

Projet de programmation informatique

Dans ce projet, nous avons à disposition des données (sous forme de fichier csv) proposées par Kandu. Ces données sont composées de :

- l'id du capteur étudié
- la température ambiante (en °C)
- l'humidité relative (en %)
- le niveau sonore (en dB)
- le niveau lumineux (en lux)
- la quantité de CO2 (en ppm)
- la date et l'heure de leur enregistrement

Le but de ce projet est alors d'exploiter ces différentes données et de les représenter dans des graphiques.

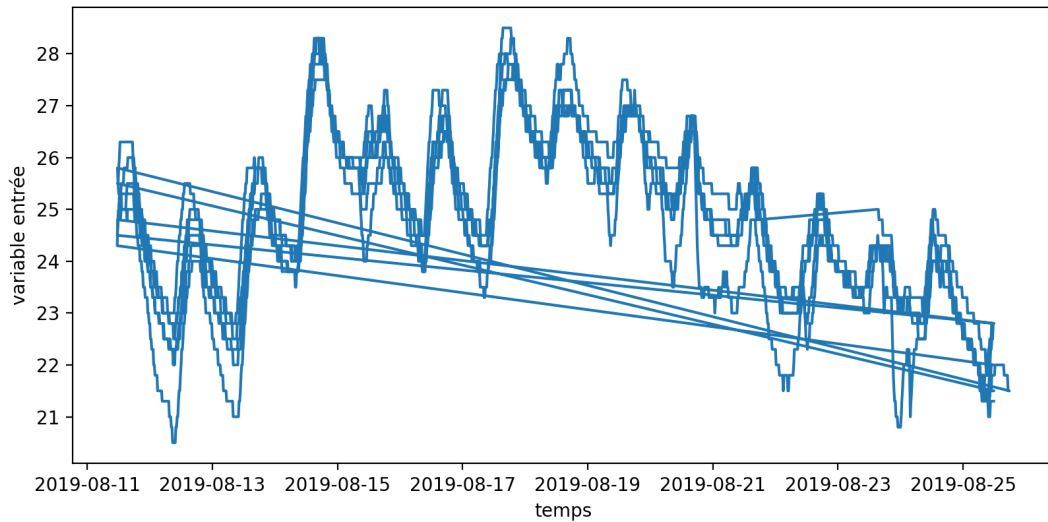
I. Le code

a. Traitement du temps dans les fonctions

Tout d'abord, nous avons réalisé un programme qui permet d'afficher la courbe montrant l'évolution d'une variable (fixée dans un premier temps) en fonction du temps. Après en avoir bien compris le mécanisme, nous avons généralisé ce programme afin que l'utilisateur puisse entrer la variable qu'il souhaite et que le programme affiche le graphique correspondant à la demande. Nous avons pour cela importé le module **pandas** de python.

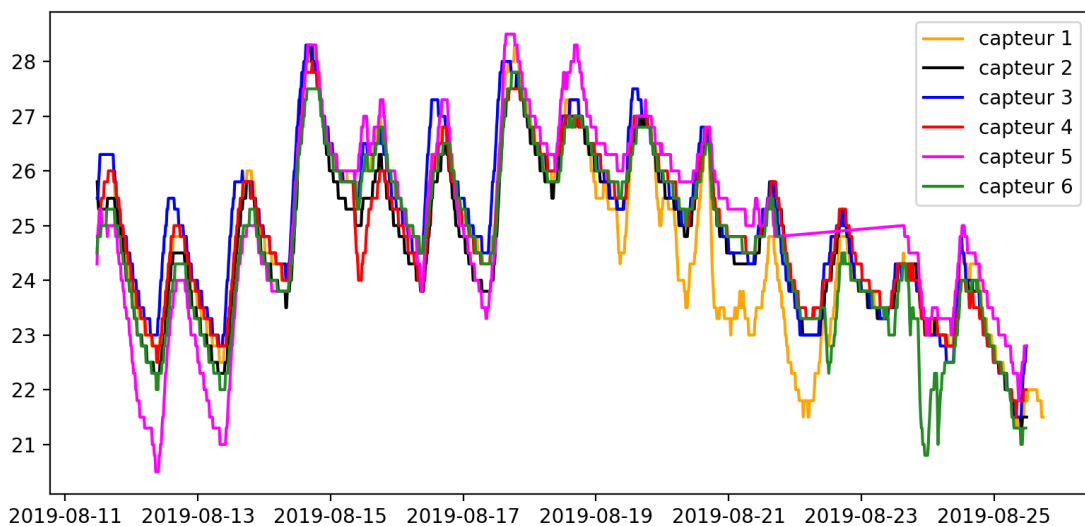
Voici le graphique obtenu lorsque l'on voulait représenter la **température en fonction du temps**. On remarque que des traits traversent le graphique et que le tracé est dédoublé. En effet, les données sont réparties sur 6 capteurs différents enregistrant chacun à des moments différents (bien que la fourchette de dates globale soit la même pour tous les capteurs), et lorsque l'on change de capteur, on revient à la date initiale.

