BOUGHDADI Ikram
GALET Chloé

Rapport du projet de programmation
Sujet 1

Introduction au sujet

Il nous est demandé de réaliser un programme permettant l'analyse de données issues de campagnes de mesure du bâtiment. Pour se faire, nous avons divisé le programme en plusieurs fonctions.
Ainsi à partir du fichier CSV contenant les données de chaque capteur, le programme est capable :

- d'afficher **l'évolution des variables en fonction du temps**
- d'afficher **les valeurs statistiques** de la courbe
- de calculer l**'indice humidex**
- de calculer l**'indice de corrélation** entre deux variables
- de **relever et d'afficher les anomalies** présentes dans les données

Pour chacune de ces fonctionnalités, il est possible de **spécifier un intervalle de temps** dans la ligne de commande.

Pour réaliser ce programme nous avons utilisé **GitHub** pour partager nos algorithmes. En effet, pour commencer, nous nous sommes réparties les fonctions à réaliser; GitHub nous a donc permis de partager au fur et à mesure notre travail, ce qui nous a été profitable étant donné que nous n'avions pas pour habitude de travailler simultanément.
De plus, même si nous ne travaillions pas toujours sur la même fonction, GitHub nous a été utile pour remarquer rapidement les changements effectués sur le programme et pour intervenir sur la partie de notre binôme lorsque l'une de nous deux était bloquée.
Cependant, nous avons aussi beaucoup utilisé la plateforme **Teams** au début de notre travail et pour rédiger le rapport car nous pouvions discuter de toutes les étapes à entreprendre, et c'était plus simple pour expliquer nos méthodologies respectives.

Lien du projet sur Git : **https://github.com/bbik/Projet-Python.git**

BOUGHDADI Ikram
GALET Chloé

Pour chaque fonction, nous avons définies les valeurs suivantes :

```
temperature= fichierCSV['temp'].to_list()
humidite=fichierCSV['humidity'].to_list()
niv_sonore= fichierCSV['noise'].to_list()
niv_lum=fichierCSV['lum'].to_list()
quantite_CO2=fichierCSV['co2'].to_list()
humidex = 'humidex'
```

Affichage des valeurs dans l'intervalle de temps spécifié:

Pour pouvoir définir la durée à prendre en considération dans la réalisation du graphique, il fallait traiter les données de temps du tableau.

Afin de réaliser cela, nous avons tout d'abord transformer chaque colonne du tableau en une **liste** afin de pouvoir accéder plus facilement aux différentes données. Puis nous avons transformé toutes les dates en **secondes** afin de faciliter le calcul et on a ensuite créé une liste contenant toutes ces secondes. Nous avons aussi **trié** à l'aide d'un **tri fusion** d'une manière croissante toutes les dates pour avoir une cohérence dans l'axe du temps.

Après cela nous avons mis deux conditions, si **aucune** requête n'a été faite concernant le temps le programme va tracer les courbes en considérant **toutes les dates** du tableau, si une **requête spécifique** a été faite pour délimiter le temps entre deux moments distincts le programme ne va tracer la courbe qu'**entre ces deux moments.**

Affichage des valeurs statistiques:

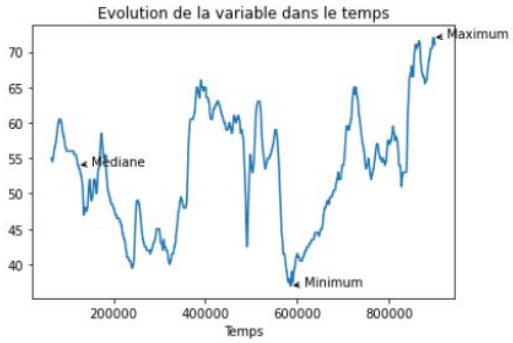
Fonction displayStat

Pour déterminer la **médiane**, ainsi que **les valeurs minimale et maximale** de la variable sélectionnée nous avons utilisé, respectivement, les fonctions min, max et numpy.median puis nous avons recherché l'index correspondant à chacune des ces valeurs, pour pouvoir déterminer le temps (en secondes) auxquelles ces index étaient associés et les afficher correctement sur le graphique.

