mytraffic

July 26, 2022

1 MY TRAFFIC TEST

L'objectif de ce test est à partir des données fournies, l'identifiant du magasin , identifiant de l'appareil, date du ping, d'inferer les horaires d'ouverture des magasins.

Je cherche dans cet exercice à mettre en avant une méthode automatique viable pour n'importe quel magasin dans n'importe quelle situation à déterminer à partir des données disponible ses horaires d'ouverture.

Une premiere approche pourrait etre d'estimer par jour la distribution des pings au cours de la journée et définir un seuil d'acceptation cependant il est d'une part difficile de trouver automatiquement ce seuil d'acceptation pour chaque magasin et rien ne dit que choisir un seuil basé sur nos données se généralisera bien à d'autres magasins.

La méthode choisie est alors la suivante, je suppose que les arrivées des clients à chaque instant suivent une loi de poisson de paramètre inconnu. Ce parametre suit une loi continue en fonction du temps et dépend d'un paramètre correspondant à l'ouverture ou non du magasin, je vais donc chercher à detecter l'instant ou le paramètre change de loi c'est à dire "saut" dans les fréquences d'arrivées.

Pour ce faire je vais chercher deux instants t1 et t2 tel et calculer le MSE entre les observations o(t) et la fonction définie par f(t) =

mean(o(t) pour t < t1) si t < t1 | mean(o(t) pour t > t2) si t > t2 | mean(o(t) pour t > t1 et t< t2) t > t1 et t< t2 |

je fais referece à cette méthode par la "méthode des moyennes"

Tous les graphiques utilisent altair et sont interactif vous pouvez zoomer en "scrollant" déplacer le graphique en cliquant et tirant et vous pouvez selectionner le centre commercial en cliquant sur la légende.

Pour plus de confort j'ai "collapsé" les cellules les moins importantes j'utilise vs code j'espere que l'affichage sera bon de votre coté

PS: j'ai supprimé mes différents essais intermédiaires pour ne garder que l'essais final avec tout le preprocessing ect…

1.1 IMPORTS

[ ]:
```
import pandas as pd
import altair as alt
import datetime
import numpy as np
import scipy.stats as stats
alt.data_transformers.disable_max_rows()
```

[ ]:
```
DataTransformerRegistry.enable('default')
```

1.2 DATASET

Visualisation des informations importantes

[ ]:
```
data = pd.read_csv("./at_home_test_data_study_centers_201909.csv")
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81838 entries, 0 to 81837
Data columns (total 3 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   shopping_center_id  81838 non-null  object
 1   device_local_date   81838 non-null  object
 2   device_hash_id      81838 non-null  object
dtypes: object(3)
memory usage: 1.9+ MB
```

[ ]:

	shopping_center_id	device_local_date	device_hash_id
0	b43e9e4f-acd1-4941-874d-e0c5650ab91e	2019-09-14 10:00:25	6fdffac307
1	b43e9e4f-acd1-4941-874d-e0c5650ab91e	2019-09-14 17:13:15	386141ebd8
2	b43e9e4f-acd1-4941-874d-e0c5650ab91e	2019-09-14 9:07:06	b06242b848
3	b43e9e4f-acd1-4941-874d-e0c5650ab91e	2019-09-14 17:14:49	c13cc52e82
4	599cb959-11ef-49aa-9eb3-e6c17b4ea6ba	2019-09-14 10:17:35	f339ddf999

[ ]:
```
min_date_serie = data[['shopping_center_id','device_local_date']].
 ↪groupby("shopping_center_id").apply(lambda x: x['device_local_date'].min()).
 ↪rename("min_date")
max_date_serie = data[['shopping_center_id','device_local_date']].
 ↪groupby("shopping_center_id").apply(lambda x: x['device_local_date'].max()).
 ↪rename("max_date")
pd.concat([min_date_serie,max_date_serie],axis=1)
```