Graphique correspondant à notre résultat initial pour l'affichage d'une variable (la température) en fonction du temps



Pour cela, nous avons décidé de regrouper les données par capteur, et on pourra ainsi afficher sur le graphique quelles valeurs correspondent à quel capteur. Après cette modification, nous avons alors obtenu un graphique beaucoup plus compréhensible et les traits parasites ont disparu.

Graphique correspondant à `graphique_par_capteurs(temperature)`



Dans notre code, les différentes variables à entrer dans les fonctions sont nommées comme suit: identity, noise, temperature, humidity, luminosity, co2 et temps.

Les **new_tempsi** avec $i \in [1;6]$ correspondent à de nouvelles listes de temps adaptées à chaque capteur car tous les capteurs n'ont pas été testés sur la même durée. Nous avons écrit ligne par ligne ces **new_temps** car lorsque l'on réalise une fonction, cela fausse les fonctions qui suivent.

Nous avons alors décidé de garder cette écriture ligne par ligne car il n'y a que 5 lignes, ce qui n'est pas excessif. De plus, nous avons converti ces listes au format datetime après avoir importé le module datetime:

```
40 new_temps1=[]
41 for k in range(len(temps1)):
42     new_temps1.append(dt.strptime(temps1[k], '%Y-%m-%d %H:%M:%S%z'))
```

L'étape suivante consistait alors à entrer une date de début et de fin pour modifier l'intervalle de temps pris en abscisse dans les graphiques. Nous avons alors créé la fonction **nouveau_temps**(start_date, end_date, temps). Cette fonction prend en variable une liste de temps et la raccourcit pour ne donner que la liste des dates comprises entre les bornes données.

Une fois cette fonction finie et testée, il fallait raccourcir les autres listes de données pour tracer des graphiques. En effet, sans réduction de la liste, on obtient l'erreur suivante:

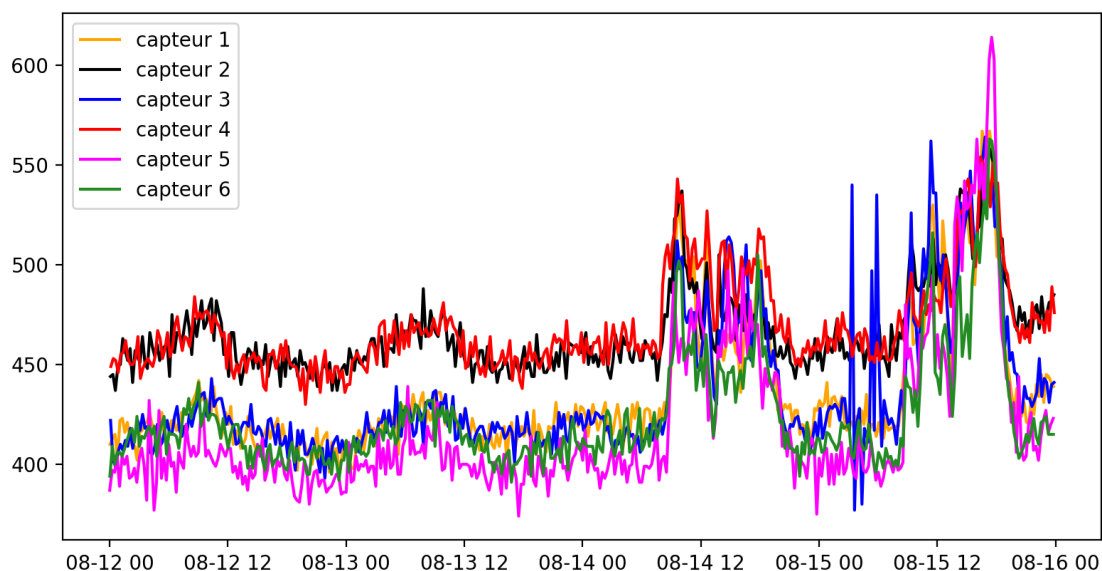
```
ValueError: x and y must have same first dimension,
but have shapes (7880,) and (1336,)
```

En effet, les deux listes du graphique doivent avoir la même dimension.