Pour la **moyenne**, l'**écart-type** et la **variance** nous avons utilisé les fonctions de la bibliothèque numpy mais leurs valeurs n'apparaissent pas sur le graphique puisque nous n'avons pas trouvé une fonction d'affichage qui nous paraissait judicieuse.

Résultat

Ainsi, par exemple, la commande suivante :
display(humidex,'2019-08-11 18:18:03+02:00','2019-08-17 18:17:39+02:00') , permet d'indiquer avec les flèches sur le graphique les valeurs minimale, maximale et la médiane.



BOUGHDADI Ikram
GALET Chloé

Calcul de l'indice humidex:

Fonction display

Afin de mieux décrire l'impression de chaleur, des météorologues canadiens ont établi le facteur **Humidex**. C'est un paramètre qui permet d'exprimer l'**effet combiné de la chaleur et de l'humidité** en une seule donnée.

La formule pour calculer l'indice humidex est la suivante :

$$\text{Humidex} = T_{\text{air}} + 0.5555\left[6.11 \times e^{5417.7530\left(\frac{1}{273.16} - \frac{1}{273.15+T_{\text{rosee}}}\right)} - 10\right]$$

Source : Wikipédia

On remarque que l'indice humidex dépend de deux variables : la **température de l'air** (Tair) qui nous est directement donnée par un capteur et la **température de rosée** que nous allons calculer grâce à la formule suivante :

$$T_p = \frac{b(\frac{aT}{b+T} + lnRH)}{a - (\frac{aT}{b+T} + lnRH)}$$

Source : PlanetCalc

Dans cette formule RH représente l'humidité relative, a et b sont deux constantes telles que a = 17,27 et b = 237,7 °C.
Nous avons défini la fonction ϕ égale à : $\frac{aT}{b+T} + lnRH$

Programmation

Nous avons donc réalisé ces calculs au fur et à mesure en commençant par exprimer les valeurs prises par ϕ, puis nous avons calculé la température de rosée associée pour, enfin, trouver l'indice humidex.

Pour commencer, il faut donc **calculer ϕ** qui dépend de deux variables : la température de l'air (Tair) et l'humidité (RH). Il faut donc que la liste ϕ parcourt **simultanément** les données des deux listes associées à ces grandeurs (entre les valeurs de temps indiquées d et f).

```
for Tair,RH in zip(temperature[d:f],humidite[d:f]):
    phi.append(((cste*Tair)/(cste2+Tair))+math.log(RH/100))
```

$$\frac{aT}{b+T} + lnRH$$

Les deux constantes "cste" et "cste 2" correspondent respectivement aux valeurs a = 17,27 et b = 237,7 °C.

Ensuite, on peut calculer les valeurs de la **température de rosée** Trosée (grâce à la formule ci-dessous) en fonction des valeurs de ϕ.

```
for i in range(len(phi)):
    Trosee.append((cste2*phi[i])/(cste-phi[i]))
```

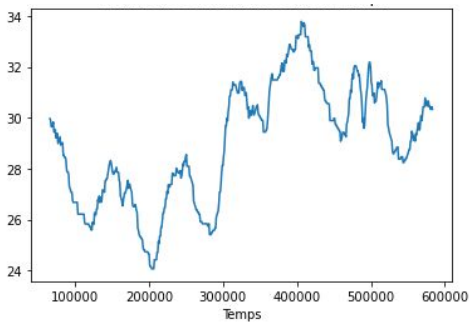
$$T_p = \frac{b(\frac{aT}{b+T} + lnRH)}{a - (\frac{aT}{b+T} + lnRH)}$$

De même, on obtient la liste **Humidex** en parcourant simultanément les listes de la température de rosée et de la température de l'air.

```
for Tair, Tr in zip(temperature[d:f],Trosee):
    Humidex.append(Tair + 0.5555*(6.11*math.exp(cste3*((1/273.16)-(1/(273.15+Tr))))-10))
print('Indice humidex = ', Humidex)
```

Résultat

Ainsi, par exemple, la commande suivante : **display(humidex,'2019-08-11 18:18:03+02:00','2019-08-17 18:17:39+02:00')** permet d'afficher la liste des valeurs prises par l'indice humidex ainsi que son évolution entre le 11 août à 18h18 et le 17 août à 18h17 sur le graphique ci-contre.