[ ]:

shopping_center_id	min_date	max_date
0cd35523-1eca-4f09-ab0d-0b506ae9d986	2019-09-01 16:18:00	2019-09-17 9:59:00

```
599cb959-11ef-49aa-9eb3-e6c17b4ea6ba   2019-09-01 0:19:53  2019-09-17 9:59:42
b43e9e4f-acd1-4941-874d-e0c5650ab91e  2019-09-01 10:03:09  2019-09-17 9:57:59
cb2d5bb6-c372-4a51-8231-4ffa288a0c28   2019-09-01 0:26:16  2019-09-17 9:59:55
```

[ ]:
```
data[['shopping_center_id','device_hash_id']].groupby('shopping_center_id').
↪nunique()
```

[ ]:
```
                                    device_hash_id
shopping_center_id
0cd35523-1eca-4f09-ab0d-0b506ae9d986            1077
599cb959-11ef-49aa-9eb3-e6c17b4ea6ba            2417
b43e9e4f-acd1-4941-874d-e0c5650ab91e            1024
cb2d5bb6-c372-4a51-8231-4ffa288a0c28            1334
```

[ ]:
```
selection = alt.selection_multi(fields=['shopping_center_id'], bind='legend')
chart = alt.Chart(data).encode(
    alt.X('yearmonthdatehoursminutes(device_local_date):T'),
    alt.Y('distinct(device_hash_id):Q',stack=None),
    color='shopping_center_id:N',
    opacity=alt.condition(selection, alt.value(1), alt.value(0))
).add_selection(selection)
chart.mark_bar(opacity=1, thickness=100,size=2).interactive()
```

[ ]:
```
alt.Chart(…)
```

Sans surprise on remarque une périodicité sur 7 jours on peut également supposer que les jours avec moins de ping sont les dimanches. Dans un premier temps je vais chercher à éliminer les ping parasites c'est à dire ceux qui se manifestent sur une periode excessiverment longue ou ceux se manifestant sur une periode anormalement courte

[ ]:
```
data['device_local_date'] = pd.to_datetime(data['device_local_date'])
data.dtypes
```

[ ]:
```
shopping_center_id            object
device_local_date     datetime64[ns]
device_hash_id                object
day                            int64
hour                           int64
dtype: object
```

[ ]:
```
data['device_local_date'] = pd.to_datetime(data['device_local_date'] )
data['day'] = data['device_local_date'].dt.day
data['hour'] = data['device_local_date'].dt.hour
data
```

[ ]:
```
                         shopping_center_id   device_local_date  \
0      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 10:00:25
```

```
1      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 17:13:15
2      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 09:07:06
3      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 17:14:49
4      599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-14 10:17:35
…                                       …                   …
81833  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-04 18:05:48
81834  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-04 18:46:00
81835  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-04 14:12:18
81836  b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-04 15:30:48
81837  0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-04 20:33:42

      device_hash_id  day  hour
0         6fdffac307   14    10
1         386141ebd8   14    17
2         b06242b848   14     9
3         c13cc52e82   14    17
4         f339ddf999   14    10
…               …  …    …
81833     b3917d13d5    4    18
81834     fa4acee13c    4    18
81835     dbdee4cd27    4    14
81836     32c7aecabc    4    15
81837     aea4ff6882    4    20

[81838 rows x 5 columns]
```

[ ]:
```
time_count = data.groupby(by=['shopping_center_id','day',"device_hash_id"]).
 ↪apply(lambda x:
x['device_local_date'].max()-x['device_local_date'].min()).rename("time_spent").
 ↪reset_index()
time_count
```