Nous avons alors créé une fonction qui adapte la liste de données d'une variable en fonction des bornes de temps données pour un seul capteur choisi. Nous avons ensuite généralisé ce mécanisme et nous avons créé la fonction **variable_bornes_capteur**(variable, start_date, end_date, nbcapt, L).

Nous pouvons désormais, grâce aux fonctions précédentes, afficher un graphique d'une variable en fonction du temps sur un intervalle donné. Le paramètre L correspond à la liste des numéros des capteurs que l'on souhaite afficher sur le graphique.

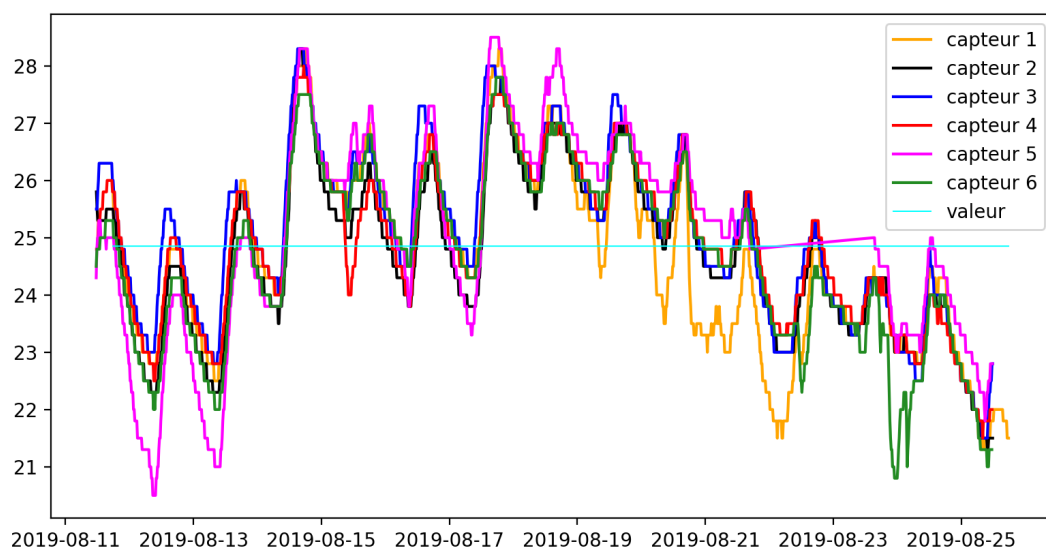
*Graphique correspondant à
graphique_par_capteurs_bornes(co2, '2020-08-12', '2020-08-15', [1,2,3,4,5,6])*



Le graphique affiche alors les valeurs comprises entre le 8 août minuit et le 15 août 23h59. Il faut faire attention à entrer les dates sous la bonne forme sans oublier les '_' sinon le programme ne fonctionnera pas.

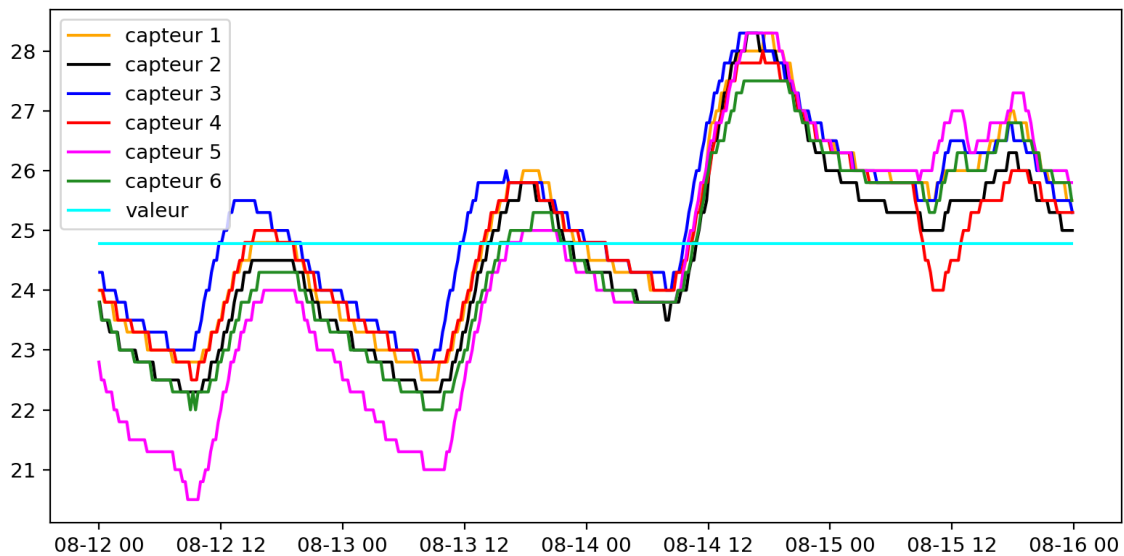
Ensuite, nous voulions afficher des valeurs caractéristiques des courbes comme la variance, le minimum, l'écart-type, la moyenne arithmétique ou encore la médiane. Nous avons choisi la moyenne arithmétique car c'est la plus courante et la plus représentative. Ainsi, nous avons créé **graphique_valeur(variable, valeur)** qui renvoie un graphique de la variable voulue en fonction du temps ainsi que la valeur caractéristique souhaitée. Les valeurs à entrer dans la fonction sont à nommer comme suit: moyenne_arithm, mediane, variance, mini, maxi et ecart_type.