Cet algorithme n'est sûrement pas le plus judicieux en terme de complexité mais il nous a permis d'effectuer les calculs étape par étape et ainsi de **détecter les erreurs** au fur et à mesure, en comparant notamment les valeurs des variables intermédiaires obtenus avec ce programme avec les valeurs obtenues à la calculatrice.

Calcul de l'indice de corrélation

Fonction corrélation

L'indice de corrélation permet de donner un sens à la **relation linéaire entre deux variables**, il est toujours compris entre -1 et 1. Ce coefficient de corrélation s'interprète de la façon suivante :

- Plus le coefficient est proche de **1**, plus la relation linéaire **positive** entre les variables est **forte**.
- Plus le coefficient est proche de **-1**, plus la relation linéaire **négative** entre les variables est **forte**.
- Plus le coefficient est proche de **0**, plus la relation linéaire entre les variables est **faible**.

L'indice de corrélation se calcule grâce à cette formule : $$Correlation = \frac{Cov\ (x,y)}{\sigma x * \sigma y}$$

BOUGHDADI Ikram
GALET Chloé

Programmation

Dans un premier temps, on calcule les **écarts types** des deux variables avec la fonction numpy.std() provenant de la bibliothèque numpy, puis la **covariance** à l'aide de la formule suivante.

$$Cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{X}).\ (Y_i - \overline{Y})$$

avec Xi et Yi : cours des actifs X et Y à l'instant i
et X et Y : moyennes du cours des actifs X et Y

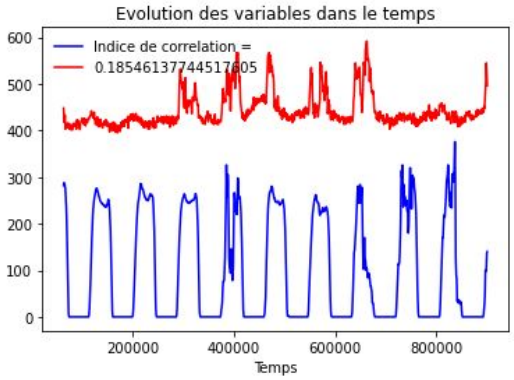
On calcule les moyennes de X et Y grâce à la fonction numpy.mean() qui provient également de la bibliothèque numpy. La covariance est alors donnée en appliquant la formule ci-dessus et en parcourant simultanément les listes de la première et de la deuxième variable :

```
sum=0
n=len(variable1)
for i in range(0,n):
    sum += ((variable1[i]-moy1)*(variable2[i]-moy2))
Cov=sum/(n)
```

Après avoir obtenu la covariance, il suffit d'appliquer la formule de l'indice de corrélation.

Résultat

Par exemple la commande **correlation(niv_lum,quantite_CO2,'2019-08-11 18:18:03+02:00','2019-08-17 18:17:39+02:00')** nous donne un indice de corrélation environ égal à **0,185**, ce qui signifie que la relation linéaire entre les deux variables est **faible**.
De plus , le graphique ci-contre affiche l'évolution des deux variables dans le temps : le niveau de luminosité est en rouge et la quantité de CO2 en bleu.



BOUGHDADI Ikram
GALET Chloé

Détection des anomalies

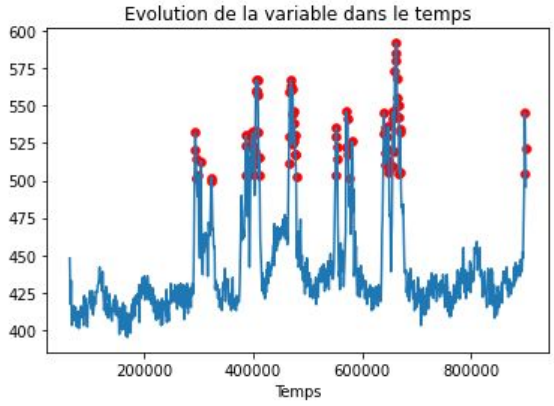
Fonction anomalies

Pour détecter les **anomalies**, nous avons utilisé la moyenne ± 2.5 écart type pour déterminer les **limites inférieure et supérieure des valeurs aberrantes**. Les limites de ces valeurs seront donc 2.5 écarts type au-dessus ou en dessous de la médiane des valeurs dans le champ de valeurs aberrantes.
Si une valeur est en dessous de la limite inférieure ou au-dessus de la limite supérieure, alors cette valeur est sauvegardée dans une nouvelle liste "anomalies".