[ ]:
```
                         shopping_center_id  day device_hash_id      time_spent
0     0cd35523-1eca-4f09-ab0d-0b506ae9d986    1     088eee52b6 0 days 00:00:00
1     0cd35523-1eca-4f09-ab0d-0b506ae9d986    1     653fdb59a5 0 days 00:00:00
2     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2     01dc964eba 0 days 00:00:00
3     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2     02f00ae4d9 0 days 00:11:10
4     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2     0415b17f49 0 days 00:00:00
…                                       …  …             …             …
8637  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17     f30a293e76 0 days 00:00:00
8638  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17     f48a741ee1 0 days 00:00:00
8639  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17     f817a1c1f8 0 days 00:00:00
8640  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17     f864c5f07b 0 days 00:14:12
8641  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17     fb2f7ca8f3 0 days 00:00:00

[8642 rows x 4 columns]
```

[ ]:
```
time_count['time_spent'] = time_count['time_spent'].dt.seconds/60
time_count
```

[ ]:

	shopping_center_id	day	device_hash_id	time_spent
0	0cd35523-1eca-4f09-ab0d-0b506ae9d986	1	088eee52b6	0.000000
1	0cd35523-1eca-4f09-ab0d-0b506ae9d986	1	653fdb59a5	0.000000
2	0cd35523-1eca-4f09-ab0d-0b506ae9d986	2	01dc964eba	0.000000
3	0cd35523-1eca-4f09-ab0d-0b506ae9d986	2	02f00ae4d9	11.166667
4	0cd35523-1eca-4f09-ab0d-0b506ae9d986	2	0415b17f49	0.000000
…	…	…	…	…
8637	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	17	f30a293e76	0.000000
8638	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	17	f48a741ee1	0.000000
8639	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	17	f817a1c1f8	0.000000
8640	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	17	f864c5f07b	14.200000
8641	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	17	fb2f7ca8f3	0.000000

[8642 rows x 4 columns]

[ ]:
```
quantiles = pd.DataFrame()
for i in range(0,21):
    q = 0.05*i
    next_quantiles = pd.DataFrame(time_count.groupby('shopping_center_id').
 ↪apply(lambda x : x['time_spent'].quantile(q = q)).rename("value"))
    next_quantiles["quantile"] = q
    quantiles = pd.concat([quantiles,next_quantiles])
quantiles.index = quantiles.index.set_names(['shopping_center_id'])
quantiles = quantiles.reset_index()
quantiles
```

[ ]:

	shopping_center_id	value	quantile
0	0cd35523-1eca-4f09-ab0d-0b506ae9d986	0.000000	0.00
1	599cb959-11ef-49aa-9eb3-e6c17b4ea6ba	0.000000	0.00
2	b43e9e4f-acd1-4941-874d-e0c5650ab91e	0.000000	0.00
3	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	0.000000	0.00
4	0cd35523-1eca-4f09-ab0d-0b506ae9d986	0.000000	0.05
..	…	…	…
79	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	289.958333	0.95
80	0cd35523-1eca-4f09-ab0d-0b506ae9d986	793.100000	1.00
81	599cb959-11ef-49aa-9eb3-e6c17b4ea6ba	1066.833333	1.00
82	b43e9e4f-acd1-4941-874d-e0c5650ab91e	1237.116667	1.00
83	cb2d5bb6-c372-4a51-8231-4ffa288a0c28	1098.016667	1.00

[84 rows x 3 columns]

[ ]:
```
selection = alt.selection_multi(fields=['shopping_center_id'], bind='legend')
chart = alt.Chart(quantiles).encode(
    x = 'quantile:Q',
```

```
    y = 'value:Q',
    color='shopping_center_id:N',
    opacity=alt.condition(selection, alt.value(1), alt.value(0))
).add_selection(selection)
chart.mark_line().interactive()
```

[ ]:
```
alt.Chart(…)
```

On observe les valeurs des différents quantiles en minutes, ennormément de pings proviennent d'utilisateurs ayant été détectés moins de une seconde je vais donc en supprimer une partie, on ne supprime pas tout car l'antenne du centre ne couvre peut etre pas toute la zone géographique. On supprime également les personnes qui passent trop de temps dans le magasin il s'agit surement d'employés ou d'appareil autres.