*Graphique correspondant à **graphique_valeur(temperature, moyenne)***



Puis, nous avons réalisé un second programme qui affiche ce même graphique sur un intervalle de temps réduit (les bornes sont toujours données par l'utilisateur) et qui affiche la valeur voulue sur l'intervalle demandé (la valeur sera calculée à partir des données de l'intervalle seulement).

*Graphique correspondant à **graphique_valeur_bornes(temperature, moyenne, '2020-08-12, '2020-08-15')***



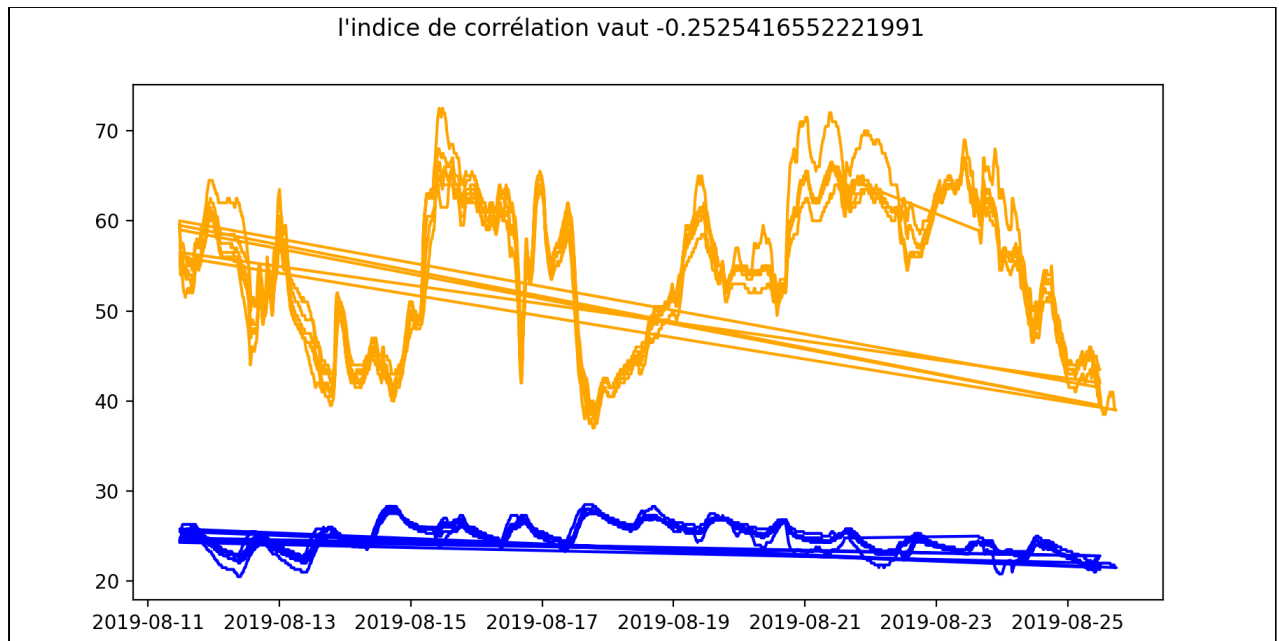
L'objectif principal de la fonction **graphique_valeur_bornes** est de créer une nouvelle liste composée des éléments de la variable demandée qui sont compris dans l'intervalle de temps voulu. C'est sur cette nouvelle liste qu'on réalisera le calcul de la valeur.

