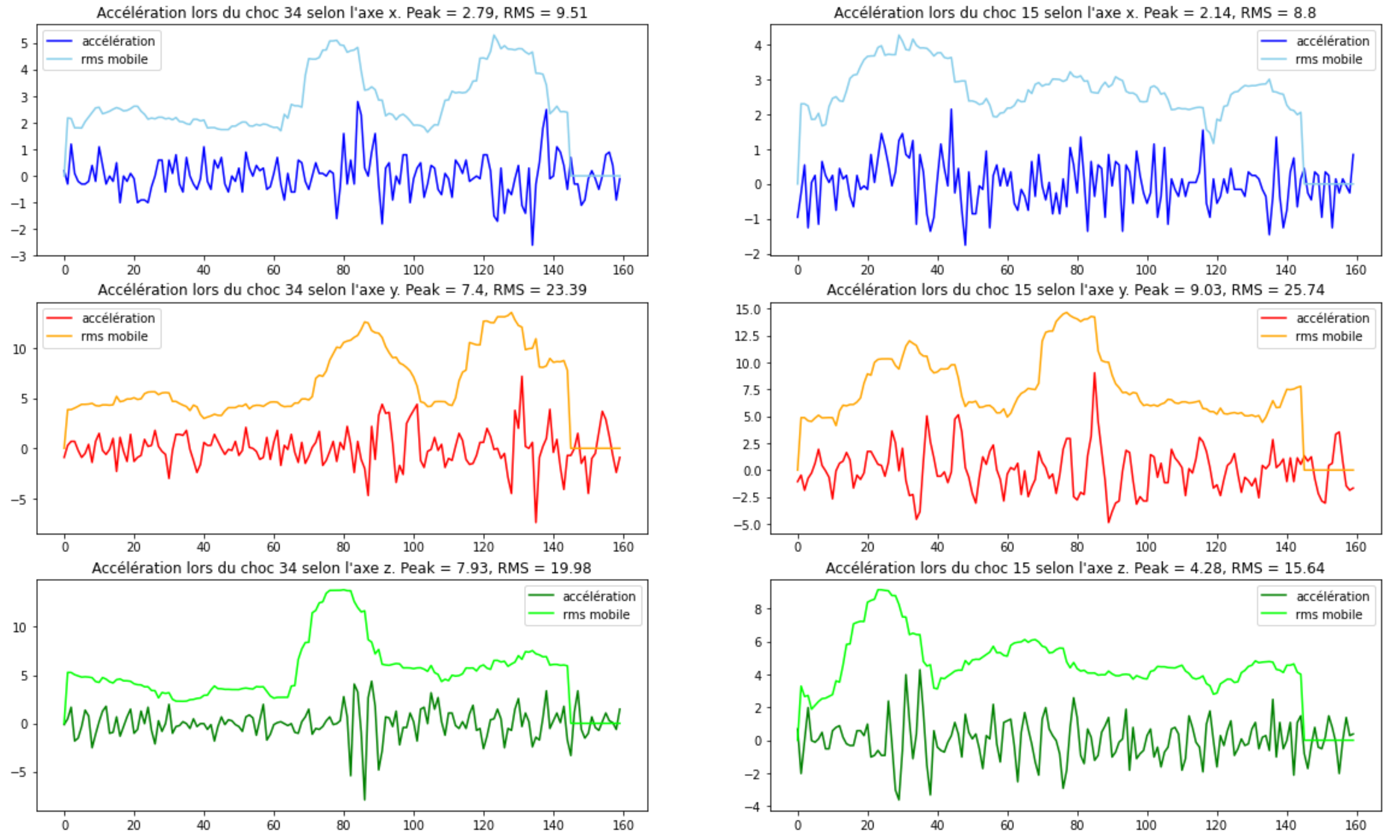
```
0 est un choc du groupe 0 et 5 est un choc du groupe 2
Différence des pics selon l'axe x : 1.14, et différence des RMS selon l'axe x : 4.92
Différence des pics selon l'axe y : 1.64, et différence des RMS selon l'axe y : 3.85
Différence des pics selon l'axe z : 3.46, et différence des RMS selon l'axe z : 9.59
```




```
comparaison_chocs(34,15,window_size=16) #XXX_1
```