[ ]:
```
time_count.groupby('shopping_center_id').apply(lambda x : x['time_spent'].
↪mean()/60)
```

[ ]:
```
shopping_center_id
0cd35523-1eca-4f09-ab0d-0b506ae9d986    0.821906
599cb959-11ef-49aa-9eb3-e6c17b4ea6ba    1.371260
b43e9e4f-acd1-4941-874d-e0c5650ab91e    0.976048
cb2d5bb6-c372-4a51-8231-4ffa288a0c28    0.815814
dtype: float64
```

un visiteur passe en moyenne 1 heure dans le centre commercial cela justifie ma décision précédente de supprimer les échantillons extremes

[ ]:
```
time_count.groupby('shopping_center_id').apply(lambda x : x['time_spent'].
↪quantile(q = 0.95))/60
```

[ ]:
```
shopping_center_id
0cd35523-1eca-4f09-ab0d-0b506ae9d986    4.539583
599cb959-11ef-49aa-9eb3-e6c17b4ea6ba    7.645125
b43e9e4f-acd1-4941-874d-e0c5650ab91e    5.819194
cb2d5bb6-c372-4a51-8231-4ffa288a0c28    4.832639
dtype: float64
```

[ ]:
```
time_count.groupby('shopping_center_id').apply(lambda x : x['time_spent'].
↪quantile(q = 0.3))
```

[ ]:
```
shopping_center_id
0cd35523-1eca-4f09-ab0d-0b506ae9d986    2.935000
599cb959-11ef-49aa-9eb3-e6c17b4ea6ba    5.751667
b43e9e4f-acd1-4941-874d-e0c5650ab91e    0.575000
cb2d5bb6-c372-4a51-8231-4ffa288a0c28    0.000000
dtype: float64
```

Je choisi donc de supprimer les utilisateurs passant moins de deux minutes et ceux passant plus de

4h30 dans les centre commerciaux

Pour automatiser ce paramètre il suffit de définir un pourcentage de donnée que l'on se permet d'abandonner et utiliser les quantiles correspondants

[ ]:
```
to_remove_hash_id  = pd.concat([time_count[(time_count["time_spent"]>4.
↪5*60)]["device_hash_id"],time_count[(time_count["time_spent"]<2)]["device_hash_id"]])
to_remove_hash_id
```

[ ]:
```
31      32aaca501c
42      4725f024bc
71      8f1846fb65
76      a29e90ce55
98      d06667544a
           …
8636    f2663218c7
8637    f30a293e76
8638    f48a741ee1
8639    f817a1c1f8
8641    fb2f7ca8f3
Name: device_hash_id, Length: 3197, dtype: object
```

[ ]:
```
clean_data = data[~data["device_hash_id"].isin(to_remove_hash_id)]
clean_data
```

[ ]:
```
                        shopping_center_id   device_local_date  \
0      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 10:00:25
1      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 17:13:15
2      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 09:07:06
6      b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-14 17:39:23
7      599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-14 12:41:58
…                                       …                   …
81826  b43e9e4f-acd1-4941-874d-e0c5650ab91e 2019-09-04 20:46:00
81828  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-04 21:16:11
81830  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-11 17:34:41
81831  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-11 10:25:11
81833  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba 2019-09-04 18:05:48

      device_hash_id  day  hour
0         6fdffac307   14    10
1         386141ebd8   14    17
2         b06242b848   14     9
6         d500fae368   14    17
7         62e3eaa686   14    12
…              …  …   …
81826     d814e5361a    4    20
81828     783906c201    4    21
81830     5770502030   11    17
```

```
81831     5ea002b1b8   11    10
81833     b3917d13d5    4    18

[44415 rows x 5 columns]
```