b. Corrélation

Ensuite, nous avons créé un programme qui donne l'indice de corrélation appelé **indice_correlation**(var1,var2). Cette fonction utilise la méthode pearson (qui nous semble être la plus adaptée pour ce projet) pour calculer l'indice de corrélation entre var1 et var2. Lors de nos tests, **indice_correlation** nous donne bien des valeurs comprises en -1 et 1 comme prévu. Par exemple, **indice_correlation**(temperature, noise) donne 0.1732798026781034

La fonction **graph_corr**(var1, var2) va alors représenter les 2 variables en fonction du temps et afficher l'indice de corrélation sur le graphique.

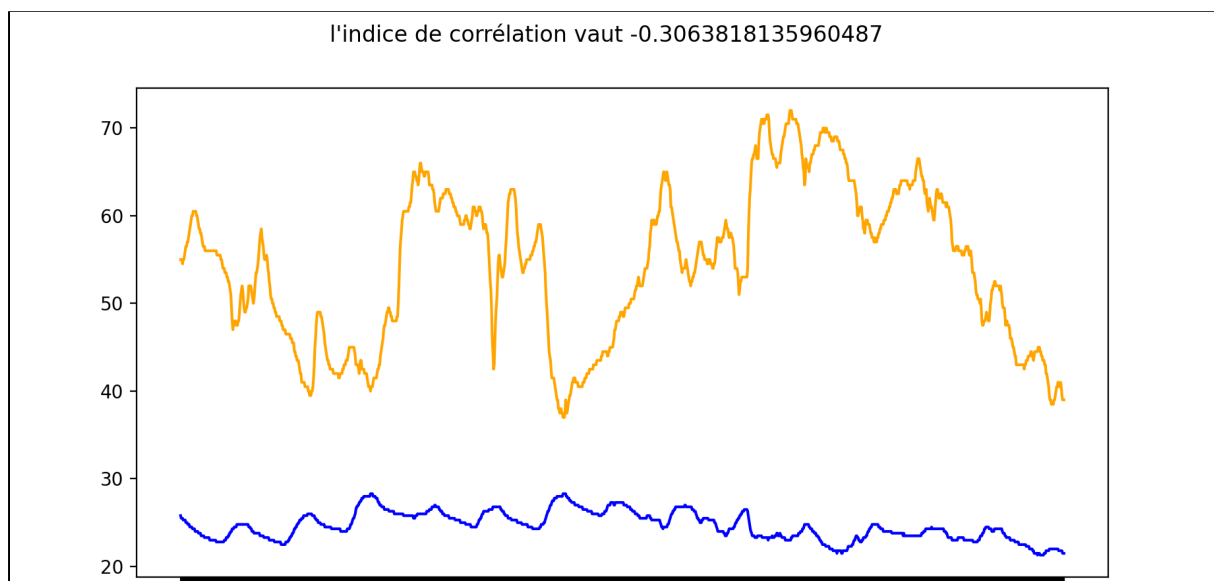
*Graphique correspondant à **graph_corr**(temperature, humidity)*



Nous avons décidé de mettre la valeur de l'indice de corrélation au-dessus du graphique pour ne pas empêcher sa lisibilité. On remarque que les traits sont de nouveau présents car nous n'avons pas séparé les capteurs. Cependant, il nous semble plus pertinent de créer une fonction **graph_corr_capteur**(var1, var2, numcapt) qui affiche la même chose que précédemment mais pour un capteur en particulier.

Nous l'avons donc réalisée :

*Graphique correspondant à **graph_corr_capteur**(temperature, humidity, 1)*



c. Humidex

D'autre part, il nous fallait réaliser une fonction humidex qui calculerait cet indice sur des binômes de variables et en afficherait le graphe pour un intervalle de temps donné. Après quelques recherches approfondies, nous avons compris qu'il y avait 2 formules possibles pour le calcul de l'indice : l'une faisait notamment intervenir la température de rosée et l'autre le taux d'humidité relative. Nous avons choisi la 2ème car nous disposons des données concernant l'humidité relative.