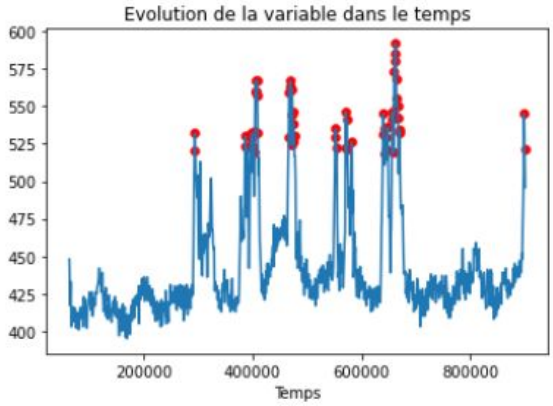
```
for i in range (len(L)):
    if L[i]>Seuil_haut or L[i]<Seuil_bas:
        anomalies.append(L[i])
        duree_anomalies.append(duree[i])
```

Les valeurs comprises dans cette liste sont ensuite affichées sur le graphique sous forme de point.

Nous avons placé les limites des valeurs aberrantes là où cela nous semblait le plus approprié; pour ce faire nous avons testé différentes positions puis **comparé** les résultats.
Par exemple, en utilisant la quantité de CO2 comme variable, nous obtenons avec le multiple 2 le graphique de droite, et avec le multiple 2.5 le graphique de gauche, dans les deux cas les points rouges représentent les anomalies.



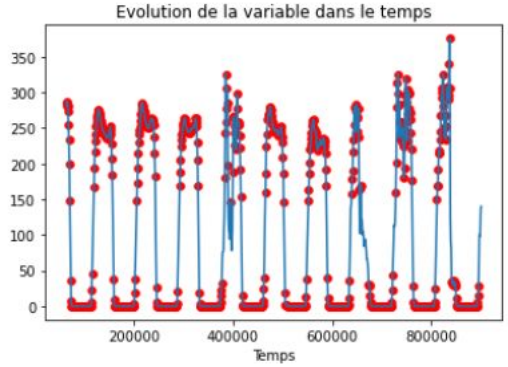
Avec moyenne + 2 écarts type



Avec moyenne +2,5 écarts type

Cette méthode nous semble la plus judicieuse puisqu'elle est **adaptée à toutes les variables**.

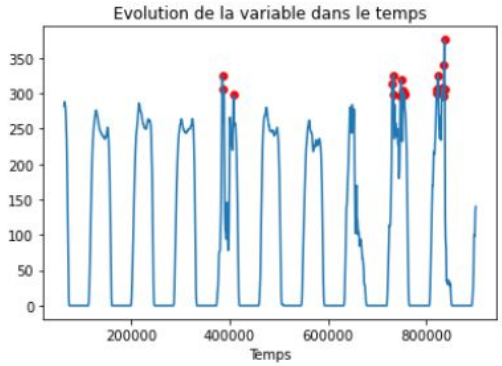
Avant de faire ce choix, nous avions essayé de définir les valeurs aberrantes comme celles **s'écartant de 50% de la moyenne**.
Cependant cette méthode ne convient pas à toutes les variables, par exemple pour le niveau de luminosité, on obtient le graphique ci-contre.



BOUGHDADI Ikram
GALET Chloé

Dans ce cas lorsque le niveau de luminosité est à 0, l'algorithme considère cette valeur comme une anomalie. Bien que la valeur 0 soit en effet une valeur éloignée de la moyenne, selon nous ce n'est pas une anomalie puisque cette valeur est récurrente (et elle correspond au cycle nuit).

En utilisant l'**écart-type et la médiane** (au lieu de la moyenne qui est plus sensible aux valeurs extrêmes) le résultat nous semble plus convenable, comme le montre le graphique ci-contre.



Cependant cette méthode ne semble pas être idéale non plus car nous avons fixé les limites de manière "**arbitraire**" en fonction des valeurs qui nous semblaient aberrantes. En effet, nous ne savions pas **au-delà de quel seuil nous pouvions considérer une valeur comme une anomalie**.