[ ]:
```
selection = alt.selection_multi(fields=['shopping_center_id'], bind='legend')
chart = alt.Chart(clean_data).encode(
    alt.X('yearmonthdatehoursminutes(device_local_date):T'),
    alt.Y('distinct(device_hash_id):Q',stack=None),
    color='shopping_center_id:N',
    opacity=alt.condition(selection, alt.value(1), alt.value(0))
).add_selection(selection)
chart.mark_bar(opacity=1, thickness=100,size=2).interactive(bind_y = False)
```

[ ]:
```
alt.Chart(…)
```

Je conserve maintenant uniquement les premier ping de chaque utilisateur pour detecter les arrivées

[ ]:
```
clean_data = clean_data.sort_values(by='device_local_date')
first_ping_data=clean_data.groupby(by=['shopping_center_id','day']).
 ↪apply(lambda x : x[['device_local_date','device_hash_id']].
 ↪drop_duplicates(subset=['device_hash_id'])).reset_index()
first_ping_data
```

[ ]:
```
                      shopping_center_id  day  level_2   device_local_date  \
0     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2    69478 2019-09-02 09:16:11
1     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2    17314 2019-09-02 09:34:15
2     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2     8565 2019-09-02 09:51:05
3     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2     9289 2019-09-02 10:12:20
4     0cd35523-1eca-4f09-ab0d-0b506ae9d986    2     9905 2019-09-02 10:20:19
…                                        …  …      …                   …
4214  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17    24212 2019-09-17 19:47:06
4215  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17    10074 2019-09-17 19:56:24
4216  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17    30692 2019-09-17 20:03:41
4217  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17     4078 2019-09-17 20:20:51
4218  cb2d5bb6-c372-4a51-8231-4ffa288a0c28   17    46406 2019-09-17 21:17:41

     device_hash_id
0        6cfc17f269
1        3abe4d2964
2        c70a591a28
3        baf8564fd3
4        bbb4d374ab
…               …
4214     17dae2574c
4215     acf78d9aa5
4216     c4ea1c5532
```

```
4217     f14302aea8
4218     a870a4f06b

[4219 rows x 5 columns]
```

[ ]:
```
selection = alt.selection_multi(fields=['shopping_center_id'], bind='legend')
chart = alt.Chart(first_ping_data).encode(
    alt.X('yearmonthdatehoursminutes(device_local_date):T'),
    alt.Y('count(device_hash_id):Q',stack=None),
    color='shopping_center_id:N',
    opacity=alt.condition(selection, alt.value(1), alt.value(0))
).add_selection(selection)
chart.mark_bar(opacity=1, thickness=100,size=2).interactive(bind_y = False)
```

[ ]:
```
alt.Chart(…)
```

Maintenant que je suis en possession des arrivées je vais les aggreger sur des tranches de 30 min, (d'expérience les centres commerciaux ont souvent des horaires par tranche de 30 min). Etant donné que j' observe pas d'anomalie graphiquement je peux raisonablement regrouper les données des différents jours de la semaine entre eux pour me ramener à une base de 7 jours.

[ ]:
```
def aggregate_device_hash_id(data:pd.DataFrame)->pd.DataFrame:
    data = data.resample('30T',on='device_local_date').nunique().
 ↪drop(['device_local_date','shopping_center_id'],axis=1)
    data.rename(columns={'device_hash_id':'nb_ping'},inplace=True)
    return data
```

[ ]:
```
agregated_data=first_ping_data.groupby(by='shopping_center_id').apply(lambda x:␣
 ↪aggregate_device_hash_id(x))
agregated_data.reset_index(inplace=True)
agregated_data
```

[ ]:
```
                      shopping_center_id   device_local_date  day  level_2  \
0     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 09:00:00    1        1
1     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 09:30:00    1        2
2     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 10:00:00    1        2
3     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 10:30:00    1        4
4     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 11:00:00    1        4
…                                      …                   …  …      …
3108  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 19:00:00    1        1
3109  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 19:30:00    1        2
3110  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 20:00:00    1        2
3111  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 20:30:00    0        0
3112  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 21:00:00    1        1

      nb_ping
0           1
```

```
1           2
2           2
3           4
4           4
…       …
3108        1
3109        2
3110        2
3111        0
3112        1

[3113 rows x 5 columns]
```