Si l'humidité relative est connue, on calcule l'indice humidex de la manière suivante :

$$\text{Humidex} = T_a + h$$

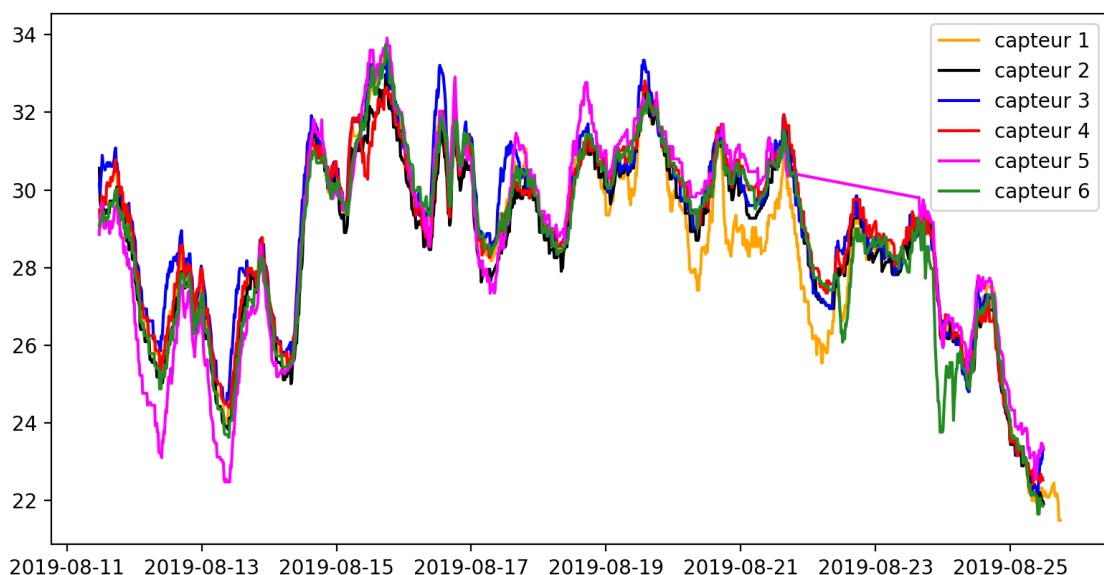
$$h = \frac{5}{9} (e - 10.0)$$

$$e = 6.112 \cdot 10 \left(\frac{7.5 + T_a}{237.7 + T_a} \right) ; \text{HR}/100$$

Où T_a = Température de l'air (°C), HR = Humidité relative (%) et e = pression de vapeur.

Après avoir créé la fonction **humidex** calculant l'indice pour un binôme ponctuel de valeurs de température ambiante et d'humidité relative, nous l'avons déclinée en **L_humidex** qui simulait la création d'une colonne "humidex" qui ne figure pas dans le tableur du fichier csv fourni. Nous avons pu, à partir de là, coder **graphique_humidex** qui est la fonction finale voulue.

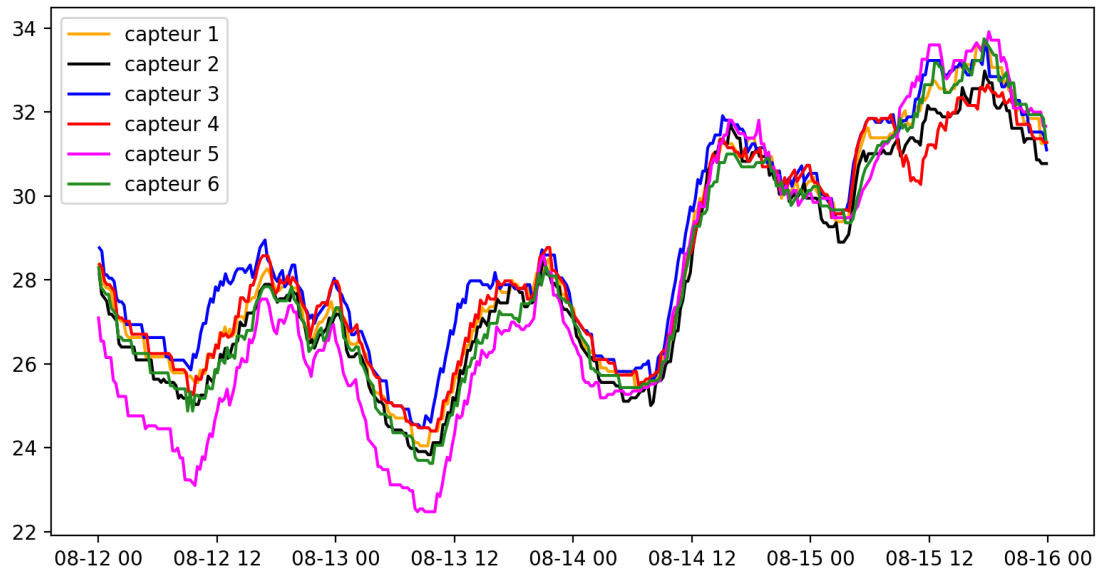
Graphique correspondant à `graphique_humidex()` :



Ici encore, on différencie les 6 capteurs puisque chaque capteur a une courbe d'humidex différente.