In [732…

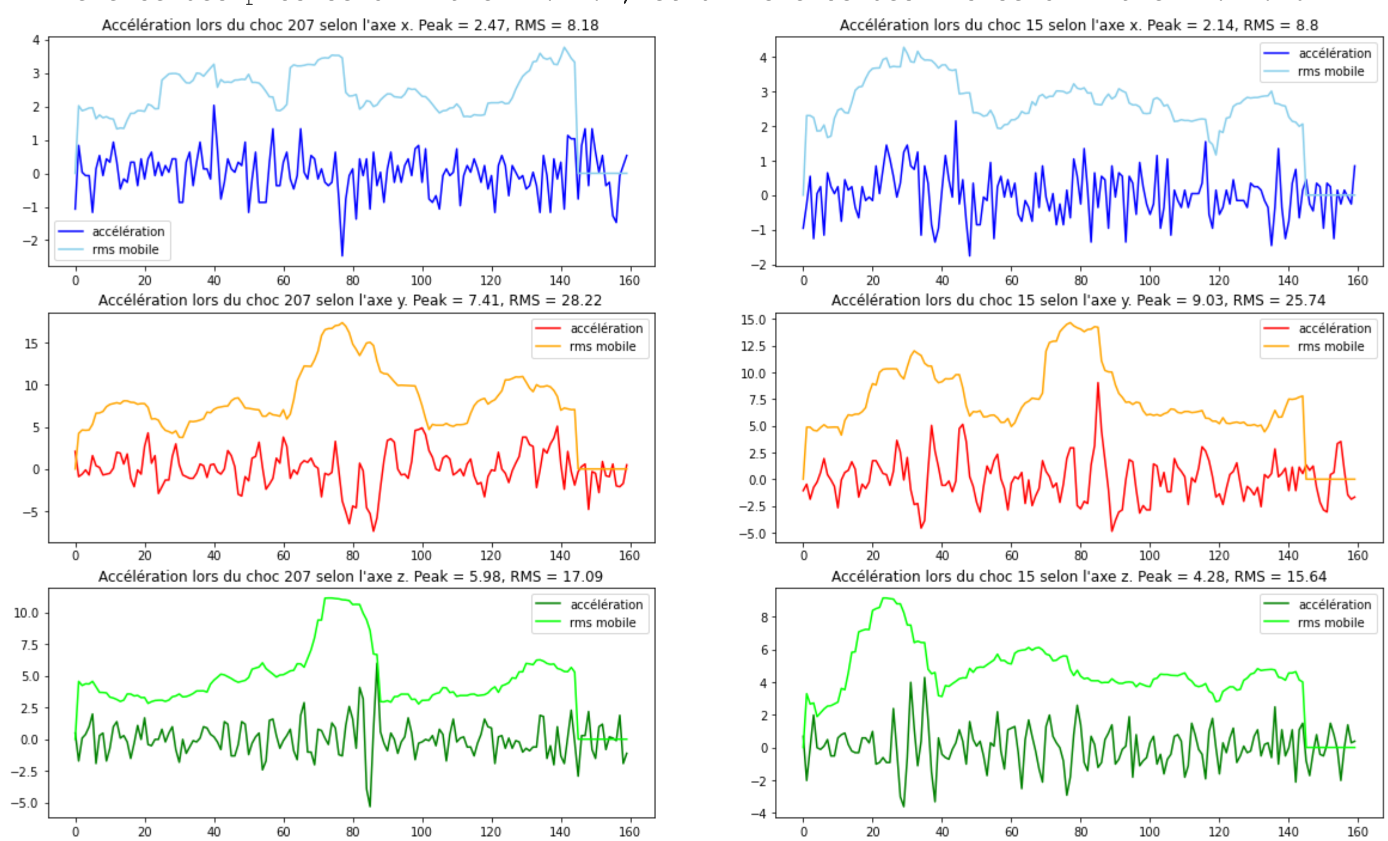
```
34 est un choc du groupe 2 et 15 est un choc du groupe 4
Différence des pics selon l'axe x : 0.65, et différence des RMS selon l'axe x : 0.71
Différence des pics selon l'axe y : 1.63, et différence des RMS selon l'axe y : 2.35
Différence des pics selon l'axe z : 3.65, et différence des RMS selon l'axe z : 4.35
```