[ ]:
```
selection = alt.selection_multi(fields=['shopping_center_id'], bind='legend')
chart = alt.Chart(agregated_data).encode(
    alt.X('yearmonthdatehoursminutes(device_local_date):T'),
    alt.Y('nb_ping:Q',stack=None),
    color='shopping_center_id:N',
    opacity=alt.condition(selection, alt.value(1), alt.value(0))
).add_selection(selection)
chart.mark_bar(opacity=1, thickness=100,size=2).interactive(bind_y = False)
```

[ ]:
```
alt.Chart(…)
```

Graphique des arrivées agrégées par jour

[ ]:
```
agregated_data["dayofweek"] = agregated_data["device_local_date"].dt.dayofweek
agregated_data['minute'] = agregated_data['device_local_date'].dt.minute
agregated_data['hour'] = agregated_data['device_local_date'].dt.hour
agregated_data
```

[ ]:
```
                        shopping_center_id   device_local_date  day  level_2  \
0     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 09:00:00    1        1
1     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 09:30:00    1        2
2     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 10:00:00    1        2
3     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 10:30:00    1        4
4     0cd35523-1eca-4f09-ab0d-0b506ae9d986 2019-09-02 11:00:00    1        4
…                                     …                  …  …      …
3108  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 19:00:00    1        1
3109  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 19:30:00    1        2
3110  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 20:00:00    1        2
3111  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 20:30:00    0        0
3112  cb2d5bb6-c372-4a51-8231-4ffa288a0c28 2019-09-17 21:00:00    1        1

      nb_ping  dayofweek  minute  hour
0           1          0       0     9
1           2          0      30     9
```

```
2           2          0       0    10
3           4          0      30    10
4           4          0       0    11
…       …          …      …    …
3108        1          1       0    19
3109        2          1      30    19
3110        2          1       0    20
3111        0          1      30    20
3112        1          1       0    21

[3113 rows x 8 columns]
```

[ ]:
```
agregated_data['time'] = agregated_data[['dayofweek','hour','minute']].
↪apply(lambda x: datetime.
↪datetime(year=2021,month=1,day=x['dayofweek']+1,hour=x['hour'],minute=x['minute']),axis␣
↪= 1)
agregated_data
```

[ ]:
```
      dayofweek  hour  minute                   shopping_center_id  nb_ping  \
0             0     0       0  0cd35523-1eca-4f09-ab0d-0b506ae9d986      0.0
1             0     0       0  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba      0.0
2             0     0       0  b43e9e4f-acd1-4941-874d-e0c5650ab91e      0.0
3             0     0       0  cb2d5bb6-c372-4a51-8231-4ffa288a0c28      0.0
4             0     0      30  0cd35523-1eca-4f09-ab0d-0b506ae9d986      0.0
…         …   …     …                                    …      …
1339          6    23       0  cb2d5bb6-c372-4a51-8231-4ffa288a0c28      0.0
1340          6    23      30  0cd35523-1eca-4f09-ab0d-0b506ae9d986      0.0
1341          6    23      30  599cb959-11ef-49aa-9eb3-e6c17b4ea6ba      0.0
1342          6    23      30  b43e9e4f-acd1-4941-874d-e0c5650ab91e      0.0
1343          6    23      30  cb2d5bb6-c372-4a51-8231-4ffa288a0c28      0.0

                    time
0    2021-01-01 00:00:00
1    2021-01-01 00:00:00
2    2021-01-01 00:00:00
3    2021-01-01 00:00:00
4    2021-01-01 00:30:00
…                    …
1339 2021-01-07 23:00:00
1340 2021-01-07 23:30:00
1341 2021-01-07 23:30:00
1342 2021-01-07 23:30:00
1343 2021-01-07 23:30:00

[1344 rows x 6 columns]
```

[ ]:
```
draw_bar_chart('time','nb_ping',agregated_data)
```

[ ]: alt.Chart(…)

Graphique des arrivées sur une base de 7 jours je remarque enormément de variations brusques qui pourrait perturber mon algorithme. je choisi donc de rentre le signal moins “sharp” en applicant la méthode des moyenne mobiles.