Nous avons également réalisé **humidex_bornes**(start_date, end_date) qui affiche la courbe de l'humidex entre 2 bornes de temps.

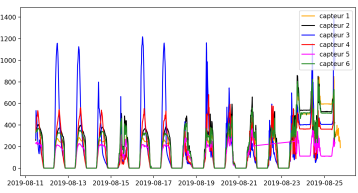
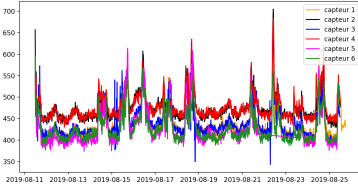
Graphique correspondant à humidex_bornes('2019-08-12', '2019-08-15')



d. Mesure des similarités

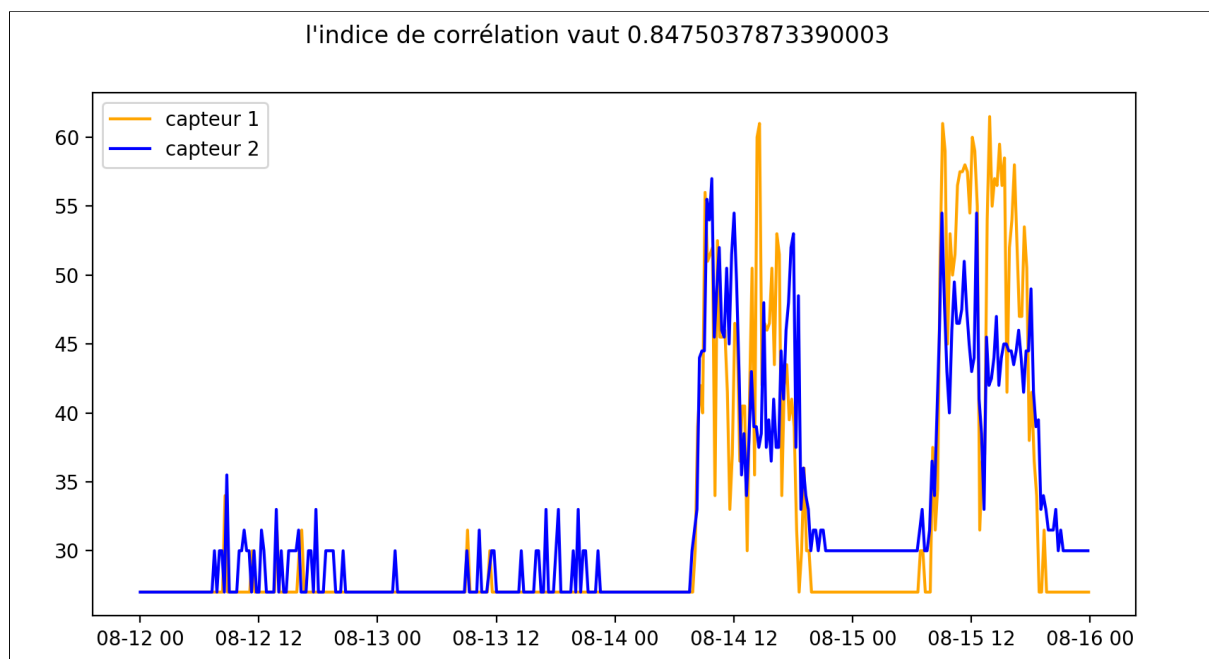
Dans cette partie, il nous faut essayer de déterminer les similarités possibles entre les capteurs. Nous avons commencé par traiter cette question par une analyse visuelle des différentes similarités entre les capteurs. Nous les avons répertoriées dans le tableau suivant:

la température ambiante (en °C)		similarités entre: - 3, 6 et 2
l'humidité relative (en %)		similarités entre: - 2, 3 et 6
le niveau sonore (en dB)		similarités entre: - 4 et 2 - 6 et 1 - 3 et 5

le niveau lumineux (en lux)		similarités entre: - 1, 5 et 6
la quantité de CO2 (en ppm)		similarités entre: - 2 et 4 - 1 et 3 - 5 et 6