In [736…

```
comparaison_chocs(207,15,window_size=16) #XXX_1
```

```
207 est un choc du groupe 3 et 15 est un choc du groupe 4
Différence des pics selon l'axe x : 0.33, et différence des RMS selon l'axe x : 0.62
Différence des pics selon l'axe y : 1.62, et différence des RMS selon l'axe y : 2.48
Différence des pics selon l'axe z : 1.7, et différence des RMS selon l'axe z : 1.46
```



Après plusieurs comparaisons de chocs, soit appartenant au même cluster, soit appartenant à des cluster différents, on constate que seule la RMSE mobile permet de caractériser correctement les accélération dûs à nos chocs

- Lorsque les chocs proviennent du même cluster, on observe une certaine cohérence entre les courbe des rms mobiles.
- Lorsque les chocs proviennent de deux cluster différents, il n'y a pas de cohérence ou alors peu entre ces courbes

Déplacement relatif :

In [681…

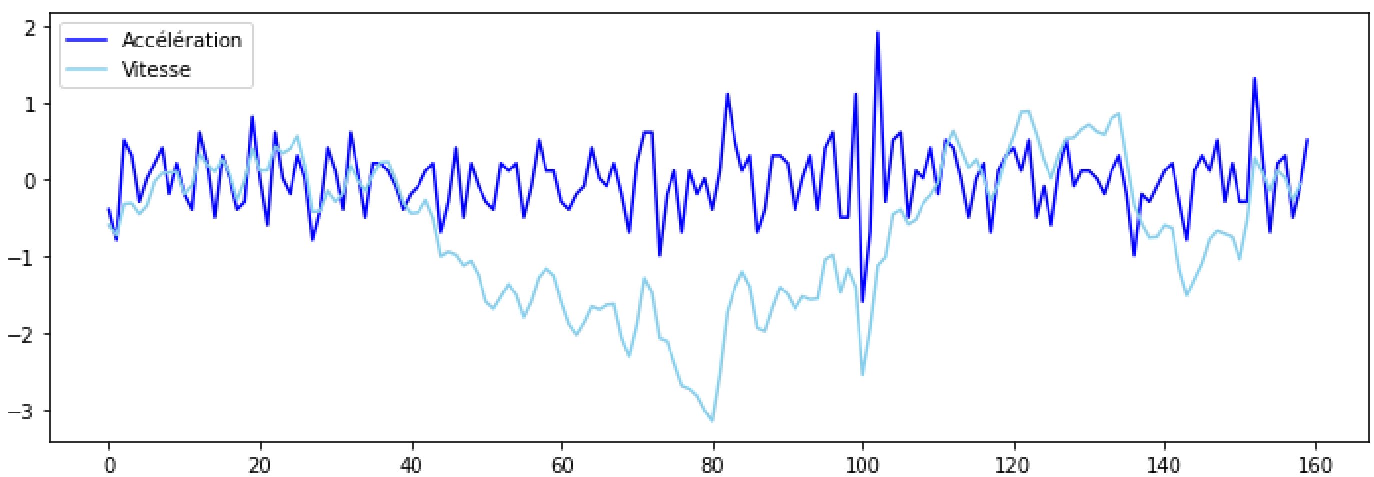
```
from scipy.integrate import cumtrapz
```