[ ]:
```
def smooth(data_per_shop:pd.DataFrame)-> pd.DataFrame:
    data_per_shop = data_per_shop.sort_values(by="time")
    data_per_shop['nb_ping'] = data_per_shop["nb_ping"].rolling(5,min_periods =␣
 ↪3,center=True).mean()
    return data_per_shop
```

[ ]:
```
smoothed_data = agregated_data.groupby(by="shopping_center_id").apply(lambda x:␣
 ↪smooth(x))
```

[ ]:
```
draw_bar_chart('time','nb_ping',smoothed_data)
```

[ ]: alt.Chart(…)

Les données sont prêtes je vais pouvoir travailler. 1 je cherche à identifier les jours ou le centre est fermé je suppose que les nombre de ping moyen par jours suivent une loi normale et je considère comme férié les jours ayant un ping moyen inférieur au premier quartile

2 j’applique la méthode des moyennes définie en introduction et j’applique une régularization par rapport à la distance entre f(t) et la valeur théorique 0 pour t< t1 et t>t2 cela est nécéssaire car les distributions sont multimodale et donc l’algorithme peut se coincer entre deux mode le paramètre de régularisation ici est 0.4 on peut clairement l’optimiser

[ ]:
```
def get_day_samples(data:pd.DataFrame)-> np.ndarray:
    return np.array([data[data['dayofweek']== i].
 ↪sort_values(by='time')["nb_ping"].to_numpy() for i in range(0,7)])

def get_closed_day(samples:np.ndarray)->np.ndarray:

    mean_ping_number_per_days = samples.sum(axis=1)
    mean = mean_ping_number_per_days.mean()
    std  = mean_ping_number_per_days.std()

    norm = stats.norm(mean, std)

    closed_day  = [i for i,n in enumerate(mean_ping_number_per_days) if n <␣
 ↪norm.ppf(0.25)]
    return closed_day

def mean_method_score(open_index:int,close_index:int,sample:np.ndarray)->float:
    if open_index!= 0:
        e_before_open =  ((sample[:open_index]-sample[:open_index].mean())**2)
    else:
```

```
        e_before_open = np.array([])
    if close_index+1 != len(sample):
        e_after_close = ((sample[close_index+1:]-sample[close_index+1:].
↪mean())**2)
    else:
        e_after_close = np.array([])

    e_between = ((sample[open_index:close_index] - sample[open_index:
↪close_index].mean())**2)
    return np.concatenate([e_before_open,e_after_close,e_between]).mean() + 0.
↪4*(sample[close_index+1:].mean() + sample[:open_index].mean())

def mean_method(sample:np.ndarray)->tuple:
    min_score = np.inf
    open_index = 0
    close_index = 1
    for i in range(len(sample)-1):
        for j in range(i,len(sample)):

            score = mean_method_score(i,j,sample)

            if score<min_score:
                open_index = i
                close_index = j
                min_score = score

    return open_index,close_index

def get_time_from_index(index):

    time= datetime.timedelta(minutes=30)*index
    return time

def get_opening_time(data):
    samples = get_day_samples(data)
    closed_days = get_closed_day(samples)
    opened_days = [i for i in np.arange(7) if i not in closed_days]
    opened_day_samples = np.array([samples[i]for i in opened_days])

    opening_time = []
    for opened_day_sample in opened_day_samples:
        open_index,close_index = mean_method(opened_day_sample)
        opening_time.
↪append([get_time_from_index(open_index),get_time_from_index(close_index)])

    day_mapper = {0:"Mon",1:"Tues",2:"Wed",3:"Thur",4:"Fri",5:"Sat",6:"Sun"}
```

```
    open_day_dataframe = pd.DataFrame({'day':opened_days,'opening_time':
↪opening_time})
    closed_day_dataframe = pd.DataFrame({'day':closed_days,'opening_time':
↪"CLOSED"})

    result = pd.concat([open_day_dataframe,closed_day_dataframe])
    result["day"] = result["day"].map(day_mapper)
    result = result.set_index("day")
    return result
```