La deuxième étape est alors de mesurer ces similarités grâce à un programme informatique. On a commencé par créer **mesure_similarites**(start, end, variable, capteur1, capteur2) qui permettait l’affichage de la corrélation entre deux capteurs pour une même dimension entre les dates choisies. D’où le test suivant :

*Graphique de **mesure_similarites**('2019-08-12', '2019-08-15', noise, 3, 6)*



A partir de là, nous avons créé **similarites**(start, end, capteur1, capteur2, ind) qui permet d’estimer la similarité entre les données relevées par capteur1 et capteur2 pour toutes les variables en prenant en référence un indice de corrélation minimal “ind”. On l’a testé :

```

In [118]: ed='2019-08-19'

In [119]: runfile('C:/Users/savan/OneDrive/Documents/BOULOT/projet info/Nouvelle
version.py', wdir='C:/Users/savan/OneDrive/Documents/BOULOT/projet info')

In [120]: sd='2019-08-12'

In [121]: similarites(sd,ed,4,6,0.99)
Les capteurs 4 et 6 ne sont pas similaires à 99.0 % pour ces données sur cette
période

In [122]: similarites(sd,ed,1,3,0.8)
Pour la variable noise, les capteurs 1 et 3 sont similaires à 80.0 % et la
corrélation est de 0.8328968194855947
Pour la variable temperature, les capteurs 1 et 3 sont similaires à 80.0 % et la
corrélation est de 0.96517831785568
Pour la variable humidity, les capteurs 1 et 3 sont similaires à 80.0 % et la
corrélation est de 0.9879924927345134
Pour la variable co2, les capteurs 1 et 3 sont similaires à 80.0 % et la
corrélation est de 0.8768809249901478

```

Enfin, on a utilisé la fonction **similarites** pour construire la fonction finale du projet :

tot_similarites_automatiques(start, end ind). Elle fait la même analyse sur tous les couples de capteurs possibles. Elle indique donc quels sont les jeux de données similaires entre deux dates choisies selon la sensibilité choisie ("ind"), pour chaque dimension.

e. Période d'occupation des bureaux

Enfin, pour finaliser ce projet, il nous fallait déterminer la période d'occupation. Pour cela, nous avons observé les différentes courbes et nous avons conclu que seule la donnée de luminosité suffisait à déterminer cette période. En effet, lorsque la luminosité est nulle, il n'y a personne, lorsqu'elle n'est pas nulle: les bureaux sont occupés.

Nous avons fait d'autres conjectures afin de répondre à cette question:

- Nous allons étudier les données d'un seul capteur (le 6) car tous les capteurs ont des périodes similaires
- Nous allons étudier les données sur une période de temps (du 12 au 19 août 2019) pour réduire la complexité du programme. Cette réduction ne semble pas impacter énormément le résultat car les périodes durent à peu près toujours 24h.

Partant de toutes ces conjectures, nous avons réalisé la fonction **horaires_bureaux()** qui renvoie les dates de début et de fin d'occupation des bureaux en moyenne. Toutes les précisions à propos de cette fonction se trouvent dans le code.

Voici alors ce que nous donne **horaires_bureaux()**:

```

>>> horaires_bureaux()
"l'heure de début est 7.0 h 47.875 min et l'heure
de fin est 20.0 h 51.625 min"

```

Ainsi, les bureaux sont occupés entre 7h47 et 20h51.

II. L'utilisation de Github

Nous avons appris à utiliser github après avoir fait les installations nécessaires (notamment l'installation de la console Git Bash sur nos ordinateurs respectifs), créé nos comptes respectifs et le "repository" qui servirait à la gestion du projet. Ce repository se nomme alors **projet_td1**

Nous avons régulièrement pratiqué le "push" and "pull", appris à maîtriser de nouvelles commandes comme git status, git add et git checkout et entrepris de créer de nouvelles branches pour simplifier le partage des parties sur lesquelles nous avons respectivement travaillé.

Quelques difficultés se sont néanmoins présentées, notamment avec une erreur persistante de git config pour l'une de nous deux que nous avons tenté de résoudre.

En dehors de cela, la gestion via github se passant sans accroc, nous avons pu avancer de façon efficace tout au long de la conception du code final. En effet, Github est assez pratique puisque qu'il permet de pouvoir travailler à distance (ce qui est encore plus utile au vu de la situation sanitaire actuelle), cela nous a permis d'échanger nos travaux facilement et de voir les modifications faites au fur et à mesure du projet (modifications qui sont récupérables à tout moment en cas de suppression d'un code par inadvertance) et ainsi assure une sécurité des travaux réalisés.