In [686…

```
plt.figure(figsize=(12,4))
plt.plot(np.array(literal_eval(choc1["x"][0])), color='blue',label = 'Accélération')
plt.plot(cumtrapz(np.array(literal_eval(choc1["x"][0]))), color='skyblue',label = 'Vites
plt.legend()
```

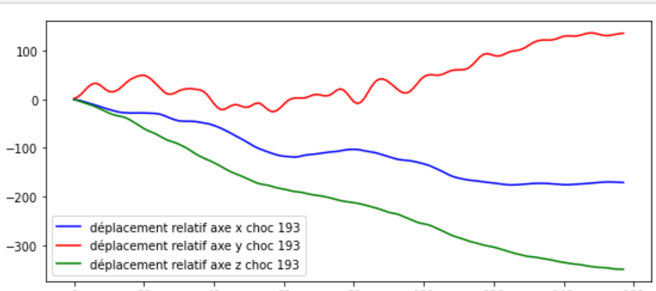
Out[686]:

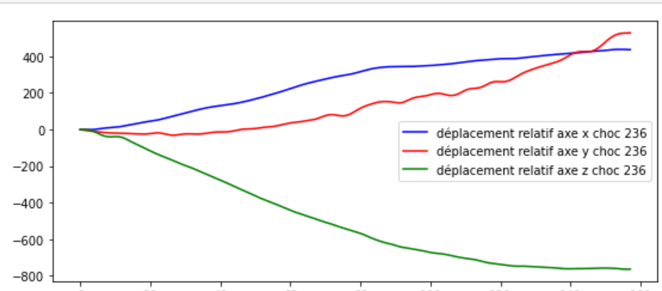
```
<matplotlib.legend.Legend at 0x2842efe7130>
```



In [757…

```
c = 193
c2 = 236
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(20,4))
ax1.plot(cumtrapz(cumtrapz(np.array(literal_eval(choc1["x"][c])))), color='blue',label =
ax1.plot(cumtrapz(cumtrapz(np.array(literal_eval(choc1["y"][c])))), color='red',label =
ax1.plot(cumtrapz(cumtrapz(np.array(literal_eval(choc1["z"][c])))), color='green',label
ax1.legend()

ax2.plot(cumtrapz(cumtrapz(np.array(literal_eval(choc1["x"][c2])))), color='blue',label
ax2.plot(cumtrapz(cumtrapz(np.array(literal_eval(choc1["y"][c2])))), color='red',label =
ax2.plot(cumtrapz(cumtrapz(np.array(literal_eval(choc1["z"][c2])))), color='green',label
ax2.legend();
```





Après l'étude des déplacements relatif provoqués par les chocs au sein des mêmes clusters, on peut caractériser un peu plus la nature des chocs :

- Cluster 0 impacte surtout l'axe des y
- Cluster 1 impacte surtout l'axe des x
- Cluster 2 impacte surtout l'axe des z et y
- Cluster 3 impacte surtout l'axe des y
- Cluster 4 impacte surtout l'axe des x et y
- Cluster 5 impacte surtout l'axe des y
- Cluster 6 impacte surtout l'axe des x,y,z
- Cluster 7 impacte surtout l'axe des y et z
- Cluster 8 impacte surtout l'axe des x,y,z
- Cluster 9 impacte surtout l'axe des y
- Cluster 10 impacte surtout l'axe des z et x
- Cluster 11 impacte surtout l'axe des x et y
- Cluster 12 impacte surtout l'axe des x
- Cluster 13 impacte surtout l'axe des x et z
- Cluster 14 impacte surtout l'axe des y
- Cluster 15 impacte surtout l'axe des y
- Cluster 16 impacte surtout l'axe des x,y,z

On aimerait maintenant répêter le clustering pour chaque jour, et effectuer des comparaisons de RMS mobile et de déplacement relatif des chocs au mêmes endroits (même cluster), mais à des jours différents.