[ ]:
```
result =smoothed_data.groupby("shopping_center_id").apply(lambda x:␣
↪get_opening_time(x))
result
```

```
/tmp/ipykernel_72929/2871621687.py:25: RuntimeWarning: Mean of empty slice.
  e_between = ((sample[open_index:close_index] -
sample[open_index:close_index].mean())**2)
/home/elio/test/.venv/lib/python3.8/site-packages/numpy/core/_methods.py:189:
RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
/tmp/ipykernel_72929/2871621687.py:26: RuntimeWarning: Mean of empty slice.
  return np.concatenate([e_before_open,e_after_close,e_between]).mean() +
0.4*(sample[close_index+1:].mean() + sample[:open_index].mean())
```

[ ]:

		opening_time
shopping_center_id	day	
0cd35523-1eca-4f09-ab0d-0b506ae9d986	Mon	[9:00:00, 21:00:00]
	Tues	[9:30:00, 20:00:00]
	Wed	[9:30:00, 20:00:00]
	Thur	[8:30:00, 20:00:00]
	Fri	[9:00:00, 20:30:00]
	Sat	[9:00:00, 20:00:00]
	Sun	CLOSED
599cb959-11ef-49aa-9eb3-e6c17b4ea6ba	Mon	[10:00:00, 20:00:00]
	Tues	[10:00:00, 20:30:00]
	Wed	[9:30:00, 20:00:00]
	Thur	[9:30:00, 20:30:00]
	Fri	[10:00:00, 20:30:00]
	Sat	[10:00:00, 19:30:00]
	Sun	CLOSED
b43e9e4f-acd1-4941-874d-e0c5650ab91e	Mon	[8:00:00, 20:00:00]
	Tues	[9:00:00, 21:30:00]
	Wed	[9:30:00, 20:30:00]
	Thur	[8:00:00, 20:30:00]
	Fri	[8:30:00, 20:30:00]
	Sat	[8:30:00, 19:00:00]
	Sun	CLOSED

```
cb2d5bb6-c372-4a51-8231-4ffa288a0c28 Tues    [9:30:00, 21:30:00]
                                     Wed     [9:30:00, 20:30:00]
                                     Thur    [9:00:00, 20:30:00]
                                     Fri     [8:30:00, 22:00:00]
                                     Sat     [9:00:00, 20:00:00]
                                     Sun     [9:30:00, 19:00:00]
                                     Mon                CLOSED
```

Les résultats sont assez mitigés le modèle est tres sensible au bruit il performe beaucoup moins bien sur les centres ayant le plus de bruit par exemple cb2d5bb6-c372-4a51-8231-4ffa288a0c28 (le bruit est visible sur les jours férié) On pourrait se débarasser de ce bruit en estimant sa distribution en utilisant les donnes des jours fermés.

[ ]:
```python
def draw_bar_chart(time:str,value:str,data:pd.DataFrame):
    selection = alt.selection_multi(fields=['shopping_center_id'],␣
 ↪bind='legend')
    chart = alt.Chart(data).encode(
        alt.X(f'yearmonthdatehoursminutes({time}):T'),
        alt.Y(f'{value}:Q',stack=None),
        color='shopping_center_id:N',
        opacity=alt.condition(selection, alt.value(1), alt.value(0))
    ).add_selection(selection)
    return chart.mark_bar(opacity=1, thickness=100,size=2).interactive()
```